Bounds and Estimates on the Average Edit Distance

Michele Schimd(✉) and Gianfranco Bilardi

Department of Information Engineering, University of Padova, Padua, Italy
{schimdmi,bilardi}@dei.unipd.it

Abstract. The edit distance is a metric of dissimilarity between strings, widely applied in computational biology, speech recognition, and machine learning. Let $e_k(n)$ denote the average edit distance between random, independent strings of n characters from an alphabet of a given size k. An open problem is the exact value of $\alpha_k(n) = e_k(n)/n$. While it is known that, for increasing n, $\alpha_k(n)$ approaches a limit α_k, the exact value of this limit is unknown, for any $k \geq 2$. This paper presents an upper bound to α_k based on the exact computation of some $\alpha_k(n)$ and a lower bound to α_k based on combinatorial arguments on edit scripts. Statistical estimates of $\alpha_k(n)$ are also obtained, with analysis of error and of confidence intervals. The techniques are applied to several alphabet sizes k. In particular, for a binary alphabet, the rigorous bounds are $0.1742 \leq \alpha_2 \leq 0.3693$ while the obtained estimate is $\alpha_2 \approx 0.2888$; for a quaternary alphabet, $0.3598 \leq \alpha_4 \leq 0.6318$ and $\alpha_4 \approx 0.5180$. These values are more accurate than those previously published.

Keywords: Edit distance · Average analysis · Upper and lower bounds · Statistical estimates

1 Introduction

Measuring distance between strings is a fundamental problem in computer science, with applications in computational biology, speech recognition, machine learning, and other fields. One commonly used metric is the *edit distance* (or *Levenshtein distance*), defined as the minimum number of substitutions, deletions, and insertions necessary to transform one string into the other.

It is natural to ask what is the average distance between two randomly generated strings, as the string size grows; knowledge of the asymptotic behavior has proved useful in computational biology (Ganguly *et al.* [10]) and in nearest neighbour search (Rubinstein [14]). In computational biology, for example, the edit distance can be used to the test the hypothesis that two subsequences

This work was partially supported by University of Padova projects CPDA152255/15 and CPGA3/13; by MIUR, the Italian Ministry of Education, University and Research, under Grant 20174LF3T8 AHeAD: efficient Algorithms for HArnessing networked Data; and by an IBM SUR Grant.

originated from the same portion of DNA. Even for the case of uniform and independent strings, the study of average edit distance appears to be challenging and little work has been reported on the problem. In contrast, the closely related problem of finding the average length of the *longest common subsequence* has been extensively studied, since the seminal work by Chvátal and Sankoff [8].

Using Fekete's lemma, it can be shown that both metrics tend to grow linearly with n (Steele [18]). Let $e_k(n)$ denote the average edit distance between two random, independent strings of length n on a k-ary alphabet; then $\alpha_k(n) = e_k(n)/n$ approaches a limit $\alpha_k \in [0, 1]$. Similarly, let $\ell_k(n)$ denote the length of the longest common subsequence; then $\gamma_k(n) = \ell_k(n)/n$ approaches a limit $\gamma_k \in [0, 1]$. The γ_k's are known as the Chvátal-Sankoff constants. The determination of the exact values of α_k and γ_k is an open problem. This paper addresses the problem of estimating and bounding α_k, for various alphabet sizes k.

Related Work. There is limited literature directly pursuing bounds and estimates for α_k. It is also interesting to review results on γ_k: on the one hand, bounds to γ_k give bounds to α_k; on the other hand, techniques for analyzing γ_k can be adapted for analyzing α_k.

The only published estimates of α_k can be found in [10] which gives $\alpha_4 \approx 0.518$ for the quaternary alphabet and $\alpha_2 \approx 0.29$ for the binary alphabet. For γ_k, the best available estimates are given by Bundschuh [5], in particular $\gamma_2 \approx 0.8126$ and $\gamma_4 \approx 0.6544$. Estimates of γ_k by sampling are given by Ning and Choi [13], their results, however, appear in contrast with estimates of [5]. In particular they conjectured that $\gamma_2 > 0.82$ contradicting the estimate of γ_2 given in [5].

The best published analytical lower bounds to α_k are $\alpha_4 \geq 0.3383$ for quaternary alphabet and $\alpha_2 \geq 0.1578$ for binary alphabet [10]. To the best of our knowledge, no systematic study of upper bounds to α_k have been published. The best known analytical lower and upper bounds to γ_2 are given by Lueker [11], who obtained $0.7881 \leq \gamma_2 \leq 0.8263$. For larger alphabets, the best results appear in Dancík [9], including $0.5455 \leq \gamma_4 \leq 0.7082$. From known relations between edit distance and the length of the longest common subsequence, it follows that $1-\gamma_k \leq \alpha_k \leq 2(1-\gamma_k)$. Thus, upper and lower bounds to α_k can be respectively obtained from lower and upper bounds to γ_k. From $\gamma_2 \leq 0.8263$ [11], we obtain $\alpha_2 \geq 0.1737$, which is tighter than the bound given in [10]. Instead $\gamma_4 \leq 0.7082$ [9] yields $\alpha_4 \geq 0.2918$, which is weaker than the bound $\alpha_4 \geq 0.3383$ [10]. From the weaker relation $(1-\gamma_4)/2 \leq \alpha_4$, [14] obtained the looser bound $\alpha_2 \geq 0.0869$. Improved bounds, for both α_2 and α_4, are derived in this paper. Some of our techniques resemble those used in Baeza-Yates *et al.* [4] for estimating γ_k.

1.1 Paper Contributions

The contributions of this paper include:

- statistical estimates of $\alpha_k(n)$ with error analysis,
- upper bounds to α_k by exhaustive computation of $\alpha_k(n)$ for small n, and
- lower bounds to α_k through analytical counting arguments.

Our numerical results for $k = 2$ and $k = 4$ are compared with previously known values in Table 1. The methods used to derive such values are presented throughout the paper, which is organized as follows. Section 2 introduces notation and definitions. Section 3 presents statistical estimates, with error analysis. Section 4 describes an algorithm for computing upper bounds. Section 5 develops lower bounds analysis based on counting edit scripts. Section 6 shows and discusses numerical results on bounds and estimates of α_k. Finally, Sect. 7 gives conclusions and future directions of investigation.

Table 1. Our results on α_2 and α_4 compared with previously published ones.

	Lower bound		Estimate		Upper bound	
	Previous	This work	Previous	This work	Previous	This work
α_2	0.1737 [11]	0.1742	0.290 [10]	0.2888	-	0.3693
α_4	0.3383 [10]	0.3598	0.518 [10]	0.5180	-	0.6318

2 Preliminaries

2.1 Notation and Definitions

Let Σ_k be a finite alphabet of size $k \geq 2$ and let $n \geq 1$ be an integer; a *string* x is a sequence of symbols $x[1]x[2]\ldots x[n]$ where $x[i] \in \Sigma_k$; n is called the *length* of x, also denoted by $|x|$. Σ_k^n is the set of all strings of length n.

Edit Distance. We consider the following *edit operations* on a string x: the *match* of $x[i]$, the *substitution* of $x[i]$ with a different symbol $b \in \Sigma_k \setminus \{x[i]\}$, the *deletion* of $x[i]$, and the *insertion* of $b \in \Sigma_k$ in position $j = 0, \ldots, n$ (insertion in j means b goes after $x[j]$ or at the beginning if $j = 0$); an *edit script* is an ordered sequence of edit operations. To each type of edit operation is associated a cost; throughout this paper, matches have cost 0 and other operations have cost 1. The cost of a script is the sum of the costs of its operations. The *edit distance* between x and y, $d_E(x, y)$, is the minimum cost of any script transforming x into y. It is easy to see that $||x| - |y|| \leq d_E(x, y) \leq \max(|x|, |y|)$.

Random Strings and the Limit Constant. A *random string* $X = X[1]X[2]\ldots X[n]$ is a sequence of *random symbols* $X[i]$ each generated according to some distribution. We will assume $X[i]$ uniformly and independently sampled from Σ_k, hence $\mathrm{P}[X = x] = k^{-n}$ for every $x \in \Sigma_k^n$. For a string x we define the *eccentricity*, $\mathrm{ecc}(x)$, as its expected distance from a random string $Y \in \Sigma_k^n$:

$$\mathrm{ecc}(x) = k^{-n} \sum_{y \in \Sigma_k^n} d_E(x, y). \tag{1}$$

The expected edit distance between two random, independent strings of Σ_k^n is:

$$\begin{aligned} e_k(n) &= k^{-2n} \sum_{x \in \Sigma_k^n} \sum_{y \in \Sigma_k^n} d_E(x,y) \\ &= k^{-n} \sum_{x \in \Sigma_k^n} \mathrm{ecc}(x). \end{aligned} \tag{2}$$

Let $\alpha_k(n) = e_k(n)/n$; it can be shown (Fekete's lemma from ergodic theory; see, for example, Lemma 1.2.1 in [18]) that there exists a real number $\alpha_k \in [0,1]$, depending only on k, such that

$$\lim_{n\to\infty} \alpha_k(n) = \alpha_k.$$

The main objective of this paper is to derive estimates and bounds to α_k.

2.2 Computing the Edit Distance

Edit distance and length of the longest common subsequence (LCS) can be computed using a *dynamic programming* algorithm. For the edit distance, given two strings x and y with length n and m respectively, the algorithm fills an $(n+1) \times (m+1)$ matrix $\mathbf{M}$. The values of $\mathbf{M}$ are computed according to the following recurrence:

$$\begin{aligned} &M_{i,0} = i && \text{for } i = 0,\dots,n \\ &M_{0,j} = j && \text{for } j = 0,\dots,m \\ &M_{i,j} = \min\{M_{i-1,j-1} + \xi_{i,j}; M_{i-1,j}+1; M_{i,j-1}+1\} && \text{otherwise} \end{aligned} \tag{3}$$

where $\xi_{i,j} = 0$ if $x[i] = y[j]$ and $\xi_{i,j} = 1$ otherwise.[1] The edit distance between x and y is the value computed in the entry $M_{n,m}$. This algorithm takes $\mathcal{O}(nm)$ time and space. From the matrix $\mathbf{M}$, an edit script realizing the transformation of x into y can be obtained using *backtracking* which produces a *path* from cell (n,m) to cell $(0,0)$ of $\mathbf{M}$. For both problems, the approach by Masek and Paterson [12] (using the method of the Four Russians) reduces the time to $\mathcal{O}(\frac{n^2}{\log n})$ (assuming $n \geq m$). It is known that both the edit distance and the length of the LCS cannot be computed in time $\mathcal{O}(n^{2-\epsilon})$, unless the *Strong Exponential Time Hypothesis (SETH)* is false (Abboud *et al.* [1], Backurs and Indyk [3]). For the edit distance, a $(\log n)^{\mathcal{O}(1/\epsilon)}$ approximation, computable in time $\mathcal{O}(n^{1+\epsilon})$, is given by Andoni *et al.* [2], while a constant approximation algorithm with running time $\mathcal{O}(n^{1+5/7})$ is given by Chakraborty *et al.* [7]. A very recent work by Rubinstein and Song [15], gives a reduction from approximate length of the longest common subsequence to approximate edit distance, proving that the algorithm in [7] can also be used to approximate the length of the LCS.

[1] A similar algorithm computes the length of the LCS. The recurrence (3) becomes $M_{i,0} = 0$, $M_{0,j} = 0$, and $M_{i,j} = \max\{M_{i-1,j-1} + (1 - \xi_{i,j}); M_{i-1,j}; M_{i,j-1}\}$.

In order to compute upper bounds to α_k, we propose an algorithm related to the approaches developed by Calvo-Zaragoza *et al.* [6] and [11]. In these works, portions of the dynamic programming matrix are associated to states of a finite state machine. Our algorithm conceptually simulates all possible execution of a machine similar to the one defined in [6].

3 Statistical Estimates of α_k

In this section, we discuss how to develop statistical estimates of $\alpha_k(n)$, by sampling. In Table 2, we show results for $k = 4$, a case of special interest in DNA analysis [10]. Although $\alpha_k < \alpha_k(n)$ for every finite n, when n is sufficiently large, estimates of $\alpha_k(n)$ provide approximations to α_k.

3.1 Estimates of $\alpha_k(n)$ and Confidence Interval

Let $(x_1, y_1), \ldots, (x_N, y_N)$ be N random and independent pairs of strings from Σ_k^n. The sample mean

$$\widetilde{e}_k(n) = \frac{1}{N}\sum_{i=1}^{N} d_E(x_i, y_i) \tag{4}$$

provides an estimate of $e_k(n)$. We determine confidence intervals of such an estimate, using the sample variance

$$S_k^2(n) = \frac{1}{N-1}\sum_{i=1}^{N} (d_E(x_i, y_i) - \widetilde{e}_k(n))^2. \tag{5}$$

Table 2 presents values of $\widetilde{e}_4(n)$ and $S_4^2(n)$ for $N = 5000$ and $n = 2^4, 2^5, \ldots, 2^{14}$. Our analysis on confidence intervals is based on the work by Saw *et al.* [16], which extends Chebyshev's inequality to the cases where both mean and variance are not known, but rather estimated with (4) and (5), respectively. For this analysis to apply, it is sufficient for a random variable to have finite first and second order moments, a condition certainly satisfied by the edit distance of a random pair of strings of a given length.

Proposition 1 (Eq. (2.2) in [16]). *Let $\widetilde{e}_k(n)$ and $S_k^2(n)$ be given by (4) and (5), respectively. For any $t \geq 1$,*

$$\mathrm{P}[|e_k(n) - \widetilde{e}_k(n)| \leq \sqrt{(N+1)/N}tS_k(n)] \geq 1 - \left(\frac{N-1}{N}\frac{1}{t^2} + \frac{1}{N}\right). \tag{6}$$

From (6) we get the confidence interval on $\alpha_k(n)$

$$\mathrm{P}[\alpha_k(n) \in [\widetilde{e}_k(n)/n \pm \sqrt{(N+1)/N}tS_k(n)]] \geq 1 - \left(\frac{N-1}{N}\frac{1}{t^2} + \frac{1}{N}\right).$$

Table 2. Results of statistical estimates of $\alpha_4(n)$ for $n = 2^4, 2^5, \ldots, 2^{14}$ obtained from $N = 5000$ samples. The table shows: n, sample mean $\widetilde{e}_4(n)$, sample variance $S_4^2(n)$, estimate $\widetilde{\alpha}_4(n)$, and the value $S_4(n)/n$ used to the compute confidence intervals.

n	$\widetilde{e}_4(n)$	$S_4^2(n)$	$\widetilde{\alpha}_4(n)$	$S_4(n)/n$
16	10.0164	2.0814	0.6260	0.0902
32	18.9460	3.3306	0.5920	0.0570
64	36.4370	5.4487	0.5693	0.0365
128	70.5634	9.1274	0.5513	0.0236
256	138.0370	14.4977	0.5392	0.0149
512	272.1636	24.3117	0.5315	0.0096
1024	538.7120	39.6606	0.5261	0.0062
2048	1070.2178	65.2186	0.5226	0.0039
4096	2131.4744	111.4540	0.5204	0.0026
8192	4251.1936	178.6490	0.5189	0.0016
16384	8486.4712	323.7883	0.5180	0.0011

The values $S_4(n)/n$ are shown in Table 2 for each n. For example with $n = 2^{14}$, $N = 5000$, and $t = 5$, we get

$$\mathrm{P}[\alpha_4(2^{14}) \in [0.518 \pm 0.0055]] \geq 1 - \left(\frac{4999}{5000}\frac{1}{5^2} + \frac{1}{5000}\right) = 0.9598.$$

Since $\alpha_k < \alpha_k(n)$, we conclude that

$$\mathrm{P}[\alpha_4 < 0.5235] \geq 0.9598. \tag{7}$$

3.2 The $(2w+1)$-bandwidth Algorithm for Approximate Distance

Although better approximations of α_k can, in principle, be obtained using larger values of n, the quadratic complexity of the dynamic programming algorithm limits the values of n we can practically test. To partially circumvent this obstacle, for $n > 2^{14}$, we have estimated $\alpha_k(n)$ resorting to an algorithm that computes an approximation (from above) to $d_E(x, y)$. The algorithm is parameterized by an integer $w \geq 0$ called *bandwidth*. It computes the portion of a matrix $\mathbf{Q}$ corresponding to the $2w + 1$ central diagonals. The algorithm performs the following steps.

1. For $h = 0, \ldots, w + 1$, set $Q_{h,0} = h$ and $Q_{0,h} = h$.
2. For $h = w + 2, \ldots, n$, set $Q_{h-w-1,h}$ and $Q_{h,h-w-1}$ to h.
3. For $j = 1, \ldots, n$ and $i = \max(1, j - w), \ldots, \min(j + w, n)$, apply (3) to $Q_{i,j}$.

We observe that the values of the entries of $\mathbf{Q}$ on the boundary of the region where such matrix is computed (steps 1 and 2) are set to upper bounds to the corresponding entries of the matrix $\mathbf{M}$, reviewed in Sect. 2.2. Since the function that updates an entry in terms of its neighbours (step 3) is monotone non-decreasing, we have that $Q_{i,j} \geq M_{i,j}$, whence the output $Q_{n,n}$ of the bandwidth algorithm computes an upper bound to $d_E(x, y)$. It can be shown that $Q_{n,n} = d_E(x, y)$ whenever there exists an optimal path in $\mathbf{M}$ confined to the $2w + 1$ central diagonals, a condition that is always met if $d_E(x, y) \leq w$.

We carried out simulations setting $w = \sqrt{n}$, so that the bandwidth algorithm runs in time is $\mathcal{O}(n^{3/2})$. Application of Eq. (6) to $n = 2^{20}$ with $t = 5$, gives for $\hat{e}_4(2^{20})/2^{20}$, the confidence interval $[0.5162, 0.5174]$ with probability at least 0.9598. Since $\hat{e}_4(n)/n$ is an estimate of an upper bound to $e_4(n)/n$, we obtain

$$\mathrm{P}[\alpha_4 < 0.5174] \geq 0.9598. \tag{8}$$

We observe that bound (8) improves on bound (7), obtained via the exact distance algorithm. This indicates that the loss of precision due to the use of an approximate algorithm is more than compensated by the ability to process larger string sizes.

4 Upper Bounds for $\boldsymbol{\alpha_k}$

This section presents methods to derive upper bounds to α_k based on the exact computation of $\alpha_k(n) = e_k(n)/n$ for some n, and on the relation $\alpha_k \leq \alpha_k(n)$, valid for all $n \geq 1$. The computation of $e_k(n)$ can be reduced to that of the eccentricity, as in Eq. (2) repeated here for convenience:

$$e_k(n) = k^{-n} \sum_{x \in \Sigma_k^n} \mathrm{ecc}(x). \tag{9}$$

If $\mathrm{ecc}(x)$ is computed according to Eq. (2) and the distance $d_E(x, y)$ is computed by the $\mathcal{O}(n^2)$-time dynamic programming algorithm for each of the k^n strings $y \in \Sigma_k^n$, then the overall computation time is $\mathcal{O}(n^2 k^n)$ for $\mathrm{ecc}(x)$ and $\mathcal{O}(n^2 k^{2n})$ for $e_k(n)$, since the eccentricity of each of the k^n strings $x \in \Sigma_k^n$ is needed in Eq. (9). Below, we propose a more efficient algorithm to speed up the computation of $\mathrm{ecc}(x)$ and, in turn, that of $e_k(n)$, achieving time $\mathcal{O}(n^2 \min{(k, 3)}^n k^n) = \mathcal{O}(n^2 3^n k^n)$. We also show how to exploit some symmetries of $\mathrm{ecc}(x)$ in order to limit its evaluation to a suitable subset of Σ_k^n.

4.1 The Coalesced Dynamic Programming Algorithm for Eccentricity

Let $\mathbf{M}(x, y)$ be the matrix produced by the dynamic programming algorithm (reviewed in Sect. 2.2) to compute $d_E(x, y)$, with $x, y \in \Sigma_k^n$. We develop a strategy to coalesce the computations of $\mathbf{M}(x, y)$ for different $y \in \Sigma_k^n$, while keeping x fixed. To this end, we choose to generate the entries of $\mathbf{M}(x, y)$, according to Eq. (3), in column-major order. Clearly, the j-th column is fully determined by x and by the prefix of y of length j. Define now the *column multiset* $\mathcal{C}_j$

Algorithm 1. Coalesced dynamic programming algorithm to compute ecc(x)

```
1: procedure ECCENTRICITY($x$)
2:     $n \leftarrow |x|$
3:     $\mathcal{C}_0 \leftarrow \{((0, 1, \ldots, n), 1)\}$
4:     for $j \leftarrow 1$ to $n$ do
5:         $\mathcal{C}_j \leftarrow \emptyset$
6:         for $\mathbf{c} \in \mathcal{C}_{j-1}$ do
7:             for $b \in \Sigma_k$ do
8:                 $\mathbf{c}' \leftarrow$ NEXTCOLUMN($x, \mathbf{c}, j, b$)
9:                 INSERT($\mathcal{C}_j, (\mathbf{c}', \mu(\mathbf{c}))$)
10:            end for
11:        end for
12:    end for
13:    $e \leftarrow 0$
14:    for $\mathbf{c} \in \mathcal{C}_n$ do
15:        $e \leftarrow e + \mu(\mathbf{c}) * \mathbf{c}[n]$
16:    end for
17:    return $e/k^n$
18: end procedure
```

containing the j-th (i.e., the last) column of $\mathbf{M}(x, y[1] \ldots y[j])$ for each string $y[1] \ldots y[j] \in \Sigma_k^j$. Multiset $\mathcal{C}_j$ is a function of (just) x, although, for simplicity, the dependence upon x is not reflected in our notation.

The *Coalesced Dynamic Programming* (CDP) algorithm described below (referring also to the line numbers of Algorithm 1), constructs the sequence of multisets $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_n$. A column multiset $\mathcal{C}$ will be represented as a set of pairs $(\mathbf{c}, \mu(\mathbf{c}))$, one for each distinct member $\mathbf{c}$, with $\mu(\mathbf{c})$ being the multiplicity of $\mathbf{c}$ in $\mathcal{C}$. The eccentricity of x is obtained (lines 13–17) as the weighted average of the n-th element of all columns in $\mathcal{C}_n$:

$$\text{ecc}(x) = k^{-n} \sum_{\mathbf{c} \in \mathcal{C}_n} \mu(\mathbf{c})\mathbf{c}[n]. \tag{10}$$

As can be seen from Eq. (3), multiset $\mathcal{C}_0$ contains just column $(0, 1, \ldots, n)$, with multiplicity 1 (line 3). For $j = 1, \ldots, n$, $\mathcal{C}_j$ is obtained by scanning all $\mathbf{c} \in \mathcal{C}_{j-1}$ (line 6) and all $b \in \Sigma_k$ (line 7), and by

– computing the j-th column $\mathbf{c}'$ resulting from Eq. (3) when the $(j-1)$-st column is $\mathbf{c}$ and $\xi_{i,j} = 0$ if $x[i] = b$ or else $\xi_{i,j} = 1$ (call to NEXTCOLUMN($x, \mathbf{c}, j, b$), line 8);
– inserting $\mu(\mathbf{c})$ copies of $\mathbf{c}'$ in $\mathcal{C}_j$, by either creating a new pair $(\mathbf{c}', \mu(\mathbf{c}))$ when $\mathbf{c}'$ is not present in the multiset or by incrementing its multiplicity by $\mu(\mathbf{c})$ otherwise (call to INSERT($\mathcal{C}_j, (\mathbf{c}', \mu(\mathbf{c}))$), line 9).

The correctness of the CDP algorithm is pretty straightforward to establish. A few observations are however necessary in order to describe and analyze a data structure that can efficiently implement, in our specific context, multisets with the insertion operation. The key property is that, for $j = 0, 1 \ldots, n$, the column of $\mathbf{M}(x, y)$ with index j satisfies the conditions (a) $M_{0,j} = j$ and (b) $(M_{i,j} - M_{i-1,j}) \in \{-1, 0, 1\}$, for $i = 1, \ldots, n$. Using this property, the set of distinct columns that belong to the multiset $\mathcal{C}_j$ can be represented as a ternary tree where each arc has a label from the set $\{-1, 0, 1\}$ and a column $(M_{0,j}, M_{1,j}, \ldots, M_{n,j})$ is mapped to a leaf v such that the n arcs in the path from the root to v have labels $(M_{1,j} - M_{0,j}), \ldots, (M_{n,j} - M_{n-1,j})$. Each leaf stores the multiplicity of the corresponding column. The size of the tree for $\mathcal{C}_j$ is $\mathcal{O}(\min(3^n, k^j))$, since there are at most 3^n columns satisfying the constrains and k^j k-ary strings that contribute (not necessarily distinct) columns. Hence, the body of the loop whose iteration space is defined in lines 4, 6, and 7 is executed $nk\mathcal{O}(\min(3^n, k^j))$ times. Considering that one call to NextColumn() as well as one call to Insert() can be easily performed in $\mathcal{O}(n)$ time, we can summarize the previous discussion as follows, where we also consider that, at any given time, the algorithm only needs to store two consecutive column multisets.

Proposition 2. *The* CPD algorithm *computes the* eccentricity $\mathrm{ecc}(x)$ *of a string x of length n over a k-ary alphabet in time $T = \mathcal{O}(n^2 k \min(3^n, k^n))$ and space $S = \mathcal{O}(\min(3^n, k^n))$. Correspondingly, the* average distance $e_k(n)$ *can be computed in time $T = \mathcal{O}(n^2 k^{n+1} \min(3^n, k^n))$ and space $S = \mathcal{O}(\min(3^n, k^n))$.*

4.2 Exploiting Symmetries of $\mathrm{ecc}(x)$ in the Computation of $e_k(n)$

The edit distance enjoys some useful symmetries, which can be easily derived from the definition. One is that, if we let $x^R = x[n] \ldots x[1]$ denote the *reverse* of string $x = x[1] \ldots x[n]$, then $d_E(x, y) = d_E(x^R, y^R)$. Another one is that if $\pi : \Sigma_k \to \Sigma_k$ is a permutation of the alphabet and $\pi(x)$ denotes the string $\pi(x[1]) \ldots \pi(x[n])$, then $d_E(x, y) = d_E(\pi(x), \pi(y))$. The following is a simple, but useful corollary of these properties.

Proposition 3. *For any $x \in \Sigma_k^n$, we have* $\mathrm{ecc}(x^R) = \mathrm{ecc}(x)$. *Furthermore, for any permutation π of Σ_k, we have* $\mathrm{ecc}(\pi(x)) = \mathrm{ecc}(x)$.

It is useful to define the equivalence class of x as the set of strings that have the same eccentricity as x, due to Proposition 3, and denote by $\nu(x)$ the cardinality of such set. If $\mathcal{R}_{k,n} \subseteq \Sigma_k^n$ contains exactly one (representative) member for each equivalence class, then Eq. (9) can be rewritten as

$$e_k(n) = k^{-n} \sum_{x \in \mathcal{R}_{k,n}} \nu(x) \mathrm{ecc}(x). \tag{11}$$

Computing $e_k(n)$ according to Eq. (11) enables one to reduce the number of strings for which the eccentricity has to be computed (via the CDP algorithm) by a factor slightly smaller than $(2k!)$, with a practically appreciable reduction in computation time.

The strategy outlined in this section has been implemented in C++ and run on a 32 core IBM Power7 server. For several alphabet sizes k, we have considered values of n up to a maximum value n_k^{ub}, under the constraint that the running time would not exceed one week. The resulting values $e_k(n_k^{ub})$ are presented and discussed in Sect. 6.

5 Lower Bounds for α_k

In this section, we prove the theoretical results that are used to obtain the lower bounds α_k^{lb} shown in Sect. 6. To obtain such lower bounds, we will derive lower bounds to $\mathrm{ecc}(x)$ by ignoring the contribution of the strings inside the ball of radius r centered at x and by setting to $r+1$ the contribution of the string outside the same ball. The objective is to determine the largest value r^* of r for which (it can be shown that) the ball of radius r contains a fraction of Σ_k^n that vanishes with n; then r^* will effectively represent a lower bound to $\alpha_k n$. Below, we formalize this idea and show that we can choose $r^* = \beta n$ for suitable values of β independent on n; this establishes that $\alpha_k \geq \beta$.

5.1 Lower Bounds to ecc(x) Using Upper Bounds to Ball Size

In this subsection, we show how to derive lower bounds to $\mathrm{ecc}(x)$ starting from upper bounds to the size of the ball of radius r centered at x. We also show that, when such bounds are valid for every x, they can be used to compute lower bounds to α_k.

Definition 1. *For a string $x \in \Sigma_k^n$, the* ball *of radius r centered at x is defined as the set of strings having distance at most r from x:*

$$B_r(x) = \{y \in \Sigma_k^n : d_E(x,y) \leq r\}.$$

Similarly, the shell *of radius r centered at x is defined as the set of strings having distance exactly r from x:*

$$S_r(x) = \{y \in \Sigma_k^n : d_E(x,y) = r\}.$$

For a given r, each string in $\Sigma_k^n \setminus B_r(x)$ has distance at least $r+1$ from x; therefore its contribution to $\mathrm{ecc}(x)$ is at least $r+1$. Given an upper bound $u_r(x) \geq |B_r(x)|$, the consequent lower bound $|\Sigma_k^n \setminus B_r(x)| \geq (k^n - u_r(x))$ yields the following lower bound to $\mathrm{ecc}(x)$.

Lemma 1. *Let $u_r(x) \geq |B_r(x)|$, then for every $r^* = 0, 1, \ldots, n$:*

$$\mathrm{ecc}(x) \geq r^* \left(1 - k^{-n} u_{r^*}(x)\right) \tag{12}$$

Proof. We can rewrite (1) as

$$\begin{aligned}
\mathrm{ecc}(x) &= k^{-n}\sum_{r=0}^{r^*} r|S_r(x)| + k^{-n}\sum_{r=r^*+1}^{n} r|S_r(x)| \\
&\geq k^{-n}(r^*+1)\sum_{r=r^*+1}^{n} |S_r(x)| \\
&= k^{-n}(r^*+1)\left(|B_n(x)| - |B_{r^*}(x)|\right) \\
&> r^*\left(1 - k^{-n}|B_{r^*}(x)|\right) \\
&\geq r^*\left(1 - k^{-n}u_{r^*}(x)\right).
\end{aligned}$$

□

In particular when $u_{r^*} \geq |B_{r^*}(x)|$ for every $x \in \Sigma_k^n$, simple manipulations of Eq. (2), recalling that $\alpha_k = \lim_{n\to\infty} \frac{e_k(n)}{n}$, yield

$$\alpha_k \geq \lim_{n\to\infty} \frac{r^*}{n}\left(1 - k^{-n}u_{r^*}\right). \tag{13}$$

The bounds presented next are based on (13). We will show that, for suitable values of β, the quantity $(k^{-n}u_{\beta n})$ converges to 0, so that $\alpha_k \geq \beta$.

5.2 Upper Bounds on Ball Size

To use Lemma 1 we need an upper bound to $|B_r(x)|$. The next proposition develops such an upper bound by (i) associating every string in $B_r(x)$ to a script of certain type with cost r or $r-1$ and (ii) counting such scripts.

Proposition 4. *For any $x \in \Sigma_k^n$ and for any $r = 1, \ldots, n$*

$$|B_r(x)| \leq (k-1)^r \sum_{d=0}^{\lfloor r/2 \rfloor} \binom{n}{d}^2 \binom{n-d+1}{r-2d} \left(\frac{k}{(k-1)^2}\right)^d. \tag{14}$$

Proof. We introduce the notion of *simple script* of cost $r \in \{0, 1, \ldots, n\}$, constructed by the following sequence of choices (shown within square brackets is the number of possible choices):

- $d \in \{0, 1, \ldots, \lfloor r/2 \rfloor\}$
- d positions to delete from x $[\binom{n}{d}]$
- $(r-2d)$ of the remaining $(n-d)$ positions to be substituted $[\binom{n-d}{r-2d}]$
- d positions to insert in y $[\binom{n}{d}]$
- the symbols in the substitutions $[(k-1)^{r-2d}]$
- the symbols in the insertions $[k^d]$

Straightforwardly, the number of simple scripts of cost r is

$$s_r = \sum_{d=0}^{\lfloor r/2 \rfloor} \binom{n}{d}^2 \binom{n-d}{r-2d} (k-1)^{r-2d} k^d. \tag{15}$$

It is easy to see that optimal scripts are simple.

Next, we prove that any $y \in B_r(x)$ can be obtained from x via a simple script of cost $r-1$ or r. Let $r' = d_E(x, y) \leq r$, we will focus to the case where $r' < r-1$, since in the complementary case the argument is trivial. Consider an optimal, simple script of cost r' that transforms x into y. By augmenting this script with $\lfloor (r-r')/2 \rfloor$ pairs of deletions and insertions, each pair acting on a matched position, we obtain a simple script of cost r, if $r-r'$ is even, or of cost $r-1$ if $r-r'$ is odd. The prescribed augmentation is always possible since the number of matches is at least $n - r' \geq r - r' \geq (r-r')/2$.

The thesis is then established by the following sequence of inequalities

$$\begin{aligned}
|B_r(x)| &\leq s_r + s_{r-1} \\
&\leq \sum_{d=0}^{\lfloor r/2 \rfloor} \binom{n}{d}^2 \binom{n-d}{r-2d} (k-1)^{r-2d} k^d + \sum_{d=0}^{\lfloor (r-1)/2 \rfloor} \binom{n}{d}^2 \binom{n-d}{r-1-2d} (k-1)^{r-1-2d} k^d \\
&\leq \sum_{d=0}^{\lfloor r/2 \rfloor} \binom{n}{d}^2 \binom{n-d}{r-2d} (k-1)^{r-2d} k^d + \sum_{d=0}^{\lfloor r/2 \rfloor} \binom{n}{d}^2 \binom{n-d}{r-1-2d} (k-1)^{r-2d} k^d \\
&\leq \sum_{d=0}^{\lfloor r/2 \rfloor} \binom{n}{d}^2 \binom{n-d+1}{r-2d} (k-1)^{r-2d} k^d,
\end{aligned}$$

where we have made use of the identity

$$\binom{n-d}{r-2d} + \binom{n-d}{r-1-2d} = \binom{n-d+1}{r-2d}.$$

□

5.3 Asymptotic Behavior of Ball Size and Bounds for α_k

The next results show that (14), divided by k^n, is bounded by an exponential function where we can choose the exponent in such a way that this function vanishes with n. This can then be used in (13) to obtain lower bounds to α_k.

Definition 2. *Let $H(\beta)$ denote the* binary entropy function

$$H(\beta) = -\beta \log_2 \beta - (1-\beta) \log_2 (1-\beta).$$

Definition 3. *For $\beta \in [0,1]$ and $\delta \in [0, \beta/2]$ we define the function*

$$\begin{aligned}
g_k(\beta, \delta) =& (\beta - 2\delta) \log_2 (k-1) - (1-\delta) \log_2 k \\
&+ 2H(\delta) + (1-\delta) H\left(\frac{\beta - 2\delta}{1-\delta}\right).
\end{aligned} \tag{16}$$

Lemma 2. *Let u_r be given by the right hand side of (14) and $g_k(\beta,\delta)$ given by (16). For every $\beta \in [0,1]$*

$$k^{-n}u_{\beta n} \le (n+1)\sum_{d=0}^{\lfloor \beta n/2 \rfloor} 2^{ng_k\left(\beta,\frac{d}{n}\right)}. \tag{17}$$

Proof. Using the relation

$$\binom{n-d+1}{r-2d} = \frac{n-d+1}{n-r+d+1}\binom{n-d}{r-2d} \le (n+1)\binom{n-d}{r-2d},$$

the bound $\binom{n}{k} \le 2^{nH(k/n)}$ (see, e.g., Eq. (5.31) in Spencer [17]), and defining $\beta = r/n$ we get

$$\begin{aligned} k^{-n}u_r \le & k^{-n}(k-1)^r \sum_{d=0}^{\lfloor r/2 \rfloor} \binom{n}{d}^2 \binom{n-d+1}{r-2d} \left(\frac{k}{(k-1)^2}\right)^d \\ \le & (n+1)\sum_{d=0}^{\lfloor r/2 \rfloor} 2^{2nH\left(\frac{d}{n}\right)+(n-d)H\left(\frac{r-2d}{n-d}\right)+(r-2d)\log_2 (k-1)^2+(d-n)\log_2 k} \\ = & (n+1)\sum_{d=0}^{\lfloor \beta n/2 \rfloor} 2^{ng_k\left(\beta,\frac{d}{n}\right)}. \end{aligned}$$

□

Theorem 1. *Letting $A_k = \{\beta \in [0,1] : \forall \delta \in [0,\beta/2]\ (g_k(\beta,\delta) < 0)\}$, we have:*

$$\alpha_k \ge \sup A_k.$$

Proof. We begin by observing that A_k is not empty, since $0 \in A_k$. In fact, when $\beta = 0$, the condition $\delta \in [0,\beta/2]$ is satisfied only by $\delta = 0$, and $g_k(0,0) = -\log_2 k < 0$. Since, by definition, $A_k \subseteq [0,1]$, we conclude that $\sup A_k$ is finite. Next, we define the function

$$G_k(\beta) = \max_{0 \le \delta \le \beta/2} g_k(\beta,\delta).$$

This definition of $G_k(\beta)$ is well posed, since $g_k(\beta,\delta)$ is a continuous function, hence it does have a maximum in the compact set $0 \le \delta \le \beta/2$. Furthermore, it follows from the definitions of $G_k(\beta)$ and A_k that $G_k(\beta) < 0$, for any $\beta \in A_k$.

For any $\beta \in A_k$, we see from Lemma 2 that

$$k^{-n}u_{\beta n} \le (n+1)\sum_{d=0}^{\lfloor \beta n/2 \rfloor} 2^{ng_k\left(\beta,\frac{d}{n}\right)} \le f(n)2^{nG_k(\beta)},$$

where we have used the relation $g_k\left(\beta,\frac{d}{n}\right) \le G_k(\beta)$ (which follows from the definition of $G_k(\beta)$ and the fact that for, any d in the summation range,

$0 \leq \frac{d}{n} \leq \beta/2$) and we have let $f(n) = (n+1)\left(\left\lfloor \frac{\beta n}{2} \right\rfloor + 1\right)$. Taking now the limit in (13) with $r^* = \beta n$ yields:

$$\alpha_k \geq \lim_{n\to\infty} \beta\left(1 - f(n)2^{nG_k(\beta)}\right) = \beta,$$

as $f(n) = \mathcal{O}(n^2)$ and $2^{nG_k(\beta)}$ is a negative exponential. In conclusion, since α_k is no smaller than any member of A_k, it is also no smaller than $\sup A_k$. $\square$

Theorem 1 gives a criterion to find lower bounds to α_k: choose β such that $g(\beta,\delta) < 0$ for every δ. The lower bounds presented in Sect. 6 are computed using a numerical evaluation of $\sup A_k$.

6 Numerical Results and Discussion

In this section, we present and discuss some numerical results obtained by applying the methodologies developed in previous sections, to alphabets of various size k. These values are reported in the Table 3 along with the indication of the string size n_k^{ub} used in the computation of α_k^{ub}.

Table 3. Results on α_k for several alphabet sizes k. The table shows lower bounds α_k^{lb}, statistical estimates $\widetilde{\alpha}_k$, upper bounds α_k^{ub}. The values of α_k^{lb} are obtained by numerically evaluating $\sup A_k$ (Sect. 5, Theorem 1). Each statistical estimate $\widetilde{\alpha}_k$ is based on $N = 5000$ sample pairs of strings of length $n = 2^{14}$ (Sect. 3.1). The values of α_k^{ub} are based on the exact determination of $\alpha_k(n_k^{ub})$ (Sect. 4). We can observe that $\alpha_k^{lb} < \widetilde{\alpha}_k < \alpha_k^{ub}$.

k	α_k^{lb}	$\widetilde{\alpha}_k$	α_k^{ub}	n_k^{ub}
2	0.1742	0.2888	0.3693	24
3	0.2837	0.4292	0.5343	17
4	0.3598	0.5180	0.6318	15
5	0.4152	0.5806	0.7020	13
6	0.4578	0.6277	0.7515	12
7	0.4918	0.6645	0.7903	11
8	0.5199	0.6946	0.8122	12
16	0.6648	0.8196	0.8955	10
32	0.7387	0.8999	0.9659	6

As already mentioned in the introduction, $1 - \gamma_k \leq \alpha_k$. Since γ_k vanishes with k (Theorem 1 in [8]), we have that $\lim_{k\to\infty} \alpha_k = 1$. The data in Table 3 show a trend consistent with this asymptotic behavior of α_k.

The gap between lower and upper bound indicates that there is room for improving both. Improving the current upper bounds requires substantial improvements in the way we compute exact values of $\alpha_k(n)$. Improving the lower bounds appears viable by tightening the upper bound to the volume of $B_r(x)$, in Proposition 4. To this end, an avenue to be explored is a refinement of script counting that exploits properties of optimal scripts, not considered in the present arguments.

We observe that, for fixed k and variable n, the values $\alpha_k(n)$ form a sequence of computable upper bounds to α_k that converges to α_k. It would be interesting to find a sequence of computable lower bounds to α_k that converges to α_k. Together, these two sequences would establish the computability of α_k, providing an algorithm that, given as input any $\epsilon > 0$, would output a rational number η such that $|\alpha_k - \eta| < \epsilon$. To the best of our knowledge, whether the α_k's are computable is an open question. In contrast, it is a simple corollary of known results that the γ_k's are computable. On the one hand, a sequence of computable lower bounds converging to γ_k is straightforwardly provided by the values $\gamma_k(n)$. On the other hand, a sequence of computable upper bounds converging to γ_k has been established, by a rather sophisticated approach, in [11] (see, in particular, Theorem 3.13).

7 Conclusions

In this paper, we have explored approaches to obtain statistical estimates, upper bounds, and lower bounds to the asymptotic constant characterizing the average edit distance between random, independent strings. We used such approaches to obtain results for some alphabet sizes k. These numerical results (Table 3) improve over previously known values [10]. There is still a gap between upper and lower bounds which deserves further investigation. The approaches proposed here can be extended to the study of other statistical properties of the edit distance (e.g., the standard deviation, widely studied in the context of the longest common subsequence).

It is interesting to explore the role of statistical properties of the edit distance in string alignment and other key problems in DNA processing. One motivation is provided by the increasing availability of reads coming from third generation sequencers (e.g., PacBio) where sequencing errors can be modelled as edit operations. In this case it will be necessary to study the behaviour of the average edit distance, when strings are generated from non-uniform distributions or from empirical distributions (e.g., the distribution of substrings from the human DNA).

References

1. Abboud, A., Backurs, A., Williams, V.V.: Tight hardness results for LCS and other sequence similarity measures. In: 2015 IEEE 56th Annual Symposium on Foundations of Computer Science, pp. 59–78 (2015). https://doi.org/10.1109/FOCS.2015.14

2. Andoni, A., Krauthgamer, R., Onak, K.: Polylogarithmic approximation for edit distance and the asymmetric query complexity. In: 2010 IEEE 51st Annual Symposium on Foundations of Computer Science, pp. 377–386 (2010). https://doi.org/10.1109/FOCS.2010.43
3. Backurs, A., Indyk, P.: Edit distance cannot be computed in strongly subquadratic time (unless seth is false). In: Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing, pp. 51–58. STOC 2015, ACM, New York, NY, USA (2015). https://doi.org/10.1145/2746539.2746612
4. Baeza-Yates, R.A., Gavaldà, R., Navarro, G., Scheihing, R.: Bounding the expected length of longest common subsequences and forests. Theor. Comput. Syst. **32**(4), 435–452 (1999). https://doi.org/10.1007/s002240000125
5. Bundschuh, R.: High precision simulations of the longest common subsequence problem. Eur. Phys. J. B - Condens. Matter Complex Syst. **22**(4), 533–541 (2001). https://doi.org/10.1007/s100510170102
6. Calvo-Zaragoza, J., Oncina, J., de la Higuera, C.: Computing the expected edit distance from a string to a probabilistic finite-state automaton. Int. J. Found. Comput. Sci. **28**(05), 603–621 (2017). https://doi.org/10.1142/S0129054117400093
7. Chakraborty, D., Das, D., Goldenberg, E., Koucky, M., Saks, M.: Approximating edit distance within constant factor in truly sub-quadratic time. In: 2018 IEEE 59th Annual Symposium on Foundations of Computer Science, pp. 979–990 (2018). https://doi.org/10.1109/FOCS.2018.00096
8. Chvátal, V., Sankoff, D.: Longest common subsequences of two random sequences. J. Appl. Probab. **12**(2), 306–315 (1975). https://doi.org/10.2307/3212444
9. Dancík, V.: Expected length of longest common subsequences. Ph.D. thesis, University of Warwick (1994)
10. Ganguly, S., Mossel, E., Racz, M.Z.: Sequence assembly from corrupted shotgun reads. arXiv preprint arXiv:1601.07086 (2016)
11. Lueker, G.S.: Improved bounds on the average length of longest common subsequences. J. ACM **56**(3), 17:1–17:38 (2009). https://doi.org/10.1145/1516512.1516519
12. Masek, W.J., Paterson, M.S.: A faster algorithm computing string edit distances. J. Comput. Syst. Sci. **20**(1), 18–31 (1980). https://doi.org/10.1016/0022-0000(80)90002-1
13. Ning, K., Choi, K.P.: Systematic assessment of the expected length, variance and distribution of longest common subsequences. arXiv preprint arXiv:1306.4253 (2013)
14. Rubinstein, A.: Hardness of approximate nearest neighbor search. In: Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, pp. 1260–1268. STOC 2018, ACM, New York, NY, USA (2018). https://doi.org/10.1145/3188745.3188916
15. Rubinstein, A., Song, Z.: Reducing approximate longest common subsequence to approximate edit distance. arXiv preprint arXiv:1904.05451 (2019)
16. Saw, J.G., Yang, M.C.K., Mo, T.C.: Chebyshev inequality with estimated mean and variance. Am. Stat. **38**(2), 130–132 (1984). https://doi.org/10.1080/00031305.1984.10483182
17. Spencer, J.: Asymptopia. Am. Math. Soc., 71 (2014)
18. Steele, J.M.: Probability Theory and Combinatorial Optimization. SIAM, Philadelphia (1997)