Bounding the Expected Length of Longest Common Subsequences and Forests *

Ricardo A. Baeza-Yates†, Ricard Gavaldà‡, Gonzalo Navarro†, and Rodrigo Scheihing†

†Dept. of Computer Science
University of Chile
Blanco Encalada 2120
Santiago, Chile

‡Dept. LSI
Technical Univ. of Catalunya
Jordi Girona Salgado 1–3
08034 Barcelona, Spain

Abstract

We present improvements to two techniques to find lower and upper bounds for the expected length of longest common subsequences and forests of two random sequences of the same length, over a fixed size, uniformly distributed alphabet. We emphasize the power of the methods used, which are Markov chains and Kolmogorov complexity. As a corollary, we obtain some new lower and upper bounds for the problems addressed as well as some new exact results for short sequences.

1 Introduction

The longest common subsequence (LCS) of two strings is one of the main problems in combinatorial pattern matching. The LCS problem is related to DNA or protein alignments, file comparison, speech recognition, etc. We say that s is a subsequence of u if we can obtain s by deleting zero or more letters of u. The LCS of two strings u and v of length n is defined as the longest subsequence s common to u and v. For example, the LCS of *longest* and *large* is *lge*. An open problem related to the LCS is its expected length for two random strings of length n over a uniformly distributed alphabet of size k, denoted by $EL_n^{(k)}$. In particular, if an alignment or common subsequence of two given sequences is relatively larger than $EL_n^{(k)}$, we may infer that it is more than a coincidence, and that the similarity should be studied further. For example, two long random DNA sequences will have on average an alignment comprising 65% of its length, and that amount of similarity might be counterintuitive.

*This work has been partially supported by the ESPRIT Long Term Research Project 20244, ALCOM IT. The first author has also been supported by Dirección General de Investigación y Técnica (Ministry of Education and Science, Spain) and Fundación Andes. The second author has also been supported by Fondecyt grants 1950569 and 1940520. The third author has also been supported by Fondecyt grant 1960881. E-mail: gavalda@lsi.upc.es, {rbaeza,gnavarro,rscheihi}@dcc.uchile.cl.

If $\ell cs(u,v)$ denotes the length of the LCS for two strings u and v, we have:

$$EL_n^{(k)} = \frac{1}{k^{2n}} \sum_{|u|=|v|=n} \ell cs(u,v) \quad .$$

Because $EL_n^{(k)}$ is superadditive (credited to Klarner and Rivest in [CS75]), that is, $EL_n^{(k)} + EL_m^{(k)} \leq EL_{n+m}^{(k)}$, it is possible to show [CS75] that

$$\gamma_k = \lim_{n\to\infty} \frac{EL_n^{(k)}}{n} = \sup_n \frac{EL_n^{(k)}}{n}$$

exists. However, the exact values of γ_k are still not known. For that reason, several lower and upper bounds have been devised for γ_k. Chvátal and Sankoff [CS75, CS83] obtained

$$0.727273 \leq \gamma_2 \leq 0.866595 \ ,$$

which was improved by Deken [Dek79, Dek83] to

$$0.7615 \leq \gamma_2 \leq 0.8575 \ .$$

These were improved by Dančík & Paterson [Dan94, DP95, PD94] to

$$0.77391 \leq \gamma_2 \leq 0.83763 \quad .$$

Also, it is known that [CS75]

$$1 \leq \gamma_k \sqrt{k} \leq e \quad .$$

First, we present exact results for small sequences, computing the exact values of $EL_n^{(2)}$ from $n = 10$ [CS75] to $n = 16$, showing that the convergence to γ_k is slow.

Second, we present new lower bounds for $2 < k < 6$ for the LCS. These new results are based on a new class of automata (following the work of Deken [Dek79] and Dančík & Paterson [Dan94, PD94]) that simulates an algorithm that computes a long common subsequence over two random infinite strings. These automata are called CSS (Common SubSequence) machines in [Dan94].

Third, to obtain upper bounds, we refine and extend the Kolmogorov complexity approach due to Tao Jiang (mentioned in Li and Vitányi [LV97]), which is simple and elegant. Kolmogorov complexity has been very useful in many areas of computer science. The reader is referred to the monograph of Li and Vitányi [LV97] for a very complete treatment of the origins, development, and applications of this concept.

We also apply both techniques to a generalization of the LCS problem, called the Longest Common Forest (LCF) by Pevzner and Waterman [PW93], obtaining the first known lower and upper bounds for the expected size of the LCF of two random sequences. In particular, we show that for large alphabets, the fraction of the expected length of the LCF is upper bounded by $e/\sqrt{k}$ as in the LCS case.

The results included here were presented in preliminary form in [BYS95, BYGN96].

2 Longest Common Subsequences and Forests

The LCS of two strings u and v can be computed using dynamic programming over a matrix L defined by $L[0,i] = L[i,0] = 0$ for $0 \leq i \leq n$ and

$$L[i,j] = \max(L[i-1,j], L[i,j-1], L[i-1,j-1] + (u[i] =?v[j])), 1 \leq i,j \leq n \quad ,$$

where $(u[i] =?v[j])$ is defined as 1 if both letters are equal, or 0 otherwise. The length of the LCS is given by $L[n,n]$. This algorithm can be implemented using $3n^2$ comparisons. For faster algorithms which solve the LCS problem we refer the reader to [GBY91, PD94, Ric95].

Longest Common Forests (LCF) are defined in [PW93] as one particular case of general alignments between strings, called the A-LCS problem. Basically, in a LCF we allow a letter to match more than one letter of the other sequence, but if we look at every match as an edge between the two sequences, then no edge crossings can exist. Hence, the alignment is a set of trees or forest. In [PW93] a cn^2 algorithm to compute the A-LCS problem is given, where c is related to the determinant of a matrix defining the generalized alignment rules. They mention that $c = 2$ for the LCF problem, but a simple algorithm is not explicitly given. In fact, the dynamic programming procedure for LCF is given by

$$L[i,j] = \max(L[i-1,j], L[i,j-1]) + (u[i] =?v[j]) \quad ,$$

which requires only $2n^2$ comparisons. If $\ell cf(u,v)$ denotes the length of the LCF for two strings u and v, in general we have

$$0 \leq \ell cf(u,v) \leq 2(|u| + |v|) - 1 \quad ,$$

where the upper bound can be seen as the longest path where we either advance in a row or a column of the matrix L. Similarly to the LCS, the LCF is superadditive. We can define

$$EF_n^{(k)} = \frac{1}{k^{2n}} \sum_{|u|=|v|=n} \ell cf(u,v)$$

and

$$f_k = \lim_{n\to\infty} \frac{EF_n^{(k)}}{n} = \sup_n \frac{EF_n^{(k)}}{n} \leq 2 \ .$$

Figure 1 shows some examples of LCFs as well as the corresponding LCS length (the solutions shown in the examples are not necessarily unique).

Table 1 and Figure 2 show some exact values of $EL_n^{(2)}/n$ and $EF_n^{(2)}/n$ for $n \leq 16$. For the LCS these results extend [CS75]. Figure 3 shows the probability distribution of LCS and LCF for $n = 15$ normalized by the length n. We can see that in both cases the distribution is centered but with significative tails, which partially explains why it is difficult to bound better their average value.

3 Lower Bounds: Markov Chains

The lower bounds are based on the work by Deken [Dek79] and Dančík & Paterson [Dan94, PD94]. They present a finite automaton that models an algorithm which finds a common subsequence (CS)

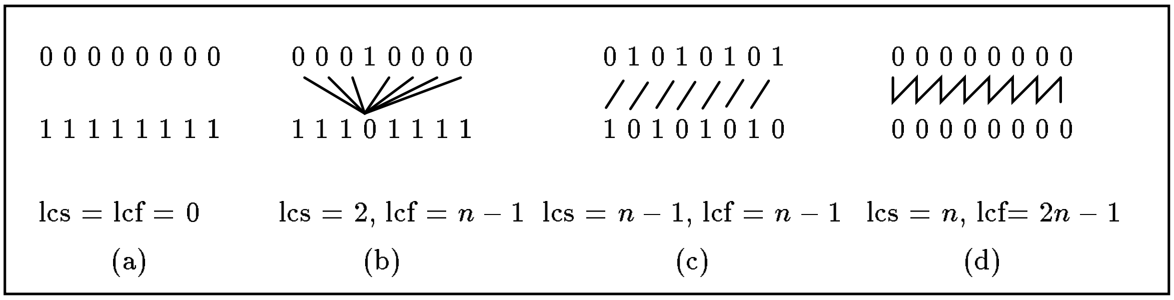

Figure 1: Some extreme LCS and LCF examples for a binary alphabet.

on two infinite strings (tapes). By analyzing the associated Markov chain, a bound on the expected length of the LCS is found. The same idea can be applied to the LCF problem.

Dančík and Paterson use an automaton that alternatively reads from each one of the two unbounded tapes. We read at the same time from both strings, allowing the possibility of applying some symmetry rules which reduce the number of states. Informally, when reading a new pair of letters of an alphabet Σ of size k with letters $\{0, 1, \cdots, k-1\}$, the automaton outputs some matches that increase the CS and computes a new state based on the letters not yet used (partial memory). Therefore, at this point, all information about the past has been lost. So, we obtain a lower bound, because potentially, a longer CS (the LCS) could have been obtained looking at the complete strings. Nevertheless, the fact that we only have to look at the current state and the future, simplifies the problem by applying the following rules. Consider that each state $s \in S$ is identified by two strings $[u, v]$ which are the letters not yet used in each tape, then:

1. We force that $|u| \geq |v|$. If it is not true, we just switch the two tapes and the behavior of the automaton is the same. This is only true because the contents of the tape are random and the letters uniformly distributed.

2. We force that $u < v$ lexicographically on their first $|v|$ letters (note that due to the previous rule, $|u| \geq |v|$). We do that by exchanging letters. If $u[1]$ is not 0, we exchange in u and v all the occurrences of $u[1]$ with 0 and vice versa. The same thing can be done with v. If $v[1] > 1$ then we exchange in u and v all the occurrences of $v[1]$ with 1 and vice versa. This is valid because the letters are indistinguishable and uniformly distributed.

These two rules diminish approximately by a factor of $2k^2$ the possible number of states that a machine like this can generate, by using classes of equivalence between states. Rule 2 can be extended recursively to $u[2]$, by permuting $u[2]$ with 2 if $u[2] > 2$, etc. We have done that for larger k, up to $k-1$ letters, reducing for every exchange the number of states by a factor of k. This symmetry is used in a similar way in [Dan94].

Formally, our CSS machine is a tuple (S, δ, O) where S is a set of states, δ is the transition function which given a state s and a pair of letters gives the new state ($s' \leftarrow \delta(s, [x, y])$), and O is the output function which given a state s and a pair of letters $[x, y]$, returns the length of the chosen CS for that transition (this is explained later). The expected behavior of a CSS machine can be modeled by a strongly connected Markov chain (no absorbing states), where the probability

n	$EL_n^{(2)}/n$	$EF_n^{(2)}/n$
1	0.5	0.5
2	0.5625	0.875
3	0.604167	1.0625
4	0.630859	1.16406
5	0.649219	1.22734
6	0.66333	1.27148
7	0.674491	1.30399
8	0.68364	1.32881
9	0.691303	1.34828
10	0.697844	1.36388
11	0.703517	1.37662
12	0.708493	1.38721
13	0.712904	1.39613
14	0.7168467	1.403736
15	0.7203977	1.4103058
16	0.7236174	1.4160315

Table 1: Exact values for EL_n/n and EF_n/n for $n \leq 16$ and $k = 2$.

of transition from one state to another state is the probability of the input letter pair associated to that transition $(1/k^2)$. In the limit, the probability of being in a given state converges to the solution of

$$\mathbf{T}\vec{p} = \vec{p}\ , \quad \sum_i p_i = 1 \quad ,$$

where $\mathbf{T}$ is the probability transition matrix and $\vec{p}$ is the steady state probability vector [CM65]. After these probabilities are obtained, a lower bound on γ_k is given by the expected length of the output

$$\gamma_k \ \geq \ \sum_{s \in S} p_s \sum_{[x,y] \in \Sigma \times \Sigma} \frac{O(s,[x,y])}{k^2} \quad .$$

CSS machines can be produced automatically as shown in [Dan94]. In our case we have a different production algorithm. The idea is that given a CSS machine $M(S, \delta, O)$, we select a subset of m states U_m from S and we expand those states. Expanding a state s means to concatenate all possible pairs of letters to s, obtaining k^2 states. We normalize each of those states by applying rules 1 and 2 defined before. That is, all the transitions of s go to these states. Of those, some of them are new. Let S' be the set of new states. For each $s' \in S'$, we compute all the possible transitions as before, but we impose the condition that the states generated by s will have at most the same number of letters of s'. If we have a larger number of letters, we drop one or two letters (we choose to delete the letters with smaller frequency). If we produce new states, we add them to S' marking s' as expanded. The condition above implies that at some point all states in S' have

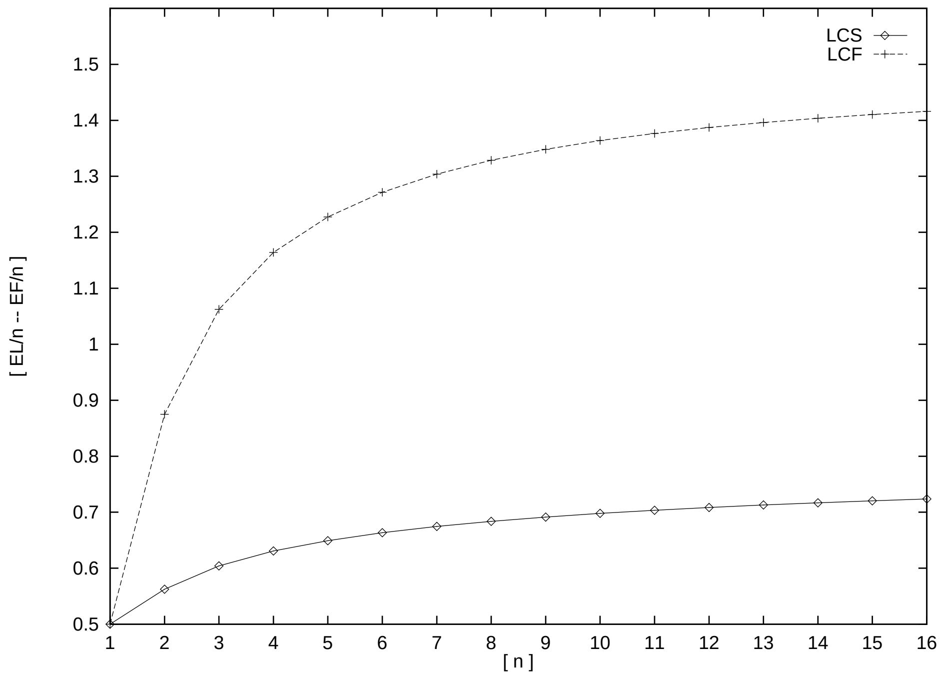

Figure 2: Exact values for EL_n/n and EF_n/n for $n \leq 16$ and $k = 2$.

been expanded, obtaining a new CSS machine M'. All states that have been expanded plus the states of M, form M'(see Figure 4).

We can repeat this process several times to obtain larger and larger CSS machines, starting with the empty state $[\lambda, \lambda]$, where λ denotes the empty string.

There are several possibilities to generate the next state in a transition. We tried several ways to do it and the most successful one was the following. Given a state $s = [u, v]$, and a pair of letters $[x, y]$, the next state is given by $s' = [u', v']$ such that $ux = w_1 u'$ and $vy = w_2 v'$ where w_1 and w_2 are the strings that maximize

$$\frac{\ell cs(w_1, w_2)}{|w_1| + |w_2|}$$

if $\ell cs(u, v) > 0$. If there is more than one candidate, we choose the one with smaller $|w_1| + |w_2|$. Otherwise, if $\ell cs(u, v) = 0$, we use $w_1 = u[1]$. In this case, for w_2, we use $w_2 = \lambda$ if $|u| > |v|$ or $|v| = 0$; otherwise $w_2 = v[1]$. This can be seen as a heuristic that locally maximizes γ_k by using the fewest possible number of letters. In practice, most of the time the cut w_1, w_2 will happen on the "best" first match from left to right. Note that it may happen that $u[1] = v[1]$ in opposition to [Dan94] where they force the starting letters to be different.

Figure 5 shows the basic CSS machine for general k for the LCS case when applying the production algorithm once starting from the empty state and using $m = 1$. The output function is shown between parenthesis, if it is not null.

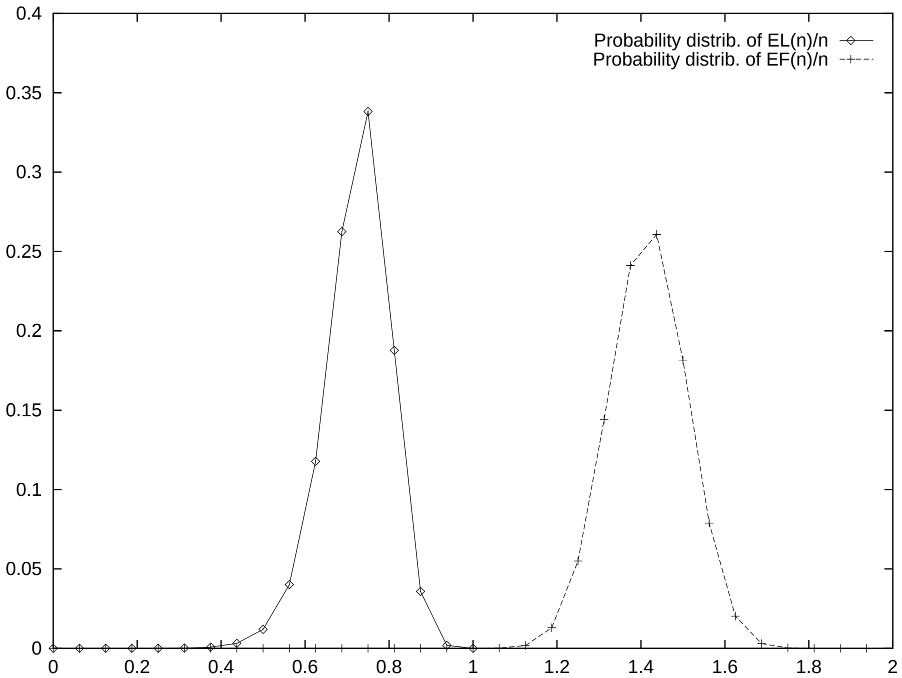

Figure 3: Probability distribution of EL_n/n and EF_n/n for $n = 16$ and $k = 2$.

The transition probability matrix of this example is

$$\mathbf{T} = \begin{bmatrix} 1/k & 1-1/k & 0 \\ (k-2)/k^2 & ((k-1)^2-(k-2))/k^2 & (2k-1)/k^2 \\ (k-1)/k^2 & (k-1)^2/k^2 & 1/k \end{bmatrix}$$

and the steady state probabilities are

$$p_0 = \frac{k^2-1}{D}\ p_1 = \frac{k^2(k-1)}{D},\ p_2 = \frac{k(2k-1)}{D},$$

with $D = k^3 + 2k^2 - k - 1$. For this automaton we have

$$\gamma_k \geq \frac{p_0}{k} + \frac{3(k-1)p_1}{k^2} + \frac{(2k-1)p_2}{k^2} = \frac{3k^2-k-1}{k^3+2k^2-k-1} = \frac{3}{k} + O(k^{-2}) \ .$$

For $k = 2$ we obtain $\gamma_2 \geq 9/13 \approx 0.6923$.

In the production algorithm we have left open the question of how to select U_m. Here, the number of states m to be expanded and the selection procedure is not fixed. In [Dan94] a next state is selected by "looking ahead" on the random input and chosing the transition where on average a longer CS is lost. Although this might be the best selection procedure, looking ahead can be computationally very expensive. They do it only for $k = 2$ using the average of all possible strings of length 6. This is not practical for $k > 2$ as the number of look ahead strings grows very

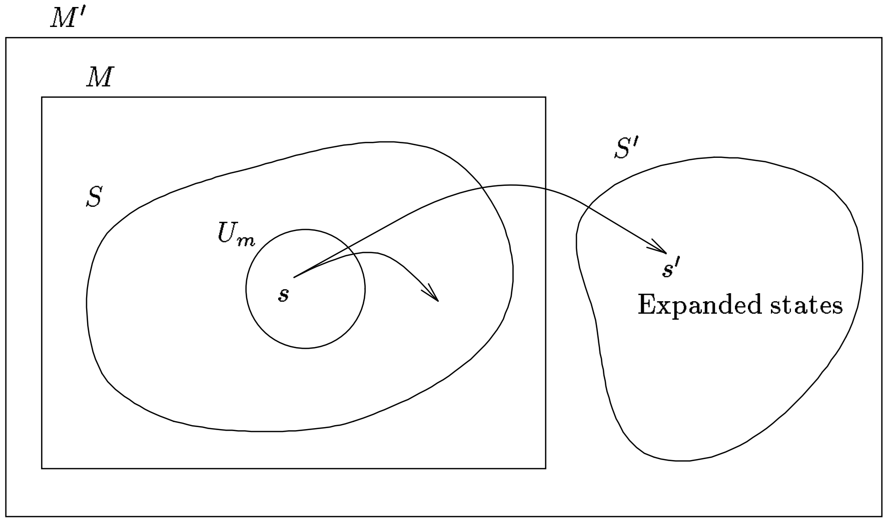

Figure 4: Production process.

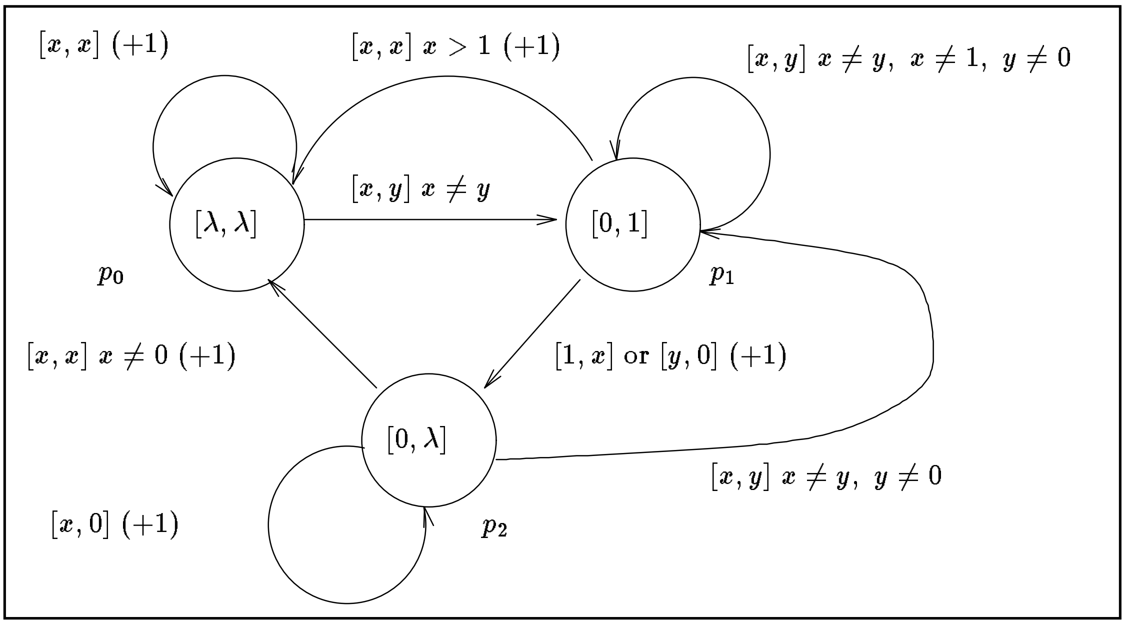

Figure 5: CSS example for LCS.

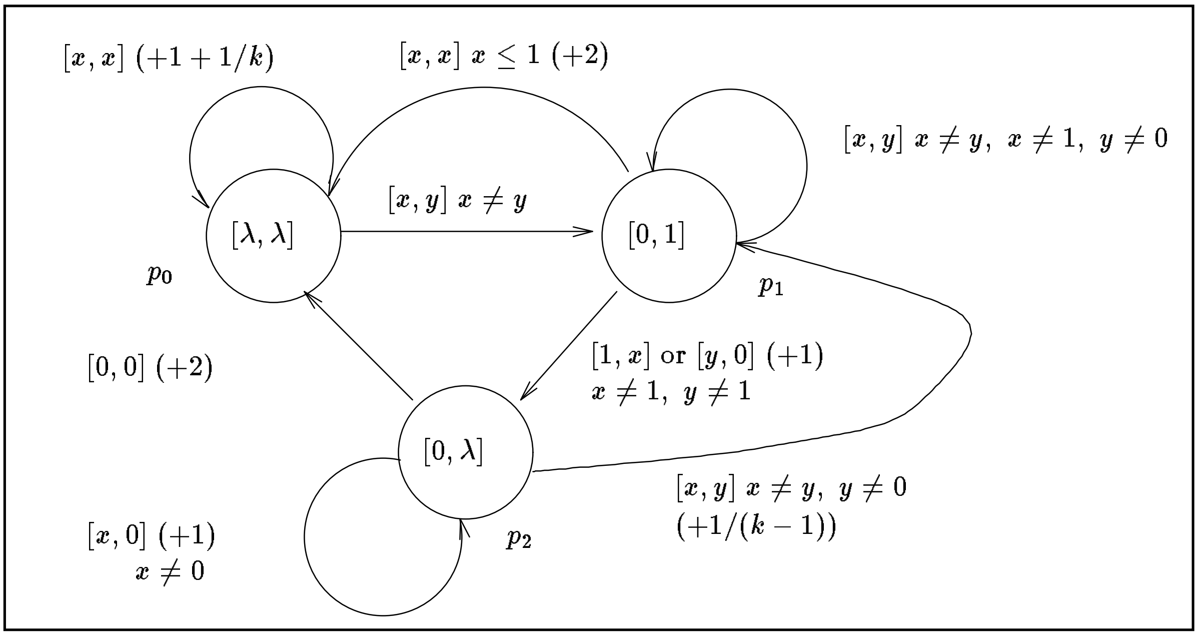

Figure 6: CSS example for LCF.

fast. For that reason, we tried different heuristic cost functions associated with a state s. The one that gave the best results was to expand the states with largest expected output, that is:

$$Cost(s) = p_s \sum_{[x,y]\in\Sigma\times\Sigma} O(s,[x,y]) \ .$$

So, the selection procedure chooses the m states with largest $Cost$ to obtain U_m. For small k we used m between 2 and 10 to speed up the growing rate of the CSS machine. For larger k, $m = 1$ was enough, as the number of states grows exponentially.

The CSS machine for the LCF problem is given in Figure 6 for the case $m = 1$. We can further improve this automaton by noticing that in states 0 and 2, the previous event is always a match. So, if one of the new letters is equal to the previous match, we can increase the LCF by 1. This has been considered in the output by adding the adequate terms which are a function of k. So, we have the following transition matrix

$$\mathbf{T} = \begin{bmatrix} 1/k & 1-1/k & 0 \\ 1/k & ((k-1)^2-(k-2))/k^2 & (2k-3)/k^2 \\ 1/k & (k-1)^2/k^2 & (k-1)/k^2 \end{bmatrix}$$

and we obtain

$$f_k \geq \frac{(k+1)p_0}{k^2} + \frac{(3k-1)}{k^2}(p_1+p_2) = \frac{3k^2-3k+2}{k^3}$$

which for $k = 2$ gives $f_2 \geq 1$.

The generation algorithm described has been implemented using the Maple symbolic algebra system [CGG+91]. Table 2 shows the lower bounds obtained so far by using our CSS machines up

k	Our γ_k	Previous γ_k	γ_k	f_k (New)	f_k
	Lower bound	[Dek79, PD94]	(Exper.)	Lower bound	(Exper.)
2	0.75875	0.77391	0.8118	**1.41031**	1.4998
3	**0.63376**	0.61538	0.7172	**1.03554**	1.2969
4	**0.55282**	0.54545	0.6537	**0.83356**	1.1426
5	**0.50952**	0.50615	0.6069	**0.67948**	1.0281
6	0.46695	0.47169	0.5701	**0.56400**	0.9403

Table 2: New lower bounds for LCS and LCF (new results in boldface), and experimental results for $n = 100,000$.

to 2000 states for the LCS and LCF problem. We were not able to obtain values for $k > 6$ due to the combinatorial explosion of the number of states. Figures 4 and 9 show these results compared to our upper bounds and experimental results.

4 Upper Bounds: Kolmogorov Complexity

The original goal of Kolmogorov complexity was to have a quantitative measure of the complexity of a finite object. Kolmogorov and others had the following idea: the regularities of an object can be used to give short descriptions of it; on the other hand, if an object is highly non-regular, or random, there should be no way of describing it that is much shorter than giving the full object itself. To formalize this notion, we first encode discrete objects as strings, as is customary in the theory of computation. Second, we want to have descriptions that can be handled algorithmically, so we identify descriptions with "programs for a sufficiently powerful model of computation".

Fix a Universal Turing Machine U whose input alphabet is $\{0,1\}$ and output alphabet is Σ. The Kolmogorov complexity of a string $s \in \Sigma^\star$ is the minimum length of a program that makes U generate s and stops.

Observe that this definition seems to depend on the choice of the Universal Turing Machine. However, it can be shown that changing the machine only affects this measure of complexity by an additive constant.

Strings whose Kolmogorov complexity are equal, or close to, their lengths are called Kolmogorov-random. These are strings that cannot be compressed algorithmically.

As there are at most $2^n - 1$ binary "programs" of length $n-1$ or less, clearly there is some string of length n whose Kolmogorov complexity is at least n. A slight generalization of this counting argument gives that for every c and n, there are at most $2^{n-c+1} - 1$ strings in Σ^n having Kolmogorov complexity $\leq n - c$.

For c even a small constant, this amounts to say that most strings, all but a fraction of 2^{-c}, are almost random: they cannot be compressed by more than c bits.

Many combinatorial properties have simple proofs via this prepackaged counting argument. Suppose that we want to show that property $P(u)$ holds for some string u. Take a Kolmogorov-random string u. Assume that $P(u)$ is false, and show that this gives a way to describe u concisely.

This is a contradiction. In fact, this argument usually gives a proof that $P(u)$ holds with high probability, as the majority of strings are Kolmogorov random up to small constants.

For example, $P(u)$ could be some static property of u, such as "the difference between zeros and ones in u is at most $2\log|u|$"[1]; or a dynamic property such as "algorithm A takes time at most $5|u|$ on input u". In fact, several lower bounds on the (worst-case and expected) running time of algorithms have been proved using Kolmogorov complexity [LV97].

To apply this kind of argument to the case of LCS, observe that if two n-bit strings have a very long LCS (that is, close to n bits), these two strings are in some sense very similar: knowing one of them gives away a lot of information about the other. Intuitively, if two strings are mutually random, knowing one of them should give essentially zero information to build the other. This must be true, in particular, if the two strings are obtained by chopping a Kolmogorov random string of $2n$ bits into two n-bit pieces. This argument is given in [LV97, Exercise 6.7.1, p.420], where is credited to Tao Jiang, though in fact it is only done for $k = 2$.

We formalize this argument for general alphabets Σ: just bear in mind that we can identify strings of length n over k letters with binary strings of length $n\log k$.

We will determine γ such that $\ell cs(u,v) \leq \gamma n$ for Kolmogorov random strings u and v. Then averaging over all strings we obtain $EL_n^{(k)} \leq \gamma_k n + O(1/n)$. Indeed, let A be the set of words uv $(u,v \in \Sigma^n)$ that have Kolmogorov complexity at least $(2n - 3\log n)\cdot\log k$. See that all but a fraction $O(1/n^3)$ of strings have this property. Then

$$\begin{aligned}
EL_n^{(k)} &= 1/k^{2n}\left[\sum_{uv\in A}\ell cs(uv) + \sum_{uv\notin A}\ell cs(uv)\right]\\
&\leq 1/k^{2n}\left[\sum_{uv\in A}\gamma n + \sum_{uv\notin A} n\right]\\
&\leq 1/k^{2n}\left[k^{2n}(1-O(1/n^3))\gamma n + k^{2n}O(1/n^3)n\right]\\
&= (1+O(1/n^3))\gamma n.
\end{aligned}$$

Assume $\ell cs(u,v) = \gamma n$. Clearly we can obtain uv if we have the following information:

- The values of n and γn.
- The length of the LCS of u and v: $\ell cs(u,v)$.
- A description of the letter positions of u and v that give the LCS.
- The sequence of letters of u that do not belong to the LCS.
- The sequence of letters of v that do not belong to the LCS.

Formally, there is a fixed program (independent of n, u, and v) that, given this information, makes the Universal Turing Machine produce uv. As uv is random, the length of writing down

[1] All logarithms in this paper are in base 2.

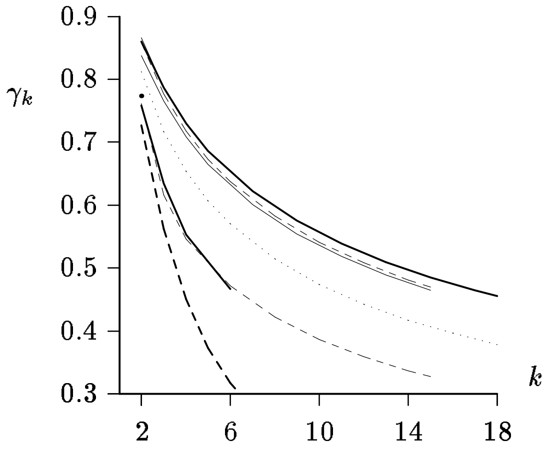

Figure 7: The bold solid lines are our lower and upper bounds on γ_k for each alphabet size k. They are compared with the results of Chvátal and Sankoff [CS75] (bold dashed line is the lower bound, the upper bound matches ours for $k > 2$), Deken [Dek79] (dashed line) and Dančík [Dan94] (solid line is the upper bound, and the bullet for $k = 2$ is the lower bound). In between we show experimental results for $n = 100,000$ (dotted line).

this information in bits, plus the size of this program, must be at least $(2n - 3\log n)\log k$. Let us estimate the bit-length of each part.

The values of n and γn can be given in $2\log n$ bits each. By assumption, the LCS can be encoded in $(\gamma n)\log k$ bits. The bits necessary to specify the letter positions is the log of the number of position sets that correspond to LCS's of two strings. Call this number $I_{n,\gamma}$.

For the last item, we use the following. A pair of strings may have several LCS's. We take as a representative that one with a lexicographically smallest set of positions: that is, if there are two choices for matching a letter we match it with the lowest index. Then, for every letter not in the LCS we can discard one out of k possibilities: if adjacent letters from positions i to j of u are not in the LCS, but letter $j+1$ is, we know that $u[k] \neq u[j+1]$, for any $i \leq k \leq j$. Hence, the $(1-\gamma)n$ letters of u not in the LCS can be encoded given as a string of length $(1-\gamma)n$ over an alphabet with $k-1$ letters, and similarly for v. In particular, for $k = 2$, this information is empty.

Adding up, we obtain the equation

$$4\log n + \gamma_k n \log k + \log I_{n,\gamma_k} + 2(1-\gamma_k)n\log(k-1) \geq (2n - 3\log n)\log k \quad .$$

Dividing the equation by n, all sublinear terms vanish asymptotically, so we obtain:

$$\frac{\log I_{n,\gamma_k}}{n} + 2(1-\gamma_k)\log(k-1) \geq (2-\gamma_k)\log k \quad . \tag{1}$$

A first upper bound on I_{n,γ_k} is the number of all subsets of $\{1\ldots n\}$ with $\gamma_k n$ elements, squared (once for choosing in u, times the choice for v). By Stirling's approximation, $\log\binom{n}{\gamma_k n} = nH(\gamma_k)(1+o(1))$, where $H(x) = -x\log(x) - (1-x)\log(1-x)$ is the binary entropy function. So we obtain the equation

$$2H(\gamma_k) + 2(1-\gamma_k)\log(k-1) \geq (2-\gamma_k)\log k \quad .$$

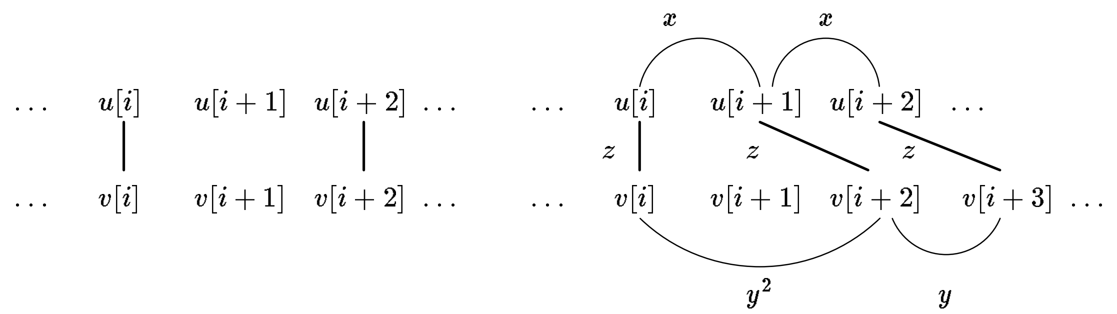

Figure 8: Forbidden case for a LCS with $k = 2$ (left), and counting variables used (right).

For every k, solving this equation numerically gives a feasible range for γ_k. For example, for $k = 2$ it gives $0.282 \leq \gamma_2 \leq 0.867$. Figure 4 plots the values of γ_k up to $k = 18$, as well as experimental results for $n = 100,000$ (average taken over ten trials). Table 3 gives some exact values. By taking the limit on k, we obtain the already known result $\gamma_k \leq e/\sqrt{k}$.

For $k = 2$ this is the result obtained by Tao Jiang using a technique from [JL95] as mentioned in [LV97]. Although this result is also equivalent to the note added in proof in [CS75] (which is explained in [CS83]), using Kolmogorov complexity is much easier to understand and simpler. We obtain a better bound for $k = 2$ by estimating more accurately the number of positions I_{n,γ_k}.

Consider the example given in Figure 8. If letters $u[i+1]$ and $v[i+1]$ are equal, we can match them and obtain a longer common sequence. If they are different, one of them equals $u[i+2] = v[i+2]$, so we can match it with either $u[i+2]$ or $v[i+2]$ and obtain a lexicographically smaller set of positions. So we have to count sets of positions that do not leave gaps simultaneously on both strings.

As we will take the log of the number of strings divided by n for large n, we disregard smaller terms such as leading polynomials, etc., without further notice. In particular, we count only those strings that end with a match; it is not hard to see that this does not affect the main term.

To count the number of LCS alignments, we use generating functions. Let $G(x, y, z)$ be

$$G(x, y, z) = \sum_{n_1, n_2, \ell} G_{n_1, n_2, \ell}\, x^{n_1} y^{n_2} z^{\ell}$$

where $G_{n_1,n_2,\ell}$ is the number of alignments such that the upper string is of length n_1, the lower string is of length n_2, and the LCS is of length ℓ. That is, in the symbolic variable x we accumulate the number of skipped letters between two consecutive positions of the upper string which participate in the alignment. We also count the aligned letter, so at the end we have in x the length of the upper string. We do the same for the lower string on the variable y. Finally, z counts the number of edges in the alignment between both strings. The counting model is depicted in Figure 8. So, we are interested in $G_{n,n,n\gamma}$.

When using generating functions to describe combinatorial objects and count their properties, the multiplication represents object composition and the addition represents union of different objects. In our case we have,

$$G(x, y, z) = \left(\frac{1}{1-y} + \frac{x}{1-x}\right) yxzG(x, y, z) + 1 = \frac{1}{1 - \left(\frac{1}{1-y} + \frac{x}{1-x}\right) xyz}$$

This formula counts all the forms to build all the possible alignments, as follows: either the alignment has an edge or not. The first case is the main part of the formula: we put zero or more initial letters either in y or in x (as we explained, there cannot be gaps in both strings), i.e. $\sum_{i\geq 0} x^i + \sum_{j\geq 0} y^j = 1/(1-x) + 1/(1-y)$. However, we subtract 1, since otherwise the case of both empty strings would be counted twice (hence $x/(1-x)$ instead of $1/(1-x)$). After this, we have an edge xyz (to count the aligned letters in x and in y, and the edge z). After this, we add more letters and edges by recursively concatenating more elements of the same form, that is $G(x,y,z)$. The recursion ends when we determine that no more edges are to be added. This corresponds to the leading $+1$ of the formula (i.e. $x^0y^0z^0$). This is because, as explained, we assume that the last letters are aligned and hence if there are no alignments the strings must be empty.

We now extract the coefficients $G_{n_1,n_2,\ell}$ from the function $G(x,y,z)$. We begin by extracting ℓ. Since $1/(1-az) = \sum_{\ell\geq 0} a^\ell z^\ell$ we have

$$G_\ell(x,y) = (xy)^\ell \left(\frac{1}{1-y} + \frac{x}{1-x}\right)^\ell = \sum_i \binom{\ell}{i} \frac{x^{i+\ell} y^\ell}{(1-x)^i (1-y)^{\ell-i}}$$

and since $x^r/(1-x)^{s+1} = \sum_{i\geq 0} \binom{i-r+s}{s} x^i$ for $s \geq 0$ (and the same property is used for y),

$$G_{n_1,n_2,\ell} = \sum_i \binom{\ell}{i} \binom{n_1-\ell-1}{i-1} \binom{n_2-i-1}{\ell-i-1}$$

where we have discarded some clumsy border conditions which do not affect the main term. Finally,

$$G_{n,n,\ell} = \sum_i \binom{\ell}{i} \binom{n-\ell-1}{i-1} \binom{n-i-1}{\ell-i-1}$$

We do not need the exact solution to the above sum, just its logarithm divided by n, for large n. Call $M_{n,\ell}$ the maximum term of the summation. Then we have

$$\begin{array}{rcccl} M_{n,\ell} & \leq & G_{n,n,\ell} & \leq & \ell M_{n,\ell} \\ \log(M_{n,\ell})/n & \leq & \log(G_{n,n,\ell})/n & \leq & \log(M_{n,\ell})/n + O\left(\frac{\log n}{n}\right) \end{array}$$

which shows that the larger term dominates the result. Moreover, we can maximize the logarithm of the term and use Stirling as before. Let $i = wn$, take the logarithm of the term i of the sum, divide by n and maximize with respect to w. We obtain that the maximum is reached for

$$w(\gamma) = \frac{2-\gamma-\sqrt{5\gamma^2-8\gamma+4}}{2}$$

that satisfies the constraints of the sum, namely $0 \leq w(\gamma) \leq \min(\gamma, 1-\gamma)$. By using this maximum term instead of the whole sum, and using the asymptotic formula $\log\binom{\alpha n}{\beta n} = \alpha n H(\beta/\alpha) + O(\log n)$, we have

$$\gamma H(w(\gamma)/\gamma) + (1-\gamma)H(w(\gamma)/(1-\gamma)) + (1-w(\gamma))H((\gamma-w(\gamma))/(1-w(\gamma)) \geq 2-\gamma$$

whose numerical solution is

$$\gamma_2 \leq 0.86019$$

which is still larger than what other more complicated theoretical models provide [Dan94], although quite close.

Let us now consider the longest common forest problem. The LCF allows a better letter representation, since in this case not only each not connected letter must be different than that of the next alignment. The letters corresponding to each tree of the forest must be different than that of the next tree (otherwise we could join both trees). Hence, we need $\log(k-1)$ bits for all letters (connected and not connected), except the first one. For example, we need only one bit for $k = 2$. Therefore, our inequality is

$$\frac{\log I_{n,f_k}}{n} + (2 - f_k)\log(k-1) \geq 2\log k \tag{2}$$

The next step is to obtain a bound for I_{n,f_k}, the number of configurations for the forest. In this case, a single letter can be matched to many, so we drop the requirement for at least one gap between two edges. However, not both gaps can be zero. Hence,

$$G(x,y,z) = \left(\frac{1}{(1-x)(1-y)} - 1\right) zG(x,y,z) + 1 = \frac{1}{1 - \left(\frac{1}{(1-x)(1-y)} - 1\right) z}$$

where the base case (leading $+1$) is as before. The recursive case considers a number of letters in the upper and lower strings (this time there can be letters in both of them simultaneously). The z represents the edge, and this time it is not coupled with xy because there may be many edges for a single letter. However, before putting a new edge we must add at least one letter in the upper or lower strings (otherwise we are putting two edges coupling the same pair of letters), and hence we subtract $1 = x^0y^0$ to the formula in parenthesis. The recursive step multiplies $G(x,y,z)$ at the end.

Computing the ℓ-th coefficient as before, rewriting $1/((1-x)(1-y)) - 1 = (x+y-xy)/((1-x)(1-y))$, and using the binomial expansion we have

$$\begin{aligned}
G_\ell(x,y) &= \frac{(x+y-xy)^\ell}{((1-x)(1-y))^\ell} \\
&= \sum_{i,j} (-1)^{\ell-i-j} \binom{\ell}{i} \binom{\ell-i}{j} \frac{x^i y^j (xy)^{\ell-i-j}}{((1-x)(1-y))^\ell}
\end{aligned}$$

Now we use the same properties as before to invert in x and y and get

$$\begin{aligned}
G_{n_1,n_2,\ell} &= \sum_{i,j} (-1)^{\ell-i-j} \binom{\ell}{i} \binom{\ell-i}{j} \binom{n_1+j-1}{\ell-1} \binom{n_2+i-1}{\ell-1} \\
&= \sum_{i} (-1)^{\ell+i} \binom{\ell}{i} \binom{n_2+i-1}{\ell-1} \sum_{j} (-1)^{j} \binom{\ell-i}{j} \binom{n_1+j-1}{\ell-1} \\
&= \sum_{i} (-1)^{\ell+i} \binom{\ell}{i} \binom{n_2+i-1}{\ell-1} (-1)^{\ell-i} \binom{n_1-1}{i-1}
\end{aligned}$$

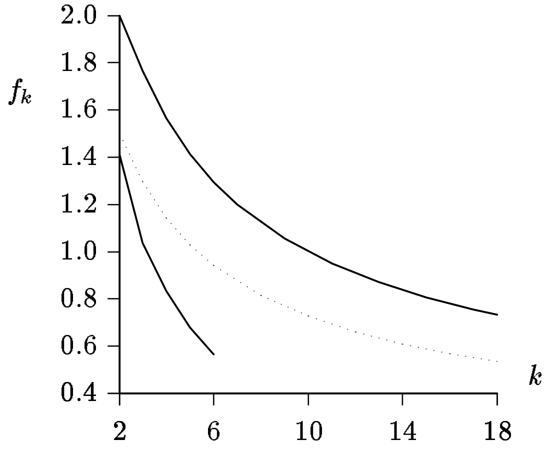

Figure 9: Bold solid lines show the upper and lower bounds for f_k, for each alphabet size k. In between we show experimental results for $n = 100,000$ (dotted line).

where we disregard again border conditions which do not affect the main term. Hence,

$$G_{n,n,\ell} = \sum_i \binom{\ell}{i} \binom{n+i-1}{\ell-1} \binom{n-1}{i-1}$$

Using the same maximizing technique as before ($i = wn$), we have

$$w(f) = \frac{-1+\sqrt{1+4f}}{2}$$

This maximum value for $i = w(f)n$ is always in the bounds of the summation (i.e. $\max(f-1,0) \leq w(f) \leq \min(f,1)$). Then, we have

$$fH(w(f)/f) + (1+w(f))H(f/(1+w(f)) + H(w(f)) \geq 2\log k - (2-f)\log(k-1) \ .$$

We can now numerically solve this inequality for each alphabet size k. Figure 9 plots the values of f_k up to $k = 18$ as well as experimental results for $n = 100,000$ (average taken over ten trials), and Table 3 shows some exact values. These are the first theoretical upper bounds for the LCF problem. Taking the limit on k, we obtain

$$f_k \leq \frac{e}{\sqrt{k}} \ + \ O(1/k) \ \ .$$

5 Final Remarks

Algorithms to find a long CS of two sequences can be considered as approximation algorithms for this problem (they can also be seen as on-line algorithms with restricted memory). The complexity of these algorithms is in general $O(n)$, which compares favorably with $O(n^2)$.

k	Our γ_k Upper bound	Previous γ_k [DP95, Dan94]	f_k (New) Upper bound
2	0.86019	0.83763	**2.00000**
3	0.78647	0.76581	**1.76704**
4	0.72971	0.70824	**1.56594**
5	0.68612	0.66443	**1.41289**
6	0.65098	0.62932	**1.29384**
7	0.62172	0.60019	**1.19855**
8	0.59676	0.57541	**1.12033**
9	0.57507	0.55394	**1.05478**
10	0.55597	0.53486	**0.99890**
15	0.48538	0.46462	**0.80753**

Table 3: Upper bounds for LCS and LCF (new results in boldface).

The general case of finding the LCS of m sequences of length n can be solved in time $O(n^{\ell})$ using dynamic programming. If ℓ is not fixed, this problem is NP-complete [Mai78]. Jiang and Li [JL95] show that it is difficult to find good approximation algorithms in the worst case for the LCS, because if there is a polynomial time approximation algorithm with performance ratio n^{δ} ($\delta > 0$), then $\mathbf{P} = \mathbf{NP}$. For that reason, it is better to look at good approximation algorithms for random inputs. For the case of ℓ sequences of length n, with ℓ a polynomial in n, simple greedy algorithms approximate the LCS of the sequences with an expected additive error of size $O(\sqrt{n^{1+\epsilon}})$ [JL95]. The expected length of the sequence in this case is n/k for an alphabet of size k. That is, $\gamma_k \to 1/k$ when $\ell = O(n)$. Note that for $\ell = 2$, $\gamma_k = O(1/\sqrt{k})$. Dančík [Dan94, Dan95] has proved that in the case of ℓ sequences

$$1 \leq \gamma_k k^{1-1/\ell} \leq e \quad .$$

The approximation ratio of our algorithms for two sequences in the worst case is unbounded. On average, CSS machines are approximation algorithms with expected additive error at most $O(n/k)$, but the exact complexity is not known. One possible measure is the ratio between the exact value of γ_k and the expected length of the CS obtained by the algorithm. In our case, because the exact value is not known, we can use an upper bound. For example, for $k = 2$ we have $\gamma_2 < 0.838$ [DP95]. So, the automaton given for the case $m = 1$ would be at most 1.22-sub-optimal for random sequences. For larger k, the ratio is at most $e/3\sqrt{k}$ for the case $m = 1$.

We are also working on upper bounds, refining the codification methods to improve the Kolmogorov bounds. For example, one can take pairs of letters and observe that some configurations are in fact not possible, thus reducing the number of alternatives to represent and hence improving the bound. However, the analysis becomes much more complex.

Finally, we would like to say a few words about the experimental results. We tried to fit several curves using least squares on the last 10 values (considering that we want to obtain the asymptotic behavior), to see the real dependency on k of the expected value. Sankoff and Mainville [SM83] have conjectured that $\gamma_k \approx 2/\sqrt{k}$. Fitting the curve a/k^{β} we got $a = 1.15$ and $\beta = 0.38$ for γ_k and

$a = 2.42$ and $\beta = 0.52$ for f_k with relative errors of 0.08% and 0.03%, respectively. Doing the same for the curve $a/\sqrt{k} + b/k$ we obtained $a = 1.90$ and $b = -1.27$ for γ_k and $a = 2.19$ and $b = 0.35$ for f_k, both with a relative error of 0.05%. This supports the same conjecture for the case of the LCF, although gives no definitive answer for the case of the LCS.

Acknowledgements

Some ideas for this work originated while the second author was visiting the University of Chile in Santiago during 1995 and attending the XV Conference of the Chilean CS Society (SCCC) in Arica. He is grateful to Eric Goles and Martín Matamala for inviting him to the first, and to the SCCC and particularly Ricardo Baeza-Yates for inviting him to the second. This work continued thanks to the kind invitation of Josep Díaz to the first author to do a sabbatical at the Technical University of Barcelona and to the third author to visit the same place during February of 1996. Finally, we thank the helful comments of the referees.

References

[BYGN96] R. Baeza-Yates, R. Gavaldá, and G. Navarro. Bounding the expected length of longest common subsequences and forests. In R. Baeza-Yates N. Ziviani and K. Guimarães, editors, *Proc. of WSP'96*, pages 1–15, Recife, Brazil, August 1996.

[BYS95] R. Baeza-Yates and R. Scheihing. New lower bounds for the expected length of longest common subsequences and forests. In *XV International Conference of the Chilean Computer Science Society*, pages 48–58, Arica, Chile, November 1995.

[CGG+91] B. Char, G. Geddes, G. Gonnet, B. Leong, M. Monagan, and S. Watt. *MAPLE V Language and Library Reference Manual.* Springer-Verlag, 1991.

[CM65] D. Cox and H. Miller. *The Theory of Stochastic Processes.* Chapman and Hall, London, 1965.

[CS75] V. Chvatal and D. Sankoff. Longest common subsequences of two random sequences. *Journal of Applied Probability*, 12:306–315, 1975.

[CS83] V. Chvatal and D. Sankoff. An Upper-Bound Technique for Lengths of Common Subsequences. In D. Sankoff and J. Kruskal, editors, *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, pages 353–357, 1983. Addison-Wesley, Reading, Mass.

[Dan94] V. Dančík. *Expected Length of Longest Common Subsequences.* PhD thesis, CS Dept, Univ. of Warwick, Warwick, UK, 1994.

[Dan95] V. Dančík. Common subsequences and supersequences and their expected length. In Z. Galil and E. Ukkonen, editors, *Proceedings of 6th Annual Symposium Combinatorial Pattern Matching*, LNCS 937, pages 55–63, Helsinki, Finland, July 1995. Springer Verlag.

[Dek79] J. Deken. Some limit results for longest common subsequences. *Discrete Mathematics*, 26:17–31, 1979.

[Dek83] J. Deken. Probabilistic Behavior of Longest-Common-Subsequence Length. In D. Sankoff and J. Kruskal, editors, *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, pages 359–362, 1983. Addison-Wesley, Reading, Mass.

[DP95] V. Dančík and M. Paterson. Upper bounds for the expected length of a longest common subsequence of two binary sequences. *Random Structures & Algorithms*, 6:449–458, 1995.

[GBY91] G.H. Gonnet and R. Baeza-Yates. *Handbook of Algorithms and Data Structures - In Pascal and C.* Addison-Wesley, Wokingham, UK, 1991. (second edition).

[JL95] T. Jiang and M. Li. On the approximation of shortest common supersequence and longest common subsequences. *SIAM Journal on Computing*, 24(5):1112–1139, Oct 1995.

[LV97] Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications.* Springer-Verlag, New York, 1997, second edition.

[Mai78] D. Maier. The complexity of some problems on subsequences and supersequences. *J.ACM*, 25:322–336, 1978.

[PD94] M. Paterson and V. Dančík. Longest common subsequences. In B. Rovan I. Privara and P. Ruzicka, editors, *19th MFCS'94*, LNCS 841, pages 127–142, Kosice, Slovakia, August 1994. Springer Verlag.

[PW93] P. Pevzner and M. Waterman. Generalized sequence alignment and duality. *Advances in Applied Mathematics*, 14:139–171, 1993.

[Ric95] Claus Rick. A new flexible algorithm for the longest common subsequence problem. In *CPM'95, 6th Annual Symposium on Combinatorial Pattern Matching*, Lecture Notes in Computer Science 937, pages 340–351, Espoo, Finland, 1995. Springer-Verlag.

[SM83] D. Sankoff and S. Mainville. Common Subsequences and Monotone Subsequences. In D. Sankoff and J. Kruskal, editors, *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, pages 363–365, 1983. Addison-Wesley, Reading, Mass.