Improved Bounds on the Average Length of Longest Common Subsequences

GEORGE S. LUEKER

University of California, Irvine, California

Abstract. It has long been known [Chvátal and Sankoff 1975] that the average length of the longest common subsequence of two random strings of length n over an alphabet of size k is asymptotic to $\gamma_k n$ for some constant γ_k depending on k. The value of these constants remains unknown, and a number of papers have proved upper and lower bounds on them. We discuss techniques, involving numerical calculations with recurrences on many variables, for determining lower and upper bounds on these constants. To our knowledge, the previous best-known lower and upper bounds for γ_2 were those of Dančík and Paterson, approximately 0.773911 and 0.837623 [Dančík 1994; Dančík and Paterson 1995]. We improve these to 0.788071 and 0.826280. This upper bound is less than the γ_2 given by Steele's old conjecture (see Steele [1997, page 3]) that $\gamma_2 = 2/(1+\sqrt{2}) \approx 0.828427$. (As Steele points out, experimental evidence had already suggested that this conjectured value was too high.) Finally, we show that the upper bound technique described here could be used to produce, for any k, a sequence of upper bounds converging to γ_k, though the computation time grows very quickly as better bounds are guaranteed.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*pattern matching*; G.2.1 [**Discrete Mathematics**]: Combinatorics—*generating functions, recurrences and difference equations*; G.3 [**Mathematics of Computing**]: Probability and Statistics

General Terms: Algorithms, Experimentation, Theory

Additional Key Words and Phrases: Longest common subsequences, average-case analysis, dynamic programming, Arratia-Steele conjecture

ACM Reference Format:

Lueker, G. S. 2009. Improved bounds on the average length of longest common subsequences. J. ACM 56, 3, Article 17 (May 2009), 38 pages. DOI = 10.1145/1516512.1516519 http://doi.acm.org/10.1145/1516512.1516519

17

This article combines results that were presented in preliminary form in "Expected length of longest common subsequences," in *DIMACS Workshop on Probabilistic Analysis of Algorithms for Hard Problems*, 1999; "Improved bounds on the average length of longest common subsequences," in *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, ACM, New York, 2003, 130–131; and "On the convergence of upper bound techniques for the average length of longest common subsequences," in *Proceedings of the 4th Workshop on Analytic Algorithmics and Combinatorics*, 2008, 169–182.

Author's address: Department of Computer Science, Donald Bren School of Information & Computer Sciences, University of California, Irvine, Irvine, CA 92697-3435.

DOI 10.1145/1516512.1516519 http://doi.acm.org/10.1145/1516512.1516519

1. *Introduction*

Given a string of characters $A = a_1a_2 \cdots a_n$, a *subsequence* of A is any string which can be obtained by deleting some of the characters in A. Given another string $B = b_1b_2 \cdots b_m$, a *longest common subsequence* (abbreviated *lcs*) of A and B is a longest string which is a subsequence of both A and B; we denote the length of such a string by $L(A, B)$.

Let a *random string* of length n be a string of length n in which each character is chosen uniformly and independently from $\Sigma = \{0, 1, \ldots, k-1\}$. Let $L_{n,n}$ be the random variable giving the length of a longest common subsequence of two independent random strings of length n. By subadditivity there exists a constant $\gamma_k > 0$, depending on k, such that [Chvátal and Sankoff 1975]

$$\lim_{n\to\infty} \frac{\mathsf{E}[L_{n,n}]}{n} = \gamma_k.$$

While the values of the γ_k are not known exactly, determination of bounds on their value has drawn attention; for an early reference, see Chvátal and Sankoff [1975]. For some history and more references, see Baeza-Yates et al. [1999], Dančík [1994], Dančík and Paterson [1995], and Steele [1997]. (For the problem of the longest common subsequence of n, rather than just 2, random strings of length n, the corresponding value is known to be $1/k$ [Jiang and Li 1995].) The breakthrough paper [Kiwi et al. 2005] showed that

$$\lim_{k\to\infty} \sqrt{k}\gamma_k = 2,$$

establishing the Sankoff-Mainville conjecture. In this article, we primarily concentrate on bounding γ_2. To my knowledge, the best previous bounds on γ_2, both upper and lower, are those of Dančík [1994] and Dančík and Paterson [1995], namely, 0.773911 and 0.837623.

Section 2 presents an improved lower bound of 0.788071, and Section 3 presents an improved upper bound of 0.826280. This upper bound finally resolves in the negative Steele's old conjecture (see Steele [1997, page 3]) that $\gamma_2 = 2/(1+\sqrt{2}) \approx 0.828427$; as Steele points out, experimental evidence had already suggested that this conjectured value was too high. Put another way, this shows that the Arratia-Steele conjecture (see Pevzner [2000, Section 6.8, page 107]) fails even for alphabet size 2. Section 3.3 shows that, for any $k \geq 2$, the method of Section 3 can produce a sequence of bounds converging to the true value of γ_k.

We hope that this article will be of wide interest because

—the longest common subsequence problem has a variety of applications and has been extensively studied,

—we show how a method involving extensive computation can be applied to the average-case analysis of this problem to improve bounds published in the mid-90s and disprove a conjecture that has been open since the 80s,

—we carefully consider the effect of roundoff error on the validity of the bounds we obtain, and

—we show that for any $k \geq 2$ our upper bound technique yields upper bounds that converge to the true value of γ_k.

2. *Lower Bounds*

We only consider lower bounds for the case of binary alphabets, that is, $k = 2$, so in this section we will simply write γ to denote γ_2.

2.1. PREVIOUS RESULTS. To obtain lower bounds, Dančík [1994] uses the following approach. Build a finite-state automaton that can read input characters, left-to-right, from the strings A and B; it produces one character of output whenever it matches a character from A and B. The behavior of this machine on random input can be modeled by a Markov chain. Under reasonable conditions on the machine, one can find the steady-state distribution of the chain and then compute the expected number of characters output at each transition as the dot product of the vector giving the stationary probability for each state and the vector giving the expected number of characters output on the next move from each state. In fact, by a result of Alexander [1994] (given in (2) below) one need not require that the machine read the strings A and B at the same rate; one can obtain a lower bound on γ by simply controlling the total number of characters read. In Dančík [1994], a specific finite state automaton, with 931 states, is used to produce a lower bound of 0.773911; a mechanical procedure that could be used to produce a sequence of machines that give lower bounds that approach the true value of γ in the limit is also presented there.

2.2. A DYNAMIC PROGRAMMING APPROACH. In this section, we briefly describe our variation and present the resulting improved lower bounds on γ. Intuitively, in order to produce a machine with a finite number of states, we simply keep a buffer of h characters from each of the two strings being matched. When possible, we match and delete the leftmost characters of the two buffers; when not, we have to discard a character from the left of one of the buffers. As these characters disappear from the left ends of the buffers, we read in new characters from the input strings at the right ends of the buffers. This is quite similar to the method discussed, for example, in Dančík [1994] and Paterson and Dančík [1994]; the key difference is that instead of being careful in choosing which states (corresponding to buffer contents of various lengths) to use, we simply let the state set be all 2^{2h} possible distinct pairs of strings of length h.

Let $x_1x_2\cdots x_n$ and $y_1y_2\cdots y_n$ be two independent random binary strings of length n; let $X(i)$ (respectively, $Y(j)$) denote the string $x_1x_2\cdots x_i$ (respectively, $y_1y_2\cdots y_j$). As noted in Dančík [1994, Theorem 2.2], if we define the *diagonal longest common subsequence* by

$$D_n = \max_{i+j=n} L(X(i), Y(j)), \tag{1}$$

then it follows from the results of Alexander [1994] that

$$\mathsf{E}[D_n] \sim \gamma n/2. \tag{2}$$

Now let s and t be specific binary strings, and define

$$v_n(s,t) = \mathsf{E}\Big[\max_{i+j=n} L(sx_1x_2\cdots x_i, ty_1y_2\cdots y_j)\Big].$$

Informally $v_n(s,t)$ tells us the expected length of the longest common subsequence we can obtain from s and t if we are allowed to first pad them randomly

with n additional characters. One easily sees that regardless of s and t,

$$\gamma = \lim_{n\to\infty} \frac{1}{n} v_{2n}(s,t); \tag{3}$$

note that the fact that we need not pad s and t with equal numbers of characters follows from (2). Let $\vec{v}_n$ be the vector whose components are all the values of $v_n(s,t)$, as s and t range over all strings of length h. Using ideas similar to the standard dynamic programming solution (see, e.g., [Pevzner 2000, Ch. 6]), it is straightforward to obtain a lower bound on each component of $\vec{v}_n$ as a function of elements of $\vec{v}_{n-1}$ or $\vec{v}_{n-2}$. For example, with $h = 3$, by matching the initial 0 in 001 and 010, we can conclude

$$v_n(001, 010) \geq 1 + \frac{1}{4} \sum_{(c,c')\in\{0,1\}\times\{0,1\}} v_{n-2}(01c, 10c'); \tag{4}$$

as another example, even though 001 and 111 begin with different characters, we can discard a character from the left end of either, so we conclude

$$v_n(001, 111) \geq \max \begin{cases} \frac{1}{2} \sum_{c\in\{0,1\}} v_{n-1}(01c, 111) \\ \frac{1}{2} \sum_{c\in\{0,1\}} v_{n-1}(001, 11c). \end{cases} \tag{5}$$

Let $\Pi_1(\vec{v})$ (respectively, $\Pi_0(\vec{v})$) denote the projection that selects just the components of $\vec{v}$ corresponding to pairs of strings whose first characters match (respectively, do not match). Let $(\cdot \parallel \cdot)$ denote the operator that reconstructs an entire vector from its two projections, so that

$$\vec{v} = (\Pi_0(\vec{v}) \parallel \Pi_1(\vec{v})). \tag{6}$$

Note that for a suitably defined T_1, all of the bounds exemplified in (4) can be written as

$$\Pi_1(\vec{v}_n) \geq T_1(\vec{v}_{n-2}).$$

Similarly, one can define a T_0 such that all the bounds exemplified in (5) can be written as

$$\Pi_0(\vec{v}_n) \geq T_0(\vec{v}_{n-1}).$$

Note that both T_0 and T_1 are *translationally invariant* in the sense that for any real r,

$$T_0(\vec{v} + r) = T_0(\vec{v}) + r \text{ and } T_1(\vec{v} + r) = T_1(\vec{v}) + r, \tag{7}$$

where addition of a real to a vector is done componentwise, for example, $(0.1, 0.2) + 3 = (3.1, 3.2)$. They are also *monotonic* in the sense that

$$\vec{v}' \geq \vec{v} \implies T_0(\vec{v}') \geq T_0(\vec{v}) \text{ and } T_1(\vec{v}') \geq T_1(\vec{v}),$$

where the inequalities hold componentwise.[1]

Also note that T_0, while not being a linear transformation, does *scale linearly* in the sense that

$$T_0(\alpha\vec{v}) = \alpha T_0(\vec{v}). \tag{8}$$

[1] This separate discussion of T_0 and T_1 rather than directly defining T may seem somewhat cumbersome, but it will provide a useful foundation for Section 2.3.2.

While T_1 does not scale linearly, one easily sees that $T_1 - 1$ does, in the sense that

$$T_1(\alpha\vec{v}) - 1 = \alpha(T_1(\vec{v}) - 1),$$

implying

$$T_1(\alpha\vec{v}) = \alpha T_1(\vec{v}) + 1 - \alpha. \tag{9}$$

Now if we define T by

$$T(\vec{v}, \vec{v}') = (T_0(\vec{v}) \parallel T_1(\vec{v}')), \tag{10}$$

all of the bounds in (4) and (5) can be written as

$$\forall n \geq 2 \quad \vec{v}_n \geq T(\vec{v}_{n-1}, \vec{v}_{n-2}). \tag{11}$$

Note that this T inherits translational invariance and monotonicity from T_0 and T_1.

OBSERVATION 2.1. *If T is monotonic and translationally invariant, and there exists a vector $\vec{v}$ and real r such that*

$$T(\vec{v}, \vec{v} - r) \geq \vec{v} + r, \tag{12}$$

then for any sequence $\vec{v}_0, \vec{v}_1, \ldots$ satisfying (11) *there exists a scalar constant c such that*

$$\forall n \geq 0 \quad \vec{v}_n \geq \vec{v} + nr - c. \tag{13}$$

PROOF. We use induction on n. Clearly, we can pick a c large enough so that (13) holds for $n \in \{0, 1\}$. For the inductive step, assume $n > 1$ and note that

$$\begin{aligned}
\vec{v}_n &\geq T(\vec{v}_{n-1}, \vec{v}_{n-2}) \\
&\geq T(\vec{v} + (n-1)r - c, \vec{v} + (n-2)r - c) \\
&\geq T(\vec{v}, \vec{v} - r) + (n-1)r - c \\
&\geq \vec{v} + r + (n-1)r - c = \vec{v} + nr - c.
\end{aligned}$$

□

Thus, if (12) holds for the T of (10), we can conclude from (3) and (13) that $\gamma \geq 2r$.

Compared to the finite state automaton analysis in work such as Dančík [1994] and Paterson and Dančík [1994], this approach produces the recurrence by brute force rather than trying to design a suitable finite state automaton. Also, it is a bit more flexible than using fixed finite automata, since we need not specify in advance which of the cases in (5) to use.

We can find a good choice for the $\vec{v}$ and r in (12) as follows. In practice, it appears that we can iterate

$$\vec{w}_n = T(\vec{w}_{n-1}, \vec{w}_{n-2}), \tag{14}$$

and have $\vec{w}_n - \vec{w}_{n-1}$ converge to a vector all of whose components are nearly equal; then $\vec{w}_n$ for some large n serves as a good choice of $\vec{v}$. (We do not prove this convergence, but this does not compromise the results of this section, since any $\vec{v}$ and r satisfying (12) serve as a witness that $\gamma \geq 2r$, regardless of how they were computed.) Thus, we need not solve a Markov chain for a steady-state distribution. (Of course, this approach could also be used for analyzing finite state automata.)

It is worth noting that the nature of the recurrences produced is such that computing successive values of $\vec{v}_n$ could be done conveniently while storing all of the

vectors on external memory. We did not use that approach in the computations reported here, however. Instead, as described in Section 2.3 below, we took advantage of a number of symmetries to enable the entire vector, with $h = 15$, to fit into the internal memory of a 1-Gigabyte machine.

Although here we only consider the case of alphabets of size 2 and longest common subsequences of 2 strings, [Kiwi and Soto 2008] shows how this lower bound method can be extended to finding lower bounds for longest common subsequences of more than 2 strings and alphabets of size greater than 2.

2.3. SOME IMPLEMENTATION DETAILS. In an effort to achieve greater efficiency, we actually carried out the computations a bit differently than described above, as described below.

2.3.1. *Some Identities.* Of course, as has been noticed before (see, e.g., Dančík [1994], Dančík and Paterson [1995], and Steele [1997]), one can take advantage of a few symmetries when dealing with such calculations. In particular,

$$v_n(s, t) = v_n(t, s), \tag{15}$$

and if an overbar represents complementation of all bits, we have

$$v_n(s, t) = v_n(\overline{s}, \overline{t}). \tag{16}$$

Also, if for some integer $m \in [1, h]$ the leftmost m bits of s and t match, then the value of $v_n(s, t)$ is independent of the values of these matching bits.

By using all three of these facts, we were able to reduce the number of components of the vector that needed to be stored by a factor of about six, from 2^{30} to 178973355 for $h = 15$.

2.3.2. *Simplifying the Recurrence.* Note that to iterate (14) in the naive way, we would simultaneously maintain values of $\vec{v}_i$, for three consecutive integers i, in memory. We kept only one vector in memory. One was written to external memory as it was computed, and another one was eliminated as follows. Suppose that instead of finding a $\vec{v}$ satisfying (12) we find a $\vec{w}$ satisfying

$$T(\vec{w}, \vec{w}) \geq \vec{w} + r'. \tag{17}$$

(Note that the second argument to T is $\vec{w}$ rather than $\vec{w} - r'$.) As before, in practice such a $\vec{w}$ can be found by iterating

$$\vec{w}_n = T(\vec{w}_{n-1}, \vec{w}_{n-1}). \tag{18}$$

This makes it possible to keep just one value of copy of $\vec{w}_i$ at a time in memory. By a simple induction $\vec{w}_n$ satisfies the same identities that were mentioned for $\vec{v}_n$ in Section 2.3.1, so we can still save a factor of about 6 when storing this vector. Of course, a $\vec{w}$ and r' satisfying (17) do not directly give us a $\vec{v}$ and r satisfying (12), but we can easily compute them in the following way.

Intuitively, using (18) instead of (14) is equivalent to charging for only one of the two new characters we draw after making a match. Equivalently, it can be viewed as giving a free extra character whenever there is a match. Thus, we expect that the true value of γ satisfies $r'n \leq (\gamma/2)(n + r'n)$. (The term $r'n$ on the right corresponds to the fact that the matches gave us $r'n$ free characters.) Thus, we expect that $\gamma \geq 2r'/(1 + r')$.

More formally, we make the following observation.

LEMMA 2.2. *Suppose that some vector $\vec{w}$ and real r' satisfy* (17). *Then, if we let*

$$\alpha = \frac{1}{1+r'}, \quad r = \alpha r', \quad \text{and} \quad \vec{v} = \alpha\vec{w}, \tag{19}$$

we have

$$T(\vec{v}, \vec{v} - r) \geq \vec{v} + r. \tag{20}$$

PROOF. Using (6) and (10), we see that (17) implies

$$T_0(\vec{w}) \geq \Pi_0(\vec{w}) + r' \quad \text{and} \quad T_1(\vec{w}) \geq \Pi_1(\vec{w}) + r'. \tag{21}$$

We now verify that this and (19) imply (20). Using (19), then (10), and finally (7), (8), and (9), we have

$$\begin{aligned} T(\vec{v}, \vec{v} - r) &= T(\alpha\vec{w}, \alpha\vec{w} - \alpha r') \\ &= (T_0(\alpha\vec{w}) \parallel T_1(\alpha\vec{w} - \alpha r')) \\ &= (\alpha T_0(\vec{w}) \parallel \alpha T_1(\vec{w}) + 1 - \alpha - \alpha r'). \end{aligned} \tag{22}$$

Applying (21) and simplifying using (19) we see that this is greater than or equal to

$$\begin{aligned} &(\alpha(\Pi_0(\vec{w}) + r') \parallel \alpha(\Pi_1(\vec{w}) + r') + 1 - \alpha - \alpha r') \\ &\quad = (\alpha\Pi_0(\vec{w}) + \alpha r' \parallel \alpha\Pi_1(\vec{w}) + 1 - \alpha) \\ &\quad = (\alpha\Pi_0(\vec{w}) + r \parallel \alpha\Pi_1(\vec{w}) + r) = \alpha\vec{w} + r = \vec{v} + r, \end{aligned} \tag{23}$$

so we have established the lemma. □

Using this lemma and Observation 2.1 above, we conclude

LEMMA 2.3. *If there exists a vector $\vec{w}$ and real r' such that*

$$T(\vec{w}, \vec{w}) \geq \vec{w} + r',$$

then

$$\gamma \geq \frac{2r'}{1+r'}.$$

2.4. THE VALUES OF THE LOWER BOUNDS. This approach to finding lower bounds on γ, and Observation 2.1 above, seem so simple and natural that one would suspect one or both have been observed before. However, if so, it seems they have not been fully exploited before, since carrying out this process for various values of h gives the results appearing in Table I. For the results reported in this figure, we used integers to implement fixed-point arithmetic and always rounded downwards during the computation of T when verifying (17). This approach makes it easy to control the effects of numerical error in the computation.

Note that the values in the table give an improvement over previous results when $h \geq 7$.

3. *Upper Bounds*

3.1. EARLIER WORK. We begin by discussing the results of Dančík [1994] and Dančík and Paterson [1995]. This discussion differs a bit from that of Dančík [1994] and Dančík and Paterson [1995], and uses a framework sufficiently general to cover our modification as well.

TABLE I. LOWER BOUNDS

h	Dimension of vector	Lower bound on γ
1	2	0.666666
2	5	0.727272
3	15	0.747922
4	51	0.758576
5	187	0.765446
6	715	0.770273
7	2795	0.773975
8	11051	0.776860
9	43947	0.779259
10	175275	0.781281
11	700075	0.783005
12	2798251	0.784515
13	11188907	0.785841
14	44747435	0.787017
15	178973355	0.788071

All values are for a binary alphabet. Parameter h controls the size of the computation.

Let a *string pair* be a pair of strings over Σ, say $a_1a_2\cdots a_i$ and $b_1b_2\cdots b_j$; we denote the string pair by $P = \binom{a_1a_2\cdots a_i}{b_1b_2\cdots b_j}$. Call $a_1a_2\cdots a_i$ the *top* string and $b_1b_2\cdots b_j$ the *bottom* string, and say P is of *size* $s(P) = i + j$. Let $lcs(P)$ be the length of a longest common subsequence of the top and bottom strings of P. Call the string pair $\binom{a_1a_2\cdots a_i}{b_1b_2\cdots b_j}$ a *match pair* if $a_i = b_j$. Say that this match pair *ends with an essential match* if any longest common subsequence must match a_i and b_j, that is, if both

$$lcs\binom{a_1a_2\cdots a_{i-1}}{b_1b_2\cdots b_j} < lcs\binom{a_1a_2\cdots a_i}{b_1b_2\cdots b_j}$$

and

$$lcs\binom{a_1a_2\cdots a_i}{b_1b_2\cdots b_{j-1}} < lcs\binom{a_1a_2\cdots a_i}{b_1b_2\cdots b_j}.$$

(This is essentially the same concept as the *minimal candidates* of Hirschberg [1977] and *dominant matches* of Apostolico and Guerra [1987].) A *minimal match pair* is a match pair $P = \binom{a_1a_2\cdots a_i}{b_1b_2\cdots b_j}$ that ends on an essential match and has $lcs(P) = 1$. We will call a_i and b_j, which must be equal, the *match character* of P, and call the other characters in P the *padding characters* of P. Call a string pair P a *null pair* if $lcs(P) = 0$, that is, if the top and bottom strings of P have no characters in common; the simplest null pair is $\binom{\epsilon}{\epsilon}$, where ϵ denotes the empty string.

Let $G(n, \ell)$ be the number of string pairs of size n that have an lcs of length at least ℓ. As pointed out in Dančík [1994] and Dančík and Paterson [1995], the following easily proven theorem has been the basis of a number of published bounds:

THEOREM 3.1 ([CHVÁTAL AND SANKOFF 1975, P. 310]). *If for some alphabet of size k, and for some $y \in (0, 1)$, we have*

$$G(2n, yn) = o(k^{2n})$$

as $n \to \infty$, then $\gamma_k \le y$.

Note that we can bound $G(n, \ell)$ by counting the number of ways we can concatenate ℓ minimal match pairs, followed by one more string pair, to obtain a string pair of size n. This bound can however be very weak; in particular, many different lists of match pairs may generate the same string pair. Much of the history of improvements in bounds on γ corresponds to improved methods for minimizing this overcounting; see Dančík [1994, Section 4.2] for more about this history. (We note that this is not the only approach presently known for computing upper bounds on expected longest common subsequence lengths; for example, see Baeza-Yates et al. [1999], which applies Kolmogorov complexity to obtain bounds.) We can abstract the process of minimizing the overcounting by defining a finite-state machine that restricts which lists of minimal match pairs will be counted. First, we give some definitions.

It will be convenient to consider lists of string pairs. Say that the *length* of such a list L is the number of elements (i.e., the number of string pairs) in the list, and that the *size* of L, written $s(L)$, is the total number of characters in all of the string pairs in L. Note that we do not use the terms size and length interchangeably; for example, the list of string pairs

$$\binom{230}{10}\binom{012}{2}\binom{13}{23}$$

has length 3 and size 13. We will use $\|$ to indicate concatenation of lists of string pairs; when an argument of $\|$ is a single string pair, we will promote it to the list with that one pair as its only element so we can for example write $L \parallel \binom{0}{0}$ to denote the list obtained by appending the pair $\binom{0}{0}$ to the list L. Following Dančík [1994] and Dančík and Paterson [1995] say that an arbitrary list $\binom{x_1}{y_1}\binom{x_2}{y_2}\cdots\binom{x_\ell}{y_\ell}$ of string pairs *generates* the string pair $\binom{x_1x_2\cdots x_\ell}{y_1y_2\cdots y_\ell}$, and write this as

$$cat\left(\binom{x_1}{y_1}\binom{x_2}{y_2}\cdots\binom{x_\ell}{y_\ell}\right) = \binom{x_1x_2\cdots x_\ell}{y_1y_2\cdots y_\ell}. \tag{24}$$

Note that the size of a list of string pairs is always the same as the size of the string pair it generates.

Now let $\mathcal{M}$ be a finite-state machine that takes as input lists of minimal match pairs; for each such list it accepts or rejects. This goes slightly outside the usual definition of finite-state machine since the input alphabet, namely, the set of minimal match pairs, has infinite cardinality, but this will cause no problems, and in practice it will be easy to restrict our attention to a finite subset of the minimal match pairs; see the comments following (36) below. We will always assume that once $\mathcal{M}$ reaches a rejecting state it will remain in that state; thus it can accept a list only if it accepts every prefix of that list. In view of this we can assume without loss of generality that there is only one rejecting state, which we denote by $\emptyset$. We will also assume that all states of $\mathcal{M}$ are reachable from the start state, since any states that were not reachable from the start state could simply be removed. Let $\mathcal{S}$ be the set of accepting states in $\mathcal{M}$; thus the entire set of states is $\mathcal{S} \cup \{\emptyset\}$.

If S is some set of objects, and each $x \in S$ has a nonnegative integer size $s(x)$, the *generating function* for S is the function $\mathcal{G}(S)$ that maps z to

$$\sum_{x \in S} z^{s(x)}.$$

Suppose S and S' are two sets of objects on which a nonnegative size function s is defined, and let $\times$ denote the Cartesian product; extend s to $S \times S'$ by letting $s(x, x') = s(x) + s(x')$. It is well known that

$$\mathcal{G}(S \times S') = \mathcal{G}(S) \cdot \mathcal{G}(S'); \tag{25}$$

we will use this fact below. Let $g_\ell(z)$ be the generating function for the set of string pairs P with $lcs(P) \geq \ell$, that is,

$$g_\ell(z) = \sum_{n=0}^{\infty} G(n, \ell) z^n. \tag{26}$$

Let $f_\ell(z)$ be the generating function for the set of lists of ℓ minimal match pairs that are accepted by $\mathcal{M}$.

Definition 3.2. Say that $\mathcal{M}$ *covers* Σ^* if for every string pair P with $lcs(P) = \ell$ there is a list L of ℓ minimal match pairs and a null pair p such that $\mathcal{M}$ accepts L, and $L \parallel p$ generates P.

THEOREM 3.3 ([DANČÍK 1994; DANČÍK AND PATERSON 1995]). *Suppose that $\mathcal{M}$ covers Σ^* and let $f_\ell(z)$ be the generating function for the set of lists of minimal match pairs of length ℓ accepted by $\mathcal{M}$. Suppose that for some real constants z and K, with*

$$z \in (0, k^{-1}) \quad \textit{and} \quad \lambda(z) \in (0, 1), \tag{27}$$

for all large enough integers ℓ we have

$$f_\ell(z) \leq K(\lambda(z))^\ell. \tag{28}$$

Then

$$\gamma_k \leq \frac{2\log(kz)}{\log \lambda(z)}. \tag{29}$$

PROOF. Let $n(z)$ be the generating function for the set of null pairs. One easily sees that $n(z)$ converges for $z \in (0, k^{-1})$ (since we can bound the number of null pairs of size n by the number of string pairs of size n, which is $(n+1)k^n$). Then by (25) and the fact that $\mathcal{M}$ covers Σ^*, we have

$$g_\ell(z) \leq \sum_{i=\ell}^{\infty} f_\ell(z) n(z). \tag{30}$$

Now pick any

$$y > \frac{2\log(kz)}{\log \lambda(z)}; \tag{31}$$

note that then

$$y > 0 \quad \text{and} \quad \lambda(z)^y < (kz)^2. \tag{32}$$

From (26), (28), and (30), for large ℓ we have

$$\begin{aligned} G(2n, \ell) &\leq \frac{g_\ell(z)}{z^{2n}} \\ &\leq \sum_{i=\ell}^{\infty} n(z) \frac{K(\lambda(z))^i}{z^{2n}} \\ &\leq \frac{K n(z)(\lambda(z))^\ell}{(1-\lambda(z))z^{2n}}. \end{aligned}$$

Now K, $n(z)$, and $\lambda(z)$ are all constants so using (32) it follows that

$$G(2n, yn) = O\left(\frac{(\lambda(z))^{yn}}{z^{2n}}\right) = o\left(\frac{(kz)^{2n}}{z^{2n}}\right) = o(k^{2n}).$$

Thus, by Theorem 3.1, any y satisfying (31) is an upper bound on γ_k, so the right-hand side of (31) is an upper bound on γ_k. □

One can bound the generating function $f_\ell(z)$ as follows. Let q_0 be the starting state of the machine $\mathcal{M}$, and let δ be its transition function; thus if $\mathcal{M}$ is in state q and reads minimal match pair p, it moves to state $q' = \delta(q, p)$. Let M be the set of all minimal match pairs. Let $f_\ell(z, q)$ be the generating function for the lists of ℓ minimal match pairs that are accepted by $\mathcal{M}$ assuming it starts in state q, so $f_\ell(z) = f_\ell(z, q_0)$. Then

$$f_0(z, q) = \begin{cases} 0 & \text{if } q = \emptyset \\ 1 & \text{if } q \in \mathcal{S} \end{cases} \tag{33}$$

and for $\ell > 0$ we have

$$f_\ell(z, q) = \sum_{p \in M} f_{\ell-1}(z, \delta(q, p)) z^{s(p)}. \tag{34}$$

Since M has infinite cardinality, this is not directly amenable to computation, but we can effectively make M finite by treating some sets of minimal match pairs as equivalent. For example, Dančík [1994] and Dančík and Paterson [1995] design a machine that treats all minimal match pairs of the form $\binom{0^+01}{1}$ in the same way. More generally, suppose there exists some finite subset M_0 of M and mapping μ from M to M_0 with the property that replacing p by $\mu(p)$ in any list L of minimal match pairs will never cause it to change from being accepted by $\mathcal{M}$ to being rejected by $\mathcal{M}$. Let $\hat{\mathcal{M}}$ be the machine whose transition function $\hat{\delta}$ is defined by

$$\hat{\delta}(q, p) = \delta(q, \mu(p)).$$

Then one easily sees that if $\mathcal{M}$ covers Σ^*, $\hat{\mathcal{M}}$ will also cover Σ^*. Let $\hat{f}_\ell(z, q)$ be the generating function for the lists of ℓ minimal match pairs that are accepted by $\hat{\mathcal{M}}$ assuming it starts in state q, so as before we have

$$\hat{f}_\ell(z, q) = \sum_{p \in M} \hat{f}_{\ell-1}(z, \hat{\delta}(q, p)) z^{s(p)}. \tag{35}$$

Define

$$w_{\mu,z}(p) = \sum_{p' \in \mu^{-1}(p)} z^{s(p')}, \tag{36}$$

that is, $w_{\mu,z}(p)$ is the generating function of the set of all minimal match pairs that map to p under μ. As a concrete example, if μ maps all minimal match pairs of the form $\binom{0^+01}{1}$ to $\binom{001}{1}$, as in Dančík [1994] and Dančík and Paterson [1995], we would have $\mu^{-1}\binom{001}{1} = \{\binom{001}{1}, \binom{0001}{1}, \binom{00001}{1}, \dots\}$ so $w_{\mu,z}\binom{001}{1} = z^4+z^5+z^6+\cdots = \frac{z^4}{1-z}$. Then, since, for any $p \in M$, $\hat{\mathcal{M}}$ treats all elements of $\mu^{-1}(p)$ identically, we have

$$\hat{f}_\ell(z, q) = \sum_{p \in M_0} \hat{f}_{\ell-1}(z, \delta(q, p)) w_{\mu,z}(p). \tag{37}$$

Thus, we need only consider a finite summation. Henceforth, we will assume that we are dealing only with machines like this that effectively have a finite alphabet, without putting hats on $\mathcal{M}$, δ, and f.

Now let $\vec{f}_\ell(z)$ denote the vector, indexed by the states of $\mathcal{M}$, in which the qth component is $f_\ell(z, q)$. Define

$$\mathbf{equal}(q, q') = \begin{cases} 1 & \text{if } q = q' \\ 0 & \text{otherwise.} \end{cases}$$

Then, the recurrence (37) can be expressed as

$$\vec{f}_\ell(z) = A_z \vec{f}_{\ell-1}(z), \tag{38}$$

where A_z is the matrix $\{a_{q,q'}\}$ in which

$$a_{q,q'} = \sum_{p \in M_0} w_{\mu,z}(p)\, \mathbf{equal}(q', \delta(q, p)).$$

Thus, as observed in Dančík [1994] and Dančík and Paterson [1995] finding the value $\lambda(z)$ to use in Theorem 3.3 is essentially the problem of finding the largest eigenvalue of A_z, where z can be selected to approximately minimize the value of the computed bound; they used Mathematica to assist with the calculations.

We conclude this section by briefly describing the machine used in Dančík [1994] and Dančík and Paterson [1995] to obtain upper bounds in the case $k = 2$. They define a *collation of order* ℓ to be a sequence of string pairs $p_1, p_2, \dots, p_\ell, p_{\ell+1}$ in which $p_1, p_2, \dots, p_\ell$ are match pairs; $p_{\ell+1}$ may be any string pair, possibly even the null pair. It is convenient to view a collation as a pair of strings in which we have indicated, by noncrossing lines, ℓ matches between characters of the upper and lower strings; see parts (a) and (b) of Figure 1. A powerful notion called *domination* in defined in Dančík [1994] and Dančík and Paterson [1995] for reducing the overcounting. Given a collation of order ℓ, define its *domination key* as follows: Let l_i be the total length of the first i match pairs in the collation; then the domination key is the tuple $(l_\ell, l_{\ell-1}, \dots, l_1)$. (In the terminology used in Dančík [1994] and Dančík and Paterson [1995] this would be the reverse of the *collation key*.) Equivalently, consider the string pair generated by the collation; let the indices of the matched characters in the top string be $i_1, i_2, \dots, i_\ell$ and the indices of the matched characters in the bottom string be $j_1, j_2, \dots, j_\ell$. Then, the domination key is $(i_\ell + j_\ell, i_{\ell-1} +$

(a) Two examples of possible matchings for the string pair $\binom{01010}{10100}$.

(b) The two collations corresponding to the matchings shown in part (a).

$$\binom{0}{10}\binom{1}{1}\binom{0}{0}\binom{10}{0}\binom{\epsilon}{\epsilon} \qquad \binom{01}{1}\binom{0}{0}\binom{1}{1}\binom{0}{0}\binom{\epsilon}{0}$$

(c) The domination keys corresponding to these two collations:

$$(10, 7, 5, 3) \qquad (9, 7, 5, 3)$$

FIG. 1. Collations and domination. Parts (a) and (b) illustrate the collations corresponding to two different matches in the same pair of strings. Part (c) gives the domination keys for these two collations. The second key is lexicographically smaller, so the second collation dominates the first.

$j_{\ell-1}, \ldots, i_1 + j_1$). One collation *dominates* another if it has the same order and a lexicographically smaller domination key. See part (c) of Figure 1.

In order to obtain their bounds, Dančík [1994] and Dančík and Paterson [1995] use a machine $\mathcal{M}$ that rejects only when it can determine that the match pairs read so far could not be the ending of an undominated collation. (Actually, their machine reads the list of minimal match pairs from right to left, instead of left to right as in this paper, but this does not affect our analysis.) Clearly, every string pair P has an undominated collation of order $\ell = lcs(P)$, and they note that each of the first ℓ string pairs in this collation is minimal. Thus, it follows that $\mathcal{M}$ covers Σ^*, so Theorem 3.3 can be used to give bounds on γ_k. For the case of a binary alphabet (i.e., $k = 2$), using a carefully constructed 52-state automaton they obtain a value of $\lambda(z_0) = 0.195960$ for $z_0 = 0.252652$, and then by Theorem 3.3 obtain the bound $\gamma_2 \leq 0.837623$. They comment

> "We have not automated the process of refinement and development of new recurrences, but further extensions could easily be found and the computation time required is not yet a limitation. ... It is an open question whether the method itself can give arbitrarily close approximations to γ." [Dančík and Paterson 1995, p. 456]

In the next section, we describe an enhancement of their method that can be shown to produce upper bounds arbitrarily close to γ_k for arbitrary k, though the computation time to guarantee better bounds increases rapidly.

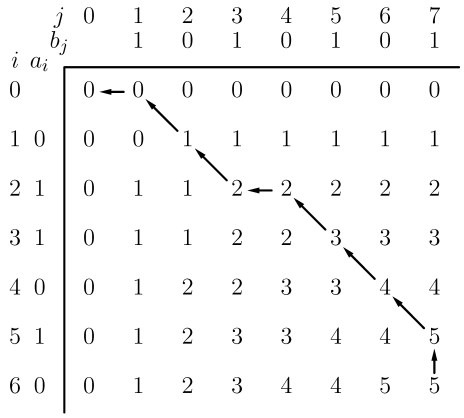
3.2. CANONICAL LONGEST COMMON SUBSEQUENCES. The modification used in this article is based on the well-known dynamic programming approach (see, e.g., Pevzner [2000, Section 6.2]) for computing longest common subsequences: Given a string pair $\binom{a_1a_2\cdots a_n}{b_1b_2\cdots b_m}$ we let $lcs[i, j]$ be the length of a longest common subsequence of $a_1a_2\cdots a_i$ and $b_1b_2\cdots b_j$ and note that for positive i and j we have the recurrence

$$lcs[i, j] = \max \begin{cases} lcs[i-1, j] \\ lcs[i, j-1] \\ lcs[i-1, j-1] + 1 \quad \text{(include only if } a_i = b_j\text{)}. \end{cases} \tag{39}$$

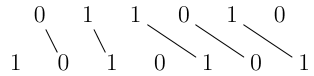
This computation is shown for the strings 011010 and 1010101 in part (a) of Figure 2.

It is well known that we can recover a longest common subsequence by backtracking through the table; when there is more than one longest common subsequence,

(a) The LCS dynamic programming table for $\binom{011010}{1010101}$ and the canonical backtracking.



(b) The canonical lcs is 01101, with the matching



(c) The canonical decomposition is

$$\binom{0}{10}\binom{1}{1}\binom{1}{01}\binom{0}{0}\binom{1}{1}\binom{0}{\epsilon}.$$

FIG. 2. Illustration of the well-known dynamic programming solution for the longest common subsequence problem, the definition of the canonical longest common subsequence, and the definition of a canonical decomposition. In part (a), the top two rows give the indices j and the characters b_j, and the left two columns give the indices i and the characters a_i. (When drawing dynamic programming tables for a string pair P, we always show the top string of P along the left edge of the table and the bottom string of P along the top edge.) The remaining entries give the lcs values computed during the dynamic programming solution; the entry indexed by i and j is the length of the lcs of $a_1a_2\cdots a_i$ and $b_1b_2\cdots b_j$. The unique canonical lcs for $a_1a_2\cdots a_6$ and $b_1b_2\cdots b_7$ is 01101, corresponding to the backtracking path shown in the diagram. Thus, the canonical decomposition for these two strings is as shown in part (c).

the order in which we consider the options as we backtrack determines the particular longest common subsequence we find. We define the *canonical* longest common subsequence to be the one we obtain by considering options in the order in which they appear in (39), that is,

(1) if $lcs[i, j] = lcs[i-1, j]$ move up from (i, j) to $(i-1, j)$,
(2) else if $lcs[i, j] = lcs[i, j-1]$ move left from (i, j) to $(i, j-1)$,
(3) else move diagonally up and to the left from (i, j) to $(i-1, j-1)$. In this case it must be that $lcs[i-1, j-1] = lcs[i-1, j] = lcs[i, j-1] = lcs[i, j]-1$, so $a_i = b_j$. We match these two characters.

Define the *canonical path* to be the sequence of entries in the lcs table followed by the canonical backtracking, and the *canonical decomposition* of a string pair to be the list of string pairs we obtain by cutting after each match in the canonical lcs. These definitions are illustrated in Figure 2.

Since the rules above completely determine the path along which we backtrack to find a longest common subsequence, one easily sees the following.

LEMMA 3.4. *Any pair of strings has exactly one canonical longest common subsequence and exactly one canonical decomposition.*

This notion of canonicity is very closely related to the notion of domination used in Dančík [1994] and Dančík and Paterson [1995]. Define the *canonicity key* of a matching as follows: Assuming that the longest common subsequence has length ℓ, let the indices of the matched characters in the top string be $i_1, i_2, \ldots, i_\ell$ and the indices of the matched characters in the bottom string be $j_1, j_2, \ldots, j_\ell$; then the canonicity key is $(i_\ell, j_\ell, i_{\ell-1}, j_{\ell-1}, \ldots, i_1, j_1)$. The canonical matching is the one that gives a longest common sequence and whose canonicity key is lexicographically smallest. For example, the first collation shown in Figure 1 has canonicity key (5, 5, 3, 4, 2, 3, 1, 2) and the second has canonicity key (5, 4, 4, 3, 3, 2, 2, 1). Thus, the first list of match pairs is not canonical.

Canonicity has a property similar to one shown in Dančík [1994] and Dančík and Paterson [1995] for undominated collations:

LEMMA 3.5. *If* $\binom{x}{y}$ *has* $lcs\binom{x}{y} = \ell$*, then the canonical decomposition of* $\binom{x}{y}$ *consists of* ℓ *or* $\ell + 1$ *string pairs, of which the first* ℓ *are minimal match pairs and the* $(\ell + 1)$*st, if present, is a null pair.*

PROOF. Since the longest common subsequence has length ℓ, and the canonical decomposition cuts the string after each match, the number of string pairs in the decomposition must be $\ell + 1$ or ℓ, depending on whether or not any characters follow the final match.

Next let $p = \binom{x'}{y'}$ be any of the first ℓ string pairs in the decomposition. Note that by construction the last character of x' must be the same as the last character of y', so p is a match pair. If p had $lcs(p) > 1$, we could find a common subsequence in $\binom{x}{y}$ longer than ℓ. If p had $lcs(p) = 1$ but did not end on an essential match, we could find a decomposition with a smaller canonicity key. Thus, p is a minimal match pair.

If the canonical decomposition consists of $\ell + 1$ match pairs, the last must be a null pair since otherwise the lcs of $\binom{x}{y}$ would be greater than ℓ. □

Definition 3.6. A list L of minimal match pairs is *canonical* if the canonical decomposition of $cat(L)$ is L itself.

Ideally, we would like to use a machine that accepts only canonical lists of minimal match pairs. To see that such a machine would cover Σ, let P be any pair of strings with a longest common subsequence of length ℓ. Form the canonical decomposition of P. Then, by Lemma 3.5, the list formed by the first ℓ pairs of P must be a list of minimal match pairs; by construction it is canonical. Also, the $(\ell+1)$st pair, if present, must be a null pair; if it is not present, we can simply append the pair $\binom{\epsilon}{\epsilon}$ to meet the conditions of Definition 3.2. Thus, this ideal machine would cover Σ^*.

Of course, for our calculations, we will want a machine that has a finite input alphabet, and only finitely many states, so we will not try to implement the ideal machine perfectly. Rather, we will have a *history* parameter h controlling how much the machine remembers in its finite state memory, and call the memory-limited machine $\mathcal{M}_h$.

The machine we construct will be allowed to accept match pair lists that are not canonical, but will never reject a canonical list of match pairs; thus it will still have the property that it covers Σ^* as required for Theorem 3.3. As in Section 3.1, we will make the set of match pairs we allow as input finite by using a function μ to map M to a finite subset M_0. For $q \in \mathcal{S}$, if the top or bottom string of p is longer than h, $\mu(p)$ removes characters on the left until only h remain. The contrapositive

of the following lemma implies that these deletions of characters will never change a canonical list of match pairs to one that is not canonical.

LEMMA 3.7. *If a given list $L = p_1, p_2, \ldots, p_\ell$ of minimal match pairs is not canonical, then no list of minimal match pairs that can be obtained by inserting additional padding characters before the matched characters of any of the p_i can be canonical.*

PROOF. Let $\binom{a_1 a_2 \cdots a_n}{b_1 b_2 \cdots b_m}$ be the string pair generated by L. For expressing and visualizing the proof without a lot of cumbersome notation, it is convenient to give an alternate way of viewing the definition of canonical. Assume that each character is at some coordinate on the real line. In particular, assume the ith character of $a_1 a_2 \cdots a_n$ is at x_i, and the jth character of $b_1 b_2 \cdots b_m$ is at y_j. Choose some assignment of coordinates such that

(1) For $1 \le i \le \ell$, the coordinate of both of the matched characters in p_i is i, and
(2) We have $x_1 < x_2 < \cdots < x_n$ and $y_1 < y_2 < \cdots < y_m$, so the ordering of the indices of the characters is the same as the ordering of the coordinates assigned to the characters.

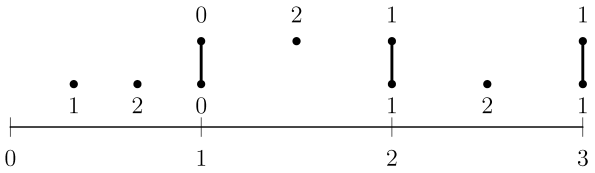
See Figure 3. Given such an assignment of coordinates to characters, and any particular matching, we define its *real canonicity key* as follows: Let the indices of the matched characters in the top string be $i_1, i_2, \ldots, i_\ell$ and the indices of the matched characters in the bottom string be $j_1, j_2, \ldots, j_\ell$; then the real canonicity key is $(x_\ell, y_\ell, x_{\ell-1}, y_{\ell-1}, \ldots, x_1, y_1)$. From (2) above, the lexicographic ordering of these real canonicity keys is the same as the lexicographic order of the original canonicity keys. Thus, in view of part (1) above, this means that L is canonical if and only if there is no match of length ℓ whose real canonicity key is lexicographically smaller than $(\ell, \ell, \ell-1, \ell-1, \ldots, 1, 1)$; see parts (a) and (b) of the figure. Now suppose that L is not canonical, so there is a match with a lexicographically smaller key K. Suppose we form a new list L' by inserting additional padding characters, without changing the real coordinates of any of the original characters, as illustrated in part (c) of the figure. Then, K is still the real canonicity key of a matching of length ℓ in L', and is of course still lexicographically smaller than $(\ell, \ell, \ell-1, \ell-1, \ldots, 1, 1)$, so L' is not canonical. □

In what follows, we first give some intuition for how the transition function δ of $\mathcal{M}_h$ is defined and computed, then give two concrete examples, and finally give pseudo-code for the general case. $\mathcal{M}_h$ will remember only the last h characters in the top and bottom of the string pair generated by the list of match pairs read so far, and the last $h+1$ entries in the rightmost column and in the bottom row of the lcs table. Thus, with $\Sigma = \{0, 1\}$ and $h = 3$, we might at some point remember the information shown in part (a) of Figure 4; assume we are in an accepting state. Now suppose we input the match pair $\binom{0}{110}$, to obtain part (b) of the figure. Unfortunately we do not have enough information to fill in the new entries in the lcs table. Instead, we will allow the entries we add to the table to be lower bounds on the correct value, with the constraint that if the input is canonical they must be tight along the canonical path. For those entries where some of the values needed in recurrence (39) are unknown (denoted by "?"), we compute a lower bound by using only the cases in (39) for which the correct value is known. This leads us to part (c) of the figure. Now we have lower bounds on all values needed to use (39)

(a) Let

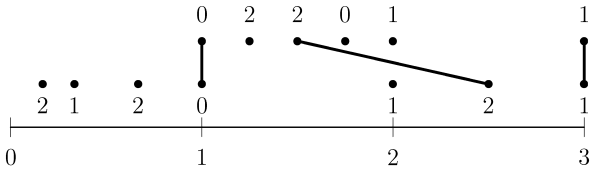
$$L = \binom{0}{120}\binom{21}{1}\binom{1}{21},$$

and assign horizontal coordinates to the characters as shown below.



L will be canonical if the real canonicity key $(3, 3, 2, 2, 1, 1)$ of the indicated match is lexicographically the smallest possible.

(b) There exists a matching with the lexicographically smaller key $K = (3, 3, 1.5, 2.5, 1, 1)$, so L is not canonical.



(c) We have now inserted some additional padding characters to the match pairs, obtaining

$$L' = \binom{0}{2120}\binom{2201}{1}\binom{1}{21}.$$

The real canonicity key of the matching shown below is still $(3, 3, 1.5, 2.5, 1, 1)$, showing that L' is not canonical.

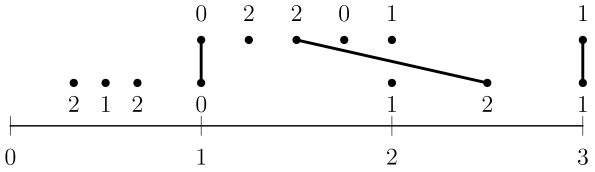


FIG. 3. Illustration for the proof of Lemma 3.7.

to fill in lower bounds for the rest of the table and obtain part (d). We then check whether the canonical backtracking starting at the bottom right entry in this figure leads us to the bottom right entry of the original state shown in part (a) of the figure. If not, we reject. Note that if the input is canonical the lcs will increase by exactly one when we add the new match pair $\binom{0}{110}$, so we are guaranteed that the lower bounds appearing along the canonical path in the table will be tight. Thus the canonical backtracking will lead us to reject only if the input is not canonical.

Unfortunately, this would still not lead to a finite set of states since the values in the lcs table increase without bound as we read more match pairs. However, it is clear that the acceptance test will not be affected if we translate all values in the table by the same amount, so we only need to remember the *differences* among the last $h + 1$ entries in the bottom row and in the rightmost column of the table. These differences are shown in a smaller font in part (d) of Figure 4. Thus, we let the accepting states of the machine be of the form (x, y, dx, dy), where x (respectively, y) records the last h characters of the top (respectively, bottom) input string, and dx (respectively, dy) records the last h differences in the last column (respectively, row) of the lcs table. Note that x and y are strings in Σ^h and dx

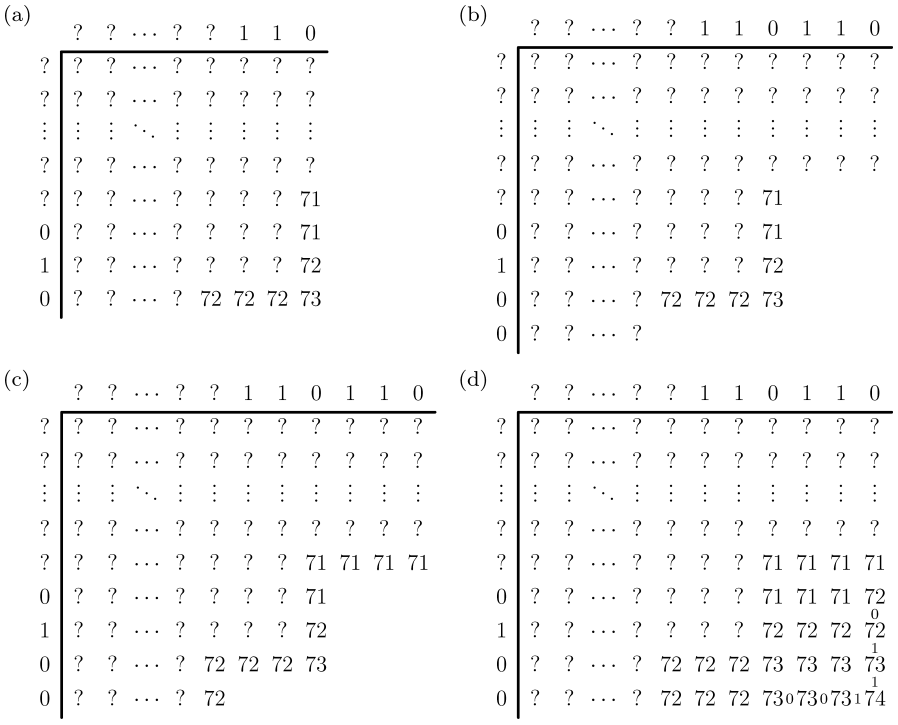


FIG. 4. Motivation for the transition function of $\mathcal{M}_h$. Here the string pair being read is $\binom{0}{110}$.

and dy are bit strings in $\{0, 1\}^h$. In particular, the state corresponding to part (d) of Figure 4 would be (100, 110, 011, 001). There is also one rejecting state $\emptyset$.

Figures 5 and 6 give two complete examples of the computation of δ. (If x is a binary string, let $\bar{x}$ be the string obtained by complementing all characters in x. Much as in Dančík [1994] and Dančík and Paterson [1995, p. 451], we took advantage of the fact that $f_\ell(z, (x, y, dx, dy)) = f_\ell(z, (\bar{x}, \bar{y}, dx, dy))$; we could not however conclude that $f_\ell(z, (x, y, dx, dy)) = f_\ell(z, (y, x, dy, dx))$, since the rules for canonical backtracking break this symmetry.)

We can now precisely state how to compute $\delta(q, p)$ in general. If q is the rejecting state $\emptyset$, we simply let $\delta(q, p) = \emptyset$. Otherwise, we let $\delta(q, p)$ be the result of calling the procedure delta shown in Figure 7. (We use a syntax loosely based on that of the Python programming language; however, we assume that strings are indexed beginning at 1 rather than at 0, for consistency with the rest of this article.)

Lines 7–13 of this procedure fill in the new portion of the lcs table using (39); whenever the machine cannot reconstruct the value stored in a needed lcs entry from the state, it uses a lower bound corresponding to one of the first two cases in (39).

At first, it might seem that it would be necessary to have some transient states in which there are fewer than h characters or differences, but a simple observation allows us to avoid this.

LEMMA 3.8. *Let $\binom{x_1}{y_1}\binom{x_2}{y_2}\cdots\binom{x_\ell}{y_\ell}$ be an arbitrary list of minimal match pairs. Then for any $c \in \Sigma$, the list $\binom{x_1}{y_1}\binom{x_2}{y_2}\cdots\binom{x_\ell}{y_\ell}$ is canonical if and only if the list $\binom{c}{c}\binom{x_1}{y_1}\binom{x_2}{y_2}\cdots\binom{x_\ell}{y_\ell}$ is canonical.*

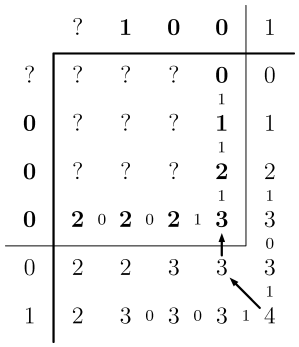


FIG. 5. Illustration of $\delta\big(\langle 000, 100, 111, 001\rangle, \binom{01}{1}\big) = \langle 001, 001, 101, 001\rangle$. The initial state $\langle 000, 100, 111, 001\rangle$ is shown by the bold numbers in the table: The last three characters of the first and second string are 000 and 100, the differences of the column of bold entries in the lcs table are 111, and the differences of the row of bold entries in the lcs table are 001. (Values left unspecified by the initial state are shown as "?".) After appending 01 to the first string and 1 to the second, and filling in the lcs table, we obtain the table shown. Now the last three characters of the first and second string are 001 and 001, the last three differences in the right column of the lcs table are 101, and last three differences in the bottom row of the lcs table are 001; thus the new state is $\langle 001, 001, 101, 001\rangle$.

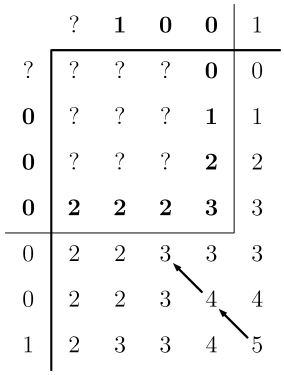


FIG. 6. Illustration of $\delta\big(\langle 000, 100, 111, 001\rangle, \binom{001}{1}\big) = \emptyset$. The initial state is the same as in the previous figure. Appending $\binom{001}{1}$ yields the table shown. This time, however, when we canonically backtrack through the table we do not reach the bottom right entry in the lcs table corresponding to the original state, so the resulting state is $\emptyset$.

PROOF SKETCH. One easily sees that if two strings begin with the same character, then the lexicographically smallest matching giving a longest common subsequence must pair the first two characters, from which the lemma follows easily. □

By Lemma 3.8, we see that reading in h match pairs $\binom{0}{0}$ before reading the actual input will not have any effect on which lists of match pairs are accepted, so we can take our initial state from the lcs table for $\binom{0^h}{0^h}$, and thus let the start state be $q_0 = (x, y, dx, dy)$, where x and y are the string 0^h and dx and dy are the bit string 1^h.

For a given z, the corresponding value of $\lambda(z)$ to use in Theorem 3.3 was found as follows. Iterate the recurrence (37) until the ratio

$$f_\ell(z, q)/f_{\ell-1}(z, q) \tag{40}$$

```
1. def delta(q, p):
2.   (x, y, dx, dy) = q
3.   lcs[h, h] = h
4.   decodeVertDiffs((h, h), dx); decodeHorzDiffs((h, h), dy)
5.   (a, b) = μ(p)
6.   a = x + a; b = y + b
7.   for i = 0 to len(a):      # fill in the new lcs table entries
8.     for j = 0 to len(b):
9.       if i > h or j > h:
10.        if i == 0: lcs[i, j] = lcs[i, j − 1]
11.        else if j == 0: lcs[i, j] = lcs[i − 1, j]
12.        else if a_i == b_j: lcs[i, j] = lcs[i − 1, j − 1] + 1
13.        else: lcs[i, j] = max(lcs[i, j − 1], lcs[i − 1, j])
14.  (i, j) = (len(a), len(b))      # now perform the canonicity test
15.  while i ≥ h and j ≥ h:
16.    if lcs[i, j] == lcs[i − 1, j]: i = i − 1          # try moving up
17.    else if lcs[i, j] == lcs[i, j − 1]: j = j − 1     # try moving left
18.    else: i = i − 1; j = j − 1                        # make a match
19.    if i == h and j == h: # canonicity test succeeds
20.      last = (len(a), len(b))
21.      return (suffix(a), suffix(b), encodeVertDiffs(last), encodeHorzDiffs(last))
22.  return ∅   # canonicity test fails
```

The following functions are used:

—$\mu(p)$ removes characters from the left of the upper and lower strings of p as needed to restrict each to length h.
—len(a) returns the length of the string a.
—suffix(a) returns the string consisting of the last h characters of a.
—encodeVertDiffs((i, j)) returns a bit vector of length h whose successive entries are the successive differences between the table entries $lcs[i-h, j]$, $lcs[i-h+1, j]$, $lcs[i-h+2, j]$, ... , $lcs[i, j]$.
—encodeHorzDiffs((i, j)) performs the operation analogous to encodeVertDiffs, but for horizontal differences. It returns a bit vector of length h whose successive entries are the successive differences between the table entries $lcs[i, j-h]$, $lcs[i, j-h+1]$, $lcs[i, j-h+2]$, ... , $lcs[i, j]$.
—decodeVertDiffs($(i, j), dx$) performs an operation inverse to encodeVertDiffs; it makes the successive differences of the table entries $lcs[i-h, j]$, $lcs[i-h+1, j]$, $lcs[i-h+2, j]$, ... , $lcs[i, j]$ be the successive bits in the bit string dx, leaving $lcs[i, j]$ unchanged.
—decodeHorzDiffs($(i, j), dy$) performs the operation analogous to decodeVertDiffs, but for horizontal differences.

The operator $+$, when applied to strings, denotes string concatenation.

FIG. 7. The transition function $\delta(q, p)$ for the machine $\mathcal{M}_h$.

is nearly independent of q. (In Section 3.3, we will show that in fact all of these ratios approach the same value for the finite state machines we use.) Let the largest of these ratios be λ. A simple induction shows that then (28) will hold for some K if we let $\lambda(z) = \lambda$; in fact, in the notation of (38), since all entries in A_z are nonnegative, simply exhibiting any particular $z \in (0, 1/K)$, $\lambda \in (0, 1)$, and positive vector $\vec{v}$ such that

$$A_z \vec{v} \leq \lambda \vec{v}, \tag{41}$$

where the inequalities are to be interpreted component-wise, is sufficient to support the claim that (29) holds with $\lambda(z) = \lambda$.

3.2.1. *The Values of the Upper Bounds for $k = 2$.* The results of this calculation are shown in Table II. The values of z shown are chosen to yield good bounds, and the values of $\lambda(z)$ shown are numerical approximations. The next section outlines the argument that the bounds shown for γ are truly upper bounds, despite

TABLE II. UPPER BOUNDS

h	Number of accepting states	z	$\lambda(z)$	Upper bound on γ
2	7	0.2118	0.131204211	0.845855
3	98	0.2422	0.177752610	0.839250
4	1020	0.2642	0.217017510	0.835072
5	10160	0.2828	0.254100606	0.831910
6	89960	0.2980	0.287155609	0.829530
7	780368	0.3104	0.316022744	0.827727
8	6419109	0.3209	0.341832259	0.826280

All values are for a binary alphabet. Parameter h controls the size of the computation.

the possibility of numerical error in the computation, provided we assume that the basic floating point operations perform as expected. Note that for $h = 4$ the method already improves the previously known upper bound, and that for $h = 7$ or 8 the bound is less than the γ given by Steele's old conjecture (see [Steele 1997, page 3]) that $\gamma = 2/(1 + \sqrt{2}) \approx 0.828427$. (As Steele [1997] pointed out, experimental evidence already suggested that this conjectured value was too high.) Put another way, this shows that the Arratia-Steele conjecture (see Pevzner [2000, Section 6.8, page 107]) fails even for alphabet size 2.

3.2.2. *Numerical Issues.* The actual computations leading to the upper bound on γ involve an iterative process and millions of floating-point operations. Thus it is appropriate to ask the extent to which rounding error casts doubt on the final results. In this section, we sketch the argument that shows that the exact bound of $\gamma \leq 826280/1000000$ is justified, despite the possibility of numerical errors in our intermediate calculations, assuming that floating point arithmetic satisfies reasonable assumptions.

We will bound the error in the computation of the expression $2\log(2z)/\log\lambda(z)$ appearing in Theorem 3.3 in two steps: First, we estimate the accuracy of our computation of $\lambda(z)$, and then we estimate the accuracy of the bound computed using this $\lambda(z)$.

As remarked earlier, any $z \in (0, 1/k)$, $\lambda \in (0, 1)$, and positive vector $\vec{v}$ satisfying (41) serve as a witness that we can apply Theorem 3.3 with $\lambda(z) = \lambda$, so we merely need to check that these inequalities hold. For the largest problem we consider there are $N = 6419109$ accepting states, so methods involving the computation of high powers of the $N \times N$ array A_z could place excessive demands on memory. Instead we start with the N-dimensional vector $(1, 1, 1, \ldots, 1)$ (see (33)) and iteratively multiply by A_z until we find that all components of the vector are increasing by approximately the same ratio between two successive iterations. Then, this serves as the vector $\vec{v}$ to use in (41).

Verification that (41) holds is simplified by the fact that, aside from the computation of $1-z$ during the calculation of $w_{\mu,z}(p)$, all expressions involve only addition, multiplication, and division of nonnegative numbers. Because of this, we can use rather crude bounds to justify the conclusion that $\gamma \leq 826280/1000000$. We take the numerical values of z and $\vec{v}$ as computed by the program to be the exact values for the witness in (41), so they have no roundoff error. We then let $\lambda(z)$ be the maximum factor by which any component of $A_z\vec{v}$ exceeds the corresponding component of $\vec{v}$. Because of roundoff error, the program computes only an approximation λ to $\lambda(z)$.

When we performed the actual calculations for this article, we used a Java program with double precision numbers. Assuming that the Java implementation

used conforms to the specification given in Gosling et al. [2005, Sections 4.2.3 and 4.2.4], we know that the value set used for a `double` includes all possible numeric values for IEEE 754 doubles, and that round-to-nearest mode is used for floating point addition, subtraction, multiplication, and division. Thus, as long as the true result r of any of these operations lies within the normalized range of double precision, if we assume exact operands we can assume, using Overton [2001, Table 4.4 and Theorem 5.1], that the computed result $\hat{r}$ satisfies $\hat{r} = r(1+\delta)$ for some $\delta \in [-2^{-53}, 2^{-53}]$, from which it follows that $\hat{r}/r \in (1/(1+2^{-52}), 1+2^{-52})$. For the remainder of this section we let

$$\rho = 1 + 2^{-52}, \tag{42}$$

and assume that we are dealing with numbers that lie within the normalized range of double precision.

Suppose that double precision numbers $\hat{x}$ and $\hat{y}$ represent positive real numbers x and y with the error bounds

$$\rho^{-i}x \le \hat{x} - x \le \rho^{i}x$$

and

$$\rho^{-j}y \le \hat{y} - y \le \rho^{j}y.$$

Then, if we let $s = x + y$, $p = xy$, and $q = x/y$, and let $\hat{s}$, $\hat{p}$, and $\hat{q}$ represent the results of the corresponding double precision operations on $\hat{x}$ and $\hat{y}$ with round-to-nearest, we can conclude that

$$\rho^{-(\max(i,j)+1)}s \le \hat{s} - s \le \rho^{\max(i,j)+1}s, \tag{43}$$

$$\rho^{-(i+j+1)}p \le \hat{p} - p \le \rho^{i+j+1}p, \tag{44}$$

and

$$\rho^{-(i+j+1)}q \le \hat{q} - q \le \rho^{i+j+1}q. \tag{45}$$

Now the approximation to $1 - z$ used in the program is accurate to within a factor of ρ, and then inspection of the program[2] and repeated use of (43), (44), and (45) establishes that the computed value λ satisfies

$$\lambda(z)\rho^{-100} \le \lambda - \lambda(z) \le \lambda(z)\rho^{100}. \tag{46}$$

Finally, we need to show that

$$\frac{2\log(2z)}{\log\lambda(z)} \le p/q, \tag{47}$$

where $p = 826280$ and $q = 1000000$. To show this, we establish that

$$\frac{\log(2z)}{\log\lambda(z)} \le p/(2q),$$

[2] Code to produce the bounds reported in this paper has been placed on the web at `http://www.ics.uci.edu/~lueker/papers/lcs/code`

which, since $\lambda(z) < 1$, is equivalent to

$$(2z)^{2q} \geq (\lambda(z))^p.$$

Let

$$R = \frac{(2z)^{2q}}{(\lambda(z))^p}, \tag{48}$$

so we need to show $R \geq 1$. We calculate a numerical approximation $\hat{R}$ to R, based on z and λ, by a routine involving repeated multiplication and division, being careful to interleave the multiplications and divisions so that we stay well within the normalized range of double precision. Again using (43), (44), and (45), starting with (46), we can compute a value e_R such that

$$R \geq \hat{R}\rho^{-e_R},$$

where it can be verified that

$$e_R \leq 2^{27}. \tag{49}$$

Then to establish that $R \geq 1$ and hence that (47) holds, it is sufficient to verify that

$$\hat{R} \geq \rho^{e_R}.$$

We can bound the right hand side of this by

$$\rho^{e_R} \leq e^{e_R \log \rho} \leq e^{e_R/2^{52}} \leq 1 + e_R/2^{51} \leq 1 + 2^{-24},$$

where we have used (49) and the fact that for $x \leq 1/2$ we have $e^x \leq 1 + 2x$. Thus it suffices to show that

$$\hat{R} - 1 \geq 2^{-24},$$

which the program verifies. (In fact, for the values used in the program it turns out that $\hat{R} - 1 \approx 1.4$, so double precision provides far more accuracy than needed.)

3.3. CONVERGENCE. We now investigate some convergence properties of the methods described in this paper. First, we show a sufficient condition for the ratios in (40) to approach a limit independent of q. Then, we show a sufficient condition for the bounds obtained from Theorem 3.3 to converge to the true value of γ_k as the history parameter h becomes large. Finally, we show that the method of Section 3.2 meets both of these conditions.

Definition 3.9. $\mathcal{M}$ is *regular*[3] if it has the following two properties:

(1) for every $q \in \mathcal{S}$ the start state is reachable from q, and
(2) there exists some state $q \in \mathcal{S}$ and some minimal match pair $p \in M$, such that if $\mathcal{M}$ is in state q and reads p, it remains in state q.

Now standard arguments show that if $\mathcal{M}$ is regular we have a convergence property.

[3]This is closely related to the definition of regular used for computing lower bounds in Dančík [1994, Definition 3.5].

THEOREM 3.10. *If $\mathcal{M}$ is regular, then for each constant $z \in (0, k^{-1})$ there exists $\lambda > 0$, $0 \leq \rho < 1$, and a vector $\vec{v}$ (indexed by states in $\mathcal{S}$), with all components positive, such that*

$$f_\ell(z, q) = \lambda^\ell v(q) + O(\rho^\ell \lambda^\ell). \tag{50}$$

PROOF. Since $\mathcal{M}$ is regular the matrix A is irreducible and aperiodic; thus, by Seneta [1981, Theorem 1.4] it is primitive,[4] so by the Perron-Frobenius Theorem for primitive matrices [Seneta 1981, Theorem 1.1] (see also MacCluer [2000]), it has a positive real eigenvalue λ of (algebraic and geometric) multiplicity one with a corresponding eigenvector in which all components are positive, and all other eigenvalues are smaller in magnitude than λ. The conclusion follows immediately. (See also Seneta [1981, Theorem 1.2].) □

COROLLARY 3.11. *If $\mathcal{M}$ is regular, then for each positive z there exists a λ such that for all $q \in \mathcal{S}$,*

$$\lim_{\ell \to \infty} \frac{f_\ell(z, q)}{f_{\ell-1}(z, q)} = \lambda.$$

Next we consider whether the upper bounds on γ_k converge to the true value as h becomes large. We note that it is not surprising that some sequence b_h of upper bounds on γ_k, where the time to compute b_h increases with h, can be shown to converge to γ_k. Alexander [1994] showed how to compute explicit values of n_0 and C such that

$$n \geq n_0 \implies \gamma_k n \geq \mathsf{E}[L_{nn}] \geq \gamma_k n - C\sqrt{n \log n}.$$

Hence, for large n

$$\gamma_k \leq \mathsf{E}[L_{nn}]/n + C\sqrt{n^{-1} \log n}. \tag{51}$$

In Alexander [1994] results from simulations with random numbers were used to estimate the expected lcs length and give bounds that held with 95% confidence, but in principle one could simply evaluate $\mathsf{E}[L_{nn}]$ for successive values of n by exhaustive enumeration to obtain from (51) a sequence of bounds guaranteed to converge to γ_k. Although this would give an error bound that converges more rapidly than the bound proven below in Theorem 3.13, it is of interest to investigate the convergence of the methods that have actually been used to produce the best bounds.

We now describe a sufficient condition for a sequence $\mathcal{M}_h$, $h = 1, 2, \ldots$, of machines to produce bounds that approach the true value of γ_k.

Definition 3.12. Say a sequence $\mathcal{M}_h$, $h = 1, 2, \ldots$, of machines *efficiently covers* Σ^* if it covers Σ^* and additionally has the following two properties:

(1) For any two lists L and L' of minimal match pairs, if $\mathcal{M}$ accepts $L \parallel L'$ then it also accepts L and L'.
(2) There exists no string pair P of size bounded by h such that two distinct lists L and L' of minimal match pairs, each of which is accepted by $\mathcal{M}_h$, both generate P.

[4] A square matrix A with no negative entries is *primitive* if for some positive integer ℓ all entries in A^ℓ are positive [Seneta 1981, p. 3].

The next section proves the following.

THEOREM 3.13. *Suppose that the sequence $\mathcal{M}_h$, $h = 1, 2, \ldots$, efficiently covers Σ^*, and that each machine in the sequence is regular. Then as $h \to \infty$ the sequence of upper bounds on γ_k produced by $\mathcal{M}_h$ approaches γ_k. In particular, the upper bound produced by $\mathcal{M}_h$ is bounded by*

$$\gamma_k + O\left(\left(\frac{\log h}{h}\right)^{1/3}\right),$$

where the hidden constants are allowed to depend on k.

3.3.1. *Proof of Theorem* 3.13. Throughout this section, we assume that the conditions of Theorem 3.13 hold and that we are dealing with a fixed alphabet $\Sigma = \{0, 1, \ldots, k-1\}$, where $k \geq 2$. Dependence on k will not always be made explicit; in particular we will allow hidden constants in asymptotic notation to depend on k, and will simply use γ instead of γ_k.

Define $F(n, \ell, h)$ to be the number of lists, with length ℓ and size n, of minimal match pairs that are accepted by $\mathcal{M}_h$, and let $f_{\ell,h}(z)$ denote the corresponding generating function, that is,

$$f_{\ell,h}(z) = \sum_{n=0}^{\infty} F(n, \ell, h) z^n. \tag{52}$$

Since this is the most complicated proof in the article, we begin by giving a very rough sketch of the proof. To get arbitrarily close bounds from Theorem 3.3 we will need to show that

$$f_{\ell,h}(z) \leq K(\lambda(z))^{\ell}$$

for some $\lambda(z)$ close to $(kz)^{2/\gamma}$. Although we are interested in the behavior of $f_{\ell,h}$ when $\ell \gg h$, we first analyze the behavior for h substantially larger than ℓ, and then show how to use this to obtain bounds for $\ell \gg h$; a similar approach was used in Kiwi et al. [2005]. For n of size up to roughly $2\ell/\gamma$, we use part (2) of Definition 3.12 to bound $F(n, \ell, h)$ by the number $G(n, \ell)$ of string pairs of size n with an lcs length of at least ℓ. Since this is roughly comparable to k^n when n is near $2\ell/\gamma$, the contribution to the sum on the right of (52) when n is near $2\ell/\gamma$ is very roughly $(kz)^{2\ell/\gamma}$, as desired. For n significantly smaller than $2\ell/\gamma$, we use a concentration inequality to show that the contribution to the sum is small, since it corresponds to strings having an lcs much longer than expected. Finally, for n significantly larger than ℓ, we use the fact that $F(n, \ell, h)$ is substantially lower than k^n (since many characters can be chosen from only $k-1$ rather than k choices), to show that the contribution to the sum is again small. We then use a subadditivity result that follows from part (1) of Definition 3.12 to extend the bound to the range $\ell \gg h$.

We now give the detailed proof. It is well known [Chvátal and Sankoff 1975] that by superadditivity we have $\mathsf{E}[L_{nn}] \leq \gamma n$, from which it follows readily that

$$\mathsf{E}[L_{ij}] \leq \frac{1}{2}\gamma(i + j). \tag{53}$$

We have

$$\Pr\{|L_{ij} - \mathsf{E}[L_{ij}]| \geq t\} \leq 2e^{-t^2/(2(i+j))} \tag{54}$$

by a standard application of Azuma's inequality and the method of bounded differences; see Alexander [1994] and Steele [1997, Section 1.3] for proofs of deviation bounds for the lcs by this method. (See Janson et al. [2000, Section 2.4] for a discussion of martingales and Azuma's inequality; one could prove a somewhat stronger bound than (54) using the bounds they present (e.g., Remark 2.28), but that is not necessary for our main results.) Recall that $G(n, \ell)$ is the number of string pairs of size n having an lcs length of at least ℓ. Considering the various possibilities for the lengths of the top and bottom strings, and using (53) and (54), we have

$$\begin{aligned} G(n,\ell) &\le \sum_{i=0}^{n} k^n \Pr\{L_{i,n-i} \ge \ell\} \\ &= \sum_{i=0}^{n} k^n \Pr\{L_{i,n-i} - \mathsf{E}[L_{i,n-i}] \ge \ell - \mathsf{E}[L_{i,n-i}]\} \\ &\le \sum_{i=0}^{n} k^n \Pr\{L_{i,n-i} - \mathsf{E}[L_{i,n-i}] \ge \ell - \gamma n/2\} \\ &\le 2(n+1)k^n e^{-(\ell-\gamma n/2)^2/(2n)}. \end{aligned} \tag{55}$$

LEMMA 3.14. *There exists an integer constant $c \ge 2$, depending only on k, such that the following holds. Choose any positive integer ℓ and real x so that*

$$x \in [0, \ell/\gamma], \tag{56}$$

and

$$n_0 = 2\ell/\gamma - x \tag{57}$$

is a positive integer. Then for any real z with

$$z \in (0, k^{-1}) \tag{58}$$

and integer h with

$$h \ge c\,\ell \tag{59}$$

we have

$$f_{\ell,h}(z) = \sum_{n=0}^{\infty} F(n,\ell,h) z^n \le 2n_0^2 e^{-\gamma^3 x^2/(16\ell)} + \frac{n_0+1}{(1-kz)^2}(kz)^{n_0} + 4k/2^{\ell}. \tag{60}$$

PROOF. From (56) and (57), we have

$$n_0 \ge \ell/\gamma. \tag{61}$$

Note that any list of ℓ minimal match pairs maps (under *cat*) to a string pair with lcs length at least ℓ; moreover, by (59) and part (2) of Definition 3.12 we know that this map is one-to-one if we restrict the domain to minimal match pair lists of size at most $c\,\ell$ that are accepted by $\mathcal{M}_h$. Thus, we have

$$n \le c\,\ell \implies F(n,\ell,h) \le G(n,\ell). \tag{62}$$

We estimate the summation in (60) by dividing the range of summation into three intervals, namely, $0 \le n < n_0$, $n_0 \le n < c\,\ell$ and $c\,\ell \le n$. (Note that (57) guarantees that if we pick c large enough we will have $n_0 \le c\,\ell$.)

Now using (62) and then (55) gives

$$\begin{aligned}\sum_{n=0}^{n_0-1} F(n,\ell,h)z^n &\le \sum_{n=0}^{n_0-1} G(n,\ell)z^n \\ &\le \sum_{n=0}^{n_0-1} 2(n+1)\,k^n\,e^{-(\ell-\gamma n/2)^2/(2n)}z^n. \end{aligned} \tag{63}$$

When $n < n_0 = 2\ell/\gamma - x$ we have $\gamma n/2 < \ell - \gamma x/2$ so

$$\frac{(\ell-\gamma n/2)^2}{2n} \ge \frac{(\gamma x/2)^2}{2\cdot 2\ell/\gamma} = \frac{\gamma^3 x^2}{16\ell}, \tag{64}$$

so using (63) and then (58) we have

$$\begin{aligned}\sum_{n=0}^{n_0-1} F(n,\ell,h)z^n &\le 2e^{-\gamma^3x^2/(16\ell)} \sum_{n=0}^{n_0-1}(n+1)(kz)^n \\ &\le 2e^{-\gamma^3x^2/(16\ell)} \sum_{n=0}^{n_0-1}(n+1) \\ &= n_0(n_0+1)e^{-\gamma^3x^2/(16\ell)} \\ &\le 2n_0^2e^{-\gamma^3x^2/(16\ell)}, \end{aligned} \tag{65}$$

where in the last step we used the fact that n_0 is a positive integer.

For the range $n_0 \le n < c\,\ell$ we note that, from (62), $F(n,\ell,h)$ is certainly bounded by the number of string pairs of size n, i.e., $(n+1)k^n$, so

$$\begin{aligned}\sum_{n=n_0}^{c\,\ell-1} F(n,\ell,h)z^n &\le \sum_{n=n_0}^{c\,\ell-1}(n+1)(kz)^n \\ &\le \sum_{n=n_0}^{\infty}(n+1)(kz)^n \\ &= \frac{n_0+1-kzn_0}{(1-kz)^2}(kz)^{n_0} \\ &\le \frac{n_0+1}{(1-kz)^2}(kz)^{n_0}. \end{aligned} \tag{66}$$

Finally, for the range $n \ge c\,\ell$ we cannot use the bound in (62), so we will bound $F(n,\ell,h)$ by the total number of sequences of ℓ minimal match pairs of size n; this number is bounded by $\binom{n}{2\ell}$ (for choosing which of the n characters to match) times k^ℓ (for choosing the values of the matched characters)

times $(k-1)^{n-2\ell}$ (for choosing the values of the padding characters), to obtain

$$\begin{aligned}\sum_{n=c\,\ell}^{\infty} F(n,\ell,h)z^n &\le \sum_{n=c\,\ell}^{\infty} \binom{n}{2\ell} k^{\ell}(k-1)^{n-2\ell}z^n \\ &\le \sum_{n=c\,\ell}^{\infty} \left(\frac{ne}{2\ell}\right)^{2\ell}(k-1)^{n-2\ell}k^{\ell}k^{-n} \\ &= \sum_{n=c\,\ell}^{\infty} \left(\frac{ne}{2\ell}\right)^{2\ell}\left(\frac{1}{k-1}\right)^{\ell}\left(\frac{k-1}{k}\right)^{n-\ell}\end{aligned}$$

where in the penultimate step we have used (58) and Motwani and Rahgavan [1995, Proposition B.2 part 3]. The ratio of successive terms in the sum on the right is bounded by

$$\left(1+\frac{1}{c\,\ell}\right)^{2\ell}\frac{k-1}{k} \le (e^{1/(c\,\ell)})^{2\ell}e^{-1/k} = \exp\left(\frac{2}{c}-\frac{1}{k}\right) \le \exp\left(-\frac{1}{3k}\right) \quad (67)$$

provided we pick $c \ge 3k$. Thus using the fact that $e^{-1/(3k)} \le 1 - 1/(4k)$ for $k \ge 2$ we have

$$\begin{aligned}\sum_{n=c\,\ell}^{\infty} F(n,\ell,h)z^n &\le \frac{1}{1-e^{-1/(3k)}}\left(\frac{ec\,\ell}{2\ell}\right)^{2\ell}\left(\frac{1}{k-1}\right)^{\ell}\left(\frac{k-1}{k}\right)^{c\,\ell-\ell} \\ &\le 4k\left(\frac{ec}{2}\right)^{2\ell}\left(\frac{1}{k-1}\right)^{\ell}\left(\frac{k-1}{k}\right)^{c\,\ell-\ell} \\ &= 4k\left(\frac{e^2c^2}{4(k-1)}\left(\frac{k-1}{k}\right)^{c-1}\right)^{\ell} \\ &\le 4k/2^{\ell}, \qquad (68)\end{aligned}$$

where the last step holds providing that we pick c large enough (depending only on k) so that

$$\frac{e^2c^2}{4(k-1)}\left(\frac{k-1}{k}\right)^{c-1} \le \frac{1}{2}.$$

Combining (65), (66), and (68) gives (60). □

For the remainder of this section we let c denote the constant whose existence is guaranteed by Lemma 3.14. We now further constrain the parameter z to appropriately balance the terms on the right of (60) and then refine the estimate.

LEMMA 3.15. *Let h and z vary with ℓ so that*

$$h \ge c\,\ell \quad (69)$$

and for some positive constant c' (depending only on k), we have

$$-\left(\frac{\log \ell}{\ell}\right)^{2/3} \le \log(kz) \le -c'\left(\frac{\log \ell}{\ell}\right)^{2/3}. \quad (70)$$

Then, as ℓ approaches ∞, we have

$$\log f_{\ell,h}(z) \le \frac{2\ell}{\gamma}\log(kz) + O(\log \ell). \tag{71}$$

PROOF. Note that (70) implies $1 - kz = \Omega(\ell^{-1})$. Choose x so that

$$0 \le x - 6k^2\,\ell^{2/3}\log^{1/3}\ell \le 1 \tag{72}$$

and

$$n_0 = 2\ell/\gamma - x$$

is a positive integer. Note that then from (70) we have $(kz)^x = \ell^{O(1)}$. Then, for large ℓ, the conditions of Lemma 3.14 apply so

$$\begin{aligned} f_{\ell,h}(z) &\le 2n_0^2 e^{-\gamma^3 x^2/(16\ell)} + \frac{n_0+1}{(1-kz)^2}(kz)^{n_0} + 4k/2^\ell \\ &= O(\ell^2)e^{-36k^4\gamma^3\ell^{4/3}\log^{2/3}\ell/(16\ell)} + O(\ell^3)(kz)^{2\ell/\gamma - x} + 4k/2^\ell \\ &= O(\ell^2)e^{-(9/4)k^4\gamma^3\ell^{1/3}\log^{2/3}\ell} + \ell^{O(1)}(kz)^{2\ell/\gamma} + 4k/2^\ell. \end{aligned} \tag{73}$$

The log of the first term on the right of (73) is asymptotic to $-\frac{9}{4}k^4\gamma^3\,\ell^{1/3}\log^{2/3}\ell$ while (from (70)) the log of the second term is bounded below by

$$-\frac{2\ell}{\gamma}\left(\frac{\log\ell}{\ell}\right)^{2/3} + O(\log\ell) \sim -\frac{2}{\gamma}\ell^{1/3}\log^{2/3}\ell,$$

so one easily sees (using the naive bound $\gamma \ge k^{-1}$) that the middle term on the right of (73) dominates the other two terms for large ℓ. Hence

$$f_{\ell,h}(z) \le \ell^{O(1)}(kz)^{2\ell/\gamma} \tag{74}$$

and thus (71) holds. □

A direct application of Lemma 3.15 will not suffice for our needs, since to apply Theorem 3.3 we want to bound the behavior of $f_{\ell,h}$ for fixed h as $\ell \to \infty$. The fact that $\mathcal{M}_h$ satisfies part (1) of Definition 3.12 will imply a subadditivity property of $\log f_{\ell,h}$ that can be used to overcome this problem. (Subadditivity of a related generating function was used in the classic paper [Alexander 1994].)

LEMMA 3.16. $\log f_{\ell+\ell',h}(z) \le \log f_{\ell,h}(z) + \log f_{\ell',h}(z)$.

PROOF. Consider any two sequences $p_1, p_2, \ldots, p_\ell$ and $p'_1, p'_2, \ldots, p'_{\ell'}$ of minimal match pairs. From part (1) of Definition 3.12, we know that $\mathcal{M}_h$ can accept $p_1, p_2, \ldots, p_\ell, p'_1, p'_2, \ldots, p'_{\ell'}$ only if it accepts $p_1, p_2, \ldots, p_\ell$ and it accepts $p'_1, p'_2, \ldots, p'_{\ell'}$. Considering all of the ways that n characters could be split between $p_1, p_2, \ldots, p_\ell$ and $p'_1, p'_2, \ldots, p'_{\ell'}$, we see that

$$F(n, \ell+\ell', h) \le \sum_{i=0}^{n} F(i, \ell, h)F(n-i, \ell', h),$$

so

$$\begin{aligned} f_{\ell+\ell',h}(z) &= \sum_{n=0}^{\infty} F(n, \ell+\ell', h) z^n \\ &\le \sum_{n=0}^{\infty} \sum_{i=0}^{n} F(i, \ell, h) F(n-i, \ell', h) z^n \\ &= f_{\ell,h}(z) f_{\ell',h}(z), \end{aligned}$$

from which the Lemma follows immediately. □

The following trivial technical lemma will be useful in the proof of Theorem 3.13. (Recall that we have fixed c to be the integer whose existence was guaranteed by Lemma 3.14.)

LEMMA 3.17. *For large enough h (depending only on k) the following will hold. Let ℓ be an integer with*

$$\ell \ge h \tag{75}$$

and let

$$d = \left\lceil \frac{2c\,\ell}{h} \right\rceil \text{ and } \bar{\ell} = \frac{\ell}{d}. \tag{76}$$

Then, if ℓ' is $\lfloor \bar{\ell} \rfloor$ or $\lceil \bar{\ell} \rceil$, we have

$$\frac{h}{3c} \le \ell' \le \frac{h}{c} \tag{77}$$

and

$$\frac{\log \ell'}{\ell'} \ge \frac{\log h}{h} \ge \frac{\log \ell'}{3c\,\ell'}. \tag{78}$$

PROOF. We have

$$d = \left\lceil \frac{2c\,\ell}{h} \right\rceil \le \frac{2c\,\ell}{h} + 1,$$

so

$$\lfloor \bar{\ell} \rfloor = \left\lfloor \frac{\ell}{d} \right\rfloor \ge \frac{\ell - (d-1)}{d} \ge \frac{\ell - 2c\,\ell/h}{2c\,\ell/h + 1} = \frac{h - 2c}{2c + h/\ell} \ge \frac{h}{3c} \tag{79}$$

where the last step holds for large h since $\ell \ge h$ and $c \ge 2$. This establishes the left inequality in (77). Also, since $d \ge 2c\,\ell/h$ we have

$$\lceil \bar{\ell} \rceil = \left\lceil \frac{\ell}{d} \right\rceil \le \frac{\ell}{2c\,\ell/h} + 1 = \frac{h}{2c} + 1 \le \frac{h}{c}$$

for large h, establishing the right inequality in (77).

Since $\log x/x$ is decreasing in x for $x > e$ we have from (77), for large enough h,

$$\frac{\log \ell'}{\ell'} \le \frac{\log(h/(3c))}{h/(3c)} \le 3c \frac{\log h}{h}$$

and

$$\frac{\log \ell'}{\ell'} \geq \frac{\log(h/c)}{h/c} = c\frac{\log(h/c)}{h} \geq \frac{c}{2} \cdot \frac{\log h}{h} \geq \frac{\log h}{h},$$

establishing (78). □

PROOF OF THEOREM 3.13. Let ℓ be any integer with

$$\ell \geq h \tag{80}$$

and define d and $\bar{\ell}$ as in (76). Decompose ℓ as

$$\ell = \sum_{i=1}^{d} \ell_i$$

where each ℓ_i is $\lfloor \bar{\ell} \rfloor$ or $\lceil \bar{\ell} \rceil$. By Lemma 3.16, we have

$$\log f_{\ell,h}(z) \leq \sum_{i=1}^{d} \log f_{\ell_i,h}(z). \tag{81}$$

Define z by

$$\log(kz) = -\left(\frac{\log h}{h}\right)^{2/3}. \tag{82}$$

Then, for large h, using Lemma 3.17 we can apply Lemma 3.15 (with the ℓ_i, which have value $\lfloor \bar{\ell} \rfloor$ or $\lceil \bar{\ell} \rceil$, playing the role of ℓ in Lemma 3.15) to the right of (81) to conclude that

$$\log f_{\ell,h}(z) \leq \frac{2\ell}{\gamma} \log(kz) + O(d \log \bar{\ell}).$$

Thus, condition (28) of Theorem 3.3 holds for large ℓ with

$$\begin{aligned} \log \lambda(z) &= \frac{2}{\gamma} \log(kz) + O\left(\frac{d}{\ell} \log \bar{\ell}\right) \\ &= \frac{2}{\gamma} \log(kz) + O(\bar{\ell}^{-1} \log \bar{\ell}) \\ &= \frac{2}{\gamma} \log(kz) + O(h^{-1} \log h), \end{aligned} \tag{83}$$

so the bound obtained from an application of Theorem 3.3 is

$$\begin{aligned} \frac{2 \log(kz)}{(2/\gamma) \log(kz) + O(h^{-1} \log h)} &= \gamma + O\left(\frac{h^{-1} \log h}{|\log(kz)|}\right) \\ &= \gamma + O\left(\frac{h^{-1} \log h}{h^{-2/3} \log^{2/3} h}\right) \\ &\qquad \text{by (82)} \\ &= \gamma + O\left(\left(\frac{\log h}{h}\right)^{1/3}\right), \end{aligned}$$

completing the proof of Theorem 3.13. □

3.3.2. *The Convergence Question for Domination-Based Methods.* As mentioned above, [Dančík 1994; Dančík and Paterson 1995], use the notion of undominated collations to produce upper bounds. They do not explicitly give a method for obtaining a sequence of machines yielding improved bounds, but here we discuss a very natural extension of their method. We conjecture that for alphabet size $k = 2$ it does meet the condition of Theorem 3.13 and thus does yield arbitrarily close upper bounds.

We (trivially) extend their definition of domination to lists of minimal match pairs by saying that a list L of ℓ minimal match pairs is undominated if the collation of order ℓ obtained by appending $\binom{\epsilon}{\epsilon}$ to L is undominated. Construct $\mathcal{M}_h$ as follows. Enumerate all lists of minimal match pairs of size at most h that are dominated, but have no contiguous proper sublist dominated; call this $\mathcal{D}_h$. Now let $\mathcal{M}_h$ be the machine that reads in lists of minimal match pairs and rejects whenever it finds one of the elements of $\mathcal{D}_h$ appearing as a contiguous sublist of its input. It is easy to see that this can be done by a finite state machine, and that this machine will never reject an undominated list, so by Theorem 3.3, it gives a valid bound for arbitrary k. We will assume in this section that the machine is minimized so it has no distinct equivalent states.

Next we show that this approach meets the condition of Theorem 3.10, that is, that $\mathcal{M}_h$ is regular. The following lemma will be useful; let $\binom{0}{0}^{\ell}$ denote a list consisting of ℓ copies of $\binom{0}{0}$.

LEMMA 3.18. *If $\mathcal{M}_h$ rejects $L \parallel \binom{0}{0}^{\lceil h/2 \rceil} \parallel L'$, then it must also reject L or L'.*

PROOF. It is not hard to see that if a list L of minimal match pairs is undominated, both the list $\binom{0}{0} \parallel L$ and the list $L \parallel \binom{0}{0}$ must also be undominated. Thus, $\mathcal{D}_h$ contains no lists beginning or ending with $\binom{0}{0}$. If $\mathcal{M}_h$ rejects $L \parallel \binom{0}{0}^{\lceil h/2 \rceil} \parallel L'$, it must be that some list D in $\mathcal{D}_h$ appears as a contiguous sublist of $L \parallel \binom{0}{0}^{\lceil h/2 \rceil} \parallel L'$. Since D has size at most h it has length at most $h/2$. Since it cannot begin or end with $\binom{0}{0}$, it must then be a contiguous sublist of either L or L', proving the lemma. □

From this lemma (and the fact that the finite state machine is minimized), it follows that whenever the machine has not yet rejected and then sees a sequence of $\lceil h/2 \rceil$ copies of $\binom{0}{0}$, it will return to the starting state. Also, if the machine is in the start state and reads $\binom{0}{0}$, it will remain in the start state. Hence, the machine is regular as required by Theorem 3.10.

Next we ask whether the sequence $\mathcal{M}_h$ efficiently covers Σ^*. The sequence $\mathcal{M}_h$ trivially meets condition (1) of Definition 3.12, but at first it might seem that any method based on undominated collations could not meet condition (2): It is easy to give a string pair that has more than two undominated collations. For example, $\binom{01}{10}$ can be written as $\binom{01}{1}\binom{\epsilon}{0}$ or as $\binom{0}{10}\binom{1}{\epsilon}$. This does not give a counterexample to condition (2) however, because the lists of minimal match pairs in these two decompositions (without including the final null pairs $\binom{\epsilon}{0}$ and $\binom{1}{\epsilon}$) do not generate the same string pair. We make the following conjecture.

CONJECTURE 3.19. *Over an alphabet of size $k = 2$, no string pair is generated by two distinct undominated lists of minimal match pairs.*

If this conjecture can be proved, it would establish that the sequence $\mathcal{M}_h$ based on domination efficiently covers $\{0, 1\}^*$ and thus gives bounds coming arbitrarily close to γ_2.

One easily establishes that this conjecture does not hold for any alphabets of size larger than 2. Two easy counterexamples are the string pair $\binom{012}{102}$, which has both $\binom{01}{1}\binom{2}{02}$ and $\binom{0}{10}\binom{12}{2}$ as undominated lists of minimal match pairs, and the string pair $\binom{10120}{0210}$, which has both $\binom{10}{0}\binom{1}{21}\binom{20}{0}$ and $\binom{10}{0}\binom{12}{2}\binom{0}{10}$ as undominated lists of minimal match pairs. Thus, we cannot apply Theorem 3.13 to the sequence $\mathcal{M}_h$ when the alphabet size is more than 2. Of course, the fact that this particular theorem does not apply does not immediately rule out the possibility that $\mathcal{M}_h$, or perhaps some other construction based on undominated collations, produces arbitrarily good approximations for an alphabet of size greater than 2.

3.3.3. *Convergence of Canonicity-Based Methods.* Now let $\mathcal{M}_h$ be the sequence of machines, based on canonicity, discussed in Section 3.2. As mentioned there, each $\mathcal{M}_h$ covers Σ^*.

We now show that each $\mathcal{M}_h$ is regular, and that the sequence $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \ldots$ efficiently covers Σ^*. We begin by establishing some properties of the transition function δ of the finite state machine. Call the portion of the table lcs constructed during the procedure delta of Figure 7 the *limited-history table*. The following two simple observations will be useful.

LEMMA 3.20. *As we move one step down, to the right, or diagonally down and to the right, among locations for which the values in the limited-history table are computed, the table values either remain the same or increase by* 1.

LEMMA 3.21. *The result returned by the procedure delta of Figure 7 remains the same if we*

(1) *initialize* $lcs[h, h]$ *to an arbitrary integer instead of* h *in line 3, or*
(2) *translate all indices into the table* lcs *by a constant amount.*

The next lemma asserts that if the machine is in any accepting state q and reads a match pair of the form $\binom{c}{c}$, it will remain in an accepting state.

LEMMA 3.22. *Let* $p = \binom{c}{c}$ *where* c *is any character in* Σ. *Then for any* $q \in \mathcal{S}$, *we have* $\delta(q, p) \in \mathcal{S}$.

PROOF. Since $q \in \mathcal{S}$, we must have

$$lcs[h, h-1] = lcs[h-1, h] = h - 1, \tag{84}$$

since otherwise q would have failed the canonicity test. The new characters read corresponding to the match pair p will be a_{h+1} and b_{h+1}; since these match we will have $lcs[h+1, h+1] = h+1$. The canonical backtracking test accepts if we move from $(h+1, h+1)$ to (h, h), which can fail to happen only if $lcs[h+1, h]$ or $lcs[h, h+1]$ is also equal to $h + 1$. But this and (84) would contradict Lemma 3.20. □

Let $\mathcal{S}_0$ be the set of states $\langle x, y, dx, dy\rangle \in \mathcal{S}$ in which dx and dy are both 1^h. Note that the start state of $\mathcal{M}_h$ is an element of $\mathcal{S}_0$.

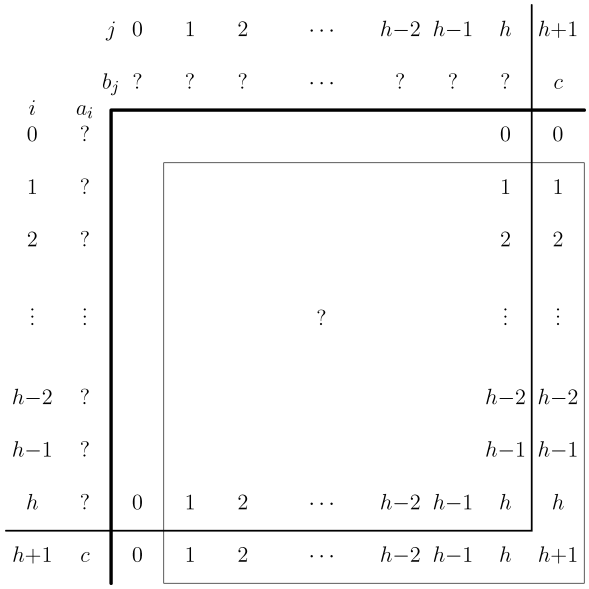


FIG. 8. Illustration for the proof of Lemma 3.23.

LEMMA 3.23. *Let* $p = \binom{c}{c}$ *where* c *is any character in* Σ. *Then, for any* $q \in \mathcal{S}_0$, *we have* $\delta(q, p) \in \mathcal{S}_0$. *In particular, if* q *is the start state* $(0^h, 0^h, 1^h, 1^h)$, *then* $\delta(q, \binom{0}{0}) = q$.

PROOF. The proof is illustrated in Figure 8; the portion of the figure above and to the left of the medium-thickness line shows the contents of the table reconstructed from q. The entry in position (h, h) is h, and then the remaining entries shown inside the box bounded by the thick and medium line are determined by the fact that dx and dy are each 1^h.

Let $\delta(q, p)$ be $q' = (x', y', dx', dy')$. The portion of the lcs table corresponding to q' is enclosed in the thin line. Following the procedure in Figure 7 produces the new entries shown. (The value at $lcs[h+1, h+1]$ is $h+1$ since a_{h+1} matches b_{h+1}; note that otherwise the values of the new entries do not depend on whether or not any of the characters match.) Thus, dx' and dy' will also consist of strings of h one bits. □

LEMMA 3.24. *Let* $q \in \mathcal{S}$ *be any accepting state. Assume that starting in state* q, *the machine reads* h *copies of* $\binom{1}{1}$ *followed by* h *copies of* $\binom{0}{0}$. *Then, the new state of the machine will be the start state* $(0^h, 0^h, 1^h, 1^h)$.

PROOF. The proof is illustrated in Figure 9; here, Lemma 3.21 is used to show multiple calls to delta in a single table, without renumbering the indices. The portion of the figure above and to the left of the thin line shows the state of the table after starting in state q and then reading in h copies of $\binom{1}{1}$. By Lemma 3.22, we know that this state is in $\mathcal{S}$, so we must have

$$lcs[h-1, h] < h \quad \text{and} \quad lcs[h, h-1] < h.$$

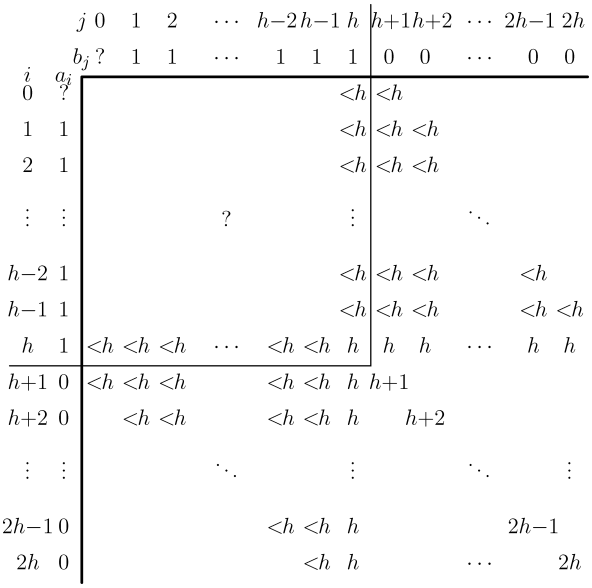


FIG. 9. Illustration for the proof of Lemma 3.24.

Thus, all of the entries to the left of or above $lcs[h, h]$ must be less than h. Note that at this point all of the characters stored in the state are ones.

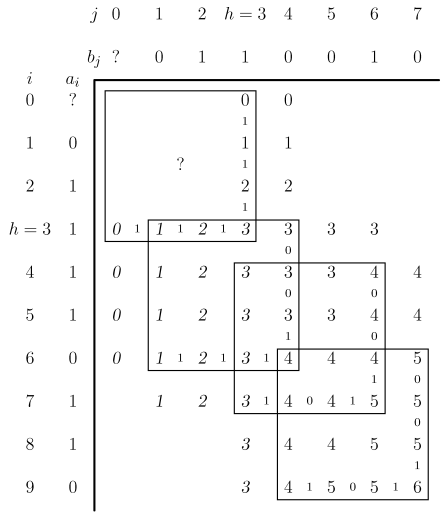
Now consider what happens when we read in the h copies of $\binom{0}{0}$, which is illustrated in the remainder of the figure. All of the new entries $lcs[i, j]$ where $j < h$ will continue to be less than h since the third case of recurrence (39) will never be used, so in particular $lcs[2h, h-1]$ is less than h. On the other hand, as we step down the main diagonal from $lcs[h, h]$ to $lcs[2h, 2h]$, the entries increase by 1 at each step since we match a 0 with a 0, so we will have $lcs[2h, 2h] = 2h$. But then, by Lemma 3.20, we must have $lcs[2h, j] = j$ for $h \le j \le 2h$. Similarly, we have $lcs[i, 2h] = i$ for $h \le i \le 2h$. Thus, encoding the bottom-right portion of the table yields the state asserted in the lemma. □

LEMMA 3.25. *Let $P = p_1, p_2, \ldots, p_m$ be any sequence of minimal match pairs, and let $q \in \mathcal{S}$ and $q' \in \mathcal{S}_0$. Then if the machine starting in state q accepts P, the machine starting in state q' must also accept P.*

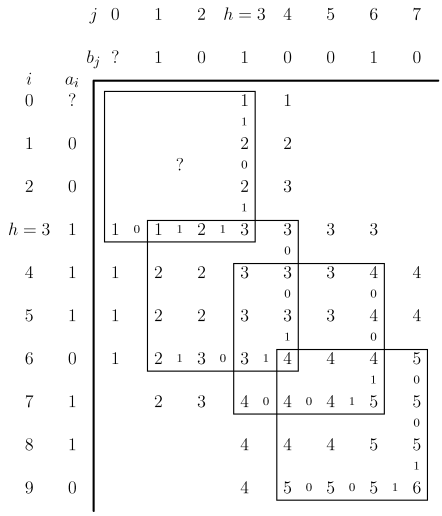
PROOF. First, consider how the limited-history lcs table evolves if we start in $q' = \langle x', y', dx', dy' \rangle$ and then read in the successive minimal match pairs of P; this is illustrated for a specific example of a P and a $q' \in S_0$ in part (a) of Figure 10. Let lcs' denote the entries in this table, so $lcs'[h, h] = h$. Note that, since dy' is a string of h 1-bits, decoding dy' yields $lcs'[h, j] = j$ for $0 \le j \le h$. It follows by inspection of the algorithm of Figure 7 and an easy induction that for any entry $lcs'[i, j]$ that we compute we have

$$i \ge h \text{ and } j \le h \implies lcs'[i, j] = j. \tag{85}$$

(In the figure, these entries are shown in a slanted font.) To see this, note that in those cases where we simply copy an entry from the one above the inductive step is trivial; on the other hand, in the cases where we compute an



(a) We start in state $q' = \langle 011, 011, 111, 111\rangle$, read the input $P = \binom{110}{0}\binom{1}{01}\binom{10}{0}$, and end in state $\langle 110, 010, 001, 101\rangle$.



(b) We start in state $q = \langle 001, 101, 101, 011\rangle$, read the input $P = \binom{110}{0}\binom{1}{01}\binom{10}{0}$, and end in state $\langle 110, 010, 001, 001\rangle$.

FIG. 10. Illustration for the proof of Lemma 3.25. The portions of the lcs table corresponding to each successive state are outlined by squares with thin lines. The small integers between certain lcs table entries show the differences between these entries.

entry $lcs'[i, j]$ from the recurrence (39), by the inductive hypothesis we have $lcs'[i-1, j-1] = lcs'[i, j-1] = j-1$ and $lcs'[i-1, j] = j$, so we set $lcs'[i, j]$ to j regardless of whether the relevant characters match. By a symmetric argument, for entries we compute we also have

$$i \le h \text{ and } j \ge h \implies lcs'[i, j] = i. \tag{86}$$

From (85) and (86), we conclude that, in particular,

$$(i \geq h \text{ and } j = h) \text{ or } (i = h \text{ and } j \geq h) \implies lcs'[i, j] = h. \tag{87}$$

Next consider how the limited-history lcs table evolves if we start in $q = \langle x, y, dx, dy \rangle$ and then read in the successive minimal match pairs of P, letting lcs denote the entries in this table. (This is illustrated in part (b) of Figure 10 for a specific example of q, and for the same P as used in part (a); one need not examine all of the details in part (b) to verify the proof.) From Lemma 3.20, it follows immediately that, for the table entries that we compute,

$$(i \geq h \text{ and } j = h) \text{ or } (i = h \text{ and } j \geq h) \implies lcs[i, j] \geq h; \tag{88}$$

Combining this with (87) gives, for the table entries that we compute,

$$(i \geq h \text{ and } j = h) \text{ or } (i = h \text{ and } j \geq h) \implies lcs[i, j] \geq lcs'[i, j]. \tag{89}$$

For any fixed sequence P, all computed entries $lcs[i, j]$ for which $i \geq h$ and $j \geq h$ can be expressed as nondecreasing functions of the set of values bounded in (89), with no further dependence on $a_1 a_2 \cdots a_h$ and $b_1 b_2 \cdots b_h$, so we have

$$i \geq h \text{ and } j \geq h \implies lcs[i, j] \geq lcs'[i, j]. \tag{90}$$

Let Γ be the canonical backtracking path for the case in which the machine starts in state q. Since the machine starting in state q accepts P, the inequality in (90) must be tight along Γ. Thus, by (90) during the canonical backtracking process the entries lying off Γ are never more attractive (i.e., larger) for the table starting in state q' than they were for the table starting in state q, so we will also accept P when we start in state q'. □

(Note that it follows that all states in $\mathcal{S}_0$ are equivalent, but we did not take advantage of this in our calculations.)

From Lemma 3.24, we see that each $\mathcal{M}_h$ satisfies part (1) of Definition 3.9, and from Lemma 3.23 we see that it satisfies part (2). Thus, it is regular, so Theorem 3.10 applies.

From Lemma 3.25, the sequence $\mathcal{M}_h$ satisfies part (1) of Definition 3.12. As long as the total size of the match pairs read does not exceed h, the machine models the process of computing the lcs table exactly, so by Lemma 3.4 we see that part (2) holds. Thus, the sequence $\mathcal{M}_h$ efficiently covers Σ^*, so we have, by Theorem 3.13,

THEOREM 3.26. *For arbitrary alphabet size k, the sequence $\mathcal{M}_h$ based on canonicity is regular and efficiently covers Σ and thus gives bounds coming arbitrarily close to γ_k.*

ACKNOWLEDGMENTS. I thank Wayne Hayes for giving me pointers to information about interval arithmetic, David Eppstein and one of the anonymous referees for asking questions that encouraged me to pursue the issues addressed in Section 3.3, Padhraic Smyth for bringing the book [Seneta 1981] to my attention, and Dan Hirschberg for discussing with me the terminology in the literature relating to dominant matches.

REFERENCES

ALEXANDER, K. S. 1994. The rate of convergence of the mean length of the longest common subsequence. *Ann. Appl. Prob. 4,* 4, 1074–1082.

APOSTOLICO, A., AND GUERRA, C. 1987. The longest common subsequence problem revisited. *Algorithmica 2*, 315–336.

BAEZA-YATES, R. A., GAVALDÀ, R., NAVARRO, G., AND SCHEIHING, R. 1999. Bounding the expected length of longest common subsequences and forests. *Theory Comput. Syst. 32*, 435–452.

CHVÁTAL, V., AND SANKOFF, D. 1975. Longest common subsequences of two random sequences. *J. Appl. Prob. 12*, 306–315.

DANČÍK, V. 1994. Expected length of longest common subsequences. Ph.D. dissertation, Department of Computer Science, University of Warwick.

DANČÍK, V., AND PATERSON, M. 1995. Upper bounds for the expected length of a longest common subsequence of two binary sequences. *Rand. Struct. Algor. 6,* 4, 449–458.

GOSLING, J., JOY, B., STEELE, G., AND BRACHA, G. 2005. *The Java*^{TM} *Language Specification*, Third ed. Addison-Wesley, Reading, MA.

HIRSCHBERG, D. S. 1977. Algorithms for the longest common subsequence problem. *J. ACM 24,* 4 (Oct.), 664–675.

JANSON, S., LUCZAK, T., AND RUCINSKI, A. 2000. *Random Graphs*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley, New York.

JIANG, T., AND LI, M. 1995. On the approximation of shortest common supersequences and longest common subsequences. *SIAM J. Comput. 24,* 5 (Oct.), 1122–1139.

KIWI, M., LOEBL, M., AND MATOUŠEK, J. 2005. Expected length of the longest common subsequence for large alphabets. *Adv. Math. 197,* 2, 480–498.

KIWI, M., AND SOTO, J. 2008. On a speculated relation between Chvátal-Sankoff constants of several sequences. *Combinatorics, Probability and Computing*. To appear. Available on-line at `http://arxiv.org/abs/0810.1066`.

MACCLUER, C. R. 2000. The many proofs and applications of Perron's theorem. *SIAM Rev. 42,* 3, 487–498.

MOTWANI, R., AND RAHGAVAN, P. 1995. *Randomized Algorithms*. Cambridge University Press, Cambridge, MA.

OVERTON, M. L. 2001. *Numerical Computing with IEEE Floating Point Arithmetic*. Society for Industrial and Applied Mathematics, Philadelphia, PA.

PATERSON, M., AND DANČÍK, V. 1994. Longest common subsequences. In *Mathematical Foundations of Computer Science 1994: 19th International Symposium, MFCS'94,* Kosice, Slovakia, August 22–26, 1994: Proceedings, I. Privara, B. Rovan, and P. Ruzicka, Eds. Lecture Notes in Computer Science, vol. 841. Springer-Verlag, Berlin, Germany, 127–142.

PEVZNER, P. A. 2000. *Computational Molecular Biology: An Algorithmic Approach*. The MIT Press, Cambridge, MA.

SENETA, E. 1981. *Non-negative Matrices and Markov Chains*, 2nd ed. Springer Series in Statistics. Springer-Verlag, New York.

STEELE, J. M. 1997. *Probability Theory and Combinatorial Optimization*. CBMS-NSF Regional Conference Series in Applied Mathematics. Society for Industrial and Applied Mathematics, Philadelphia, PA.

RECEIVED JULY 2005; REVISED DECEMBER 2008; ACCEPTED JANUARY 2009