# Bounding the Expected Length of Longest Common Subsequences

George S. Lueker

## 1. Introduction

A *subsequence* of $A = a_1 a_2 \cdots a_n$ is any string which can be obtained from $A$ by deleting some of the characters.

Given two strings $A$ and $B$, the *longest common subsequence* (abbreviated *lcs*) of $A$ and $B$ is the longest string which is a subsequence of both $A$ and $B$.

We denote its length by $L(A, B)$.

Let $A$ and $B$ each be generated $n$ independent coin flips, and let $L_n$ denote $L(A, B)$.

[CS75]: By subadditivity there exists a $\gamma > 0$ such that

$$\lim_{n\to\infty} \frac{\mathsf{E}[L_n]}{n} = \gamma.$$

In fact Alexander [Alex94] has proved

$$\mathsf{E}[L_n] \geq \gamma n - O(\sqrt{n \log n}).$$

Value of $\gamma$ is not known.

Goal: bound $\gamma$ from above and below.

Let $X$ and $Y$ be two random binary strings (generated by independent coin flips) of length $2n$, and let $X(i)$ (resp. $Y(i)$) denote the prefix of $X$ (resp. $Y$) of length $i$. Alexander [Alex94] also proved that

$$\mathsf{E}\left[L(X(n),Y(n))\right] \sim \mathsf{E}\Big[\max_{i+j=2n} L\big(X(i),X(j)\big)\Big].$$

## 2. Lower bounds on $\gamma$

Best known previous: 0.773911 [Danč94, DP95]

Approach

- devise a finite-state automaton to read characters from $A$ and $B$ and produce a common subsequence
- model by Markov chain
- find the steady-state distribution
- compute the expected number of characters output per character read

[Danč94]

- produces a lower bound of $\gamma > 0.773911$ by considering a specific finite state automaton with 931 states.
- produces a mechanical procedure which could be used to produce a sequence of machines which give lower bounds which approach the true value of $\gamma$ in the limit.

([BYGNS99] introduce some other approaches, but for the particular problem of bounding the value of $\gamma$, they do not lead to improvements.)

## New Bounds

The Recurrence

Idea: let states be pairs of strings $k$ characters (buffers) from each string.

- When leftmost characters match, delete and add new characters on the right of each string.
- When leftmost characters don't match, discard one from one buffer and append new character at right end

Similar to method [Danč94, DP95], but without careful design of automaton.

If $s$ and $t$ are binary strings, let

$$v_n(s,t) = \mathsf{E}\Big[\min_{i+j=n} L\big(sX(i), tY(j)\big)\Big].$$

(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.)

Then

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

## The recurrences

Example:

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

General: When $\lfloor i/2^{k-1} \rfloor = \lfloor j/2^{k-1} \rfloor$,

$$\begin{aligned}\vec{v}_n(i,j) \geq 1 + \frac{1}{4}\Big(&\vec{v}_{n-2}(2i \bmod 2^k, 2j \bmod 2^k)\\ &+\vec{v}_{n-2}(2i \bmod 2^k, 2j+1 \bmod 2^k)\\ &+\vec{v}_{n-2}(2i+1 \bmod 2^k, 2j \bmod 2^k)\\ &+\vec{v}_{n-2}(2i+1 \bmod 2^k, 2j+1 \bmod 2^k)\Big).\end{aligned}$$

Example:

$$v_n(001,111) \geq \max \begin{cases} \frac{1}{2}\sum_{c\in\{0,1\}} v_{n-1}(01c, 111) \\ \frac{1}{2}\sum_{c\in\{0,1\}} v_{n-1}(001, 11c) \end{cases}. \qquad (3)$$

General: When $\lfloor i/2^{k-1} \rfloor = \lfloor j/2^{k-1} \rfloor$,

$$\vec{v}_n(i,j) \geq \max \begin{cases} \frac{1}{2}\Big(\vec{v}_{n-1}(2i \bmod 2^k, j) + \vec{v}_{n-1}(2i+1 \bmod 2^k, j)\Big) \\ \frac{1}{2}\Big(\vec{v}_{n-1}(i, 2j \bmod 2^k) + \vec{v}_{n-1}(i, 2j+1 \bmod 2^k)\Big) \end{cases}$$

Note that this defers the design of the FSA.

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 $k$ and
- $T$ be such that the entire set of such recurrences expressed by

$$\forall n \geq 2 \quad \vec{v}_n \geq T(\vec{v}_{n-1}, \vec{v}_{n-2}). \tag{4}$$

The number of variables in each $\vec{v}_n$ is $2^{2k}$.

$T$ is *monotonic* in the sense that

$$\vec{v}' \geq \vec{v} \implies T(\vec{v}') \geq T(\vec{v})$$

where the inequalities hold componentwise.

$T$ is also *translationally invariant* in the sense that for any real $r$,

$$T(\vec{v}+r, \vec{v}'+r) = r + T(\vec{v}, \vec{v}').$$

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

$$T(\vec{v}+r, \vec{v}) \geq \vec{v} + 2r, \tag{5}$$

*then for any set of boundary values $\vec{v}_0$ and $\vec{v}_1$, there exists a vector $\vec{v}'$, such that the solution to (4) satisfies*

$$\forall n \geq 0 \quad \vec{v}_n \geq \vec{v}' + nr. \tag{6}$$

It will follow that $\gamma \geq 2r$.

## Properties of method

1. It is almost trivial to mechanically produce the inequalities yielding $T$.
2. It eliminates the need to solve a Markov chain for a steady-state distribution.
3. Simply iterate to find $\vec{v}$.
4. Rather immune to numerical problems.
5. Amenable to computation on external memory.

(2, 3, and 4 could of course also be applied to the Markov chains of [Danč94, DP95].)

## Use of External Memory

The recurrences are amenable to solution on external media such as disks; index pattern is similar to perfect shuffle.

## Some Implementation Details

The actual computations we carried were not exactly as described above, in an effort to achieve greater efficiency.

Some identities:

1. $v_n(s,t) = v_n(\overline{s}, \overline{t})$
2. $v_n(s,t) = v_n(t,s)$.
3. If the leftmost bits of $s$ and $t$ match, then the value of $v_n(s,t)$ is independent of whether this matching bit is a 0 or 1.

We used 1 and 3.

We did not use 2 to avoid problems with external memory usage.

## Simplifying the Recurrence

We actually computed

$$\vec{v}_n \geq T(\vec{v}_{n-1}, \vec{v}_{n-1}) \tag{7}$$

rather than

$$\vec{v}_n \geq T(\vec{v}_{n-1}, \vec{v}_{n-2}) \tag{8}$$

to save a factor of 2 on space.

- equivalent to only charging for one of the two new characters we draw after making a match
- equivalent to giving a free extra character whenever there is a match.

Let $\gamma'$ be the value computed using (8), and $\gamma$ be that computed (7). Then

$$\gamma' n = \gamma(n + \gamma' n/2),$$

so

$$\gamma = \frac{\gamma'}{1 + \gamma'/2}.$$

Approximate Results

Lower bounds obtained with various values of $k$

| $k$ | Bound |
|---|---|
| 5 | 0.7654 |
| 6 | 0.7702 |
| 7 | 0.7739 |
| 8 | 0.7768 |
| 9 | 0.7792 |
| 10 | 0.7812 |
| 11 | 0.7829 |
| 12 | 0.7845 |
| 13 | 0.7858 |
| 14 | 0.7870 |
| 15 | 0.7880 |

This beats the previous bound for $k$ around 8. For that value, the computation takes only a few seconds on a 300 MHz Pentium II.

For $k = 15$, the table of variable values uses 1.5 Gb, and each iteration takes about 45 min.

## 3. Upper bounds

[Danč94] points out that all upper bounds as of his writing use the following basic idea.

Let $F(i, n)$ be the number of pairs of strings of length $n$ which have a longest common subsequence of length $i$.

Let $G(i, 2n)$ be the number of pairs of strings of total length $2n$ which have a longest common subsequence of length $i$.

Then

$$\mathsf{E}\left[L_n\right] = \frac{1}{4^n} \sum_{i=1}^{n} iF(i, n). \tag{9}$$

and

$$F(i, n) \leq G(i, 2n)$$

Therefore if

$$\sum_{i=\lceil yn \rceil}^{n} G(i, 2n) = o(4^n), \tag{10}$$

then

$$\gamma \leq y.$$

**Bounding $G(i, 2n)$**

For a given string $S$ of length $i$, let an *$n$-collation* of $S$ be any way of inserting additional characters into two copies of $S$ to form two strings $A$ and $B$ of total length $n$.

We say this collation *yields* $A$ and $B$.

Let $C_n(S)$ be the number of distinct $n$-collations possible for string $S$. Then certainly

$$G(i, 2n) \leq \sum_{S \in \{0,1\}^{*i}} C_{2n}(S). \tag{11}$$

This bound may be very weak, since a given pair of strings with an lcs of length $i$ may be counted many times.

To improve bounds, minimize overcounting. (See [Danč94, Section 4.2] for a history of such improvements.)

a) One ten-collation of $S = 0100$, yielding the strings 01010 and 10100, is

$$\begin{cases} \hat{0} & \hat{1} & \hat{0} & 1 & \hat{0} \\ 1 & \hat{0} & \hat{1} & \hat{0} & \hat{0} \end{cases}$$

The characters from $S$ are marked.

b) Another ten-collation (this time of 1010) yielding the same pair as the collation in (a) is

$$\begin{cases} 0 & \hat{1} & \hat{0} & \hat{1} & \hat{0} \\ \hat{1} & \hat{0} & \hat{1} & \hat{0} & 0 \end{cases}$$

A collation is also *dominated* if it does not end in a pair of matched characters. So (b) is also dominated.

Two collations are considered distinct if they yield the same pair of strings but have different marks.

[Danč94, DP95] define a notion of a *dominated* collation.

Define the decomposition of a collation into *match pairs* to be the result of chopping both strings just after each match.

If there are $k$ match pairs, define the *collation key* to be the vector of length $k$ given by $\ell(1), \ell(2), \ldots, \ell(k)$, where $\ell(i)$ is the total number of characters in the first $i$ match pairs.

If two collations have equal upper strings and equal lower strings, the first *dominates* the second if the reverse of the first's collation key lexicographically precedes the reverse of the second's.

a) $\left\{\begin{array}{ccccc} \hat{0} & \hat{1} & \hat{0} & 1 & \hat{0} \\ 1 & \hat{0} & \hat{1} & \hat{0} & \hat{0} \end{array}\right.$. Collation key: (3,5,7,10).

b) $\left\{\begin{array}{ccccc} 0 & \hat{1} & \hat{0} & \hat{1} & \hat{0} \\ \hat{1} & \hat{0} & \hat{1} & \hat{0} & 0 \end{array}\right.$. Collation key: (3,5,7,9).

So, (b) dominates (a).

In order to obtain their best bounds, [Danč94, DP95] attempt to rule out dominated collations.

Define a finite state automaton which reads in match pairs and rejects when it can determine that the collation is dominated. This enables one to compute an upper bound on the number of non-dominated collations.

Let $q$ be a state of the FSA.

Let $H_q^{(k)}(z)$ be the generating function in which the coefficient of $z^n$ gives the number of $n$-collations strings of length $k$ which leave the automaton in state $q$.

[Danč94, DP95] give a recurrence on these polynomials, expressible as a matrix of polynomials. They show that, for any arbitrary value of $z_0 \in (0, 1/2)$, if the largest eigenvalue of the matrix is $\lambda(z_0)$, then we have

$$\gamma \leq 2 \frac{\ln(2z_0)}{\ln\big(\lambda(z)\big)}.$$

A carefully constructed 52-state automaton yields a bound of $\gamma \leq 0.837623$.

## New bounds

Further reduce the number of times a given collation $(A, B)$ of a string is counted.

Suppose we implement the dynamic programming solution.

Say a match is *canonical* if it is obtained by backtracking through the table, trying options in the order

a) move left,

b) move up,

c) make a match and move diagonally up and to the left.

Call the resulting match the *canonical* match.

| | | 0 | 1 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 2 | 2 | 2 |
| 1 | 0 | 1 | 2 | 2 | 3 | 3 |
| 1 | 0 | 1 | 2 | 2 | 3 | 4 |
| 1 | 0 | 1 | 2 | 2 | 3 | 4 |
| 0 | 0 | 1 | 2 | 3 | 3 | 4 |
| 1 | 0 | 1 | 2 | 3 | 4 | 4 |

Canonical match: $\begin{cases} \hat{0} \;\; \hat{1} \;\; \hat{0} \;\; \hat{1} \;\; 1 \\ 1 \;\; \hat{0} \;\; \hat{1} \;\; 1 \;\; 1 \;\; \hat{0} \;\; \hat{1} \end{cases}$

Now define $C_{k,i}$ to be the number of ways of choosing a binary string $S$ of length $k$ and then collating a total of $i$ additional characters into two copies of $S$ so that $S$ is the canonical longest common subsequence of the collation. Then we have no overcounting.

As before an FSA can't remember the entire dynamic programming table in its state.

Maintain in the state the last $k$ characters of strings $A$ and $B$ and a *lower bound* on bottom-right last $k \times k$ entries in the dynamic programming table, which is tight for the canonical match. (Actually, of course we only need to maintain differences of values in the table.)

Thus this modified table enables us to construct upper bounds on the numbers of canonical collations, since lowering table values corresponding to non-canonical matches only makes them less attractive during the backtracking.

Example: Initial state is

$$< 011, 001, 11, 01 > .$$

$$\begin{array}{c|cccc}
 & 0 & 0 & 1 & 1 \\
\hline
0 & & & 0 & 0 \\
1 & & & 1 & 1 \\
1 & 1 & 1 & 2 & 2 \\
0 & 1 & 2 & 2 & 2 \\
0 & 1 & 2 & 2 & 2 \\
1 & 1 & 2 & 3 & \leftarrow 3
\end{array}$$

State after appending $\binom{001}{1}$ would be

$$< 001, 011, 10, 01 >$$

but we reject since this is not canonical.

Approximate Results

Upper bounds obtained with various values of $k$

| $k$ | Bound | Number of states |
|---|---|---|
| 2 | 0.8602 | 4 |
| 3 | 0.8459 | 25 |
| 4 | 0.8393 | 254 |
| 5 | 0.8351 | 2424 |
| 6 | 0.8320 | 22734 |
| 7 | 0.8296 | 194335 |

(Note: The program to compute these is a bit involved; I plan to do some further checking of the logic and programming before publishing these numbers.)

## References

[Alex94] Kenneth S. Alexander. The rate of convergence of the mean length of the longest common subsequence. *Annals of Applied Probability*, 4(4):1074–1082, 1994.

[BYGNS99] R. A. Baeza-Yates, R. Gavaldà, G. Navarro, and R. Scheihing. Bounding the expected length of longest common subsequences and forests. *Theory Comput. Systems*, 32:435–452, 1999.

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

[Danč94] Vladimír Dančík. *Expected Length of Longest Common Subsequences.* Ph.D. thesis, Department of Computer Science, University of Warwick, September 1994.

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