hexagon:2611.00467v2

CERTIFIED UPPER BOUNDS FOR THE CHVÁTAL–SANKOFF CONSTANT

TEJASVI SINGH TOMAR

Abstract. The expected length of a longest common subsequence of two independent uniformly random binary strings of length n is $\gamma_2 n + o(n)$, where γ_2 is the Chvátal–Sankoff constant. Simulations suggest $\gamma_2 \approx 0.8122$. In 2009 Lueker obtained the bound $\gamma_2 \le 0.826280$ with a weighted finite automaton that remembers the last $h = 8$ symbols of each string. We prove $\gamma_2 \le 0.821963$ by the same method with $h = 13$, where the automaton has about $1.66 \cdot 10^{11}$ reachable states up to symmetry. The proof rests on an explicit vector $v > 0$ and a number $\lambda < 1$ with $Av \le \lambda v$, where A is the weighted transition matrix; a program checks the inequality $Av \le \lambda v$ entry by entry, in exact arithmetic. The bound is also proved in the Lean 4 proof assistant.

1. Introduction

For strings A and B let $\mathrm{LCS}(A, B)$ be the length of a longest common subsequence of A and B. Let A_n and B_n be independent uniformly random strings of length n over $\Sigma = \{0, 1\}$ and put $L_n = \mathrm{LCS}(A_n, B_n)$. Chvátal and Sankoff [2] showed that $\mathbb{E}L_n/n$ converges as $n \to \infty$; the limit γ_2 is called the Chvátal–Sankoff constant. Since $\mathbb{E}L_{m+n} \ge \mathbb{E}L_m + \mathbb{E}L_n$, the limit is also the supremum of $\mathbb{E}L_n/n$.

The value of γ_2 is not known. Simulations suggest $\gamma_2 \approx 0.8122$ [1]. Dančík [3] proved $\gamma_2 \ge 0.773911$. Dančík and Paterson [4] (see also [3]) proved $\gamma_2 \le 0.837623$ with a 52-state automaton. Lueker [7] improved both bounds, to $0.788071 \le \gamma_2 \le 0.826280$; the lower bound has since been improved to 0.792665992 [11]. Lueker's upper-bound technique uses finite automata M_h that read a pair of strings as a sequence of minimal match pairs (Section 2.1); here h is the history length: M_h remembers the last h symbols of each string. The bound 0.826280 comes from M_8, which has 6,419,109 reachable states after a symmetry reduction [7, Table II], [8]. This bound shows that the value $2/(1 + \sqrt{2}) \approx 0.828427$ conjectured by Steele is too large [7]. Lueker also proved that the bounds obtainable from M_h converge to γ_2 as $h \to \infty$. Jack [6] has computed the bounds of Lueker's method for $h = 9$ and $h = 10$: $\gamma_2 \le 0.825101$ and $\gamma_2 \le 0.824120$, respectively. The known bounds on γ_2 are collected in [5, page 31a]. We prove the following bound.

Theorem 1.1. $\gamma_2 \le 0.821963$.

Bukh and Cox [1, Section 5] found a binary word P of length 28 with the following property. Let P_n be the n-letter periodic word with period P; then the expected length of a longest common subsequence of P_n and a uniformly random binary string of length n is $\gamma_P n + O(\sqrt{n})$, with $\gamma_P \ge 0.82118$. An upper bound $\gamma_2 < 0.82118$ would show that this periodic word is closer to a random string than an independent random string is, in the sense of the expected length of a longest common subsequence. Bukh and Cox note that the bound $\gamma_2 \le 0.826280$ does not

2020 *Mathematics Subject Classification.* 60C05; 68R15, 68Q45, 65G20, 68V20.

Key words and phrases. Chvátal–Sankoff constant, longest common subsequence, random strings, Lueker's method, finite automata, computer-assisted proof, Lean 4.

suffice for this comparison; the bound of Theorem 1.1 does not suffice either, but it comes within 0.0008 of 0.82118.

We follow Lueker's method. A pair of strings decomposes canonically into minimal match pairs followed by a final pair of strings with no matching symbols (Lemma 2.3). There is one minimal match pair for each symbol of a particular longest common subsequence, the one found by the canonical traceback of Section 2.2. The automaton M_h reads the minimal match pairs one at a time. Its state records the last h symbols of each string, the last h increments down the last column of the dynamic programming table, and the last h increments along its last row. When the canonical traceback rule shows that the pair just read cannot be the next pair of a canonical decomposition, the automaton rejects this pair. Each minimal match pair of size s (the total length of its two strings) gets the weight z^s; let A_z be the weighted transition matrix of M_h. If $0 < z < 1/2$ and a vector $v > 0$ on the states satisfies $A_z v \le \lambda v$ for some $0 < \lambda < 1$, then $\gamma_2 \le 2\ln(2z)/\ln\lambda$ (Theorem 4.1). Lueker introduced the automata M_h and this way of bounding γ_2 with them [7]; the counting bound of Theorem 4.1 is due to Dančík and Paterson [3, 4], in the form stated in [7, Theorem 3.3]. A certificate consists of three parts: the parameter z; the states reachable from the start state, up to the symmetry that complements every symbol (Lemma 4.2); and a vector v on these states, found by a power iteration. Our checking program, the checker of Section 5, computes λ from these data.

Theorem 1.1 comes from a certificate for $h = 13$. Table 1 lists the bounds that we obtain with the same method for $h = 8, \dots, 13$, each at one chosen value of z (Section 5.6); for $h = 8$ we recover Lueker's value 0.826280, and for $h = 9$ and $h = 10$ we recover the values of [6].

Our verification differs from the earlier computations in two respects. First, the final bound of [7] is justified there under the assumption that the floating-point operations of Lueker's program perform as expected [7, Section 3.2.2]. The computations of [6] use floating-point arithmetic with upward rounding, together with a recheck for $h = 10$ that does not depend on the rounding mode. Our checking programs (the checker and two further implementations of its checks; Section 5.2) verify the required inequalities (4.1) of Section 4 exactly. They work in integer arithmetic, for a rational parameter z, with the weights of the inputs rounded up to integer multiples of 2^{-64} (Section 5). Second, Lueker explains why the automaton never rejects the canonical decomposition of a pair of strings [7, Section 3.2]; we give a detailed proof of this property, which we call the covering property (Lemma 3.1).

Theorem 1.1 has also been proved in the Lean 4 proof assistant (Section 6). The formal proof does not use `native_decide`, a tactic that executes compiled code; instead, the certificate of Section 5 is split into parts, and Lean's kernel checks each part. The formal proof covers the whole argument, from the definition of γ_2 to the check of the certificate.

Section 2 collects facts about longest common subsequences and the canonical decomposition. Section 3 defines M_h and proves the covering property, and Section 4 proves the counting bound. Section 5 describes the certificates, the checker and the results. Section 6 describes the formal proofs, and Section 7 contains remarks.

2. Longest common subsequences

Throughout, $\Sigma = \{0, 1\}$ and $\bar{c} = 1 - c$ for $c \in \Sigma$. For a string A we write $|A|$ for its length, $A[i]$ for its ith symbol and $A[i..j]$ for $A[i]A[i+1]\cdots A[j]$. For strings A, B let $L(i, j) = \mathrm{LCS}(A[1..i], B[1..j])$, with $L(i, 0) = L(0, j) = 0$. For $i, j \ge 1$,

$$L(i,j) = \begin{cases} L(i-1, j-1) + 1, & A[i] = B[j], \\ \max\{L(i-1, j), L(i, j-1)\}, & A[i] \ne B[j]. \end{cases} \tag{2.1}$$

Lemma 2.1. *Let $a_1, \dots, a_m, b_1, \dots, b_n \in \Sigma$ and let $W\colon \{0,\dots,m\} \times \{0,\dots,n\} \to \mathbb{Z}$ satisfy, for all $r, s \ge 1$, the recurrence (2.1) with W, a_r, b_s in place of $L, A[i], B[j]$. If $W(r,0) - W(r-1,0) \in \{0,1\}$ and $W(0,s) - W(0,s-1) \in \{0,1\}$ for all $r, s \ge 1$, then $W(r,s) - W(r-1,s) \in \{0,1\}$ and $W(r,s) - W(r,s-1) \in \{0,1\}$ for all $r, s \ge 1$.*

Proof. By induction on $r+s$. Put $t = W(r-1,s-1)$, $t' = W(r-1,s)$ and $t'' = W(r,s-1)$; by induction $t', t'' \in \{t, t+1\}$. If $a_r = b_s$, then $W(r,s) = t+1$ lies in $\{t', t'+1\}$ and in $\{t'', t''+1\}$. Otherwise $W(r,s) = \max\{t', t''\}$, which lies in $\{t', t'+1\}$ and in $\{t'', t''+1\}$ because $|t' - t''| \le 1$. □

In particular all increments of L lie in $\{0,1\}$, so L is nondecreasing in each argument.

2.1. **Minimal match pairs.** A pair of nonempty strings (u, w) is a *minimal match pair* if $u[|u|] = w[|w|]$ and $u[i] \ne w[j]$ for every pair of positions $(i,j) \ne (|u|, |w|)$. Its *size* is $|u| + |w|$. For $c \in \Sigma$ and $l \in \mathbb{Z}$ put

$$T_{c,l} = (\bar{c}^{\,l}c,\ c) \quad (l \ge 0), \qquad T_{c,l} = (c,\ \bar{c}^{\,|l|}c) \quad (l < 0),$$

a pair of size $|l| + 2$.

Lemma 2.2. *The minimal match pairs over Σ are exactly the pairs $T_{c,l}$, $c \in \Sigma$, $l \in \mathbb{Z}$.*

Proof. Each $T_{c,l}$ is a minimal match pair. Conversely, let (u, w) be a minimal match pair with last symbol c. If $|u|, |w| \ge 2$, then $u[1] \ne w[|w|] = c$ and $u[|u|] = c \ne w[1]$ give $u[1] = w[1] = \bar{c}$, which contradicts $u[1] \ne w[1]$. So $|u| = 1$ or $|w| = 1$, and every symbol of the other string except its last differs from c. □

2.2. **The canonical decomposition.** The *canonical traceback* starts at a cell (i,j) and, while $i, j \ge 1$, moves up to $(i-1, j)$ if $L(i-1,j) = L(i,j)$, else left to $(i, j-1)$ if $L(i,j-1) = L(i,j)$, and else diagonally to $(i-1, j-1)$. By Lemma 2.1 and (2.1), a diagonal step from (i,j) happens only if $A[i] = B[j]$ and $L(i-1,j) = L(i,j-1) = L(i-1,j-1) = L(i,j) - 1$.

Lemma 2.3 (cf. [7, Lemmas 3.4 and 3.5])**.** *Let $A, B \in \Sigma^n$ and let $(i_1, j_1), \dots, (i_m, j_m)$, with $i_1 < \cdots < i_m$, be the cells at which the canonical traceback from (n,n) takes a diagonal step; put $(i_0, j_0) = (0,0)$. Then $m = L(n,n)$ and $L(i_k, j_k) = k$ for each k. Each pair $p_k = (A[i_{k-1} + 1..i_k],\ B[j_{k-1} + 1..j_k])$ is a minimal match pair, and $A[i] \ne B[j]$ for all $i > i_m$ and $j > j_m$. Moreover $L(i_k, j_k - 1) = L(i_k - 1, j_k) = k - 1$. If $p_k = T_{c,l}$ with $l < 0$ and $k \ge 2$, then*

$$L(i_{k-1} - 1, j) < L(i_{k-1}, j) \qquad \text{for } j_{k-1} < j < j_k. \tag{2.2}$$

We call $p_1, \dots, p_m$ the *canonical decomposition* of (A, B); the remaining pair $(A[i_m + 1..n], B[j_m + 1..n])$ has no matching symbols.

Proof. The value of L is constant along up steps and left steps, and it drops by 1 at diagonal steps; the traceback ends where $L = 0$. Hence $L(i_k, j_k) = k$ and $m = L(n,n)$. The last symbols of p_k agree, since a diagonal step is taken at (i_k, j_k). Suppose $A[i] = B[j]$ for a cell (i,j) of $(i_{k-1}, i_k] \times (j_{k-1}, j_k]$ other than (i_k, j_k). Then $L(i,j) \ge L(i-1, j-1) + 1 \ge L(i_{k-1}, j_{k-1}) + 1 = k$. If $i < i_k$, then $L(i_k - 1, j_k) \ge L(i,j) \ge k$, so the traceback moves up at (i_k, j_k); if $i = i_k$ and $j < j_k$, then $L(i_k, j_k - 1) \ge k$, so it moves up or left. Both contradict the diagonal step at (i_k, j_k). An equality $A[i] = B[j]$ with $i > i_m$ and $j > j_m$ would give $L(n,n) \ge L(i_m, j_m) + 1 = m + 1$. The equalities $L(i_k, j_k - 1) = L(i_k - 1, j_k) = k - 1$ hold at every diagonal step, by the remark before Lemma 2.3.

For $k \ge 2$, between the diagonal steps at (i_k, j_k) and (i_{k-1}, j_{k-1}), the traceback moves from $(i_k - 1, j_k - 1)$ to (i_{k-1}, j_{k-1}) by up and left steps only. If $p_k = T_{c,l}$ with $l < 0$, then $i_k = i_{k-1} + 1$, so all these steps are left steps along row i_{k-1}, and $L = k - 1$ at every cell they pass. A left step at

(i_{k-1}, j) with $j \geq 1$ is taken only if the rule does not move up, that is, $L(i_{k-1}-1, j) < L(i_{k-1}, j)$, which is (2.2). □

Lemma 2.4. *For strings U, V and $c \in \Sigma$, $\mathrm{LCS}(cU, cV) = 1 + \mathrm{LCS}(U, V)$. Consequently, the following holds for $A, B \in \Sigma^n$ and an integer $h \geq 0$. Put $A^* = 0^h A$ and $B^* = 0^h B$, and let $L^*(r, s) = \mathrm{LCS}(A^*[1..r], B^*[1..s])$; then $L^*(h+i, h+j) = h + L(i, j)$ for $0 \leq i, j \leq n$. Moreover, $L^*(r, s) = \min\{r, s\}$ whenever $\min\{r, s\} \leq h$.*

Proof. Let S be a longest common subsequence of cU and cV. If S begins with c, the rest of S is a common subsequence of U and V; otherwise S itself is one. Hence $\mathrm{LCS}(cU, cV) \leq 1 + \mathrm{LCS}(U, V)$, and the reverse inequality is clear. The other statements follow by induction on h. □

By Lemma 2.4, at a cell $(h + i, h + j)$ with $i, j \geq 1$ the traceback rule makes the same comparisons in L^* as at (i, j) in L. So the canonical decomposition of (A, B) is read off from L^* in the same way, with all coordinates shifted by h. Moreover, in L^* the shifted inequality (2.2) also holds for $k = 1$, since $L^*(h-1, h+j) = h - 1 < h = L^*(h, h+j)$.

3. Lueker's automaton

We write out Lueker's automaton [7, Section 3.2 and Figure 7] in the form in which we use it. Our form differs from Figure 7 of [7] in two inessential ways: each minimal match pair is truncated by the map μ (defined below) before it is read, and we normalize the table W attached to a state by $W(h, h) = 0$ instead of Lueker's value h (states and their tables are defined below); this normalization does not affect the transitions [7, Lemma 3.21]. The start state below corresponds to prepending 0^h to both strings, as in [7, Lemma 3.8]. The symmetry reduction at the end of this section is the one used in Lueker's programs [8]. Fix $h \geq 2$. A *state* is a quadruple $q = (x, y, d_x, d_y)$ of strings in Σ^h. To q we attach a partial table W and the strings $a = x$ and $b = y$, so that $a_r = x[r]$ and $b_s = y[s]$ for $1 \leq r, s \leq h$. The table W is defined on row h and column h by

$$W(h, h) = 0, \quad W(r, h) - W(r-1, h) = d_x[r], \quad W(h, s) - W(h, s-1) = d_y[s] \quad (1 \leq r, s \leq h).$$

The *inputs* are the pairs $T_{c,l}$ with $c \in \Sigma$ and $|l| \leq h - 1$. The transition $\delta(q, T_{c,l})$ is computed as follows.

(1) Append the first string of $T_{c,l}$ to a and the second to b; let α and β be the new lengths, so that $(\alpha, \beta) = (h + l + 1, h + 1)$ if $l \geq 0$ and $(h + 1, h + |l| + 1)$ if $l < 0$.
(2) Extend row 0 beyond column h and column 0 beyond row h by constants: put $W(0, s) = W(0, h)$ for $h < s \leq \beta$ and $W(r, 0) = W(h, 0)$ for $h < r \leq \alpha$.
(3) Fill every cell (r, s) with $1 \leq r \leq \alpha$, $1 \leq s \leq \beta$ and $\max\{r, s\} > h$, in row-major order, by (2.1) with W, a_r, b_s in place of $L, A[i], B[j]$.
(4) Starting at (α, β), apply the canonical traceback rule to W until the current cell is (h, h) or has a coordinate smaller than h. The input is *accepted* if the traceback stops at (h, h), and *rejected* otherwise.
(5) If the input is accepted, the new state is (x', y', d'_x, d'_y) with $x' = a[\alpha - h + 1..\alpha]$, $y' = b[\beta - h + 1..\beta]$, $d'_x[t] = W(\alpha - h + t, \beta) - W(\alpha - h + t - 1, \beta)$ and $d'_y[t] = W(\alpha, \beta - h + t) - W(\alpha, \beta - h + t - 1)$ for $1 \leq t \leq h$.

The *start state* is $q_0 = (0^h, 0^h, 1^h, 1^h)$. Steps (3)–(5) read only cells that are defined: in step (3), because the neighbours of a cell being filled are filled cells, cells of row h or column h, or cells set in step (2); in steps (4) and (5), because these steps read only cells with $\max\{r, s\} > h$ and cells of row h or column h. The rectangle $\{h, \ldots, \alpha\} \times \{0, \ldots, h\}$ has a first row with increments $d_y[s]$ and a constant first column, so by Lemma 2.1 its increments lie in $\{0, 1\}$; then the rectangle $\{0, \ldots, \alpha\} \times \{h, \ldots, \beta\}$ has a constant first row and a first column (column h) whose increments

lie in $\{0,1\}$: up to row h they are given by d_x, and below row h by the first rectangle. So Lemma 2.1 applies again, and the increments of W on the filled region, the union of these two rectangles, lie in $\{0,1\}$. Hence the new d'_x and d'_y are bit strings. On an accepted input the traceback takes exactly one diagonal step, at (α,β). It takes at least one, because $a_\alpha = b_\beta$ and monotonicity give $W(\alpha,\beta) = W(\alpha-1,\beta-1)+1 > W(h,h)$, while up and left steps keep the value. It takes at most one. A diagonal step from a cell of row h or column h leaves the region of cells with both coordinates at least h, so the input would be rejected. A diagonal step at a filled cell (r,s) with $r,s>h$ requires $a_r = b_s$ by (2.1), and this happens only at (α,β), since $T_{c,l}$ is a minimal match pair.

For $q=(x,y,d_x,d_y)$ put $\bar q = (\bar x,\bar y,d_x,d_y)$. Complementing every symbol of a, b and of the input does not change any comparison. Hence $T_{\bar c,l}$ is rejected at $\bar q$ if and only if $T_{c,l}$ is rejected at q, and otherwise

$$\delta(\bar q, T_{\bar c,l}) = \overline{\delta(q,T_{c,l})}. \tag{3.1}$$

Let $\mathrm{rep}(q) = \bar q$ if $x[1]=1$, and let $\mathrm{rep}(q)=q$ otherwise. We call a state q *reduced* if $x[1]=0$. The *reduced automaton* has the reduced states as its states and $\delta_{\mathrm{red}} = \mathrm{rep}\circ\delta$ as its transition function; it is the automaton of Lueker's programs.

Truncation sends an arbitrary minimal match pair to an input: $\mu(T_{c,l^\circ}) = T_{c,l}$ with $l = \mathrm{sign}(l^\circ)\min\{|l^\circ|, h-1\}$. For $0<z<1$ the weight of an input is the sum of z^s over the minimal match pairs that truncate to it, where s is the size of the pair:

$$w_z(T_{c,l}) = z^{|l|+2} \quad (|l|<h-1), \qquad w_z(T_{c,l}) = \sum_{j\ge h-1} z^{j+2} = \frac{z^{h+1}}{1-z} \quad (|l|=h-1). \tag{3.2}$$

Lemma 3.1 (covering)**.** *Let $A,B\in\Sigma^n$ have the canonical decomposition $p_1,\dots,p_m$. Then the inputs $\mu(p_1),\dots,\mu(p_m)$, fed to M_h from q_0, are all accepted.*

Proof. We work with $A^* = 0^hA$, $B^* = 0^hB$ and L^* as in Lemma 2.4. Let $q_k = (x_k,y_k,d_{x,k},d_{y,k})$ be the state after $\mu(p_1),\dots,\mu(p_k)$, let W_k be its table, and let $I = h+i_k$ and $J = h+j_k$; then $L^*(I,J) = h+k$. We prove by induction on k that the inputs are accepted and that the following invariant holds:

> (Inv_k): there are an integer C and strictly increasing maps $\phi,\psi\colon\{0,\dots,h\}\to\{0,1,\dots\}$ such that $\phi(h)=I$, $\psi(h)=J$, $x_k[r]=A^*[\phi(r)]$ and $y_k[s]=B^*[\psi(s)]$ for $1\le r,s\le h$, and
>
> $$W_k(r,h)+C \le L^*(\phi(r),J), \qquad W_k(h,s)+C\le L^*(I,\psi(s)) \qquad (0\le r,s\le h),$$
>
> with equality at $r=s=h$.

For $k=0$ the table of q_0 is $W_0(r,h)=r-h$, $W_0(h,s)=s-h$, while $L^*(r,h)=r$ and $L^*(h,s)=s$ by Lemma 2.4; so (Inv_0) holds with $\phi=\psi=\mathrm{id}$ and $C=h$.

Assume (Inv_k) and let $p_{k+1} = T_{c,l^\circ}$, so that the automaton reads the input $\mu(p_{k+1}) = T_{c,l}$. Write $p_{k+1} = (u,w)$ and let $I' = I+|u|$, $J' = J+|w|$, so that $(I',J') = (h+i_{k+1}, h+j_{k+1})$ is the next cell, in the order of the decomposition, at which the traceback of L^* steps diagonally. Extend ϕ to $\{0,\dots,\alpha\}$ by sending $h+1,\dots,\alpha$ to the last $\alpha-h$ rows up to I', that is, $\phi(r) = I'-\alpha+r$ for $h<r\le\alpha$, and extend ψ in the same way. The extensions are strictly increasing, since $\phi(h) = I \le I'-(\alpha-h)$. They also preserve symbols, that is, $a_r = A^*[\phi(r)]$ and $b_s = B^*[\psi(s)]$ hold for the new r and s as well. Indeed, the input keeps the last $\alpha-h$ symbols of u and the last $\beta-h$ symbols of w, since, for $|l^\circ|>h-1$, the truncation removes symbols $\bar c$ at the left end of the longer string.

We claim that $W(r,s)+C\le L^*(\phi(r),\psi(s))$ for every cell of the filled region, where W is the table of step (3). For the cells set in step (2), the claim follows from (Inv_k) and the monotonicity

of L^*; for the cells of row h and column h, the claim is (Inv_k). For a cell filled in step (3), induct in the order of filling. If $a_r = b_s$, then $A^*[\phi(r)] = B^*[\psi(s)]$, since ϕ and ψ preserve symbols. As $\phi(r-1) \le \phi(r) - 1$ and $\psi(s-1) \le \psi(s) - 1$, we get

$$L^*(\phi(r), \psi(s)) \ge L^*(\phi(r) - 1, \psi(s) - 1) + 1 \ge L^*(\phi(r-1), \psi(s-1)) + 1$$
$$\ge W(r-1, s-1) + C + 1 = W(r, s) + C.$$

Otherwise $L^*(\phi(r), \psi(s)) \ge \max\{L^*(\phi(r-1), \psi(s)), L^*(\phi(r), \psi(s-1))\} \ge W(r,s) + C$.

Acceptance when $l \ge 0$. Then $\beta = h+1$, $J' = J+1$, and Lemma 2.3 gives $L^*(I', J) = L^*(I' - 1, J') = h + k$. Hence $W(\alpha, h) + C \le h + k = W(h, h) + C$, and since W is nondecreasing, $W(r, h) = W(h, h)$ for $h \le r \le \alpha$. Likewise $W(\alpha - 1, h + 1) + C \le L^*(I' - 1, J') = h + k = W(h, h) + C$, because $\psi(h + 1) = J'$ and $\phi(\alpha - 1) = I' - 1$ (also when $\alpha = h + 1$, since then $l^\circ = 0$). Since also $W(\alpha - 1, h+1) \ge W(\alpha - 1, h) = W(h, h)$, we get $W(\alpha - 1, h+1) = W(h, h)$. Since $a_\alpha = b_\beta$, we get $W(\alpha, \beta) = W(\alpha - 1, h) + 1 = W(h, h) + 1$, which exceeds both $W(\alpha - 1, \beta) = W(\alpha - 1, h+1)$ and $W(\alpha, \beta - 1) = W(\alpha, h)$. So the traceback steps diagonally at (α, β) and then moves up along column h, where W is constant, to (h, h): the input is accepted.

Acceptance when $l < 0$. Then $\alpha = h+1$, $I' = I+1$, and Lemma 2.3 gives $L^*(I, J') = L^*(I', J' - 1) = h + k$. As before $W(h, s) = W(h, h)$ for $h \le s \le \beta$ and $W(h + 1, \beta - 1) = W(h, h)$, so the traceback steps diagonally at (α, β) to $(h, \beta-1)$. At a cell (h, s) with $h < s < \beta$ the value $\psi(s)$ lies strictly between J and J'. Since $h+k = L^*(I, J) \le L^*(I, \psi(s)) \le L^*(I', J' - 1) = h+k$, we have $L^*(I, \psi(s)) = h + k$, and the shifted form of (2.2) for the pair p_{k+1} gives $L^*(I - 1, \psi(s)) < h + k$ (for $k = 0$, that is, for the pair p_1, this form holds by the remark after Lemma 2.4). Hence

$$W(h-1, s) + C \le L^*(\phi(h-1), \psi(s)) \le L^*(I - 1, \psi(s)) \le h + k - 1 < W(h, s) + C.$$

So the traceback does not move up. Since $W(h, s-1) = W(h, s)$, it moves left, and in this way it reaches (h, h): the input is accepted.

The invariant. Now $W(\alpha, \beta) + C = W(h, h) + 1 + C = h + k + 1 = L^*(I', J')$, and the cells that define q_{k+1} are filled cells, for which the claim above gives the inequalities of (Inv_{k+1}). The table of q_{k+1} is $W_{k+1}(r, h) = W(\alpha - h + r, \beta) - W(\alpha, \beta)$ and $W_{k+1}(h, s) = W(\alpha, \beta - h + s) - W(\alpha, \beta)$, where $W(\alpha, \beta) = W(h, h) + 1 = 1$. So ($\mathrm{Inv}_{k+1}$) holds with the maps $r \mapsto \phi(\alpha - h + r)$, $s \mapsto \psi(\beta - h + s)$ and the constant $C + 1$. □

4. The counting bound

Theorem 4.1 (Dančík–Paterson; [7, Theorem 3.3 and (41)])**.** *Let $0 < z < 1/2$ and let S be a set of states of M_h that contains q_0 and is closed under the transitions of M_h, that is, $\delta(q, p) \in S$ for every $q \in S$ and every input p accepted at q. If $v\colon S \to (0, \infty)$ and $0 < \lambda < 1$ satisfy*

$$\sum_{p \text{ accepted at } q} w_z(p)\, v(\delta(q, p)) \le \lambda\, v(q) \qquad (q \in S), \tag{4.1}$$

then $\gamma_2 \le 2 \ln(2z) / \ln \lambda$.

Proof. Fix n and k. A pair $(A, B) \in \Sigma^n \times \Sigma^n$ with $L(n, n) \ge k$ is determined by the first k pairs $p_1, \dots, p_k$ of its canonical decomposition and by the symbols of A and B after these k pairs. By Lemma 3.1 the sequence $\mu(p_1), \dots, \mu(p_k)$ is accepted from q_0. The total size s of $p_1, \dots, p_k$ is at most $2n$, and the pair (A, B) has probability 2^{-2n}, so

$$\Pr(L_n \ge k) \le \sum_{p_1, \dots, p_k} 2^{-s} \le (2z)^{-2n} \sum_{p_1, \dots, p_k} z^s,$$

where the sums run over the sequences of minimal match pairs of total size $s \le 2n$ whose truncations are accepted from q_0; the first inequality counts the 2^{2n-s} choices of the remaining symbols, and the second uses $2z < 1$. If we drop the condition $s \le 2n$ and group the sequences

by their truncations, then (3.2) shows that the last sum is at most $e_{q_0}^{\mathsf{T}} A_z^k \mathbf{1}$, where e_{q_0} is the unit vector at q_0, the vector $\mathbf{1}$ has all entries 1, and A_z is the $S \times S$ matrix whose entry $A_z(q, q')$ is the total weight of the inputs that take q to q'. By (4.1), which says that $A_z v \le \lambda v$, we have $A_z^k v \le \lambda^k v$, so $e_{q_0}^{\mathsf{T}} A_z^k \mathbf{1} \le \lambda^k v(q_0)/\min_S v$ (S is finite, since there are 2^{4h} states). Let $\theta > 2\ln(2z)/\ln\lambda$ and $k = \lceil \theta n \rceil$. Then

$$\Pr(L_n \ge \theta n) \le \frac{v(q_0)}{\min v} \exp\bigl(n(-2\ln(2z) + \theta \ln\lambda)\bigr),$$

which decays geometrically in n. Since $L_n \le n$, we have $\mathbb{E}L_n \le \theta n + n \Pr(L_n \ge \theta n)$, so $\gamma_2 \le \theta$. Let θ decrease to $2\ln(2z)/\ln\lambda$. □

By the Collatz–Wielandt bound, a positive vector v with $A_z v \le \lambda v$ shows that the Perron root of A_z is at most λ; the proof above uses the inequality $A_z v \le \lambda v$ only through the iterated inequalities $A_z^k v \le \lambda^k v$.

The certificates are stated for the reduced automaton; the next lemma transfers them to M_h with the same λ.

Lemma 4.2. *Let S_{red} be a set of reduced states that contains q_0 and satisfies $\delta_{\mathrm{red}}(q, p) \in S_{\mathrm{red}}$ for every $q \in S_{\mathrm{red}}$ and every input p accepted at q. Let z and λ be as in Theorem 4.1, and let $v\colon S_{\mathrm{red}} \to (0, \infty)$ satisfy (4.1) with δ_{red} in place of δ. Then $S = S_{\mathrm{red}} \cup \{\bar{q} : q \in S_{\mathrm{red}}\}$, with v extended by $v(\bar{q}) = v(q)$, satisfies the hypotheses of Theorem 4.1 with the same λ.*

Proof. The extension of v is well defined because $\bar{q}$ is not reduced when q is; it satisfies $v(q') = v(\mathrm{rep}(q'))$ for all $q' \in S$. For $q \in S_{\mathrm{red}}$ and an accepted input p, the states $\delta(q, p)$ and $\delta_{\mathrm{red}}(q, p) = \mathrm{rep}(\delta(q, p))$ differ at most by complementation, and $\delta_{\mathrm{red}}(q, p) \in S_{\mathrm{red}}$. So $\delta(q, p) \in S$, and the left side of (4.1) at q is the same for δ and δ_{red}. At $\bar{q}$, (3.1) gives $\delta(\bar{q}, T_{c,l}) = \overline{\delta(q, T_{\bar{c},l})} \in S$, and since $w_z(T_{c,l})$ does not depend on c,

$$\sum_{c,l} w_z(T_{c,l})\, v(\delta(\bar{q}, T_{c,l})) = \sum_{c,l} w_z(T_{\bar{c},l})\, v(\delta_{\mathrm{red}}(q, T_{\bar{c},l})) \le \lambda v(q) = \lambda v(\bar{q}),$$

where the left sum runs over the inputs accepted at $\bar{q}$ and the right sum over the inputs $T_{\bar{c},l}$ accepted at q. □

5. Certificates and computations

5.1. **Certificates.** A certificate for history length h consists of a rational $z \in (0, 1/2)$ given as a decimal, a sorted list of states, and a vector $v > 0$ on the list. The entries of v are stored as double-precision floating-point numbers (single-precision for $h = 13$), and we read them as exact dyadic rationals. Each state is stored as the integer $x \cdot 2^{3h} + y \cdot 2^{2h} + d_x \cdot 2^h + d_y$, where each bit string is read as a binary number with its first symbol as the most significant bit. The checker computes an integer Λ such that (4.1) holds with $\lambda = \Lambda/2^{64}$.

5.2. **The checker.** The checker rebuilds the transitions with its own implementation of steps (1)–(5) and of rep. This implementation is a direct dynamic programming computation of the table W of steps (1)–(5); it marks which cells are defined and stops if a cell read in steps (3) or (4) is undefined (step (5) reads only cells filled in step (3)). It also asserts two facts: all increments lie in $\{0, 1\}$, and an accepted input causes exactly one diagonal step. The checker also checks that the list is strictly increasing and consists of integers below 2^{4h}, that the list contains q_0, that $\delta_{\mathrm{red}}(q, p)$ is listed for every listed state q and every input p accepted at q, and that every entry of v is a positive normal floating-point number. For (4.1) the checker uses the integer weights $\hat{w}_j = \lceil 2^{64} w_z(T_{c,l}) \rceil$, where $j = |l|$. A separate script computes these weights from the exact rational z, and the same z is used for the final bound. The checker writes each

entry of v exactly as $m \cdot 2^e$ with integers m and e. For every state q it then evaluates, in 128-bit integer arithmetic, the sum of the terms $\hat{w}_{|l|}\, v(\delta_{\mathrm{red}}(q, T_{c,l}))$ over the accepted inputs, with each term scaled to the exponent of $v(q)$; terms that are shifted to the right are rounded up. Write $v(q) = m_q \cdot 2^{e_q}$ and let σ_q be the scaled sum at q; the checker reports $\Lambda = \max_q \Lambda_q$, where $\Lambda_q = \lceil \sigma_q / m_q \rceil$. If the sum at some state q (the row of q in $A_z v \le \lambda v$) overflows, the checker reports $\Lambda = 2^{64} - 1$, which gives no useful bound; otherwise $A_z v \le (\Lambda/2^{64}) v$ holds exactly, and no rounding occurs except the upward roundings described. Finally $2\ln(2z)/\ln(\Lambda/2^{64})$ is evaluated in interval arithmetic and rounded up to six decimals.

A second checker, written independently from the same definitions, regenerated the state sets for $h = 9$, 10, 11 and 12 by its own breadth-first search and verified these four certificates; in each case the smallest Λ for which its check passes equals the value reported by the first checker. The second checker also accepted the certificate for $h = 8$.

The certificate for $h = 13$ was checked by a third implementation of the same checks, written for the size of this certificate. The third implementation computes the transitions by an incremental version of the dynamic programming computation and evaluates (4.1) in the same exact arithmetic. Before the third implementation was used, its transition function was compared with that of the checker on every reachable state and every input for $h \le 10$; the README of the archive (see Data availability) describes this comparison and further tests of the checking programs, among them tests with altered certificates. The third implementation also reported the same Λ as the checker on the certificates for $h = 8$, 9 and 10. For $h = 13$ the numbers of accepted transitions counted by the third implementation agree with those found by the breadth-first search of Section 5.3, which uses a different transition function. The Lean proof of Section 6 checks the inequality (4.1) at every state again.

5.3. **Search.** The vectors v come from a power iteration. Any positive vector that passes the check gives a valid certificate; the iteration only serves to make Λ small. For $h = 12$ the certificate comes from a deterministic version of the iteration, whose output does not depend on the processor or the number of threads. For $h = 13$, the certificate comes from a version of the iteration distributed over four machines, with the iterates stored in single precision. This iteration starts from the vector for $h = 12$: each state for $h = 13$ gets the value of that vector at the state for $h = 12$ obtained by deleting the oldest symbol of each of x, y, d_x, d_y and reducing by the symmetry, or, if there is no such state, the smallest entry of that vector. The state sets are found by breadth-first search from q_0. The search programs for $h = 12$ and $h = 13$ use a faster, bit-parallel transition function (Section 5.4); none of the checking programs uses any code of the search programs. The README describes all programs and runs of Sections 5 and 6, with the machines and the running times.

5.4. **Validation of the automaton.** For $h = 2$ our programs reproduce the transition table of the 7-state automaton printed in the description of Lueker's programs [8]. For $h = 2, \ldots, 8$ the numbers of reachable reduced states are 7; 98; 1,020; 10,160; 89,960; 780,368; and 6,419,109, as in [7, Table II]. For $h = 9$ and $h = 10$ they are 51,425,006 and 400,253,317, as in [6]. Our transition tables for $h = 9$ and $h = 10$ agree with those of [6]. For $h \le 10$, the transition function of the checker agrees, on every input at every reachable state, with both the direct and the bit-parallel transition functions of the search programs.

5.5. **Results.** Table 1 summarizes the certificates. The checking programs do not verify that the listed states are reduced, and this is not needed: the set R of states reachable from q_0 under δ_{red} consists of reduced states, lies in the list by the check that $\delta_{\mathrm{red}}(q, p)$ is listed (Section 5.2), and is closed under δ_{red}; moreover, the inequality (4.1) holds at each state of R. Theorem 4.1,

together with Lemma 4.2 applied to R and the restriction of v, gives the bound in each row of Table 1. Theorem 1.1 follows from the row $h = 13$.

TABLE 1. Certified upper bounds on γ_2 from Lueker's automaton with history length h. The numbers of reachable states and accepted transitions refer to the reduced automaton. The column headed "$\lambda \le$" gives $\lambda = \Lambda/2^{64}$ rounded up to 12 decimals. The next column gives $2\ln(2z)/\ln\lambda$, computed with the exact λ and truncated to 10 decimals; the last column gives the same quantity rounded up to six decimals.

h	states	accepted transitions	z	$\lambda \le$	$2\ln(2z)/\ln\lambda$	$\gamma_2 \le$
8	6,419,109	91,508,397	0.3209	0.341832258719	0.8262791785	0.826280
9	51,425,006	790,078,170	0.3298	0.364708361170	0.8251002729	0.825101
10	400,253,317	6,574,257,580	0.3376	0.385532597280	0.8241195406	0.824120
11	3,045,043,879	53,133,194,463	0.3445	0.404566031012	0.8232896731	0.823290
12	22,670,087,101	418,139,991,128	0.3505	0.421581925965	0.8225783634	0.822579
13	165,695,219,469	3,217,333,414,315	0.3568	0.439973329782	0.8219629194	0.821963

5.6. **The choice of z.** For each $h \le 12$ the parameter z was chosen near the value that minimizes the bound. For $h = 11$, power iterations at $z = 0.3425$, 0.3445 and 0.3465 indicate that the optimal z lowers the bound by less than 10^{-7}, which does not change the rounded value. For $h = 12$ we used only $z = 0.3505$. For an iterate v of the power iteration, the smallest ratio $(A_z v)(q)/v(q)$ is at most the Perron root of A_z, by the Collatz–Wielandt bound; a floating-point evaluation of this ratio indicates that no positive vector gives a bound below 0.822578 at this z. For $h = 13$ we used only $z = 0.3568$; the same evaluation for the 99th iterate, the iterate just before the one used in the certificate, indicates that no positive vector gives a bound below 0.821958 at this z.

6. Formal verification

The formal proofs use the Lean 4 proof assistant [10] with the Mathlib library [9]. The formal statements define three objects: $\mathrm{LCS}(A, B)$, for lists A and B of bits, as the largest length of a list that is a subsequence of both; $\mathbb{E}L_n$ as the average of $\mathrm{LCS}(A, B)$ over the 4^n pairs of strings of length n; and γ_2 as the limit of $\mathbb{E}L_n/n$. The formalization proves that this limit exists, using the inequality $\mathbb{E}L_{m+n} \ge \mathbb{E}L_m + \mathbb{E}L_n$ and Fekete's lemma. These three definitions are the only ones that occur in the statements of the theorems below; the automaton and the checking programs are auxiliary definitions, and everything needed about them is proved. The automaton M_h, the reduced transition function and the weights are defined in Lean as in Sections 3 and 4. The covering property (Lemma 3.1) is proved for every $h \ge 2$, in the form used in the proof of Theorem 4.1. Theorem 4.1 is proved, together with Lemma 4.2, in the following form, which avoids logarithms: if (i) $h \ge 2$, $0 < z < 1/2$, $\lambda > 0$ and $\theta > 0$, (ii) S is a set of states that contains q_0 and is closed under δ_{red}, (iii) $v > 0$ on S, (iv) the inequality (4.1) with δ_{red} in place of δ holds at every $q \in S$, and (v) $\lambda^\theta < (2z)^2$, then $\gamma_2 \le \theta$. A checking program written in Lean verifies a certificate, and Lean proofs show that every certificate accepted by this program yields data that satisfy these hypotheses; the transition function of the program is a fast implementation that is proved to agree with the definition of M_h.

Theorem 1.1 is proved in Lean by the theorem `K13TopH13.gamma2_le`, and the bounds for $h = 9, \ldots, 12$ in Table 1 are proved by four further theorems. All five are in Lean files written by generator programs, and these programs are archived (see Data availability). These proofs

use no `native_decide`, and their trusted base is Lean's kernel, including the kernel's built-in arithmetic on natural numbers. Each certificate is split into parts, and the kernel checks each part with the tactic `decide +kernel`. Compiled code runs only while the data modules, which hold the certificate, are compiled; at that stage, compiled code builds the Lean terms of these modules from the certificate files or from pieces cut from these files. For $h \geq 11$ the checking program packs up to 64 consecutive states of the list into one natural number and processes them together, using bit operations. Lean proofs show that every certificate accepted by this packed version of the program also yields data that satisfy the hypotheses above. For $h = 13$ the checking program looks up, in the data modules, each entry that it needs, and the Lean proofs assume nothing about these entries, not even their order or the uniqueness of their keys, so the proof does not depend on how the pieces were cut.

Each of the five theorems for $h = 9, \ldots, 13$ depends only on the axioms `propext`, `Classical.choice` and `Quot.sound`. For Theorem 1.1 this statement about axioms rests on two things: the tables of axioms that are stored in the compiled module files and read by Lean's `#print axioms`, and a scan of the proof terms. The scan covered all modules of this proof except 4,414 of its 4,478 data modules.[1]

Proofs that evaluate the checks with `native_decide`, which executes compiled code, also exist for $h = 12$ and $h = 13$; their trusted base includes the Lean compiler, through one auxiliary axiom for each compiled check.

7. Remarks

Lueker's proof of convergence [7, Theorems 3.13 and 3.26] also gives a rate: for history length h the bounds can be made at most $\gamma_2 + O((\log h/h)^{1/3})$. The improvements in Table 1 decrease slowly: from $h = 8$ to $h = 9$, the bound improves by 0.001179, and from $h = 12$ to $h = 13$ by 0.000616. Meanwhile the number of states grows by a factor that decreases from 8.0 (from $h = 8$ to $h = 9$) to 7.3 (from $h = 12$ to $h = 13$). Theorem 1.1 leaves open whether $\gamma_2 < \gamma_P$ for the word P of Bukh and Cox (Section 1).

Data availability

The checking programs, the search programs, the weights, the Lean files of Section 6 and the logs of the checks are archived at doi:10.5281/zenodo.24414180 and available at `https://github.com/tejasvi/chvatal-sankoff-certificates/tree/v6`. For the proofs that use no `native_decide`, the archive holds the programs that generate their Lean files and the SHA-256 checksums of these files, not the generated files themselves. The Lean files use Lean 4.35.0-rc3 and Mathlib at its tag `v4.35.0-rc3`, as pinned in `lean/lake-manifest.json`. The archive also contains the certificates for $h = 8$, 9, 10 and 11, compressed losslessly to 28.3 GB.

The certificate for $h = 12$ consists of two files of 181 GB each and is not archived; the archived programs regenerate it. A breadth-first search from the start state gives the list of states; its SHA-256 checksum begins with `bef385e5`. Then 240 iterations of the deterministic power iteration with $z = 0.3505$ give the vector; its checksum begins with `915ac393`. For these two files the checker reports $\Lambda = 7776813894375449830$, and any vector for which the checker reports $\Lambda \leq 7776819092424481736$ gives the same bound 0.822579. The README of the archive gives the commands, the full checksums and the computing resources needed; regenerating and

[1]The scan found no reference to `sorryAx`, `Lean.ofReduceBool`, `Lean.ofReduceNat` or `Lean.trustCompiler`, and no scanned constant is an axiom or an `opaque`, `implemented_by`, `extern` or `unsafe` declaration. For the 4,414 unscanned modules the statement about axioms rests on the stored tables alone; the kernel checked these modules again, and the compiled files of these modules match the checksums recorded when the files were built.

checking the certificate takes less than two and a half hours on a machine with 288 virtual cores and 2.2 TB of memory.

The certificate for $h = 13$ is not archived either, because of its size: the list of states takes 1.33 TB, and the vector, stored in single precision, takes 663 GB. The breadth-first search regenerates the list. Its SHA-256 checksum begins with `8454e843`. Its 64-bit fingerprint, as printed by the checking program used for $h = 13$ (the third implementation in Section 5), begins with `9e2d85eb`. Any positive vector accepted by this program gives a certificate. The checked vector has a checksum beginning with `b5456a64`; it comes from the distributed power iteration of Section 5.3 with $z = 0.3568$, started from the vector for $h = 12$. For the checked vector the program used for $h = 13$, run on row ranges that cover all states, reports $\Lambda = 8116075413730513074$, and any vector for which it reports $\Lambda \le 8116076067129428569$ gives the same bound 0.821963. The README of the archive gives the commands and the full checksums; the breadth-first search ran on a machine with 160 virtual cores and 3.8 TB of memory, and so did each of the four parts of the iteration and each of the four parts of the check.

For the Lean proof of Theorem 1.1 the archive also contains its hand-written Lean files; the programs that cut the certificate into pieces and generate the other 2,921,665 source files; the SHA-256 checksums of all generated source files and of the compiled modules that hold the certificate data; and the records of the scan described in Section 6. The generated files are not archived: once compiled, the 4,478 data modules of this proof take 3.50 TB on their own. The README gives the commands that regenerate these files and the resources needed to do so.

Acknowledgements

Generative AI tools were used to write code, run analyses and draft text; the author directed the work and takes full responsibility for its content.

References

[1] B. Bukh and C. Cox, *Periodic words, common subsequences and frogs*, Ann. Appl. Probab. **32** (2022), 1295–1332.

[2] V. Chvátal and D. Sankoff, *Longest common subsequences of two random sequences*, J. Appl. Probab. **12** (1975), 306–315.

[3] V. Dančík, *Expected length of longest common subsequences*, PhD thesis, University of Warwick, 1994.

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

[5] D. Davis, P. Ivanisvili, T. Tao and contributors, *Optimization constants in mathematics*, GitHub repository, `https://github.com/teorth/optimizationproblems`, page `constants/31a.md` at commit `7d762bbb`, 2026.

[6] L. Jack, *An improved upper bound on the binary Chvátal–Sankoff constant*, GitHub repository `https://github.com/leandrejack-afk/chvatal-sankoff-bound`, commit `472c18de`, 2026.

[7] G. S. Lueker, *Improved bounds on the average length of longest common subsequences*, J. ACM **56** (2009), no. 3, Article 17.

[8] G. S. Lueker, *Overview of the code*, with the Java programs for the bounds of [7], `https://www.ics.uci.edu/~lueker/papers/lcs/code/`.

[9] The mathlib Community, *The Lean mathematical library*, in: Proc. 9th ACM SIGPLAN Int. Conf. Certified Programs and Proofs (CPP 2020), 2020, 367–381.

[10] L. de Moura and S. Ullrich, *The Lean 4 theorem prover and programming language*, in: Automated Deduction – CADE 28, Lecture Notes in Comput. Sci. **12699**, Springer, 2021, 625–635.

[11] D. Soiffer, A. Salls, C. Miller, D. Reichman, G. Sárközy and G. Heineman, *Improved lower bounds on the expected length of longest common subsequences*, in: Proc. 2025 IEEE International Symposium on Information Theory (ISIT), 2025, 1–6; arXiv:2407.10925.

ORCID: `https://orcid.org/0000-0003-4668-3639`