Theoretical Computer Science 409 (2008) 394–404

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Algorithms for subsequence combinatorics

Cees Elzinga[a,*], Sven Rahmann[b], Hui Wang[c]

[a] *Department of Social Science Research Methods, VU University Amsterdam, The Netherlands*

[b] *Bioinformatics for High-throughput Technologies, Computer Science 11, TU Dortmund, Germany*

[c] *School of Computing and Mathematics, University of Ulster, Northern Ireland, UK*

ARTICLE INFO

Article history:
Received 10 November 2007
Received in revised form 30 May 2008
Accepted 25 August 2008
Communicated by A. Apostolico

Keywords:
Subsequence
Combinatorics
Dynamic programming
Algorithm

ABSTRACT

A subsequence is obtained from a string by deleting any number of characters; thus in contrast to a substring, a subsequence is not necessarily a contiguous part of the string. Counting subsequences under various constraints has become relevant to biological sequence analysis, to machine learning, to coding theory, to the analysis of categorical time series in the social sciences, and to the theory of word complexity. We present theorems that lead to efficient dynamic programming algorithms to count (1) distinct subsequences in a string, (2) distinct common subsequences of two strings, (3) matching joint embeddings in two strings, (4) distinct subsequences with a given minimum span, and (5) sequences generated by a string allowing characters to come in runs of a length that is bounded from above.

1. Introduction

Over the last decades, combinatorial theory on sequences of symbols has found wide application, most notably in computational biology and information sciences, but also in physics and in the social sciences. Usually, the symbols of these sequences are called *characters* or *letters*, the set of all characters is called an *alphabet* and the sequences themselves are called *strings* or *words*. For an overview on word combinatorics, the reader is referred to [23,24].

The terminology and notation concerning parts of strings have not yet arrived at a commonly accepted standard. Therefore, we start by explicitly defining two basic concepts, *substring* and *subsequence*.

A *substring* is a contiguous (possibly empty) part of a string, i.e., if $s = s_1 \dots s_n$ denotes a string, then for any $1 \leq i \leq j \leq n$, $s_i \dots s_j$ is a substring, and so is the empty string. In other words, a substring is constructed by deleting a prefix and a suffix from the original string. Often, substrings are also called *factors* or *subwords* in the literature (e.g. [26,7]), but unfortunately they are sometimes also referred to as subsequences. Substrings play an important role in the vast literature on approximate string matching (e.g. [14]).

Subsequences are constructed by deleting characters anywhere in the given string. So, characters that are adjacent in the remaining subsequence, are not necessarily adjacent in the original string. Subsequences are unfortunately also called subwords or *scattered* subwords (e.g. [27]).

A string of n characters has $\binom{n+1}{2} + 1 = \Theta(n^2)$ substrings but $\sum_k \binom{n}{k} = 2^n$ subsequences. Therefore, we may expect that counting subsequences is much more involved than counting substrings. However, it will appear that the subsequence problems can be solved by dynamic programming algorithms efficiently. Surprisingly, the problems of counting and enumerating subsequences and variants thereof have received little attention so far (but see [5,26,18,19,12]).

* Corresponding address: Department of Social Science Research Methods, Faculty of Social Sciences, VU University, De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands. Tel.: +31 20 5986889.
E-mail addresses: ch.elzinga@fsw.vu.nl (C. Elzinga), sven.rahmann@tu-dortmund.de (S. Rahmann), h.wang@ulster.ac.uk (H. Wang).

0304-3975/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.08.035

The number of distinct subsequences of a string can be used to quantify "complexity" of strings: the more distinct subsequences, the more complex (see [2,11]). Identifying fixed-length strings that maximize the number of distinct subsequences is useful to normalize such complexity measures. Furthermore, in coding theory, such "maximizing" strings and (bounds on) their numbers of distinct subsequences are central to the problem of reconstructing a binary string from a subset of its subsequences, i.e. the problem of correcting insertions and deletions (see [20,21,16,22,28]). In [28], an expression is provided for the number of subsequences that a binary sequence will have after two deletions.

In modern life sciences, DNA microarrays have become an indispensable tool to monitor the activity of thousands of genes simultaneously in a certain cell or tissue type. They consist of billions of single stranded DNA molecules of different types. During the manufacturing process, they are synthesized as subsequences of a common supersequence of a certain structure and length [25]. Here the question arises, how many distinct sequences of a given length can be constructed from the supersequence, and which supersequence maximizes the number of distinct subsequences. We answer these questions in Section 3.

Different strings may have certain substrings or subsequences in common and such common objects and their properties are important in pattern matching, especially longest common substrings and longest common subsequences [15,14,3,4]. Applications arise in the comparison of categorical time series [10] and the computation of string kernels in kernel-based machine learning methods, where the number of common subsequences is used to define proximity or similarity between strings [29]. In Section 4, we present a theorem that enables us to efficiently count common subsequences with a length of at least $k \geq 0$, generalizing results published in [5,25,29,9]. In order to take into account the frequency of subsequences in strings, we further consider the problem of counting matching embeddings, i.e., the frequency weighted common subsequences. This is relevant for time series in which the repetition of certain patterns is important, as for example in job careers. A theorem is presented, that implies an algorithm that counts such matches much faster than a previous algorithm proposed in [8].

In Section 5, we introduce the constraint that the subsequences we consider must span a certain range in the original sequence. This has again applications to the analysis of categorical time series in social sciences, and also allows us to introduce different notions of word complexity.

Finally, we consider the problem of counting sequences that are generated by length restricted runs. This is a generalization of the subsequence counting problem, and has an interesting application in high-throughput DNA sequencing: Some modern DNA sequencing technologies allow that a whole run of a nucleotide is sequenced at once. A run refers to a contiguous repetition of a single letter. For example, the string AACCCTA consists of four runs (A,C,T,A) of lengths (2, 3, 1, 1), respectively. The sequencing machine tests in turn for each nucleotide (A, C, G, T), if a run of this type is available to be sequenced. A run can be reliably sequenced if it is not too long (e.g., at most 4 or 5 characters). This leads to the question of how many distinct sequences of a given length can be sequenced in, say 100 turns. The theorems of Section 6 give efficiently computable answers to such questions.

2. Preliminaries

In this section, we define the key concepts formally and introduce the necessary notation. Refinements and special cases of these concepts will be introduced when appropriate.

Let $\Sigma = \{\sigma_1, \ldots, \sigma_d\}$ be a finite *alphabet*, and let $\Sigma^\star$ denote the set of finite *strings* that are constructed from the characters of Σ by concatenation. We say that a string $x = x_1 \ldots x_n$ has *length* $|x| = n$ or that x is *n-long* if it consists of n, not necessarily distinct, characters from Σ. The *empty string* or *empty sequence*, which has a length of zero, is denoted by λ. The set Σ^n denotes the set of all *n-long strings* over Σ. If a string x is n-long, it has n nonempty *prefixes* $x^i = x_1 \ldots x_i$ (in particular, $x^n = x$), and the empty prefix $x^0 = \lambda$.

A string y is a *substring* of another string x if there exist not necessarily distinct and possibly empty strings $v_1, v_2 \in \Sigma^\star$ such that $v_1 y v_2 = x$.

A k-long string $y = y_1 \ldots y_k$ is a *subsequence* of x if there exist $k+1$, not necessarily distinct and possibly empty, strings $v_1, \ldots, v_{k+1} \in \Sigma^\star$, such that $v_1 y_1 \ldots v_k y_k v_{k+1} = x$ and we write $y \preceq x$ to denote this fact. Clearly, if $y = x_{i_1} \ldots x_{i_k}$ is a substring of x, then $i_{j+1} - i_j = 1$ for all j, i.e., the characters that are adjacent in y are adjacent in x, too. This is not required if y is a subsequence. The set of all subsequences of x is denoted by $\mathcal{S}(x)$. If $u \preceq x$ and $u \preceq y$, we write $u \preceq (x, y)$ and we say that u is a common subsequence of x and y and we will write $\mathcal{S}(x, y)$ to denote the set of all common subsequences of x and y.

A subsequence $u \preceq x$ may be *embedded* in x more than once. For example $u = ab$ is embedded three times in $x = abab$. We will write $|x|_u = r$ to denote the fact that $u \preceq x$ has r distinct embeddings in x. We formally define an *embedding* $i_u(x)$ of $u \preceq x$ as a sequence of indices $1 \leq i_1 < \cdots < i_{|u|} \leq |x|$, such that $u_j = x_{i_j}$ for $1 \leq j \leq |u|$ and we write $I_u(x)$ for the set of embeddings of u in x. So, $|I_u(x)| = |x|_u$. We say that an embedding $\hat{\imath}_u(x) = \hat{\imath}_1 \ldots \hat{\imath}_{|u|}$ is *canonical* (sometimes also called *left-most*) if each of its indices is as small as possible: for each $i_u(x) \in I_u(x), \hat{\imath}_j \leq i_j$ for $1 \leq j \leq |u|$. The *right-most* embedding is defined similarly. Each $u \preceq x$ has exactly one canonical embedding. Hence, setting $\hat{\imath}_\lambda = 0$, there exists a bijective mapping of the set of canonical embeddings $\hat{I}(x) = \{\hat{\imath}_u(x) : u \in \mathcal{S}(x)\}$ to the set of subsequences $\mathcal{S}(x)$.

If $u \preceq (x, y)$, we say that the pair $(\hat{\imath}_u(x), \hat{\imath}_u(y))$ is the *joint canonical embedding* of u in x and y. There exists a set of such pairs $\hat{I}(x, y)$ with $\left|\hat{I}(x, y)\right| = |\mathcal{S}(x, y)|$.

The canonical embedding can be considered as resulting from an algorithm (e.g. [6]) that verifies whether or not $u \preceq x$: find the index i_1 of the first appearance of u_1 in x, then find the index of the first appearance of u_2 after that, etc. and $u \preceq x$ if the first occurrence of $u_{|u|}$ occurs at or before x_n. If the algorithm fails, $u \npreceq x$. The next lemma provides a straightforward (not the most efficient) algorithm to determine $|x|_y$ and $\hat{\imath}_y(x)$.

Lemma 1. *Let $x, y \in \Sigma^\star$ with $y \preceq x$ and $\hat{\imath}_y(x) = \hat{\imath}_1 \dots \hat{\imath}_{|y|}$. Let $E = \left(e_{ij}\right)$ be a $(|y| \times |x|)$-matrix with $e_{ij} = 1$ if and only if $y_i = x_j$ and $e_{ij} = 0$ otherwise. Furthermore, let $F = \left(f_{ij}\right)$ be a $(|y| \times |x|)$-matrix with $f_{1j} = e_{1j}$ and $f_{ij} = e_{ij} \cdot \sum_{k<j} f_{i-1,k}$ for $1 \le j \le |x|$ and $2 \le i \le |y|$. Then*

$$|x|_{y^i} = \sum_j f_{ij},$$

$$\hat{\imath}_j = \min\left\{k : f_{jk} > 0\right\}.$$

Proof. The proof is straightforward by induction, once one realizes that f_{ij} equals the number of embeddings of y^i in x^j that end at position j in x. □

We conclude this section with a note on the complexity of the algorithms put forward in this paper. Since the number of subsequences may grow exponentially with the length of the input string, storing this number requires $\Theta(n)$ bits for an n-character text, so it is inappropriate to assume that arithmetic operations can be done in constant time. For example, additions take $\Theta(n)$ time and multiplications $\Theta(n \log n)$ time. We have, however, measured the running time of the algorithms in terms of arithmetic operations. In some applications, it is reasonable to assume a constant-size alphabet $|\Sigma| = \Theta(1)$; in others, however, we may need to allow that $|\Sigma| = \Theta(n)$ in order to be realistic. Where necessary, we include $|\Sigma|$ as a parameter in the complexity analysis.

3. Counting distinct subsequences

In this section, we present an efficient solution to the problem of counting all distinct subsequences of length $k \ge 0$ of a given string, and study strings that have maximally many distinct subsequences. We begin with the simple problem of counting all distinct subsequences of a given string, since this allows us to introduce a basic proof technique in a simple, straightforward manner.

The number of distinct subsequences of an n-long string x can be computed with $\Theta(n)$ arithmetic operations as follows: when we elongate the prefix x^{n-1} with a character x_n that does not yet occur in x^{n-1} (we say that x_n is *new to* x^{n-1}), the number of distinct subsequences doubles: we retain all the subsequences already counted for x^{n-1}, and we further generate new subsequences by elongating all of these subsequences with x_n. If x_n is *not* new to x^{n-1}, we must compensate the doubling by subtracting all the elongated subsequences that were really new when $x_\ell = x_n$ was used to elongate some previous, shorter prefix $x^{\ell-1}$ with $\ell < n$. We repeat this procedure, starting with the first prefix x^1, which elongates λ, elongating it with x_2, etc. This reasoning is formalized in the next lemma.

Lemma 2 ([9]). *Let $x \in \Sigma^n$ be a nonempty string and let $\phi(x) := |\mathcal{S}(x)|$ denote the number of distinct subsequences of x. Furthermore, let $\ell(x, \sigma)$ denote the last position of the character $\sigma \in \Sigma$ in x: $\ell(x, \sigma) := \max\{i : x_i = \sigma\}$ if $\sigma \preceq x$ and $\ell(x, \sigma) := 0$ otherwise; for brevity we write $\ell := \ell(x^{n-1}, x_n)$. Then*

$$\phi(x) = \begin{cases} 2\phi(x^{n-1}) & \text{if } x_n \npreceq x^{n-1}, \\ 2\phi(x^{n-1}) - \phi(x^{\ell-1}) & \text{if } x_n \preceq x^{n-1}. \end{cases} \tag{1}$$

Proof. We observe that the set of canonical embeddings can be written as

$$\hat{I}(x) = \hat{I}(x^{n-1}) \cup \left\{(\hat{\imath}_u(x^{n-1}), n) : \hat{\imath}_u(x^{n-1}) \in \hat{I}(x^{n-1}) \setminus \hat{I}(x^{\ell-1})\right\}.$$

The two sets are disjoint, since the sequences of the first one never end on n, while the sequences in the second one always do. So,

$$\left|\hat{I}(x)\right| = 2\left|\hat{I}(x^{n-1})\right| - \left|\hat{I}(x^{\ell-1})\right|.$$

Using the existence of a bijective map from $\mathcal{S}(x)$ to $\hat{I}(x)$ and the fact that $\left|\hat{I}(x^{\ell-1})\right| = 0$ when $x_n \npreceq x^{n-1}$ then yields the required result (1). □

Lemma 2 implies a simple dynamic programming algorithm with $\Theta(|x|)$ arithmetic operations that specifies how to count subsequences, irrespective of their length.

However, in some applications (e.g., [25,9]) it is necessary to count subsequences that have a length of precisely k. The next Lemma is a specialization of Lemma 2 in the sense that it is about subsequences of a particular length k. It implies an algorithm that, given a string of length $|x|$, uses $\Theta(k\,|x|)$ arithmetic operations.

Lemma 3 (*[5,9]*). *Let $x \in \Sigma^n$ be a nonempty string, $\mathcal{S}(x|k)$ the set of its k-long subsequences and $\phi(x|k) := |\mathcal{S}(x|k)|$. Then*

$$\phi(x|k) = \begin{cases} \phi(x^{n-1} \mid k) + \phi(x^{n-1} \mid k-1) & \text{if } x_n \not\preceq x^{n-1}, \\ \phi(x^{n-1} \mid k) + \phi(x^{n-1} \mid k-1) - \phi(x^{\ell-1} \mid k-1) & \text{if } x_n \preceq x^{n-1}. \end{cases}$$

Proof. The proof is analogous to that of Lemma 2, additionally tracking the length of the canonical embeddings. □

Maximizing the number of subsequences. Whenever the number of distinct subsequences is used to express the complexity of a string, it is interesting to know which string $z \in \Sigma^n$ maximizes the quantity $\phi(\cdot)$ among all strings of length n. For example, if we define the complexity of a string $x \in \Sigma^n$ as $\phi(x)$, it becomes relevant to study the relative complexity $\phi(x)/\phi(z)$ where $|x| = |z| = n$. Therefore we concisely discuss some of the properties of this ϕ-maximizing z. We let $\Sigma := \{\sigma_1, \ldots, \sigma_d\}$ and $z \in \Sigma^n$ such that

$$z = \sigma_1 \ldots \sigma_d \sigma_1 \ldots \sigma_d \sigma_1 \ldots \sigma_{\text{mod}(n,d)},$$

i.e., z is constructed as a repeated permutation of the alphabet. In [5,16,12], it is shown that $\phi(z|k) \geq \phi(x|k)$ for any $x \in \Sigma^n$ and any k, equality holding for all $k \geq 0$ only if x also consists of a repeated, possibly different permutation of Σ.

The reader notes that we could use Lemma 2, to prove that the repeated permutation z maximizes the number of distinct subsequences. The next lemma was also provided in [12]; it shows that $\phi(z)$ satisfies a generalized Fibonacci-recurrence. Because of Lemma 2, the proof has become almost trivial.

Lemma 4 (*[12]*). *Let $z^n \in \Sigma^n$ be a repeated permutation of an alphabet Σ of size $|\Sigma| = d$. Then*

$$\phi(z^n) = \sum_{k=1}^{d} \phi(z^{n-k}) + 1 \quad \text{if } n \geq d,$$

and $\phi(z^n) = 2^n$ for $0 \leq n < d$.

Proof. Because of Lemma 2 and the fact that z is a repeated permutation of Σ, we have

$$\phi(z^n) = \begin{cases} 2^n & \text{if } 0 \leq n < d, \\ 2\phi(z^{n-1}) - \phi(z^{n-1-d}) & \text{if } n \geq d. \end{cases}$$

Using the recursive part of the above equation to repeatedly expand terms of the form $2\phi(z^{n-j})$ as

$$\phi(z^{n-j}) + 2\phi(z^{n-j-1}) - \phi(z^{n-j-1-d}),$$

and adding, ultimately yields

$$\phi(z^n) = \sum_{k=1}^{d} \phi(z^{n-k}) + 2^d - 2^{d-1} - \cdots - 2^0 = \sum_{k=1}^{d} \phi(z^{n-k}) + 1,$$

as required. □

To discuss a result that is relevant to coding theory, we refine our notation: we write $\phi_d(z|k)$ to denote the number of k-long subsequences of a repeated permutation z of an alphabet Σ of size $|\Sigma| = d$. Interestingly, in [16, Theorem 2.6], it is proven that

$$\phi_d(z^n|k) = \sum_{i=0}^{n-k} \binom{k}{i} \phi_{d-1}(z^{n-k}|n-k-i).$$

However, using Lemma 3, it is immediate that

$$\phi_d(z^n|k) = \phi_d(z^{n-1}|k) + \phi_d(z^{n-1}|k-1) - \phi_d(z^{n-d-1}|k-1)$$

and $\phi_d(z^n|k) = \binom{n}{k}$ for $0 \leq n \leq d$, a recursion that does not need the $\phi_{d-1}(\cdot)$.

4. Counting common subsequences

Next we turn our attention to counting common subsequences of two strings. We write $\phi(x, y) := |\mathcal{S}(x, y)|$ and start with a simple lemma, stated in [29].

Lemma 5 (*[29]*). *Let the strings $x, y \in \Sigma^\star$ have lengths m and n, respectively. Let neither of them contain multiple occurrences of any of the characters of Σ, i.e. $|x|_\sigma$, $|y|_\sigma \in \{0, 1\}$ for all $\sigma \in \Sigma$. Then*

$$\phi(x, y) = \begin{cases} 2\phi(x^{m-1}, y^{n-1}) & \text{if } x_m = y_n, \\ \phi(x^{m-1}, y) + \phi(x, y^{n-1}) - \phi(x^{m-1}, y^{n-1}) & \text{if } x_m \neq y_n. \end{cases}$$

Proof. Suppose $x_m = y_n$. By assumption, $x_m \not\preceq x^{m-1}$ and $y_n \not\preceq y^{n-1}$, so $ux_m \notin \mathcal{S}(x^{n-1}, y^{n-1})$ but $ux_m \in \mathcal{S}(x, y)$ if $u \in \mathcal{S}(x^{m-1}, y^{n-1})$. Therefore, if $x_m = y_n$, then $\mathcal{S}(x, y) = \mathcal{S}(x^{m-1}, y^{n-1}) \cup \{ux_m : u \in \mathcal{S}(x^{m-1}, y^{n-1})\}$, so $\phi(x, y) = 2\phi(x^{m-1}, y^{n-1})$, since these sets are disjoint.

Now suppose $x_m \neq y_n$. Then either $x_m \not\preceq y$ or $x_m \preceq y^{n-1}$. If $x_m \not\preceq y$, $\mathcal{S}(x, y) = \mathcal{S}(x^{m-1}, y)$ and $\mathcal{S}(x, y^{n-1}) = \mathcal{S}(x^{m-1}, y^{n-1})$. On the other hand, if $x_m \preceq y^{n-1}$, then $\mathcal{S}(x, y) = \mathcal{S}(x^{m-1}, y) \cup \mathcal{S}(x, y^{n-1})$ but these sets may not be disjoint. □

Evidently, Lemma 5 implies a simple dynamic programming algorithm [29, Algorithm 3.1]. Equally evident is that the lemma is not correct if a character occurs repeatedly in either string; then $\phi(x, y)$ could well be much smaller than $2\phi(x^{m-1}, y^{n-1})$ when $x_m = y_n$. To obtain a more general result, we must explicitly account for multiple embeddings. This is exactly what the next lemma does: for $|x| = m$, it relates $\phi(x, y)$ to $\phi(x^{m-1}, y)$ under three different conditions.

First, if $x_m \not\preceq y$, elongation of x^{m-1} with x_m does not increase the number of common subsequences. Under the second condition, we suppose that x_m is contained in y and that x_m is new to x^{m-1}. New common subsequences arise by elongating the common subsequences of (x^{m-1}, y^{ℓ_y-1}) with x_m where $\ell_y := \ell(y, x_m)$. Under the third condition, we again assume that $x_m \preceq y$ but now we also suppose that x_m is not new to x^{m-1}. We must compensate the number of new common subsequences that arise by elongation, by subtracting those that were formed when elongating with x_m at the previous time we encountered x_m.

Lemma 6 (*[9]*). *Let x and y be finite, nonempty strings over Σ with lengths $|x| = m$ and $|y|$, respectively. For each $\sigma \in \Sigma$, let $\ell(x, \sigma) := \max\{i : x_i = \sigma\}$ with $\ell(x, \sigma) := 0$ if $\sigma \not\preceq x$. For brevity, we set $\ell_x := \ell(x^{m-1}, x_m)$ and $\ell_y := \ell(y, x_m)$. Then*

$$\phi(x, y) = \begin{cases} \phi(x^{m-1}, y) & \text{if } x_m \not\preceq y, \\ \phi(x^{m-1}, y) + \phi(x^{m-1}, y^{\ell_y-1}) & \text{if } x_m \preceq y,\ x_m \not\preceq x^{m-1}, \\ \phi(x^{m-1}, y) + \phi(x^{m-1}, y^{\ell_y-1}) & \\ \qquad - \phi(x^{\ell_x-1}, y^{\ell_y-1}) & \text{if } x_m \preceq y,\ x_m \preceq x^{m-1}. \end{cases}$$

Proof. If $x_m \not\preceq y$ then $\hat{I}(x^{m-1}, y) = \hat{I}(x, y)$ hence $\phi(x, y) = \phi(x^{m-1}, y)$.

If $x_m \preceq y$ and $x_m \not\preceq x^{m-1}$ then

$$\hat{I}(x, y) = \hat{I}(x^{m-1}, y) \cup C$$

with

$$C = \left\{ \left(i_{ux_m}(x), i_{ux_m}(y^{\ell_y})\right) : \left(i_u(x^{m-1}), i_u(y^{\ell_y-1})\right) \in \hat{I}(x^{m-1}, y^{\ell_y-1}) \right\},$$

and these sets are disjoint, since the embeddings in C end with m whereas the embeddings in $\hat{I}(x^{m-1}, y)$ do not. Note that the joint embeddings in C are canonical since $x_m \not\preceq x^{m-1}$. Hence it follows that

$$\phi(x, y) = \phi(x^{m-1}, y) + \phi(x^{m-1}, y^{\ell_y-1}),$$

as required.

If $x_m \preceq y$ and $x_m \preceq x^{m-1}$ then

$$\hat{I}(x, y) = \hat{I}(x^{m-1}, y) \cup D$$

with

$$D = \left\{ \left(i_{ux_m}(x), i_{ux_m}(y^{\ell_y})\right) : \left(i_u(x^{m-1}), i_u(y^{\ell_y-1})\right) \in E \right\}$$

and

$$E = \hat{I}(x^{m-1}, y^{\ell_y-1}) \setminus \hat{I}(x^{\ell_x-1}, y^{\ell_y-1}).$$

Therefore

$$\phi(x, y) = \phi(x^{m-1}, y) + \phi(x^{m-1}, y^{\ell_y-1}) - \phi(x^{\ell_x-1}, y^{\ell_y-1}),$$

as required. □

Lemma 6 implies a dynamic programming algorithm that requires $\Theta(|x||y|)$ arithmetic operations. The reader notes that in Lemma 6, we presume that the numerical values of ℓ_x and ℓ_y are known. This implies that these values have to be computed before the dynamic algorithm is put to work.

Finally, we generalize Lemma 6 to the problem of computing the number $\phi(x, y|\overrightarrow{k})$ of all common subsequences that are at least k-long.

Theorem 1. *Let x and y be finite, nonempty strings over Σ with lengths $|x| = m$ and $|y|$ respectively. Furthermore, for each $\sigma \in \Sigma$, let $\ell(x, \sigma) := \max\{i : x_i = \sigma\}$ with $\ell(x, \sigma) := 0$ if $\sigma \npreceq x$. For brevity, we set $\ell_x := \ell(x^{m-1}, x_m)$ and $\ell_y := \ell(y, x_m)$. Then, for $0 \le k \le \min\{|x|, |y|\}$,*

$$\phi(x, y|\overrightarrow{k}) = \begin{cases} \phi(x^{m-1}, y|\overrightarrow{k}) & \text{if } x_m \npreceq y, \\ \phi(x^{m-1}, y|\overrightarrow{k}) + \phi(x^{m-1}, y^{\ell_y - 1}|\overrightarrow{k-1}) & \text{if } x_m \preceq y,\ x_m \npreceq x^{m-1}, \\ \phi(x^{m-1}, y|\overrightarrow{k}) + \phi(x^{m-1}, y^{\ell_y - 1}|\overrightarrow{k-1}) & \\ \qquad\qquad - \phi(x^{\ell_x - 1}, y^{\ell_y - 1}|\overrightarrow{k-1}) & \text{if } x_m \preceq y,\ x_m \preceq x^{m-1}. \end{cases}$$

Proof. We first define $\hat{I}(x, y|\overrightarrow{k})$ as the set of all joint canonical embeddings of x and y that are at least k-long. If $x_m \npreceq y$ then $\hat{I}(x^{m-1}, y|\overrightarrow{k}) = \hat{I}(x, y|\overrightarrow{k})$; hence $\phi(x, y|\overrightarrow{k}) = \phi(x^{m-1}, y|\overrightarrow{k})$.

If $x_m \preceq y$ and $x_m \npreceq x^{m-1}$ then

$$\hat{I}(x, y|\overrightarrow{k}) = \hat{I}(x^{m-1}, y|\overrightarrow{k}) \cup C$$

with

$$C = \left\{ \left(i_{ux_m}(x), i_{ux_m}(y^{\ell_y})\right) : \left(i_u(x^{m-1}), i_u(y^{\ell_y - 1})\right) \in \hat{I}(x^{m-1}, y^{\ell_y - 1}|\overrightarrow{k-1}) \right\};$$

and these sets are disjoint, since the embeddings in C end with m whereas the embeddings in the first set do not. Note that the joint embeddings in C are canonical since $x_m \npreceq x^{m-1}$. Hence it follows that

$$\phi(x, y|\overrightarrow{k}) = \phi(x^{m-1}, y|\overrightarrow{k}) + \phi(x^{m-1}, y^{\ell_y - 1}|\overrightarrow{k-1}),$$

as required.

If $x_m \preceq y$ and $x_m \preceq x^{m-1}$ then

$$\hat{I}(x, y|\overrightarrow{k}) = \hat{I}(x^{m-1}, y|\overrightarrow{k}) \cup D$$

with

$$D = \left\{ \left(i_{ux_m}(x), i_{ux_m}(y^{\ell_y})\right) : \left(i_u(x^{m-1}), i_u(y^{\ell_y - 1})\right) \in E \right\}$$

and

$$E = \hat{I}(x^{m-1}, y^{\ell_y - 1}|\overrightarrow{k-1}) \setminus \hat{I}(x^{\ell_x - 1}, y^{\ell_y - 1}|\overrightarrow{k-1}).$$

Therefore,

$$\phi(x, y|\overrightarrow{k}) = \phi(x^{m-1}, y|\overrightarrow{k}) + \phi(x^{m-1}, y^{\ell_y - 1}|\overrightarrow{k-1}) - \phi(x^{\ell_x - 1}, y^{\ell_y - 1}|\overrightarrow{k-1}),$$

as required. □

Clearly, Theorem 1 implies a dynamic programming algorithm with $\Theta(k \cdot |x| \cdot |y|)$ arithmetic operations. Obviously, the possibility to compute the quantity $\phi(x, y|\overrightarrow{k})$ suffices to compute the related quantities

$$\phi(x, y|k) = \phi(x, y|\overrightarrow{k}) - \phi(x, y|\overrightarrow{k+1})$$

and

$$\phi(x, y|\overleftarrow{k}) = \phi(x, y) - \phi(x, y|\overrightarrow{k}),$$

where $\phi(x, y|\overleftarrow{k})$ denotes the number of common subsequences with a length that is smaller than k.

Counting matching embeddings. In Section 2, we discussed the concept of an embedding of $u \preceq x$: a sequence of integers $1 \le i_1 < \cdots < i_{|u|} \le i_{|x|}$ such that $u = x_{i_1} \ldots x_{i_{|u|}}$. For example, $ab \preceq abab = x$ has 3 embeddings: $I_u(x) = \{12, 14, 34\}$. In the analysis of categorical time series, it may be important to account for the fact that certain subsequences have, or do not have, multiple embeddings. For example, consider a set of job career sequences, built from three characters: E for being employed, U for being unemployed and T for a spell of government supported vocational training. Then the subsequence $u = TEU$ both occurs in $x = TEUTEUTEU$ and in $y = TEUE$. But the fact that this "failure"-subsequence occurs 10 times in x and only once in y is not accounted for, if one just counts common subsequences. This example illustrates that it is often useful to account for the number of embeddings, too. This is achieved by evaluating the number $\mu(x, y)$ of matching nonempty embeddings,

$$\mu(x, y) := \sum_{u \in \mathcal{S}(x,y), u \neq \lambda} |x|_u \cdot |y|_u .$$

So, in $\mu(x, y)$, each nonempty common subsequence is weighted according to the number $|x|_u \cdot |y|_u$ of its joint embeddings $(i_u(x), i_u(y))$. In other words, each nonempty common subsequence is weighted by the product of the frequencies of it in either of the sequences.

Let us write $\mu(x, y|k)$ for the number of matching embeddings of length k. Clearly then, we have that $\mu(x, y) = \sum_{k \ge 1} \mu(x, y|k)$. The next lemma implies that we first calculate $\mu(x, y|1)$, then the 1-long subsequences are, if possible, elongated to 2-long common subsequences upon which $\mu(x, y|2)$ is calculated, etc.

Lemma 7 ([8]). *Let $x, y \in \Sigma^\star$, and let $\mathbf{E}^{(k)} = \{e_{ij}^{(k)}\}$ denote $|x| \times |y|$-matrices as follows. We set $e_{ij}^{(1)} := 1$ if $x_i = y_j$, and $e_{ij}^{(1)} := 0$ otherwise. For $2 \le k \le \min\{|x|, |y|\} =: M$, we set $e_{ij}^{(k)} := e_{ij}^{(1)} \sum_{a>i,b>j} e_{ab}^{(k-1)}$. Furthermore, we set $S_k := \sum_{ij} e_{ij}^{(k)}$. Then, for $k \ge 1$,*

$$\mu(x, y|k) = S_k.$$

Proof. By induction over k, $e_{ij}^{(k)}$ equals the number of k-long joint embeddings that start at position i in x and at position j in y and spell the same string. Hence also $S_k = \mu(x, y|k)$. □

To calculate $\mu(x, y)$, we need $\mu(x, y|k)$ for each $1 \le k \le M$ and this requires that we construct each $\mathbf{E}^{(k)}$ and add its elements. So, the implied algorithm needs $\Theta(M \cdot |x| \cdot |y|)$ arithmetic operations. The algorithm uses only little information in each of its M steps, so we suspect the existence of a faster algorithm. Indeed, such an algorithm exists and it requires only $\Theta(|x| \cdot |y|)$ operations. To prove its correctness, we first need the next lemma:

Lemma 8. *Let $x \in \Sigma^m$ and $u \in \Sigma^\star$. Then*

$$|x|_u = \begin{cases} |x^{m-1}|_u & \text{if } x_m \ne u_{|u|} \\ |x^{m-1}|_u + |x^{m-1}|_{u^{|u|-1}} & \text{if } x_m = u_{|u|}. \end{cases}$$

Proof. If $x_m \ne u_{|u|}$, then the embeddings of u in x must be embedded in x^{m-1} and if there exist embeddings of u that end on x_m, these embeddings must have their prefixes $u^{|u|-1}$ embedded in x^{m-1}. □

The reader notes that, since $|x|_{u^0} = |x|_\lambda = 1$, Lemma 8 implies a dynamic algorithm, requiring only $\Theta(|x| \cdot |u|)$ operations, which counts the number of embeddings of $u \preceq x$.

Theorem 2. *Let $x \in \Sigma^m$ and $y \in \Sigma^n$. Then*

$$\mu(x, y) = \begin{cases} \mu(x^{m-1}, y) + \mu(x, y^{n-1}) - \mu(x^{m-1}, y^{n-1}) & \text{if } x_m \ne y_n, \\ \mu(x^{m-1}, y) + \mu(x, y^{n-1}) + 1 & \text{if } x_m = y_n. \end{cases}$$

Proof. Since $|x| = m$, x_m denotes the last character of x and, for any string u, $u_{|u|}$ denotes the last character of u. By definition, we have that $|x_m|_{u_{|u|}} = 1$ if $x_m = u_{|u|}$ and $|x_m|_{u_{|u|}} = 0$ otherwise. So, we reformulate Lemma 8 as

$$|x|_u = |x^{m-1}|_u + |x^{m-1}|_{u^{|u|-1}} \cdot |x_m|_{u_{|u|}},$$

and we use this fact in

$$\begin{aligned} \mu(x, y) &= \sum_u |x|_u \cdot |y|_u \\ &= \sum_u |x^{m-1}|_u \cdot |y^{n-1}|_u + \sum_u |x^{m-1}|_u \cdot |y^{n-1}|_{u^{|u|-1}} \cdot |y_n|_{u_{|u|}} \\ &\quad + \sum_u |x^{m-1}|_{u^{|u|-1}} \cdot |y^{n-1}|_u \cdot |x_m|_{u_{|u|}} + \sum_u |x^{m-1}|_{u^{|u|-1}} \cdot |y^{n-1}|_{u^{|u|-1}} \cdot |x_m|_{u_{|u|}} \cdot |y_n|_{u_{|u|}}, \end{aligned}$$

where it is understood that $u \in \Sigma^\star \setminus \{\lambda\}$. Now it is not difficult to see that

$$\sum_u |x^{m-1}|_u \cdot |y^{n-1}|_u = \mu(x^{m-1}, y^{n-1}),$$

$$\sum_u |x^{m-1}|_u \cdot |y^{n-1}|_{u^{|u|-1}} \cdot |y_n|_{u_{|u|}} = \mu(x^{m-1}, y^n) - \mu(x^{m-1}, y^{n-1}),$$

$$\sum_u |x^{m-1}|_{u^{|u|-1}} \cdot |y^{n-1}|_u \cdot |x_m|_{u_{|u|}} = \mu(x^m, y^{n-1}) - \mu(x^{m-1}, y^{n-1}).$$

Substituting these identities in the above expression for $\mu(x, y)$ then yields

$$\mu(x, y) = \mu(x^{m-1}, y) + \mu(x, y^{n-1}) - \mu(x^{m-1}, y^{n-1}) + \sum_u |x^{m-1}|_{u^{|u|-1}} \cdot |y^{n-1}|_{u^{|u|-1}} \cdot |x_m|_{u_{|u|}} \cdot |y_n|_{u_{|u|}}.$$

Whenever $x_m \ne y_n$, we must have that $|x_m|_{u_{|u|}} \cdot |y_n|_{u_{|u|}} = 0$ for any u so the above expression then reduces to

$$\mu(x, y) = \mu(x^{m-1}, y) + \mu(x, y^{n-1}) - \mu(x^{m-1}, y^{n-1}),$$

proving the first part of the theorem.

On the other hand, if $x_m = y_n$, the summands are only positive when $x_m = u_{|u|} = y_n$. For $u^{|u|-1} \ne \lambda$, the sum equals $\mu(x^{m-1}, y^{n-1})$ and for $x_m = u = y_n$, we must have that

$$|x^{m-1}|_{u^{|u|-1}} \cdot |y^{n-1}|_{u^{|u|-1}} \cdot |x_m|_{u_{|u|}} \cdot |y_n|_{u_{|u|}} = 1.$$

So, for $x_m = y_n$, we have that

$$\mu(x, y) = \mu(x^{m-1}, y) + \mu(x, y^{n-1}) + 1,$$

proving the second part of the theorem. □

Again, a dynamic programming algorithm follows immediately from Theorem 2.

Unpublished work by Greenberg [13] shows how to compute the number of distinct *longest* common subsequences and the number of matching embeddings thereof.

We finally note, that by choosing $y = (1, 2, \ldots, |\Sigma|)$ as one of the sequences and x arbitrarily, the algorithms presented in this section compute the number of strictly increasing subsequences of x.

5. Span of subsequences

In this section, we examine a property of subsequences that we call the *span*. Let x be an n-long string and $u \preceq x$ with embedding $i_u(x) = i_1 \ldots i_{|u|}$. Then we say that the *span of this embedding* equals

$$\widehat{i_u(x)} := i_{|u|} - i_1 + 1.$$

Since a subsequence u may have multiple embeddings in x, we define the *span of a subsequence* as the largest span of its embeddings:

$$\widehat{u \preceq x} := \max\left\{\widehat{i_u(x)} : i_u \in I_u(x)\right\}.$$

We furthermore define the span of the empty embedding as zero, and consequently also $\widehat{\lambda \preceq x} := 0$. If a subsequence has a span of m we call it an *m-span subsequence*.

The span is an interesting property in at least two contexts. First, in social sciences, one may not want to consider subsequences that consist of states that are too remote, i.e., that have too big a time lapse or too many other states in between. For example, consider a categorical time series that represents a job career and that contains short spells of unemployment. Such spells may affect the kinds of jobs directly following the unemployment but this effect will fade away with jobs that are more remote from the unemployment spells. So, it may be interesting to consider only subsequences of labor market statuses that have limited span.

A second context is that of complexity of finite strings (e.g. [7]). Normally, the (subword) complexity of a finite string can be expressed as the number of distinct substrings that it contains. [17,18] proposed a special kind of subsequence, the d-substring, and a class of complexity measures that relies on it. A *d-substring* from x is a subsequence $u = x_{i_1} \ldots x_{i_{|u|}}$, such that $1 \le i_{j+1} - i_j \le d$ for all $1 \le j \le |u| - 1$, or such that $|u| \in \{0, 1\}$. Indeed, for $d = 1$, these objects are the ordinary substrings and for $d > 1$, objects arise that are subsequences with bounded gaps (gaps of at most d). The d-complexity $K_d(x)$ is then taken to be the number of distinct d-substrings of x.

If no character occurs twice in x, it is not difficult to see that

$$K_d(x) = K_d(x^{n-1}) + [K_d(x^{n-1}) - K_d(x^{n-1-d})] + 1.$$

For by elongating x^{n-1} with x_n, new d-substrings arise by appending x_n to all substrings that have their last characters in $\{x_{n-d}, \ldots, x_{n-1}\}$, and x_n is new, too. So, in the right hand side of the above expression, the first term refers to the d-substrings of x^{n-1}, the second term to the new d-substrings that arise by concatenation of x_n to the substrings that have their last character in $\{x_{n-d}, \ldots, x_{n-1}\}$ and the third term, 1, refers to x_n itself that is by definition new. Unfortunately, we were unsuccessful in finding a recurrence that extends to the general case.

The concept of an m-span subsequence is related to that of a d-substring, in the sense that d fixes the maximal gap size, whereas m fixes the average gap size for a subsequence. While no efficient algorithm for counting d-substrings is known to us, it is comparatively easy to count the subsequences of any given span.

We first note that the span of a subsequence is completely determined by the position of its first character in its canonical (left-most), and of its last character in its right-most embedding. The following lemma makes this precise.

Lemma 9. *Let $a, b \in \Sigma$ be two (not necessarily distinct) characters and $v \in \Sigma^\star$ (possibly empty) such that $avb \preceq x$. Then*

$$\widehat{avb \preceq x} = \widehat{ab \preceq x}.$$

Proof. The proof is straightforward and left to the reader. □

Let us write $\widehat{\mathcal{S}}(x|m)$ for the set of subsequences of a string x, the subsequences having a span of exactly m, and let $\vartheta(x|m)$, denote the cardinality of this set.

Clearly, $\widehat{\mathcal{S}}(x|0) = \{\lambda\}$, so $\vartheta(x|0) = 1$ for any string x. Also, $\widehat{\mathcal{S}}(x|1)$ is simply the set of characters that occur in x, so $\vartheta(x|1) = |\{\sigma : \ell(x, \sigma) > 0\}|$. The following theorem states how to compute $\widehat{\mathcal{S}}(x|m)$ and $\vartheta(x|m)$ for $m \ge 2$, and is a direct consequence of the fact that the first and last character determine the span of a subsequence. The theorem translates again into a dynamic programming algorithm. It needs $\Theta(|\Sigma||x|)$ arithmetic operations, assuming $|\Sigma| \in \Theta(|x|)$.

Lemma 10. *As before, let $\ell(x, \sigma) = \ell_\sigma$ denote the rightmost position of $\sigma \in \Sigma$ in $x \in \Sigma^\star$, or $\ell(x, \sigma) := 0$ if $\sigma \npreceq x$. Conversely, let $f(x, \sigma) = f_\sigma$ denote the leftmost position of σ in x, or $f(x, \sigma) := 0$ if $\sigma \npreceq x$. Let $m \geq 2$. Then*

$$\widehat{\mathcal{S}}(x|m) = \bigcup_{\substack{(a,b)\in\Sigma^2:\\ \ell_b - f_a + 1 = m}} \{avb : v \in \mathcal{S}(x_{f_a+1} \dots x_{\ell_b-1})\}.$$

Consequently,

$$\vartheta(x|m) = \sum_{\substack{(a,b)\in\Sigma^2:\\ \ell_b - f_a + 1 = m}} \phi(x_{f_a+1} \dots x_{\ell_b-1}).$$

Proof. Obviously, ab belongs to $\widehat{\mathcal{S}}(x|m)$ if and only if $\ell_b - f_a + 1 = m$. By Lemma 9, the same holds for any subsequence of the form avb with $v \preceq x_{f_a+1} \dots x_{\ell_b-1}$. Since the sequence sets are disjoint for distinct (a, b), the set recurrence immediately translates to the sum formula. $\square$

We conclude this section, by noting that it is equally easy to keep track of the length of the m-span subsequences; this follows as in Lemma 10.

Corollary 1. *Let $\widehat{\mathcal{S}}(x|m; k)$ denote the set of m-span subsequences of length k and $\vartheta(x|m; k)$ its cardinality. Then for $m \geq 2$ and $k \geq 2$,*

$$\widehat{\mathcal{S}}(x|m; k) = \bigcup_{\substack{(a,b)\in\Sigma^2:\\ \ell_b - f_a + 1 = m}} \{avb : v \in \mathcal{S}(x_{f_a+1} \dots x_{\ell_b-1} \mid k-2)\},$$

$$\vartheta(x|m; k) = \sum_{\substack{(a,b)\in\Sigma^2:\\ \ell_b - f_a + 1 = m}} \phi(x_{f_a+1} \dots x_{\ell_b-1} \mid k-2).$$

6. Sequences generated by length-restricted runs

Consider the string $x = x_1 \dots x_r$, with $x_i = \sigma \in \Sigma$ for each $1 \leq i \leq r$, i.e., a string that consists of a *run* of r repetitions of the character σ. In this section, we will denote such runs as $\sigma^{(r)}$ and in particular, $\sigma^{(0)} = \lambda$ for all $\sigma \in \Sigma$. Now consider an n-long string x, and let $y \preceq x$. Then there exists a sequence of integers $r_1, \dots, r_n$ with $r_i \in \{0, 1\}$ for all $1 \leq i \leq n$ such that

$$y = x_1^{(r_1)} x_2^{(r_2)} \dots x_n^{(r_n)}.$$

In fact, the sequence (r_i) is a binary representation of an embedding of y in x. It is natural to say that y is *generated* from x and with run-lengths r_i. An obvious generalization is to allow the r_i to take a wider range $\{0, \dots, \rho\}$ and call the resulting objects the *ρ-generated sequences* of x. We will write $y \preceq_\rho x$ to denote that y is ρ-generated from x. With $\rho = 1$, these objects are the ordinary subsequences of x and the relation $\preceq_1$ is the same as $\preceq$.

Although in a ρ-generated sequence y from x, the individual runs are confined to lengths $\{0, \dots, \rho\}$, this does not imply that the runs in y cannot exceed ρ. For example, with $x = aba$, we can 2-generate $a^{(2)}b^{(0)}a^{(1)} = a^{(3)} \preceq_2 aba$. Motivated by modern DNA sequencing technology that can sequence a whole run at a time, but only reliably so when the run is not too long (about 4 or 5 characters), we are interested in counting sequences in which no run-length exceeds ρ. Therefore we define the set Σ_ρ^k of k-long *ρ-restricted* sequences

$$\Sigma_\rho^k := \left\{ u = u_1^{(r_1)} \dots u_m^{(r_m)} : u_{i+1} \neq u_i \text{ and } 1 \leq r_i \leq \rho \text{ for all } i; \sum_{i=1}^m r_i = k \right\}.$$

Now we define the set of k-long, ρ-restricted sequences that are ρ-generated from $x \in \Sigma^n$ as

$$\mathcal{S}_\rho(x|k) := \left\{ u \in \Sigma_\rho^k : u \preceq_\rho x \right\},$$

and we wish to count the sequences in this set. We subdivide $\mathcal{S}_\rho(x|k)$ into

$$\mathcal{S}_\rho(x|k; \sigma) := \{u \in \Sigma_\rho^k : u \preceq_\rho x, u_m = \sigma\},$$

i.e., we condition on the last letter of the subsequence. Furthermore we set

$$\mathcal{S}_\rho(x|k; \overline{\sigma}) := \mathcal{S}_\rho(x|k) \setminus \mathcal{S}_\rho(x|k; \sigma).$$

Note that for $k > 0$ we have $\mathcal{S}_\rho(x|k) = \cup_\sigma \mathcal{S}_\rho(x|k; \sigma)$ as a disjoint union. However, for $k = 0$ we have $\mathcal{S}_\rho(x|0) = \{\lambda\} \neq \{\} = \cup_\sigma \mathcal{S}_\rho(x|k; \sigma)$. We write $\phi_\rho(x|k)$, $\phi_\rho(x|k; \sigma)$, and $\phi_\rho(x|k; \overline{\sigma})$ for the respective cardinalities of the above sets.

The next theorem [25] again implies a dynamic programming algorithm to compute these quantities.

Table 1
Subsequence counting problems solved in this paper, along with their Java implementation

Object	Symbol	Reference	Java: `count...`
Distinct subsequences of x	$\phi(x)$	Lemma 2	`Subsequences(x)`
Distinct subsequences of length k	$\phi(x\|k)$	Lemma 3	`Subsequences(x,k)`
Common subsequences	$\phi(x, y)$	Lemma 6	`CommonSubsequences(x,y)`
Common subsequences of length $\geq k$	$\phi(x, y\|k)$	Theorem 1	`CommonSubsequences(x,y,k)`
Embeddings of y in x	$\|x\|_y$	Lemma 8	`Embeddings(x,y)`
Matching embeddings	$\mu(x, y)$	Theorem 2	`MatchingEmbeddings(x,y)`
Matching embeddings of length k	$\mu(x, y\|k)$	Lemma 7	`MatchingEmbeddings(x,y,k)`
Subsequences with span m	$\vartheta(x\|m)$	Lemma 10	`SubsequencesSpan(x)`
Subsequences of length k	$\vartheta(x\|m; k)$	Corollary 1	`SubsequencesSpan(x,k)`
ρ-generated ρ-restricted seq's	$\phi_\rho(x)$	Theorem 3	`GeneratedSequences(x)`
ρ-generated ρ-restricted seq's of length k	$\phi_\rho(x\|k)$	Theorem 3	`GeneratedSequences(x)`

Theorem 3 *([25]). Let $x \in \Sigma^n$ and $k \geq 1$. Then*

$$\phi_\rho(x|k;\sigma) = \begin{cases} \phi_\rho(x^{n-1}|k;\sigma) & \text{if } \sigma \neq x_n, \\ \sum_{r=1}^{\min\{\rho,k\}} \phi_\rho(x^{n-1}|k-r;\overline{\sigma}) & \text{if } \sigma = x_n. \end{cases}$$

Proof. If $x_n \neq \sigma$, then the sequences in $\mathcal{S}_\rho(x|k;\sigma)$ cannot end at x_n; hence they are already in $\mathcal{S}_\rho(x^{n-1}|k;\sigma)$. Hence the cardinalities of these sets are equal.

If, on the other hand, $x_n = \sigma$, we may take any $u \in \mathcal{S}(x^{n-1}|k-r;\overline{\sigma})$, and append a σ-run $\sigma^{(r)}$ for any $1 \leq r \leq \rho$ (unless $r > k$); this yields a distinct $u\sigma^{(r)} \in \mathcal{S}(x|k;\sigma)$ in each case. Conversely, every string in this set is obtained by appending a σ-run to such a prefix. Hence $\mathcal{S}(x|k;\sigma) = \cup_{1\leq r\leq\min\{\rho,k\}} \mathcal{S}(x^{n-1}|k-r;\overline{\sigma})$, and since the union is disjoint, the cardinalities sum up. □

We point out the special case $\rho = 1$, which counts the number of k-subsequences of x, in which no two adjacent characters are equal:

$$\phi_1(x|k;\sigma) = \begin{cases} \phi_1(x^{n-1}|k;\sigma) & \text{if } \sigma \neq x_n, \\ \phi_1(x^{n-1}|k-1;\overline{\sigma}) & \text{if } \sigma = x_n. \end{cases}$$

In fact, the formula for general ρ can be derived from this special case by the identity

$$\phi_\rho(x|k) = \sum_{m\geq 0} \phi_1(x|k) \cdot c(k, m, \rho),$$

where $c(k, m, \rho)$ denotes the number of compositions [1] of k with m summands that do not exceed ρ.

7. Conclusion

Up to now, the systematic study of subsequences of one or several strings has received relatively little attention, compared with the study of substrings. With the present paper, we have collected and extended efficient algorithms to count subsequences under different constraints; each problem we consider is motivated by at least one application field (e.g., the study of categorical time series in the social sciences, biological sequence analysis in bioinformatcs, string complexity considerations and string kernel construction in machine learning). We provide a Java implementation using exact arbitrary-precision arithmetic of the algorithms contained in this paper at http://gi.cebitec.uni-bielefeld.de/comet/subseq. Table 1 summarizes the counting problems considered in this paper, along with the respective Java function call.

It remains open to find an efficient algorithm to count d-substrings (see Section 5) or prove that this problem is NP-hard. We have noted that by choosing $y = (1, 2, 3, \ldots)$ in the algorithms that count common subsequences or matching embeddings, we can count increasing subsequences or embeddings in x, but possibly for this problem, more efficient algorithms exist.

Acknowledgements

The remarks of two anonymous reviewers were very helpful in substantially improving this paper by pointing out relevant literature and a number of annoying flaws in the first version.

References

[1] G.E. Andrews, The Theory of Partitions, Cambridge University Press, Cambridge, UK, 1998.
[2] M.-C. Anisiu, Z. Blázsik, Z. Kása, Maximal Complexity of Finite Words, Pure Mathematics and Applications 13 (1–2) (2002) 39–48.

[3] L. Bergroth, H. Hakonen, T. Raita, A survey of longest common subsequence algorithms, in: Proceedings of the Seventh International Symposium on String Processing and Information Retrieval, SPIRE'00, IEEE Computer Society, 2000, pp. 39–48.
[4] L. Bergroth, H. Hakonen, J. Väisänen, New refinement techniques for longest common subsequence algorithms, in: M.A. Nascimento, E. Silvo de Moura, A.L. Oliveira (Eds.), String processing and Information Retrieval, 10th International Symposium, SPIRE 2003, Springer, New York, 2003, pp. 287–303.
[5] P. Chase, Subsequence numbers and logarithmic concavity, Discrete Mathematics 16 (1976) 123–140.
[6] M. Crochemore, W. Rytter, Text Algorithms, Oxford University Press, New York, 1994.
[7] A. de Luca, On the combinatorics of finite words, Theoretical Computer Science 218 (1999) 13–39.
[8] C.H. Elzinga, Combinatorial representation of token sequences, Journal of Classification 22 (1) (2005) 87–118.
[9] C.H. Elzinga, Turbulence in categorical time series, Sociological Methods & Research (submitted for publication).
[10] C.H. Elzinga, A.C. Liefbroer, De-standardization and differentiation of family life trajectories of young adults: A cross-national comparison using sequence analysis, European Journal of Population 23 (3–4) (2007) 225–250.
[11] S. Ferenczi, Z. Kása, Complexity for finite factors of infinite sequences, Theoretical Computer Science 218 (1999) 177–195.
[12] A. Flaxman, A.W. Harrow, G.B. Sorkin, Strings with maximally many distict subsequences and substrings, The Electronic Journal of Combinatorics 11 (2004) R8.
[13] R.I. Greenberg, Computing the number of longest common subsequences, Computational Science Research Repository. arXiv:cs/0301034v1[cs.DS] (http://arxiv.org/abs/cs/0301034), 2003.
[14] D. Gusfield, Algorithms on Strings, Trees and Sequences, Computer Science and Computational Biology, Cambridge University Press, Cambridge, UK, 1997.
[15] D.S. Hirschberg, Algorithms for the longest common subsequence problem, Journal of the ACM 24 (4) (1977) 664–675.
[16] D.S. Hirschberg, M. Regnier, Tight bounds on the number of string subsequences, Journal of Discrete Algorithms 1 (1) (2000) 123–132.
[17] A. Iványi, On the *d*-complexity of words, Annales "Universitatis Scientiarum Budapestinensis de Rolando Eötvös nominatae, Sectio Computatorica 8 (1987) 69–90.
[18] Z. Kása, On the d-complexity of strings, Pure Mathematics and Applications 9 (1–2) (1998) 119–128.
[19] F. Levé, P. Séébold, Proof of a conjecture on word complexity, Bulletin of the Belgian Mathematical Society Simon Stevin 8 (2) (2001) 277–291.
[20] V.I. Levenshtein, Binary codes capable of correcting deletions, insertions and reversals, Soviet Physics Doklady 10 (1966) 707–710.
[21] V.I. Levenshtein, On perfect codes in insertion and deletion metric, Discrete Mathematics and Applications 2 (3) (1992) 241–258.
[22] V.I. Levenshtein, Efficient reconstruction of sequences from their subsequences or supersequences, Journal of Combinatorial Theory, Series A 93 (2) (2001) 310–332.
[23] M. Lothaire, Combinatorics on Words, in: The Encyclopedia of Mathematics and its Applications, vol. 17, Addison-Wesley, Reading, MA, 1983.
[24] M. Lothaire, Algebraic Combinatorics on Words, in: The Encyclopedia of Mathematics and its Applications, vol. 90, Cambridge University Press, Cambridge, UK, 2002.
[25] S. Rahmann, Subsequence combinatorics and applications to microarray production, DNA sequencing and chaining algorithms, in: M. Lewenstein, G. Valiente (Eds.), Combinatorial Pattern Matching (CPM), in: Lecture Notes in Computer Science (LNCS), vol. 4009, Springer, New York, 2006, pp. 153–164.
[26] J. Sakarovitch, I. Simon, Subwords, in: M. Lothaire (Ed.), Combinatorics on Words, Addison-Wesley, Reading, MA, 1983 (Chapter 6).
[27] A. Salomaa, Counting (scatterred) subwords, Bulletin of the European Association for Theoretical Computer Science (81) (2003) 165–179.
[28] T.G. Swart, H.C. Ferreira, A note on double insertion/deletion correcting codes, IEEE Transaction on Information Theory 49 (1) (2003) 269–273.
[29] H. Wang, All common subsequences, in: M.M. Veloso (Ed.), IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, 2007.