The following paper is the draft of the Elzinga/Wang paper which is the formal basis for the Elzinga/Studer/Wang paper intended for Sociological Methods & Research. The latter will contain some applications of soft-matching (something equivalent to substitution cost in OM) to demographical data. My presentation will informally deal with the metrics and discuss the merits of OM and the methods presented here in the context of the demography-applications.

Cees Elzinga

Versatile String Kernels

Cees H. Elzinga[a,*], Hui Wang[b]

[a] *Department of Sociology, PARIS Research Program, VU University Amsterdam, The Netherlands*
[b] *Computer Science Research Institute, School of Computing and Mathematics, University of Ulster, Northern Ireland, UK*

Abstract

This paper proposes string kernels that can be easily modified to handle a variety of subsequence-based features. Slight adaptations of the basic algorithm allow for weighing subsequence lengths, restricting or soft-penalizing gap-size, character-weighing and soft-matching of characters. An easy extension of the kernels allows for comparing run-length encoded strings with a time-complexity that is independent of the length of the original, uncompressed strings. Such kernels have applications in image processing, computational biology, demography and in comparing partial rankings.

Keywords: String kernel, string matching, subsequences, gap-penalizing, soft matching, run-length encoding, partial rankings.

1. Introduction

In many branches of science and engineering, data come as strings of characters from a finite alphabet, the characters standing for states, events or objects and the order of the characters corresponding to the order of states or events in time or the order of objects in space. An example of such a time series is a labor market career with states like "employed" and "retired", an example of a sequence in space is a strand of DNA where the objects are nucleobases and another example is a consumer's ranking of different brands of lipsticks. Analyzing such data requires sensible classification of the strings

*Corresponding author
Email addresses: `c.h.elzinga@vu.nl` (Cees H. Elzinga), `h.wang@ulster.ac.uk` (Hui Wang)

and thus we need algorithms to efficiently gauge similarity and distance between the strings. Classical methods of string matching have been amply described in very readable books like [17, 35, 41]. String kernels were first introduced by [42, 18] and a first, thorough applications of the concepts was developed in [30] for text classification and in [28] for protein classification. Since then, string kernels have been used in quite different sciences: in the social sciences to study life courses and labor market careers [5, 6, 10], in meteorology to study wind patterns [33], in biology to study animal behavioral patterns [26], in decision making to analyze concordance [13] and in computational biology, for example, to recognize splice sites in eukaryotes[4]. The spectrum kernels, using the concept of substrings, as introduced by Christina Leslie and her collaborators [28] comprise a quite natural way to compare strings when contiguity of the symbols is important as it is, for example in analyzing proteins. The kernel introduced by Huma Lodhi [30] compares strings by a weighted count of their (possibly noncontiguous) subsequences, penalizing the gaps in the embeddings of subsequences. Lodhi's kernel and its variants were amply discussed in [37, see Chapter 11]. Driven by applications in the social sciences, [10, 11] proposed a more efficient subsequence based string kernel and [39, 40] also proposed kernels to count all common subsequences.
Driven by problems in social demography [19, 38] and in comparing partial rankings [1, 2, 14], this paper discusses extensions of the subsequence kernel proposed in [10, 12, Theorem 7]. We will discuss extensions that weigh subsequences according to their length, according to the characters included and the gaps spanned, discuss soft matching and we will address the problem that is known in computational biology as the problem of "run lengths" [8, 15] and in social demography as the problem of "duration" of states.
To set the stage, we will first, in the next section, deal with some concepts and notation pertaining to subsequences and their feature vectors and discuss the basic version of the kernel in Section 3. In Section 4, we discuss differential weighing of the characters appearing in the subsequences. Section 5 deals with weighing subsequence length and restricting gap-sizes. In Section 6, we concisely deal with another kernel, the Trail-algorithm, that is particularly suitable for soft gap-penalizing in the context where repetitions of characters are few or even absent, like for example in comparing (partial) rankings. In Section 7, we discuss soft-matching through using an elliptical inner-product. In Section 8, we discuss an extension of the Grid-algorithm that very efficiently compares run-length encoded (RLE) strings: indepen-

dently from the lengths of the original, uncompressed strings. Finally, in Section 9, we summarize and conclude.

2. Preliminaries

Throughout this paper, we will use the notation $[n]$ to denote the first n natural numbers $\{1, \ldots, n\}$. Let $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ denote a set, also called an alphabet, of n characters. A string over Σ is a sequence $x = x_1 x_2 \ldots x_n$ that arises by concatenation of characters from the alphabet: $x_i \in \Sigma$ for $i \in [n]$. We let Σ^* 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 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 $y \sqsubseteq x$ to denote this fact. The set of all subsequences of x is denoted by $\mathcal{S}(x)$.
Let $u \sqsubseteq x$ with $u = x_{j_1} \ldots x_{j_{|u|}}$. Then the *gaps* of u in x are defined as $g_m(u|x) = j_m - j_{m-1} - 1$ for $m \in [|u| - 1]$. Hence, as is intuitive, $g_j(x|x) = 0$. The *width* $w(u|x)$ of u in x is defined as $j_{|u|} - j_1 + 1$ and $w(x|x) = |x|$.
If $u \sqsubseteq x$ and $u \sqsubseteq y$, we write $u \sqsubseteq (x, y)$ 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 with $\phi(x, y) = |\mathcal{S}(x, y)|$.
Whenever we want to confine the length of the (common) subsequences to some $k \geq 0$, we use the symbols $\mathcal{S}_k$ and ϕ_k. So, for example, $\mathcal{S}_k(x, y)$ denotes the set of all common k-long subsequences of x and y and there exist $\phi_k(x, y) = |\mathcal{S}(x, y)|$ of such subsequences.
Let $u = u_1 \ldots u_{|u|} \sqsubseteq x$. Then the integer sequence $i_x(u) = i_1, \ldots, i_{|u|}$ is called an embedding of u in x, precisely when $x_{i_1} \ldots x_{i_{|u|}} = u$. Some subsequences may be embedded more than once in a string; for example $u = ac$ is embedded twice in $x = abac$. We denote the number of embeddings of $u \sqsubseteq x$ as $|x|_u$; so, $|x|_{ac} = 2$.
In the sections to come, we will discuss kernels to evaluate inner products of feature vectors $\mathbf{x} = (\mathrm{x}_1, \mathrm{x}_2, \ldots)$ that represent strings $x = x_1 x_2 \ldots x_n$. Such vectors can be constructed through an arbitrary but fixed mapping

$r : x \in \Sigma^\star \to r(x) \in \mathbb{Z}^+$ of all finite strings to the nonnegative integers and then defining the coordinates x_i of the vector $\mathbf{x} = (\mathrm{x}_1, \mathrm{x}_2, \ldots)$ through

$$\mathrm{x}_{r(u)} = \begin{cases} f(u,x) \text{ if } u \sqsubseteq x \\ \\ 0 \text{ otherwise} \end{cases} \tag{1}$$

wherein $f(u,x)$ maps into some number-field.

3. The basic Grid-algorithm

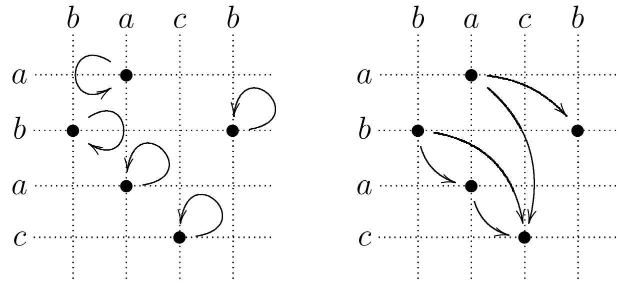
In this section, we deal with the most simple form of the kernel as first proposed in [10], evaluating the inner product $\mathbf{x}'\mathbf{y}$ of vectors constructed according to

$$\mathrm{x}_{r(u)} = \begin{cases} |x|_u \text{ if } u \sqsubseteq x \\ \\ 0 \text{ otherwise} \end{cases} . \tag{2}$$

Hence, the inner product adds products of the form $|x|_u \cdot |y|_u$, i.e. the number of times that an embedding of u in the one string matches to an embedding of u in the other string. So, we say that $\mathbf{x}'\mathbf{y}$ counts the number of matching embeddings or, equivalently, the number of matching subsequences. Furthermore, we discuss some of its properties and algorithmic details. Before we formally present the algorithm, we illustrate its principles through a simple example. Thus, writing $\mu(x, y)$ for the total number of matching embeddings of x and y, we have that $\mathbf{x}'\mathbf{y} = \mu(x, y)$ and we write $\mu_k(x, y)$ to denote the number of matching k-long embeddings.
Fig. 1 shows two grids, each representing a different stage of the algorithm applied to the strings $x = abac$ and $y = bacb$, constructed from $\Sigma = \{a, b, c\}$. In both grids, common subsequences are represented by (sequences of) arrows. In the left grid, only the common 1-tuples are shown, hence the arrows point to the node from which they departed, the nodes themselves representing the fact that $x_i = y_j$. In the right grid, the arrows point to all nodes that are to the South-East relative to the nodes from which they departed, hence the arrows represent all common 2-tuples. The common 3-tuples arise by combining two arrows with one common node. We constructed $\mathbf{M}^1$ according to the rule $m^1_{ij} = 1$ iff $x_i = y_j$ and $e^1_{ij} = 0$ otherwise. Then we keep track of the number of k-long South-East going paths that depart from each node by adding the numbers South-East of each non-zero cell in $\mathbf{M}^{k-1}$ and

storing these results in the corresponding cell in $\mathbf{M}^k$. Adding the values in each matrix $\mathbf{M}^k$ then results in the number of common k-long subsequences of the pertaining sequences. The process stops as soon as all cells of $\mathbf{M}^{k+1}$ equal zero, implying that x and y have no subsequences in common with a length of $k+1$ or longer and that all common subsequences have been counted but the empty λ. The process described and illustrated in Fig. 1, is formalized in the next theorem 1.

Figure 1: Counting 1-, 2- and 3-long common subsequences of $x = abac$ and $y = bacb$ using a grid: common subsequences are represented by paths of arrows that are either self-referencing in case of 1-tuples (left grid) or that are pointing to the "South-East" (right grid). The resulting counts are stored in matrices $\mathbf{M}^1$, $\mathbf{M}^2$ and $\mathbf{M}^3$. There are no common 4-long subsequences, hence we conclude that $\mu(x, y) = 5 + 5 + 1 + 1 = 12$, the last "+1" counting λ, the empty sequence.



$$\mathbf{M}^1 = \begin{pmatrix} & 1 & & \\ 1 & & & 1 \\ & 1 & & \\ & & 1 & \end{pmatrix} \quad \mathbf{M}^2 = \begin{pmatrix} & 2 & & \\ 2 & & & 0 \\ & 1 & & \\ & & 0 & \end{pmatrix} \quad \mathbf{M}^3 = \begin{pmatrix} & 0 & & \\ 1 & & & 0 \\ & 0 & & \\ & & 0 & \end{pmatrix}$$

Theorem 1. *[10, 12] Let $x, y \in \Sigma^*$. Furthermore, let $\mu(x, y)$ denote the number of matching embeddings of the pair (x, y), let $\mu_k(x, y)$ denote the number of k-long embeddings of x and y and let $\mathbf{M}^k = \left(m_{ij}^k\right)$ denote $|x| \times |y|$-matrices as follows. We set $m_{ij}^1 = 1$ if $x_i = y_j$, and $m_{ij}^1 = 0$ otherwise. For $2 \leq k \leq n$, we set $m_{ij}^k = m_{ij}^1 \sum_{a>i,b>j} m_{ab}^{k-1}$. Then $\mu_k(x, y) = \sum_{ij} m_{ij}^k$ and*

$$\mu(x, y) = 1 + \sum_{k=1} \mu_k(x, y) \tag{3}$$

Proof. By induction over k, m_{ij}^k equals the number of k-long common embeddings that start at position i in x and at position j in y and spell the same string. $\square$

Let $M = \min\{|x|, |y|\}$. Common subsequences of (x, y) cannot be longer than M and thus theorem 1 implies an algorithm of complexity $O(M\cdot|x|\cdot|y|)$. A faster algorithm, of complexity $O(|x|\cdot|y|)$, was proposed in [12, Theorem 2] but it is much less versatile than the algorithm implied by theorem 1. The latter algorithm is shown in pseudo-code as Algorithm 1.

The algorithm is initialized in lines 1-11, constructing the matrix $\mathbf{M}^1$ after which the algorithm continues in the while-loop in lines 12 and 33, constructing successive $\mathbf{M}^k$. There are two main optimizations in Algorithm 1 compared to theorem 1. The first one is in line 32, where the algorithm diminishes the size of the matrix processed in each cycle of the while-loop. The second optimization prevents actually carrying out the many additions implied by the theorem's statement that

$$m_{ij}^k = m_{ij}^1 \sum_{a>i,b>j} m_{ab}^{k-1}. \tag{4}$$

Instead, the Algorithm first evaluates, in lines 13-17, the "backwards" row-sum

$$m_{ij}^k = \sum_{m=1} \sum_{b=j+1,1} m_{ib}^{k-1} \tag{5}$$

and then, in lines 19-31, evaluates, the "backwards" column-sum

$$m_{ij}^k = m_{ij}^1 \sum_{j=1} \sum_{a=i+1,1} m_{aj}^{k-1}. \tag{6}$$

It is not difficult to see that the algorithmic complexity of the Grid-algorithm when applied to sequences of length $|x|$ and $|y|$ is of order $O(M|x||y|)$ with $M = \min\{|x|, |y|\}$. The reader notes that by changing the summation in Eq. (4) to

$$m_{ij}^k = m_{ij}^1 \max_{a>i,b>j} m_{ab}^{k-1}, \tag{7}$$

Algorithm 1: Grid-algorithm, implements theorem 1.

```
Require: $x = x_1 \dots x_r, \quad y = y_1 \dots y_c$
1: $s \leftarrow 0$
2: for $i = 1$ to $r$ do
3:    for $j = 1$ to $c$ do
4:       if $x_i = y_j$ then
5:          $m^1_{ij} \leftarrow 1, \quad m_{ij} \leftarrow 1, \quad s \leftarrow s + 1$
6:       end if
7:    end for
8: end for
9: if $s = 0$ then
10:    return $s + 1$
11: end if
12: while $(r > 0) \wedge (c > 0)$ do
13:    for $i = 1$ to $r$ do
14:       $s \leftarrow 0$
15:       for $j = c$ to $1$ do
16:          $t \leftarrow s, \quad s \leftarrow s + m_{ij}, \quad m_{ij} \leftarrow t$
17:       end for
18:    end for
19:    $\mu_k \leftarrow 0$
20:    for $j = 1$ to $c$ do
21:       $s \leftarrow 0$
22:       for $i = r$ to $1$ do
23:          $t \leftarrow s, \quad s \leftarrow s + m_{ij}$
24:          $m_{ij} \leftarrow t \times m^1_{ij}$
25:          $\mu_k \leftarrow \mu_k + m_{ij}$
26:       end for
27:    end for
28:    if $\mu_k = 0$ then
29:       return $\mu + 1$
30:    end if
31:    $\mu \leftarrow \mu + \mu_k$
32:    $r \leftarrow r - 1, \quad c \leftarrow c - 1$
33: end while
34: return $\mu + 1$
```

the algorithm evaluates an inner product of vectors constructed according to

$$\mathrm{x}_{r(u)} = \begin{cases} 1 \text{ if } u \sqsubseteq x \\ \\ 0 \text{ otherwise} \end{cases}, \tag{8}$$

i.e. it ignores embedding frequency of the subsequences. Grids are related to edit graphs which have been used to find longest common subsequences [e.g. 3, 17].

4. Weighing the characters

In some applications, it might be worthwhile to put more weight on the occurrence of some characters or states, like for example "infected" or "unemployed"; such states could be medically or demographically significant. In this section we discuss the principles of such weighing and we discuss a specific interpretation of it, that allows for comparing partial rankings as arise in e.g. decision theory and statistics.

4.1. Principles

Weighing characters is easily accomplished by defining a weight-function $w : \Sigma \to \mathbb{R}^+$ of the characters to the non-negative real numbers and defining the weighing

$$\omega(u) = \prod_i w(u_i) \tag{9}$$

and thus constructing vectors $\mathbf{x} = (\mathrm{x}_1, \mathrm{x}_2, \ldots)$ according to

$$\mathrm{x}_{r(u)} = \begin{cases} \omega(u)|x|_u \text{ if } u \sqsubseteq x \\ \\ 0 \text{ otherwise} \end{cases}, \tag{10}$$

Implementation into the Grid-algorithm is easy: replace line 5 by

$$5' : \quad m^1_{ij} \leftarrow w^2(x_i), \quad m_{ij} \leftarrow w^2(x_i), \quad s \leftarrow s + m_{ij}$$

where $w(\cdot)$ is a predefined array of weights.

4.2. Partial rankings as bucket strings

A ranking is an ordered set, e.g. a set of produce ordered according to the preferences of a judge, a set of politicians ordered according to the attitudes of a voter or a set of patients, ordered according to surgeon's treatment priority. Often, judges are not able or do not care to order all pairs of items and as a result, "ties" occur in which case the ordering is called "partial". Formally, let Σ with $|\Sigma| = d$ denote a set of items. Following [14], we call the $2^d - 1$ non-empty subsets of Σ "buckets" $\mathcal{B}_i$. A partial ordering is a string of such buckets $\mathcal{B} = \mathcal{B}_1 \dots \mathcal{B}_m$ with $1 \le m \le d$; within buckets, items are unordered and between different buckets, items are ordered. For example, let $\Sigma = \{a, b, c, d, e, f\}$ denote the set of items and define $\mathcal{B}_1 = \{b, e\}$, $\mathcal{B}_2 = \{a\}$ and $\mathcal{B}_3 = \{c, d, f\}$, then $\mathcal{B} = \mathcal{B}_1\mathcal{B}_2\mathcal{B}_3 = \{b, e\}\{a\}\{c, d, f\}$ is a bucket string representing a partial ordering of the items of Σ. Given Σ, there exist $\sum_k \left\{ {d \atop k} \right\} k!$ of partial orderings of all items, where $\left\{ {d \atop k} \right\}$ denotes a Stirling number of the second kind [24, 25].
Clearly, a bucket string of length m can be interpreted as set of *full* orderings of only m items. We say that an m-long full ordering $x = x_1 \dots x_m$ is compatible with an n-long bucket string $\mathcal{B} = \mathcal{B}_1 \dots \mathcal{B}_n$, $m \le n$, when $x_i \in \mathcal{B}_{k_i}$ and $x_j \in \mathcal{B}_{k_j}$ implies $k_i < k_j$ iff $i < j$ and we write $x \preceq \mathcal{B}$ to denote such compatibility. For example, $x = abc$ is compatible with $\mathcal{B} = \{a, d\}\{e, b\}\{c, f, g\}$ but $z = acf$ is not. Clearly, a bucket string $\mathcal{B} = \mathcal{B}_1 \dots \mathcal{B}_n$ has $\prod_i |\mathcal{B}_i|$ distinct compatible orderings. Writing $\mathcal{B}_{\mathcal{Y};i}$ to denote the i^{th} bucket of bucket string $\mathcal{Y}$, we say that the ordering x is compatible to both bucket strings $\mathcal{Y}$ and $\mathcal{Z}$, precisely when $x_i \in \mathcal{B}_{\mathcal{Y};i} \cap \mathcal{B}_{\mathcal{Z};i}$. Hence, we can use the Grid-algorithm to compare partial rankings through the implementation

$$5'' : \quad m^1_{ij} \leftarrow |\mathcal{B}_{\mathcal{X};i} \cap \mathcal{B}_{\mathcal{Y};i}|, \quad m_{ij} \leftarrow |\mathcal{B}_{\mathcal{X};i} \cap \mathcal{B}_{\mathcal{Y};i}|, \quad s \leftarrow s + m_{ij}.$$

The algorithm then evaluates an inner product of bucket string representing vectors that are constructed according to

$$\mathrm{x}_{r(u)} = \begin{cases} 1 \text{ if } u \preceq \mathcal{B} \\ \\ 0 \text{ otherwise} \end{cases}, \tag{11}$$

hence the algorithm counts the number of matching suborders. Clearly, the more matching suborders, the more similar the rankings.

5. Weighing Subsequence Lengths and Gaps

Life courses and careers normally share most, if not all states: most people live with their parents during childhood, go to school, find partners and jobs and sooner or later become parents. Similarly, proteins are build from the same, limited set of nucleotides. So, it is not surprising that many kinds of strings share many short subsequences. Therefore, one might want to penalize for shorter sequences. As the Grid-algorithm counts the common subsequences in an orderly manner, it is easy to weigh according to length through introducing a convex weighing function in Equation 3, yielding

$$\mu(x, y) = 1 + \sum_k L(k, \mu_k(x, y)). \tag{12}$$

Such a modification is particularly easy to implement by modifying line 31 of Algorithm 1:

$$31' : \mu \leftarrow \mu + L(k, \mu_k).$$

Let $u \in \mathcal{S}(x, y)$, i.e. u is common to both x and y. When the characters of u are widely spaced in either or both originals, then u may be quite insignificant in quantifying the similarity between x and y. A way to deal with this is to weigh the count of common subsequences with the size of their gaps. A particularly simple method would be to simply exclude all subsequences which show gaps that exceed some fixed limit [20, 21, 22]. We call this a "hard" gap-penalty of size h. Hard-penalizing is easily implemented into the Grid-algorithm by simply limiting the elongation of subsequences already counted: we just evaluate the limited sum (compare to Equation (4)):

$$m_{ij}^k = m_{ij}^1 \sum_{a=i+1}^{i+h} \sum_{b=j+1}^{j+h} m_{ab}^{k-1} \tag{13}$$

$$= m_{ij}^1 \left(\sum_{a>i,b>j} m_{ab}^{k-1} - \sum_{a>i+h,>j+h} m_{ab}^{k-1} \right), \tag{14}$$

Equation (14) reflecting the way the hard-penalizing can be implemented in the Grid-algorithm. Thereto, one modifies line 16 to

$$16' : \quad t \leftarrow s, \quad s \leftarrow s + m_{ij} - m_{i,j-h}, \quad m_{ij} \leftarrow t - m_{i,j-h}$$

and line 23 to

$$23' : \quad t \leftarrow s, \quad s \leftarrow s + m_{ij} - m_{i-h,j},$$

thus evaluating an inner product of vectors $\mathbf{x} = (\mathrm{x}_1, \mathrm{x}_2, \ldots)$ constructed through

$$\mathrm{x}_{r(u)} = \begin{cases} |x|_u \text{ if } (u \sqsubseteq x) \wedge (\max_i\{g_i(u|x)\} < h) \\ \\ 0 \text{ otherwise} \end{cases} . \qquad (15)$$

Soft-penalizing gaps would then mean that we assign a penalty to each gap, the size of it depending upon the size of the gap. However, soft-penalizing is not so easy to implement in the Grid-algorithm because specific information about the size of the gaps is lost in the summations of Equations (5) and (6) (the loops starting in lines 13 and 20 of the Grid-algorithm).
To implement flexible soft-penalizing of gaps, we need an algorithm that sequentially deals with all gaps and that is, consequently, slower than the Grid-algorithm. However, in the context of comparing (partial) ranks, wherein each character can occur only once, this alternative has an algorithmic complexity of order $O(n^2)$ where n denotes the size of the item set. In the next section, we concisely deal with the alternative, called the Trail-algorithm.

6. Soft gap-penalizing and the Trail-algorithm

In the Grid-algorithm, relatively much time is spend on adding the elements of the matrices $\mathbf{M}^k$, most of which equal 0. This useless adding of zero's is avoided in the Trail-algorithm, where we first construct a trail $\mathbf{T}$ of the non-zero entries of $\mathbf{M}^1$. This trail is a 2-column array where each row contains the row- and column-indices of the non-zero entries of $\mathbf{M}^1$. Thus, for the toy-sequences $x = abac$ and $y = baca$, the (transpose of) $\mathbf{T}$ would equal

$$\mathbf{T}' = \begin{pmatrix} 1 & 1 & 2 & 3 & 3 & 4 \\ 2 & 4 & 1 & 2 & 4 & 3 \end{pmatrix} .$$

Next, for each row of $\mathbf{T}$, say the i^{th} row, we create a set of row-indices $\mathcal{D}_i$ such that

$$j \in \mathcal{D}_i \quad \text{iff} \quad (t_{i1} < t_{j1}) \wedge (t_{i2} < t_{j2}). \qquad (16)$$

Hence, $j \in \mathcal{D}_i$ implies that each common subsequence of the pair (x, y) that ends on x_i can be elongated to a longer common subsequence that ends on x_j. For the running example, this would yield $\mathcal{D}_1 = \{5, 6\}$, $\mathcal{D}_3 = \{4, 5, 6\}$,

$\mathcal{D}_4 = \{6\}$ and the remaining $\mathcal{D}_i$ being empty.
Once available, we can use the sets $\mathcal{D}_i$ to recursively construct a matrix $\mathbf{Q} = (q_{ij})$ of which each entry q_{ij} contains the number of j-long matching subsequences that start with $x_{t_{i1}} = y_{t_{i2}}$ by first defining $q_{i1} = 1$ for all i and than evaluating the next columns through using $q_{ik} = \sum_{j \in \mathcal{D}_i} q_{j(k-1)}$. Finally, we obtain $\mu(x, y) = 1 + \sum_{ij} q_{ij}$.
We formalize this procedure in the next theorem, a special case of [13, Theorem 3]

Theorem 2. *Let x and y denote nonempty strings over some alphabet Σ and let $\mathbf{T} = (t_{i1}, t_{i2})$ be constructed such that $x_{t_{i1}} = y_{t_{i2}}$ for all i. Let $\mathcal{D}_i$ be sets such that $j \in \mathcal{D}_i$ iff $(t_{i1} < t_{j1}) \wedge (t_{i2} < t_{j2})$ and let the matrix $\mathbf{Q} = (q_{ij})$ be defined by $q_{i1} = 1$ for all i and, for $i > 1$, $q_{ij} = \sum_{k \in \mathcal{D}_i} q_{k(j-1)}$. Then $\mu(x, y) = 1 + \sum_{ij} q_{ij}$.*

Proof. The proof is by induction and left to the reader. □

For strings of lengths m and n, the construction of the matrix $\mathbf{T}$ requires $m \cdot n$ operations that yield a matrix $\mathbf{T}$ with, say, k rows, k depending on the number of repetitions of characters in the strings. Then constructing the $\mathcal{D}_i$ takes $k(k-1)/2 = O(k^2)$ comparisons and each next column of $\mathbf{Q}$ requires at most $O(k^2)$ additions and there will be at most k columns in $\mathbf{Q}$. Therefore, the algorithm is of complexity $O(k^3)$. This can be quite a big number. If, on the other hand, $k = |\Sigma|$ as in comparing (partial) rankings, the complexity is quite acceptable. Therefore, the algorithm is presented in pseudo-code as Algorithm 2

Soft-penalizing gaps is easy now since we can modify line 26 of the Trail-algorithm to

$$26' : \quad q_{ij} \leftarrow q_{ij} + q_{D_{im}(j-1)} \times f(t_{i1}, t_{i2}, t_{D_{im}1}, t_{D_{im}2})$$

wherein f is an arbitrary function that can be made to operate on the gaps $t_{D_{im}1} - t_{i1}$ and $t_{D_{im}2} - t_{i2}$ in any way.

7. Soft-Matching the Characters

So far, we have dealt with "hard" matching of characters or states: a character from the one sequence matches or does not match to a particular character of another sequence, and if one or more characters do not match,

Algorithm 2: Trail-algorithm, implements theorem 2.

```
Require: $x = x_1 \dots x_r, \quad y = y_1 \dots y_c$
1: $k \leftarrow 0, \quad M \leftarrow min\{r, c\}$
2: for $i = 1$ to $r$ do
3:    for $j = 1$ to $c$ do
4:       if $x_i = y_j$ then
5:          $k \leftarrow k + 1$
6:          $I_{k1} \leftarrow k, \quad I_{k2} \leftarrow j$
7:          $\ell_k \leftarrow 0$
8:       end if
9:    end for
10: end for
11: for $i = 1$ to $k - 1$ do
12:    for $j = i + 1$ to $k$ do
13:       if $(I_{i1} < I_{j1}) \wedge (I_{i2} < I_{j2})$ then
14:          $\ell_k \leftarrow \ell_k + 1, \quad D_{i\ell_k} \leftarrow j$
15:       end if
16:    end for
17: end for
18: for $i = 1$ to $k$ do
19:    $q_{i1} \leftarrow 1$
20: end for
21: $\mu \leftarrow k$
22: for $j = 2$ to $M$ do
23:    $\mu_j \leftarrow 0$
24:    for $i = 1$ to $k$ do
25:       for $m = 1$ to $\ell_i$ do
26:          $q_{ij} \leftarrow q_{ij} + q_{D_{im}(j-1)}$
27:       end for
28:       $\mu_j \leftarrow \mu_j + q_{ij}$
29:    end for
30:    if $\mu_j = 0$ then
31:       return $\mu + 1$
32:    end if
33:    $\mu \leftarrow \mu + \mu_j$
34: end for
35: return $\mu + 1$
```

the subsequences do not match. However, in some applications it is more sensible to account for a certain degree of proximity or matching between different characters. For example, the replacement of a particular nucleotide by some others may be much more likely than average. Likewise, certain states in social demography may be less dissimilar than others; for example "Married" and "Cohabitating" may be considered as less dissimilar than "Married" and "Single".
Soft-matching means that we compare unequal characters or subsequences and this cannot be attained by evaluating the standard inner product $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}'\mathbf{y} = \sum_i \mathrm{x}_i \mathrm{y}_i$. because this involves only comparing identically indexed coordinates. However, it is well known [e.g. 32] that $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}'\mathbf{A}\mathbf{y}$ is an inner product too, provided $\mathbf{A}$ is positive semi-definite.
Now suppose that we have a square, symmetric($|\Sigma| \times |\Sigma|$)-matrix $\mathbf{M} = (m(\sigma_i, \sigma_j)) = (m_{ij})$ with $0 \leq m_{ij} \leq 1$ and $m_{ii} = 1$ and that m_{ij} denotes a "degree of matching" between characters σ_i and σ_j. Perhaps, this $\mathbf{M}$ derives from expert judgment or perhaps from empirical transition or replacement frequencies. It is convenient to assume $m(\sigma_i, \lambda) = m(\lambda, \sigma_i) = 0$ for all characters $\sigma_i \in \Sigma$. Furthermore, suppose that we define the matching $m^*(u, v)$ between two subsequences u and v as resulting from

$$m^*(u, v) = \prod_i m(u_i, v_i) \tag{17}$$

This in turn generates a matrix $\mathbf{M}^*$. An example of the structure of such a matrix is shown in Fig. 2. Clearly, if $\mathbf{M}$ is positive semi-definite, so is $\mathbf{M}^*$ and hence $\mathbf{x}'\mathbf{M}^*\mathbf{y}$ is an inner product with any of the string-representations discussed so far.

Evaluating this inner product just requires a simple modification of the Grid-algorithm: we replace its line 5 by

$$5: \quad m^1_{ij} \leftarrow m_{ij}, \quad s \leftarrow s + m_{ij}, \quad m_{ij} \leftarrow m(x_i, y_j).$$

8. Representing run-lengths

Run-length encoding is a simple data-compression technique that can be very efficient for strings in which long series of repetitions of the same characters occur. Such strings are easily compressed to much shorter sequences

Figure 2: Structure of the matrix $\mathbf{M}^*$, given an alphabet $\Sigma = \{a, b, c\}$ and lexicographic ordering of Σ^*.

$$\mathbf{M} = \begin{pmatrix} 1 & p & q \\ p & 1 & r \\ q & r & 1 \end{pmatrix}, \quad \mathbf{M}^* = \begin{pmatrix} \mathbf{M} & \dots & \mathbf{0} & \dots & \mathbf{0} & \dots \\ \vdots & \mathbf{M} & p\mathbf{M} & q\mathbf{M} & \vdots & \\ \mathbf{0} & p\mathbf{M} & \mathbf{M} & r\mathbf{M} & \mathbf{0} & \dots \\ \vdots & q\mathbf{M} & r\mathbf{M} & \mathbf{M} & \vdots & \\ & & \vdots & & \mathbf{M} & \dots \\ \mathbf{0} & \dots & \mathbf{0} & \dots & \vdots & \ddots \\ \vdots & & \vdots & & q^2\mathbf{M} & \end{pmatrix}$$

of pairs of characters and run-length counters like e.g.

$$x = aaabbccccccccccdddd \quad \mapsto \quad x' = a^3b^2c^9d^4.$$

Run-length encoding (RLE) is widely used in many different fields, for example in computational biology [34] to encode strands of DNA, in digital image processing of representations of simple graphics [31] and holograms [23] and in social demography to encode strings of life course states [29] and their durations. RLE has been amply studied [e.g. 27], in particular in string comparison algorithms that use aligning techniques [9, 15]. The complexity of these algorithms depends on both the lengths of the uncompressed strings and the lengths of the compressed strings. Let N and n denote the length of the uncompressed string x, respectively of the compressed string x' and likewise, let M and n denote the lengths of y and y'. With this notation, the very general algorithm as proposed in [9] is of complexity $O(Nm + Mn)$, i.e. the complexity still depends on the lengths of the uncompressed strings. In this section, we will present a comparison algorithm that is of complexity $O(\min\{m, n\}mn)$, i.e a Grid-based algorithm that is independent of the lengths of the uncompressed strings. This is a substantial improvement when the uncompressed sequences are long.

To discuss the algorithm, it is convenient to write a run-length or duration encoded string as a pair (x, t_x), where $x = x_1 \dots x_n$ is a string and $t_x = t_{x1}, \dots t_{xn}$ is a sequence of real-valued non-negative durations or run-lengths. Furthermore, for a subsequence $x_{i1} \dots x_{i_{|u|}} = u \sqsubseteq x$, we write

$T_x(u) = \sum_j t_{i_j}$ for the duration of the subsequence. However, $T_x(u)$ is ill-defined when multiple embeddings of the same subsequence occur as for example in $(x = abac, t_x = 2, 4, 3, 5)$: $T_x(ac) = 7$ and $T_x(ac) = 8$. Therefore, we introduce the concept of an embedding.
Let $u = u_1 \dots u_{|u|} \sqsubseteq x$. As said, the integer sequence $i_x(u) = i_1, \dots, i_{|u|}$ is called an embedding of u in x, precisely when $x_{i_1} \dots x_{i_{|u|}} = u$. Clearly, for a particular $x \in \Sigma^*$ and any $u \in \Sigma^*$, there exists a set $I_x(u)$ of such embeddings and this set may or may not be empty. Now we define the duration $T_x(u)$ of u in x as the sum of the durations of all embeddings:

$$0 \leq T_x(u) = \sum_{i_x(u) \in I_x(u)} \sum_{j \in i_x(u)} t_{xj}. \tag{18}$$

Now $T_x(u)$ is well-defined and we immediately use this definition to construct a vector $\mathbf{x} = (\mathrm{x}_1, \mathrm{x}_2, \dots)$ representing the pair (x, t_x) through

$$\mathrm{x}_{r(u)} = T_x(u) \tag{19}$$

and hence, we are interested in evaluating inner products of the form

$$\begin{aligned} \mathbf{x}'\mathbf{y} &= \sum \mathrm{x}_i \mathrm{y}_i \\ &= \sum_{u \in \mathcal{S}(x,y)} \left(\sum_{i_x(u) \in I_x(u)} \sum_{j \in i_x(u)} t_{xj} \right) \left(\sum_{i_y(u) \in I_y(u)} \sum_{j \in i_x(u)} t_{yj} \right) \qquad (20) \\ &= \sum_{u \in \mathcal{S}(x,y)} \left(|x|_u \overline{T}_x(u) \right) \left(|y|_u \overline{T}_y(u) \right), \qquad (21) \end{aligned}$$

where $\overline{T}_x(u)$ denotes the average duration of the embeddings of u in x. So, the last Equation (21) shows that we weigh the common subsequences for embedding frequency and for average duration.

For fixed u and fixed embeddings, Equation (20) reduces to

$$\left(\sum_{j\in i_x(u)} t_{xj}\right)\left(\sum_{k\in i_x(u)} t_{yj}\right) \tag{22}$$

$$=(t_{xj_1}+\ldots+t_{xj_{|u|}})(t_{yk_1}+\ldots+t_{yk_{|u|}})=P\cdot Q \tag{23}$$

Now suppose that we can elongate this fixed subsequence u with some common state σ to $u\sigma$, i.e. to a common subsequence embedding of length $|u|+1$ and, furthermore, suppose that the duration of σ in x equals a and that its duration in y equals b. Then evaluating this particular product of sums of durations requires us to evaluate

$$(t_{xj_1}+\ldots+t_{xj_{|u|}}+a)(t_{yk_1}+\ldots+t_{yk_{|u|}}+b) \tag{24}$$

$$=(P+a)(Q+b) \tag{25}$$

$$=PQ+aQ+bP+ab \tag{26}$$

Hence, most of the calculations done to evaluate Equation (23) ca be recycled when evaluating Equation (26) and the principle embodied in Equation (26) can easily be implemented as an extension of the Grid algorithm:

Theorem 3. *Let (x,t_x) and (y,t_y) denote run-length encoded strings with $x\in\Sigma^m$, $y\in\Sigma^n$ and $\Sigma=\{\sigma_1,\ldots,\sigma_d\}$. Define the the $(m\times n)$-arrays $\mathbf{M}^1=\left(m^1_{ij}\right)$, $\mathbf{U}^1=\left(u^1_{ij}\right)$, $\mathbf{V}^1=\left(v^1_{ij}\right)$ and $\mathbf{W}^1=\left(w^1_{ij}\right)$ as follows: if $x_i=y_j$,*

$$m^1_{ij}=1,\quad u^1_{ij}=t_{xi},\quad v^1_{ij}=t_{yj},\quad w^1_{ij}=t_{xi}t_{yj}$$

and otherwise $m^1_{ij}=u^1_{ij}=v^1_{ij}=w^1_{ij}=0$. For $1<k\le M=\min\{m,n\}$, define the arrays $\mathbf{M}^k=\left(e^k_{ij}\right)$, $\mathbf{U}^k=\left(u^k_{ij}\right)$, $\mathbf{V}^k=\left(w^k_{ij}\right)$ and $\mathbf{W}^k=\left(v^k_{ij}\right)$

through

$$m_{ij}^k = m_{ij}^1 \sum_{a>i,b>j} m_{ab}^{k-1}, \tag{27}$$

$$u_{ij}^k = m_{ij}^1 \left(\sum_{a>i,b>j} u_{ab}^{k-1} + m_{ij}^k t_{xi} \right), \tag{28}$$

$$v_{ij}^k = m_{ij}^1 \left(\sum_{a>i,b>j} v_{ab}^{k-1} + m_{ij}^k t_{yj} \right), \tag{29}$$

$$w_{ij}^k = m_{ij}^1 \left(\sum_{a>i,b>j} \left(w_{ab}^{k-1} + t_{xi} v_{ab}^{k-1} + t_{yj} u_{ab}^{k-1} \right) + m_{ij}^k t_{xi} t_{yj} \right). \tag{30}$$

Let $\mathbf{x}$ *and* $\mathbf{y}$ *denote the representing vectors, constructed according to Eqn.* (20)*, of the run-length encoded strings* (x, t_x) *and* (y, t_y)*. Then*

$$\mathbf{x}'\mathbf{y} = 1 + \sum_{k=1}^{M} \sum_{i,j} w_{ij}^k \tag{31}$$

Proof. The proof is by induction and is left to the reader. □

In Fig. 3, we illustrate the algorithm implied by Theorem 3 by applying the Theorem to the RLE-strings ($x = abac, t_x = (2,3,4,2)$) and ($y = baca, t_y = (2,4,3,2)$).

Actually implementing the algorithm implied by Theorem 3 amounts to an extension of the code sketched in Algorithm 1. This extension is quite straightforward when the calculations are organized according to the following scheme: after initializing the matrices $\mathbf{M}$, $\mathbf{U}$, $\mathbf{V}$ and $\mathbf{W}$, we first we

Figure 3: Grid-algorithm for RLE-strings: Matrices (omitting initial zero's) resulting from Theorem 3 applied to $(x = abac, t_x = (2,3,4,2))$ and $(y = baca, t_y = (2,4,3,2))$.

k	$k=1$	$k=2$	$k=3$
$\mathbf{M}^k$	$\begin{pmatrix} & 1 & & 1 \\ 1 & & & \\ & 1 & & 1 \\ & & 1 & \end{pmatrix}$	$\begin{pmatrix} & 2 & & 0 \\ 3 & & & \\ & 1 & & 0 \\ & & 0 & \end{pmatrix}$	$\begin{pmatrix} & 0 & & 0 \\ 1 & & & \\ & 0 & & 0 \\ & & 0 & \end{pmatrix}$
$\mathbf{U}^k$	$\begin{pmatrix} & 2 & & 2 \\ 3 & & & \\ & 4 & & 4 \\ & & 2 & \end{pmatrix}$	$\begin{pmatrix} & 10 & & 0 \\ 19 & & & \\ & 6 & & 0 \\ & & 0 & \end{pmatrix}$	$\begin{pmatrix} & 0 & & 0 \\ 9 & & & \\ & 0 & & 0 \\ & & 0 & \end{pmatrix}$
$\mathbf{V}^k$	$\begin{pmatrix} & 4 & & 2 \\ 2 & & & \\ & 4 & & 2 \\ & & 3 & \end{pmatrix}$	$\begin{pmatrix} & 13 & & 0 \\ 15 & & & \\ & 7 & & 0 \\ & & 0 & \end{pmatrix}$	$\begin{pmatrix} & 0 & & 0 \\ 9 & & & \\ & 0 & & 0 \\ & & 0 & \end{pmatrix}$
$\mathbf{W}^k$	$\begin{pmatrix} & 8 & & 4 \\ 6 & & & \\ & 16 & & 8 \\ & & 6 & \end{pmatrix}$	$\begin{pmatrix} & 64 & & 0 \\ 95 & & & \\ & 42 & & 0 \\ & & 0 & \end{pmatrix}$	$\begin{pmatrix} & 0 & & 0 \\ 81 & & & \\ & 0 & & 0 \\ & & 0 & \end{pmatrix}$

calculate

$$m_{ij}^{k} = \sum_{a>i,\ b>j} m_{ab}^{k-1} \tag{32}$$

$$u_{ij}^{k} = \sum_{a>i,\ b>j} u_{ab}^{k-1} \tag{33}$$

$$v_{ij}^{k} = \sum_{a>i,\ b>j} v_{ab}^{k-1} \tag{34}$$

$$w_{ij}^{k} = \sum_{a>i,\ b>j} w_{ab}^{k-1} \tag{35}$$

$$\tag{36}$$

through the addition-principle demonstrated in Algorithm 1 and then update the matrices through calculating (in the order as indicated below)

$$e_{ij}^{k} = e_{ij}^{1} e_{ij}^{k}, \tag{37}$$

$$w_{ij}^{k} = e_{ij}^{1} \left(w_{ij}^{k} + u_{ij}^{1} v_{ij}^{k} + v_{ij}^{1} u_{ij}^{k} + e_{ij}^{k} w_{ij}^{1} \right), \tag{38}$$

$$u_{ij}^{k} = e_{ij}^{1} \left(u_{ij}^{k} + u_{ij}^{1} e_{ij}^{k} \right), \tag{39}$$

$$v_{ij}^{k} = e_{ij}^{1} \left(v_{ij}^{k} + v_{ij}^{1} e_{ij}^{k} \right). \tag{40}$$

Using this approach, it is also immediate how to extend the Trail-algorithm to handling RLE-strings.
Clearly, complexity of the algorithm is, despite the extensions due to the added computation of the matrices $\mathbf{U}$, $\mathbf{V}$ and $\mathbf{W}$, still of order $O(\min\{m, n\}mn)$. The reader notes that, because of the multiplications in Equations (28)-(30), the numbers in the entries of the matrices can become quite big. On the other hand, the counter-sequences t_x and t_y are in no way confined to be integers. Therefore, it may be convenient to scale these sequences in such a way that the average or median counter-value equals 1.0. Finally, the reader notes that the numbers in the counter-sequences t_x need not refer to just run-length or duration; they could be used to convey any quantifiable property of the characters. An example of such a quantifiable property is in comparing rankings: we know that common sub-ranks of highly ranked items should get more weight than common sub-ranks that mainly consist of low-ranked items. So, we could assign a weight to a each item in a ranking, the weight being some function of its position.

9. Conclusions

So far, we presented two very flexible kernels to compare strings, along with pseudo-code to implement them. We discussed variations of the Grid-algorithm to accommodate applications in social demography, in computational biology and in decision making. The counting of gap-restricted embeddings, as described in Equations (14) and (15), was mentioned as an open problem in [12]. All of these variations could have been implemented as variations of the Trail-algorithm too; we leave this as a small challenge to the interested reader who wants to compare strings that only have few character repetitions.
An extension of the Grid-algorithm, embodied in Theorem 3 and illustrated in Figure 3, compares strings with a quantified character feature. It is natural to interprete this feature as "run-length" or "duration" but other interpretations are possible, for example when comparing rankings. Remarkably, the time-complexity of this extension is only $O(\min\{m,n\}mn)$ and fully independent of the length of the "uncompressed" strings.
The Gramm matrix that results from application of kernels, especially of sophisticated string kernels, is known to have relatively big diagonal elements, expressing the fact that strings have much more features in common with themselves than with any other string, the more so when the strings are very long. This may result in small values of similarity metrics like for example

$$s(x,y) = \frac{\mathbf{x}'\mathbf{y}}{\mathbf{x}'\mathbf{x} + \mathbf{y}'\mathbf{y} - \mathbf{x}'\mathbf{y}}.$$

However, in most applications, there is no objection in applying some concave transformation to the inner products before calculating similarity, at the risk of losing the metric properties [7] of the similarity. For more sophisticated methods to deal with this problem in the context of machine learning and SVM's the reader is referred to [36, 43].

Acknowledgements

The paper gained considerable clarity thanks to the discussions with Matthias Studer in the course of applying the algorithms discussed in this paper to the analysis of life course data and implementing them into the TraMineR-software[1] [16].

[1]TraMineR is freely available from http://mephisto.unige.ch/traminer/

References

[1] N. Ailon, Aggregation of partial rankings, p-ratings and top-m lists, Algorithmica 57 (2) (2010) 284–300.

[2] M. S. Bansal, D. Fernández-Baca, Computing distances between partial rankings, Information Processing Letters 109 (4) (2009) 238–241.

[3] M. Barski, U. Stege, A. Thomo, C. Upton, Shortest path approaches for the longest common subsequence of a set of strings., in: Proceedings of the 7th IEEE International Conference on Bioinformatics and Bioengineering (BIBE2007), IEEE, 2007.

[4] A. Ben-Hur, C. S. Ong, S. Sonnenburg, B. Schölkopf, G. Rätsch, Support vector machines and kernels for computational biology, PLoS Computational Biology 4 (10) (2008) .

[5] H. Bras, A. C. Liefbroer, C. H. Elzinga, Standardization of pathways to adulthood? An analysis of Dutch cohorts born between 1850 and 1900, Demography 47 (4) (2010) 1013–1034.

[6] C. Brzinsky-Fay, Lost in transition? Labour market entry sequences of school leavers in Europe, European Sociological Review 23 (4) (2007) 409–422.

[7] S. Chen, B. Ma, K. Zhang, On the similarity metric and the distance metric, Theoretical Computer Science 410 (24-25) (2009) 2365–2376.

[8] M. Crochemore, L. Illiec, W. Rytter, Repetitions in strings: Algorithms and combinatorics, Theoretical Computer Science 410 (50) (2009) 5227–5235.

[9] M. Crochemore, G. Landau, M. Ziv-Ukelson, A subquadratic sequence alignment algorithm for unrestricted scoring matrices, SIAM Journal of Computing 32 (6) (2003) 1654–1673.

[10] C. H. Elzinga, Combinatorial representation of token sequences, Journal of Classification 22 (1) (2005) 87–118.

[11] 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.

[12] C. H. Elzinga, S. Rahmann, H. Wang, Algorithms for subsequence combinatorics, Theoretical Computer Science 409 (3) (2008) 394–404.

[13] C. H. Elzinga, H. Wang, Z. Lin, Y. Kumar, Concordance and consensus, Infomation Sciences 181 (2011) 2529–2549.

[14] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, E. Vee, Comparing partial rankings, SIAM Journal of Discrete Mathematics 20 (3) (2006) 628–648.

[15] V. Freschi, A. Bogliolo, Longest common subsequence between run-length-encoded strings: a new algorithm with improved parallelism, Information Processing Letters 90 (2004) 167–173.

[16] A. Gabadinho, G. Ritschard, N. S. Müller, M. Studer, Analyzing and visualizing state sequences in R with TraMineR, Journal of Statistical Software 40 (4) (2011) 1–37.

[17] D. M. Gusfield, Algorithms on Strings, Trees and Sequences, Computer Science and Computational Biology, Cambridge University Press, Cambridge, 1997.

[18] D. Haussler, Convolution kernels on discrete structures, Tech. Rep. UCSC-CRL-99-10, University of California, Computer Science Department, Santa Cruz (1999).

[19] M. N. Hollister, Is optimal matching sub-optimal?, Sociological Methods & Research 38 (2009) 235–264.

[20] A. Iványi, On the d-complexity of words, Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös nominatae,Sectio Computatorica 8 (1987) 69–90.

[21] Z. Kása, On the d-complexity of strings, Pure Mathematics and Applications 1-2 (1998) 119–128.

[22] Z. Kása, On scattered subwords, Acta Universitatis Sapientiae, Informatica 3 (2011) 127–136.

[23] S.-C. Kim, E.-S. Kim, Fast computation of hologram patterns of a 3D object using run-length encoding and novel look-up table methods, Applied Optics 48 (6) (2009) 1030–1041.

[24] D. E. Knuth, The Art of Computer Programming: Fundamental Algorithms, vol. 1, 3rd ed., Addison Wesley, Reading (MA), 1997.

[25] D. E. Knuth, Two notes on notation, in: Selected Papers on Discrete Mathematics, chap. 2, Center for the Study of Language and Information, Stanford (MA), 2003, pp. 15–44.

[26] F. Legendre, T. Robillard, L. Desutter-Grandcolas, M. F. Whiting, P. Grandcolas, Phylogenetic analysis of non-stereotyped behavioural sequences with a successive event-pairing method., Biological Journal of the Linnean Society 94 (2008) 853–867.

[27] D. Lemire, O. Kaser, K. Aouiche, Sorting improves word-aligned bitmap indexes, Data & Knowledge Engineering 69 (1) (2010) 3–28.

[28] C. Leslie, E. Erskin, W. S. Noble, The spectrum kernel: a string kernel for SVM proteine classification, in: R. Altman, A. Dunker, L. Hunter, K. Lauderdale, T. Klein (eds.), Proceedings of the Pacific Symposium of Biocomputing, No. 7, World Scientific Publishing Company, 2002.

[29] A. C. Liefbroer, C. H. Elzinga, Intergenerational transmission of behavioral patterns: How similar are parents'and children's demographic trajectories, Advances in Life Course Research 17 (1) (2012) 1–10.

[30] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, C. Watkins, Text classification using string kernels, Journal of Machine Learning Research 2 (2002) 419–444.

[31] C. Messom, G. Sen Gupta, S. Demidenko, Hough transform run length encoding for real-time image processing, IEEE Transactions on Instrumentation and Measurement 56 (3) (2007) 962–967.

[32] C. D. Meyer, Matrix Analysis and Applied Linear Algebra, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000.

[33] S. Pakalapati, S. Beaver, J. A. Romagnoli, A. Palazoglu, Sequencing diurnal air flow patterns for ozone exposure assessment around Houston, Texas, Atmospheric Environment 43 (3) (2009) 715–723.

[34] S. Rahmann, Subsequence combinatorics and applications to microarray production, DNAsequencing and chaining algorithms., in: M. Lewenstein, G. Valiente (eds.), Combinatorial Pattern Matching (CPM), Lecture Notes in Computer Science(LNCS), Springer, New York, 2006, pp. 153–164.

[35] D. Sankoff, J. B. Kruskal (eds.), Time Warps, String Edits and Macro-Molecules. The Theory and Practice of String Comparison., Reading: Addison-Wesley, 1983 (1983).

[36] B. Schölkopf, J. Weston, E. Eskin, C. Leslie, W. S. Noble, A kernel approach for learning from almost orthogonal patterns, in: Machine Learning: ECML 2002. Lecture Notes in Computer Science, vol. 2430, Springer, 2002, pp. 151–167.

[37] J. Shawe-Taylor, N. Christianini, Kernel Methods for Pattern Recognition, Cambridge University Press, Cambridge, 2004.

[38] M. Studer, Analyse de données séquentielles et application à l'étude des inégalités sociales en début de carrière académique, Ph.D. thesis, Faculté des sciences économiques et sociales de l'Université de Genève (2012).

[39] 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.

[40] H. Wang, Z. Lin, A novel algorithm for counting all common subsequences, in: 2007 IEEE Conference on Granular Computing, IEEE, 2007.

[41] M. S. Waterman, Introduction to Computational Biology, Interdisciplinary Statistics, Chapman & Hall/CRC, Boca Raton, 1995.

[42] C. Watkins, Dynamic alignment kernels, in: A. Smola, P. Bartlett, B. Schölkopf, D. Schuurmans (eds.), Advances in Large Margin Classifiers, MIT Press, 2002, pp. 39–50.

[43] J. Weston, B. Schölkopf, E. Eskin, C. Leslie, W. Stafford Noble, Dealing with large diagonals in kernel matrices, Anals of the Institute of Statistical Mathematics 55 (2) (2003) 391–408.