Quantifying Sequential Subsumption

Hui Wang*, Cees H. Elzinga†, Zhiwei Lin, Jordan Vincent

Abstract

Subsumption is used in knowledge representation and ontology to describe the relationship between concepts. Concept A is subsumed by concept B if the extension of A is always a subset of the extension of B, irrespective of the interpretation. The subsumption relation is also useful in other data analysis tasks such as pattern recognition – for example in image analysis to detect objects in an image, and in spectral data analysis to detect the presence of a reference pattern in a given spectrum. Sometimes the subsumption relation may not be 100% true, so it is useful to quantify this relationship.

In this paper we study how to quantify subsumption for sequential patterns. We review existing work on subsumption, give an axiomatic characterisation of subsumption, and present one general approach to quantification in terms of set intersection operation over concept extension. Constructing the concept extension set explicitly is impossible without specifying the domain of discourse and the interpretation. Instead, we focus on concept intension for sequences as patterns and propose to represent concept intension of a sequence by its subsequences. We further consider different types of concept intension set – subsequence set, subsequence multiset, embedding set and embedding set with constraints such as warping and selection. We then present a general algorithmic framework for computing set intersections, and specific algorithms for computing different concept intension sets. We also present an experimental evaluation of these algorithms with regard to their runtime performance.

Keywords: Subsumption characterisation; Subsumption quantification; Sequence and subsequence analysis; Sequence similarity; Grid algorithm

1 Introduction

According to Oxford Dictionaries, "*[to] subsume is to include or absorb something in something else*", and "*subsumption is the act of subsuming or the state of being subsumed*". Here "thing" can be anything but typically it is a concept, which is a set of all objects that share a set of properties or, dually, a set of all properties that are shared by a set of objects [1]. The notion of subsumption is often used in knowledge engineering, description logic and ontology, and formal concept analysis [8, 35, 11, 29, 22] to describe the generality relationship between concepts. Informally concept A is 'subsumed by' concept B if A is 'a kind of' B. For example, a Chair is a kind

*Hui Wang gratefully acknowledges support by European Union's Horizon 2020 research and innovation programme under grant agreement Nos 690238 and 700381. Ulster University, h.wang@ulster.ac.uk

†Cees H. Elzinga gratefully acknowledges support by European Research Council under grant agreement No 324178 FP7/ERC (Project: Contexts of Opportunity, PI: Aart C. Liefbroer)

[1] https://en.wikipedia.org/wiki/Formal_concept_analysis (Accessed 7 July 2018).

of Furniture [2]. Academic studies of subsumption usually use the notion of concept intension and extension. Thus, concept A is subsumed by concept B if the extension of A is always a subset of the extension of B, irrespective of the interpretation of the concepts. This is achieved by the properties they possess or, simply, their intensions.

The subsumption relation is also useful in other applications such as chemometrics, and pattern recognition when we need to check if some object of interest is present in a sample under consideration. In chemometrics (aka spectral data analysis) [30], we may, for example, want to know if hydrogen is present in Earth's atmosphere. When light travels through Earth's atmosphere, it will be absorbed at some wavelength depending on what atoms there are in the atmosphere. We can thus sense the atmosphere using a spectrometer and represent the sensing result as the solar spectrum (called *target* pattern). We can also represent hydrogen by its emission/absorption spectrum (called *reference* pattern) (see Figure 1). We now want to check if the reference pattern is included in the target pattern. It is clearly not appropriate to do so by measuring the similarity of these two spectra. We believe it is appropriate to do so by measuring the degree to which the reference pattern is included in the target pattern; or simply the degree of subsumption. In another application, facial recognition [1], we need to determine if a person is present in a scene[3]. We can sense the scene and the person's face by images, represent the scene image as a feature vector (such as LBP image descriptor) (again, called *target* pattern) and represent the face image as another feature vector using the same features (again, called *reference* pattern), and finally check if the reference pattern (the person) is included in or *subsumed by* the target pattern (the scene).

In general, a pattern is a salient combination of properties, the saliency deriving from its frequency or from the way we use it to define classes. When a pattern is interpreted as the set of all objects in a domain satisfying these properties, the pattern is a concept in the sense of formal concept analysis [29]. The properties may be structured or unstructured, as follows:

- (Unstructured: Set) The pattern consists of an unstructured set of properties, for example, $\{x = 5, y > 18, z = -3\}$. This pattern can be represented as $\{A, B, C\}$. Therefore pattern $\{A, B, C\}$ is a subsumption of $\{A, B\}$, or $\{B, A\}$.

- (Structured: Vector) The pattern consists of an ordered set of properties where the position index provides meanings for the property values, so the pattern representation can be simplified. For example, pattern $\{x_1 = 2, x_2 = 1, x_3 = 5\}$ can be represented simply as a vector $< 2, 1, 5 >$ or $< A, B, C >$ for short when the position index provides meanings of the values A, B, C. Therefore pattern $< A, B, C >$ is a subsumption of $< A, B, * >$ but not $< B, A, * >$ where $*$ means any.

- (Structured: Sequence) The pattern consists of an ordered set of properties that have a spatial or temporal ordering where the order of the properties provide meanings for the property values. For example, an emission spectrum $\{x_1 =$

[2] We thank one anonymous reviewer for suggesting this example.

[3] We usually achieve this through machine learning: we gather a large number of images as the training data set – some contain this person's face and some do not. We train a (SVM, neural network, decision tree, etc) model and use it to decide if the person is present or not. This approach needs a large set of images for training, which is not always possible. For example, there is growing demand for computer systems to help immigration officers at airports to verify identities of passengers by comparing photos on their passports against their face images on the spot. There are not enough images about each passenger, and there is not enough time to build a model in real time.

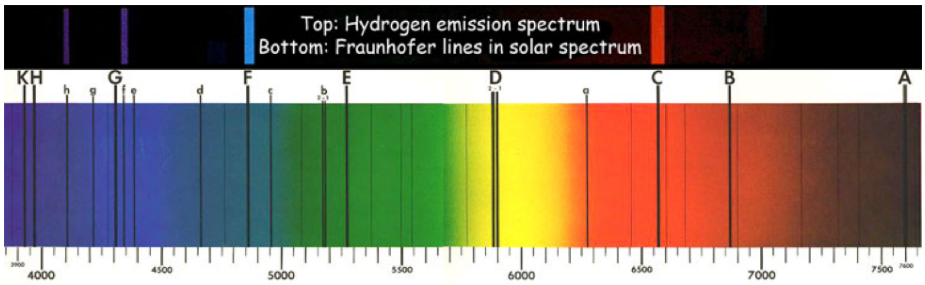


Figure 1: A comparison of the hydrogen emission spectrum with the Fraunhofer absorption lines in the solar spectrum. The numbers at the bottom of the image correspond to the wavelength of various kinds of light measured in Ångstroms (millionths of a centimeter). The hydrogen emission lines are called (from right to left, or red to violet) H-alpha, H-beta, H-gamma, and H-delta. The H-alpha and -beta emission lines correspond exactly to the C and F Fraunhofer lines in the solar spectrum, and the H-gamma and -delta emission lines correspond exactly to the f and h Fraunhofer lines, thereby demonstrating the presence of hydrogen atoms in the solar atmosphere. *Source: https://cseligman.com/text/physics/absorptionemission.htm.* To get a computer to make this decision, hydrogen and Earth's atmosphere should be represented (here as spectra) and then calculate the degree of subsumption between the two representations. If the degree is high enough, it can be concluded that hydrogen is present in Earth's atmosphere.

$2, x_2 = 1, x_3 = 5\}$ where x_i means light intensity at wavelength i, a time series $\{x_1 = 2, x_2 = 1, x_3 = 5\}$ where x_i means price on day i. Such patterns are called sequential patterns and are denoted by e.g. ABC. Therefore ABC is a subsumption of AC but not CA. More real world examples of sequential patterns can be found in [12, 28, 13].

The subsumption relation may not be perfect in cases such as the examples above, so it is useful to quantify this relationship. The bulk of the (recent) literature related to subsumption is in the field of fuzzy description logics where the problem of reasoning in the context of an ontology requires (preliminary) decisions about *degree* of subsumption in ontology alignment or ontology integration. See e.g. Borgwardt & Penaloza [9, 6]. Lots of applications can be found in the field of medical ontologies where vague terminology is abundant (e.g. about discolorations and deformations of skin or other tissues, life periods like "perinatal", "infanthood" and "adolescence". In navigating in fuzzy ontologies and in aligning different ontologies, the degree of subsumption is important. Bobillo and Straccia (2008) pointed out [7] : "*all instances of a concept C are instances of a concept D to degree . . .*".

As far as the authors are aware, there is no work in the literature on quantification of subsumption. Any quantification must be done in a specific setting. This paper aims to fill this gap in a specific setting where the properties in a pattern are ordered or, in other words, the patterns are sequences of properties. This setting is quite broad. In the case of image analysis, images (reference or target) can be represented as vectors of properties (using image descriptors such as LBP or HOG [4]) which may be viewed as sequences if the chosen descriptors are ordered, e.g. the LBP patterns corresponding to different radii/scales. In the case of spectral data analysis, spectra can be represented as sequences – intensity values arranged in the order of their corresponding wavelengths.

This setting is potentially valid for many other applications where sampling at different spatial and temporal scales is used in data representation.

To present our proposal, we organise this paper as follows. In the next section we give an overview of subsumption including a review, an axiomatic characterisation and a quantification in terms of set inclusion and set intersection operations, with different interpretations. We then provide formulation of subsumption for a particular case where concepts/patterns can be viewed as sequences. We also provide efficient algorithms for calculating subsequence set intersections. Finally we present an evaluation of the algorithms in terms of their runtimes.

2 Subsumption

In this section we present a review of subsumption and characterisations of subsumption in the contexts of sets and sequences.

2.1 A review of subsumption

The notion of subsumption is often used in knowledge engineering, description logic and ontology [8, 35, 11, 3] to describe the generality relationship between concepts. Informally concept A is subsumed by concept B if the extension of A is always a subset of the extension of B, irrespective of the interpretation. A more formal definition can be given in terms of model theory and formal language [34].

From a computer science point of view, model theory is the study of models of theories in a formal language. A set of sentences in a formal language (such as first order logic) is a theory; a model of a theory is a structure (e.g. groups, fields, graphs, universes of set theory) that satisfies the sentences of that theory through an interpretation. A formal language describes the world in terms of *instances* which are things with individual identities, *properties* of instances that distinguish them from other instances, and *relations* that hold among sets of instances. First Order Logic is a formal language, consisting of terms (including constants and variables) and sentences (predicates applied to zero or more terms, simple or compound). An interpretation is a function that maps a term to an instance in the *domain of discourse* and a predicate to a relation in the domain. If a predicate has only one argument (i.e., it is applied to only one term), it is then mapped to a unary relation which is a set of instances.

Concepts can be formally described by a formal language as predicates. The intension of a concept is the description of the predicate, and the extension is given by a model-theoretic interpretation of the predicate (i.e. a unary relation). The extension of a concept consists of the instances in the domain to which the concept applies, and its intension consists of the properties that are possessed by all the instances in the extension.

There are various formal definitions of subsumption. Perhaps the best definition of subsumption is that of William Woods [34]. He gives definitions of 5 different versions of subsumption: extensional, structural, recorded, axiomatic and deduced. The first two versions are relevant, so we quote and paraphrase them here.

Definition 1 (Extensional subsumption [34])**.** *One concept A is said to subsume another concept B if the extension of the subsuming concept A contains the extension of the subsumed concept B.*

Definition 2 (Structural Subsumption [34])**.** *The subsuming concept is more general than the subsumed concept by virtue of the formally specified subsumption criteria applied to the structure of the descriptions of the concepts.*

One view is that subsumption is fundamentally extensional, but extensional subsumption can be established through an inspection of the structure of the descriptions of the concepts.

Falquet [20] and Ferilli et al. [21] provide definition of subsumption in the context of description logic and ontology, which is quoted and paraphrased below.

Definition 3 (Subsumption [20, 21])**.** *Let U be a domain of discourse (i.e., the set of all instances in the world of interest), let C be the set of concepts, and let $f : C \rightarrow 2^U$ be an interpretation function. Let $r, t \in C$ be two concepts. r is subsumed by t iff for any U and any interpretation function f,*

$$f(r) \subseteq f(t).$$

Other definitions of subsumption in the context of ontology are provided in [23, 27].

Subsumption is traditionally used in term subsumption languages [35, 11]. Brachman and Levesque [11] described a simple term subsumption language that has a sound and complete algorithm for the subsumption test, and has a well defined logic-based semantics. Due to its formal semantics, a term subsumption system can automatically infer the subsumption relationship between concepts defined in the system. He provides the following definition, which is both extensional and structural.

Definition 4 (Term subsumption [11])**.** *Type A subsumes type B if, by virtue of the form of A and B, every instance of B must be an instance of A. In other words, it can be determined that being a B is implicit in being an A, based only on the structure of the two terms.*

For example, without any world knowledge, we can determine that the type (or concept) 'person' subsumes the type (or concept) 'person each of whose male friends is a doctor'.

Subsumption test is a major operation in a term subsumption system, which determines whether a concept subsumes (i.e., is more general than) another. John Yen [35] generalises term subsumption languages to using fuzzy logic, where concept c_1 subsumes concept c_2 if and only if the extension of c_1 is a fuzzy superset of the extension of c_2.

Borgwardt et al. [8, 9, 10] study the complexity of subsumption reasoning in description logics (DLs) and fuzzy DLs. It was shown that the complexity of reasoning in finitely valued fuzzy DLs is often not higher than that of the underlying classical DL. However this does not hold for fuzzy extensions of the light-weight DL $\mathcal{EL}$, which is used in many biomedical ontologies such as SNOMED CT and Gene Ontology. The complexity of reasoning increases from polynomial to exponential, even if only one additional truth value is introduced.

2.2 Axiomatic characterisation of subsumption

Let $r, t \in C$ be any two concepts and $f : C \rightarrow 2^U$ be an interpretation. According to the definition of subsumption (Def. 3), we have that t subsumes r, precisely

when $f(r) \subseteq f(t)$; otherwise, there is no such subsumption. Now let us consider the following truth-function:

$$T(t \text{ subsumes } r) = \begin{cases} 1 & \text{if } f(r) \subseteq f(t) \\ 0 & \text{if } f(r) \not\subseteq f(t) \end{cases}, \tag{1}$$

which is equivalent to stating that

$$T(t \text{ subsumes } r) = \begin{cases} 1 & \text{if } f(r) \cap f(t) = f(r) \\ 0 & \text{if } f(r) \cap f(t) \subset f(r) \end{cases}, \tag{2}$$

Apparently, the value of the binary truth-function depends on the nature of the intersection of the extensions of r and t. Quantifying "imperfect" subsumption is intended to take into account that some but not all elements of $f(r)$ are also elements of $f(t)$, that there is a wide range of possibilities between the extremes $f(r) \cap f(t) = \emptyset$ and $f(r) \cap f(t) = f(r)$. This is accomplished by creating a multi-valued "truth-function" $s(r,t) : |f(r) \cap f(t)| \rightarrow [0,1]$ that satisfies the following properties, which are collectively called *Axioms of Subsumption*:

Axiom 1 (Axioms of Subsumption)**.**

1. $s(r,t) = 1$ *if* $f(r) \cap f(t) = f(r)$
2. $0 < s(p,t) < s(r,t) < 1$ *if* $|f(p) \cap f(t)| < |f(r) \cap f(t)|$
3. $s(r,t) = 0$ *if* $f(r) \cap f(t) = \emptyset$

Of course, there are many such truth functions that satisfy the above requirements. For example

$$s(r,t) = \left(\frac{|f(r) \cap f(t)|}{|f(r)|} \right)^w, \qquad w > 0, \tag{3}$$

$$s(r,t) = \begin{cases} 1 & \text{if } f(r) \cap f(t) = f(r) \\ \frac{|f(r) \cap f(t)|}{c + |f(r) \cap f(t)|} & \text{otherwise} \end{cases}. \tag{4}$$

and

$$s(r,t) = \begin{cases} 1 & \text{if } f(r) \cap f(t) = f(r) \\ 1 - e^{-|f(r) \cap f(t)|} & \text{otherwise} \end{cases}. \tag{5}$$

each satisfy the above requirements.

2.3 Subsumption and operations on sets

We have shown that subsumption can be formulated through the notion of set-inclusion $\subseteq$, and quantified through the notion of set-intersection $\cap$. Therefore, the notions of set and set-operations are crucial to the understanding and quantification of subsumption. It is thus worthwhile to be a bit more precise about what exactly is meant by using these notions of sets and set-operators. Sets can be specified, relative to some universe U, through their generating function: every set A is determined by a generating function 1_A that is defined as

$$\forall x \in U : \; 1_A(x) = \begin{cases} 1 & \text{if } x \in A, \\ 0 & \text{otherwise} \end{cases}$$

with $|A| = \sum_{x \in U} 1_A(x)$. Set inclusion is decided by $A \subseteq B$ iff $\forall x \in A : 1_A(x) \leq 1_B(x)$. The intersection $A \cap B$ is generated by the pointwise product $1_{A \cap B} = 1_A \cdot 1_B$ and, analogously,

$$|A \cap B| = \sum_x 1_A(x) \cdot 1_B(x) \stackrel{def}{=} \phi(A, B). \tag{6}$$

An alternative but equivalent definition of the intersection generating function is the pointwise $1_{A \cap B} = \min\{1_A, 1_B\}$.

However, this concept of the simple set may not be adequate for the application of subsumption in the context of, for example, face recognition or spectral analysis. One reason is that often certain features have multiple occurrences like moles and scars in a face or patterns of energy peaks along the wavelength axis. Therefore, it seems wise to consider $f(t)$ and $g(t)$ as multisets [4] [5, 32]: pairs of a simple set and a multiplicity function χ. For example, the multiset $\{2,1,2,2,3,3,5,5,1,5,5\}$ consists of the (simple) set $\{1,2,3,5\}$ and the multiplicity function defined by $\chi(1) = 2, \chi(2) = 3, \ldots$

If A is a set and $\chi : A \mapsto \mathbb{N}$ is a function, the pair (A, χ_A) is a multiset; the simple set A is a special case since $(A, 1_A)$ is a multiset too. A is called the domain or support of the multiset. If the extensions $f(r)$ and $f(t)$ are interpreted as multisets, we need to say something about multiset inclusion and multiset intersection.

To define set inclusion, we generalize the definition for simple sets: let $\mathcal{A} = (A, \chi_A)$ and $\mathcal{B} = (B, \chi_B)$ denote multisets, then $\mathcal{A} \subseteq \mathcal{B}$ iff $\forall x \in B : \quad \chi_B(x) \leq \chi_A(x)$. For set intersection the generalization of the different constructions for simple sets lead to quite different results. The first construction, $1_{A \cap B} = 1_A \cdot 1_B$, is congruent with the truth function τ for logical conjunction $\wedge$ as can be seen from

		$\tau(x \in B) = 1_B(x)$	
		0	1
$\tau(x \in A) = 1_A(x)$	0	0	0
	1	0	1

Extending this construction to multisets, we write $\chi_{\mathcal{A} \cap \mathcal{B}} = \chi_{\mathcal{A}} \cdot \chi_{\mathcal{B}}$:

		$\chi_{\mathcal{B}}$	
		0	$\chi_{\mathcal{B}}(x)$
$\chi_{\mathcal{A}}$	0	0	0
	$\chi_{\mathcal{A}}(x)$	0	$\chi_{\mathcal{A}} \cdot \chi_{\mathcal{B}}(x)$

The quantity $\chi_A(x) \cdot \chi_B(x)$ equals the number of matches of elements of "type x" when we compare each element of A with each element of B. Under this interpretation, we have that

$$|\mathcal{A} \cap \mathcal{B}| = \sum_x \chi_{\mathcal{A}}(x) \cdot \chi_{\mathcal{B}}(x) \stackrel{def}{=} \mu(\mathcal{A}, \mathcal{B}) \tag{7}$$

So, this generalization through extending the truth-function from binary to multi-valued, yields a "count of matches".

The second construction to generate the intersection of sets, $1_{A \cap B} = \min\{1_A, 1_B\}$, is also straightforward to generalize as $1_{\mathcal{A} \cap \mathcal{B}} = \min\{\chi_{\mathcal{A}}, \chi_{\mathcal{B}}\}$ with

$$|\mathcal{A} \cap \mathcal{B}| = \sum_x \min\{\chi_{\mathcal{A}}(x), \chi_{\mathcal{B}}(x)\} \stackrel{def}{=} \psi(\mathcal{A}, \mathcal{B}) \tag{8}$$

[4] In computer science, multisets are also known as "bags", "heaps" or "assemblies".

and the latter construction yields a "count of shared elements", numerically quite different from the above "count of matches". Although conceptually and numerically quite different, the reader notes that

$$\phi(\mathcal{A},\mathcal{B}) = 0 \text{ iff } \psi(\mathcal{A},\mathcal{B}) = 0 \text{ iff } \mu(\mathcal{A},\mathcal{B}) = 0, \tag{9}$$

$$\phi(\mathcal{A},\mathcal{B}) \leq \psi(\mathcal{A},\mathcal{B}) \leq \mu(\mathcal{A},\mathcal{B}). \tag{10}$$

The count ψ has, for good reasons, become the standard way of gauging multiset-intersection. But the decision which of the above counts, ϕ, ψ or μ, to use in an actual application should not be justified by that fact: such a decision should be based on the requirements of the application at hand.

2.4 What's next?

Subsumption has been formally characterised in terms of concept extension. In practice, however, concept extensions are not accessible without a specification of the domain of discourse and the interpretation function. We will need to establish the subsumption relation 'by virtue of concept descriptions'.

In the rest of this paper we consider a particular case where a concept (i.e. pattern) is represented by an ordered set of properties or feature values – a sequence. We show that, instead of using concept extension, we can use subsequence relation to determine subsumption. The set intersection operation, needed to quantify subsumption, can be computed efficiently by the algorithms presented in this paper.

3 Subsumption based on sequences

In this section we consider the case where the concepts are (described or represented by) sequences. So, we introduce some concepts and notation pertaining to sequences and subsequences. Then we will specialise subsumption in this specific case. Finally, we will focus on various ways to adapt or constrain straightforward counts of intersections to the needs of specific applications.

3.1 Preliminaries on sequences

Let $\Sigma = \{\sigma_1, \ldots, \sigma_d\}$ be a finite *alphabet* of *symbols*, and let $\Sigma^\star$ denote its Kleene closure [2] that is constructed from the symbols of Σ by concatenation. We say that a string $x = x_1 \cdots x_n$ has *length* $|x| = n$ or that x is n*-long* if it consists of n, not necessarily distinct, symbols from Σ. The *empty string*, which has a length of zero, is denoted by λ. If a string x is n-long, it has n *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 $y \sqsubseteq x$ to denote this fact. If y is a subsequence of x, then x is a *supersequence* of y. 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 symbols that are adjacent in y are adjacent in x too. This is not required if y is a subsequence of x. The set of all subsequences of x is denoted by $\mathcal{S}(x)$. If $u \sqsubseteq x$ and $u \sqsubseteq y$, u is a common subsequence of x and y. The set

of all common subsequences of x and y is denoted by $\mathcal{S}(x, y)$. Let $x = x_1 x_2 \cdots x_n$. If $u \sqsubseteq x$, we say that u is *embeddable* in x; we then have $u = x_{i_1} x_{i_2} \cdots x_{i_{|u|}}$. We call the integer sequence $i_x(u) = i_1, i_2, \cdots, i_{|u|}$ an *embedding* of u in x. Some sequences may be embeddable more than once in another sequence; for example $u = ac$ is twice embeddable in $x = abac$. We denote the number of embeddings of $u \sqsubseteq x$ as $|x|_u$; so, $|x|_{ac} = 2$.

3.2 Specialisation of subsumption for sequences

Here we consider subsumption for a special case where a concept can be represented as a sequence. We call this type of concept *sequential concept* and this type of subsumption *sequential subsumption*.

A property is taken as an attribute-value pair. A sequential concept is a set of homogeneous properties arranged in an order, where the attributes of the properties are of the same type so the values of the properties are directly comparable. For example, a sequential concept may be a colour histogram which is a sequence of frequency values for a range of colours in an image. A sequential concept may also be a spectrum which is a sequence of light intensity values at a range of wavelengths. For brevity we will call sequential concepts simply as sequences. More formally,

Definition 5 (Sequential concept)**.** *A* sequential concept *is a sequence* $x =< p_1, p_2, \cdots, p_n >$ *where* $p_i = (a_i, v_i)$ *is a property such that* a_i *is an attribute and* v_i *is a single value or a range of values. All attributes have the same domain, so the properties are homogeneous. For brevity we write this concept as* $v_1 v_2 \cdots v_n$*. These properties together serve as the* intension *of the sequential concept.*

The role of 'attribute' here is a bridge between the general notion of concept and a specialised one – sequential concept, where the attributes can be safely dropped with only values left. Thus, a sequential concept can be simply represented as a sequence of values.

Given a sequential concept x an interpretation $f(x)$ is the set of all instances in the domain of discourse U such that each instance can be represented as a sequence which is x or a supersequence [5] of x. For example, if x is a colour histogram representing an image of interest, then an instance in $f(x)$ is any image whose colour histogram y is equal to x or is a supersequence of x (e.g. by way of including more colours). Clearly, being identical is difficult so we can try to quantify their relationship by similarity or subsumption. More formally, we have

Definition 6 (Interpretation and representation)**.** *Let* U *be a domain of discourse and let* x *be a sequential concept. Then an* interpretation *of* x *is a set*

$$f_g(x) = \{e \in U : g(e) \sqsupseteq^C x\} \tag{11}$$

Here g *is a* representation function *that maps an element of* U *to a sequence* $v_1 \ldots v_k$*.* $\sqsupseteq^C$ *means supersequence relation under constraint* C*.*

Clearly, the interpretation depends upon the nature of the representation function g. It also depends on the constraint C imposed on the supersequence relation $\sqsupseteq^C$. For brevity we will drop C from the supersequence relation $\sqsupseteq$ unless this is necessary. See Section 3.3 for some examples of the constraint. Again, for brevity we will drop g unless this is necessary.

[5] Note that if y is a supersequence of x then x is a subsequence of y.

The following theorem establishes 1-1 mapping between extensional subsumption, as defined in the domain of discourse (Def. 3), and embedding of sequential representation using g.

Theorem 1. *Let U be a domain of discourse; x and y be two sequential concepts; and g be a representation function used in the construction of interpretations. Then $x \sqsubseteq y$ iff $f_g(x) \supseteq f_g(y)$.*

Proof. (i) Suppose $x \sqsubseteq y$. Suppose further that $U \ni e \in f_g(y)$. Then $g(e) \sqsupseteq y$ and by the transitivity of $\sqsupseteq$, $g(e) \sqsupseteq x$. This is true for any $U \ni e \in f_g(y)$ and any g, so we conclude that $f_g(x) \supseteq f_g(y)$ if $x \sqsubseteq y$.

(ii) Suppose that $f_g(x) \supseteq f_g(y)$ for some concepts x and y and some representation function g. Let $U \ni e \in f_g(y)$. Then $e \in f_g(x)$ and hence $g(e) \sqsupseteq x$. As this is true $\forall e \in f_g(y)$, it must also be true for $u \in U : g(u) = y$ since $u \in f_g(y)$ too and hence $y \sqsupseteq x$. So, we conclude that $x \sqsubseteq y$ if $f_g(x) \supseteq f_g(y)$. □

According to the above theorem and Def. 3, in order to determine subsumption relationship between sequences, it suffices to check if they have subsequence relation. Furthermore, in order to quantify subsumption, we need to use the set of subsequences $\mathcal{S}(x)$ of sequence x .

Corollary 1. *Let x, y be two sequences. $\mathcal{S}(x) \subseteq \mathcal{S}(y)$ if and only if $f(x) \supseteq f(y)$ for any interpretation function f.*

Thus the more general Axiom 1 can be made more specific as follows:

Axiom 2 (Axiom of Subsumption for Sequences)**.**

1. $s(r,t) = 1$ *if* $\mathcal{S}(r) \cap \mathcal{S}(t) = \mathcal{S}(t)$
2. $0 < s(p,t) < s(r,t) < 1$ *if* $|\mathcal{S}(p) \cap \mathcal{S}(t)| < |\mathcal{S}(r) \cap \mathcal{S}(t)|$
3. $s(r,t) = 0$ *if* $\mathcal{S}(r) \cap \mathcal{S}(t) = \emptyset$

One such function is the following:

$$s(r,t) = \left(\frac{|\mathcal{S}(r) \cap \mathcal{S}(t)|}{|\mathcal{S}(t)|} \right)^w, \qquad w > 0, \tag{12}$$

In a sequential context, gauging $|\mathcal{S}(r) \cap \mathcal{S}(t)|$ amounts to counting the elements of the set of all common subsequences of r and t and, i.e., $\mathcal{S}(r,t)$, depending on the $\cap$-operator chosen, determining the joint multiplicity or embedding-frequency of the elements of that set. In this context, the counts ϕ, ψ and μ as discussed in subsection 2.3 on set operations are now reformulated as

$$\phi(r,t) = \sum_{u \in \mathcal{S}(r,t)} 1, \tag{13}$$

$$\psi(r,t) = \sum_{u \in \mathcal{S}(r,t)} \min\{|r|_u, |t|_u\}, \tag{14}$$

$$\mu(r,t) = \sum_{u \in \mathcal{S}(r,t)} |r|_u \cdot |t|_u, \tag{15}$$

wherein the embedding frequencies $|r|_u$ and $|t|_u$ are the multiplicity functions of the multisets $(\mathcal{S}(r), |r|_.)$ and $(\mathcal{S}(t), |t|_.)$ that live in the universe of the free monoid Σ^*. However, depending on the application, such counts may be weighted or constrained in various ways. Here, we discuss such constraints in a qualitative way. When we turn to algorithms, we will be more precise and formal about some of these constraints.

3.3 Constraints on special cases

3.3.1 Constraints on matching

To construct and gauge the set $\mathcal{S}(x, y)$, we have to somehow compare the subsequences of r to those of t. If the subsequences match, they add to the count of $|\mathcal{S}(r, t)|$. However, depending upon the constituting symbols, the subsequences may not be equally important and thus should not equally add to the total count: the matches between subsequences are then weighted, depending on the symbols involved. Furthermore, matching of subsequences may be "soft" in the sense that some symbols may be more similar to each other than others. So, instead of "hard", binary matching, one might opt for accounting for the degree of "similarity" between subsequences. Weighing subsequences depending on their content and soft-matching are options in the algorithms that we will present in the sequel.

3.3.2 Constraints on gap-widths

Subsequences may have gaps in the sense that its constituting adjacent symbols may not be adjacent in either or both of r and t. Some subsequences in, say, r may have many more or less gaps than the same subsequences in t or the gaps may be of different sizes. Depending on the application, the significance of gaps may vary, depending on their size. Such considerations invite for a kind of gap-penalizing that is either hard, i.e not accepting subsequences that have gap-widths beyond a certain limit, or soft, i.e. the penalty depends on the width of the gap.

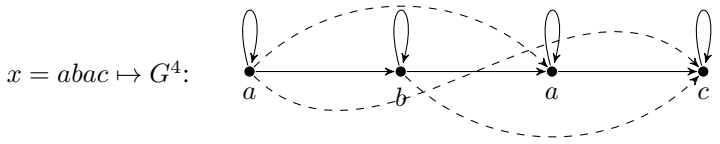
On the other hand, particular gap-patterns might be interesting or even mandatory and then one accounts for (the degree of) uniform warping of time or space. For example, gap-width might or should vary linearly with gap-position. In the sequel, we will discuss algorithms that specifically account for some forms of warping.

3.3.3 Constraints on subsequence lengths

Sometimes only longer subsequences are interesting since shorter subsequences are likely to occur anyway. In such applications, subsequences could be weighted according to their length. An extreme way of doing this is confining to the longest common subsequences as is done in edit-based algorithms on strings.

In light of the fact that the constraint may be different from one application to another and the subsumption test is a service in applications, we need an algorithm to measure subsumption that is fast, can accommodate a broad range of constraints and can be easily extended to numerical sequences.

Figure 2: The sequence $x = abac$ mapped to G^4. The dashed arrows represent the subsequences aa, ac and bc as tickets (the corresponding minors and representing paths are not shown).



4 Computing subsequence intersection

4.1 Intersection as a product graph: the Grid

When quantifying subsumption in a sequential context, we have to evaluate the number and nature of the common subsequences of r and t. An important class of algorithms to do just that is known as Grid-algorithms: these algorithms use the direct product of the graph representations of the sequences involved. Therefore, we discuss these graphs, their direct product and the Grid.

An n-long sequence $x = x_1, \ldots, x_n$ can be represented as digraph $G^n = (V, A, L)$ along n vertices $V = \{v_1 \ldots, v_n\}$ through $2n-1$ arcs $A = \{v_i v_j : (v_i, v_j \in V) \wedge ((j = i + 1) \vee (i = j))\}$ and vertex-labels $L = \{l(v_1), \ldots, l(v_n)\}$ that correspond to the symbols $x_i \in \Sigma$. Then substrings of x are represented as directed paths on G^n with terminals v_i and v_j with $1 \leq i \leq j \leq n$. As the symbols themselves are substrings of the sequence, it is convenient to assign an arc on itself to each vertex: if $v_i \in V$, $(v_i, v_i) \in A$. Subsequences that are not substrings of x are paths on minors $M(G^n)$ of G^n that arise by contracting one or more arcs and eliminating pertaining vertices [14]; such paths on $M(G^n)$ are called "tickets" on G^n [17]. These ideas are illustrated in Figure 2. Now suppose that we have two sequences, say r and t with representing graphs $G_r = (V_r, A_r, L_r)$ and $G_t = (V_t, A_t, L_t)$. Then the direct product graph $G_{rt} = (V_{rt}, A_{rt}, L_{rt})$ of these sequences is defined as [24]:

$$V_{rt} = \{(v_r, v_t) : (v_r \in V_r) \wedge (v_t \in V_t) \wedge (l(v_r) = l(v_t))\} \tag{16}$$

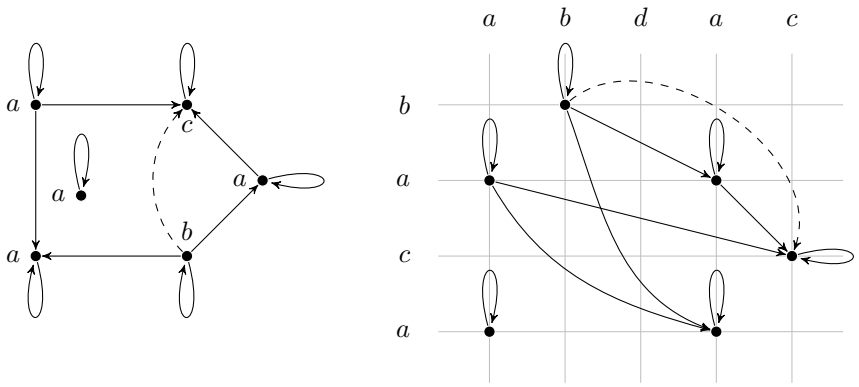
$$A_{rt} = \{((v_r, v_t), (v'_r, v'_t)) \in V_r \times V_t : ((v_r, v'_r) \in A_r) \wedge ((v_t, v'_t) \in A_t)\} \tag{17}$$

$$L_{rt} = L_r \cap L_t \tag{18}$$

In Figure 3, we present two visualizations of the direct product graph of the sequences $r = baca$ and $t = abdac$: one arbitrary and one embedded in a grid that is spanned by the sequence-graphs: the $(r \times t)$-grid. The reader will note that in the grid-embedding, all arcs and tickets point "South-East", i.e. they represent the common subsequences of r and t and thus reflect the sequential character of the factors. The reader also notes that some subsequences are represented by two arcs; for example, there are two arcs connecting vertices labeled b and a because the common subsequence ba is embedded twice in r and just once in t.

The grid-representation of the graph G_{rt} is complete in the sense that all paths of

Figure 3: Two visual representations of the direct product graph of the sequences $r = baca$ and $t = abdac$ and the ticket (dashed) bc on that graph. The left representation respects the adjacencies but is arbitrary otherwise. The representation on the right side shows an embedding in a grid that is spanned by the sequence graphs. Black points (the nodes) are placed at grid-locations that have the same symbols as coordinates



the graph are embedded. So, the grid-representation shows all common substrings but, unfortunately, not all common subsequences since the latter may contain symbols that are not adjacent in either or both of the sequences. So, if we want to evaluate $f(r) \cap f(t) = \mathcal{S}(r,t)$ through a graph-embedding on the $(r \times t)$-grid, we implicitly use tickets on G_{rt}. Evaluating the intersection $f(r) \cap f(t)$ amounts to counting all paths and tickets and adding the counts. This is accomplished through using what was called a "Grid-algorithm" [19]. The next sections discuss the Grid-algorithm and some relevant parameterizations.

4.2 The Grid-algorithm and its tuning

The Grid-algorithm operates on the $r \times t$-grid and counts all paths and tickets in an orderly fashion. Clearly, the set of all nonempty common subsequences may be partitioned as the disjoint union of sets of common subsequences of equal length:

$$\mathcal{S}(x,y) = \bigcup_{i=1}^{\min\{|x|,|y|\}} \mathcal{S}(x,y:i) \tag{19}$$

wherein $\mathcal{S}(x,y:i)$ denotes the set of all common subsequences of length i. As these sets are disjoint, we must have that

$$|f(r) \cap f(t)| = |\mathcal{S}(x,y)| = \sum_{i=1}^{\min\{|x|,|y|\}} |\mathcal{S}(x,y:i)|. \tag{20}$$

All parameterizations of the Grid-algorithm calculate the above sum, proceeding through $i = 1, 2, \ldots, \min\{|x|,|y|\}$. Here we discuss the most simple parameterization as first published in [15] and illustrated in Figure 4.

The common subsequences of length 1 are represented by paths of length zero, i.e the loops on the grid. Common subsequences of length two (consisting of two symbols) are created by defining arcs from each vertex to all other vertices that are located to the South-East. Counting these arcs then yields the number of 1-long paths. Next, for each vertex, the arcs are counted that leave the vertices that are located to the South-East: the number of 2-long paths from each of the vertices since they elongate the 1-long paths. This process of counting elongations stops when none of the paths counted can be elongated any more.

So, we see that the embedding of the direct product graph in the $r \times t$-grid allows us to count ever longer common paths and tickets on G_{rt} since moving simultaneously along the prefixes of the pertaining sequences amounts to moving in South-Easterly direction on the grid-embedded product graph. The reader notes that in the example of Figure 4, the 2-long common subsequence ba is embedded once in $r = abac$ and thrice in $t = babcac$ and thus all three corresponding arcs in G_{rt} are counted: the number of times that an occurrence of ba in one sequence is matched by occurrences in the other sequence. Hence, in its basic guise, the Grid-algorithm counts the number of matching subsequences $\mu(r, t)$:

$$\mu(r,t) = \sum_{u \in \mathcal{S}(r,t)} |r|_u \cdot |t|_u \tag{21}$$

which implies a specific definition of the intersection of multisets. Fortunately, the Grid-algorithm can be adapted such that different definitions of multiset intersection can be used in determining the size of the intersection $|f(r) \cap f(t)|$ of the reference and the target.

We now formulate the basic version in the theorem below and then generalize it to a guise that allows for a systematic discussion of the variants that are relevant to the quantification of subsumption.

Theorem 2. *[16] Let $x, y \in \Sigma^*$ be represented as $x = x_1 \dots x_n$ and $y = y_1 \dots y_m$. Let $\mathbf{M}^1 = \{m^1_{ij}\}$ denote a $n \times m$ - matrix defined through*

$$m^1_{ij} = \begin{cases} 1 & \textit{iff } x_i = y_j \\ 0 & \textit{otherwise} \end{cases} \tag{22}$$

Let the $(n \times m)$-matrices $\mathbf{M}^k = \{m^k_{ij}\}$ be recursively defined for $k \geq 2$ through

$$m^k_{ij} = m^1_{ij} \sum_{a>i, b>j} m^{k-1}_{ab}. \tag{23}$$

Then

$$\mu(x,y) = \sum_k \mu^k(x,y). \tag{24}$$

where $\mu^k(x,y) = \sum_{ij} m^k_{ij}$.

Clearly, in Eq. (22), the algorithm is initialized by defining the 0-long paths. Then follows the recursion of Eq. (23) of counting ever longer paths and tickets in parts of the grid that are defined by the range of summation in Eq. (23). Finally, there is the way that the results for subsequences are aggregated into the final result, i.e. Eq. (24).

To allow for discussing interesting variants, Theorem 2 is generalised to a general Grid-algorithm, the *Grid-based Algorithmic Framework* (GAF), as follows:

Theorem 3. *Let $x, y \in \Sigma^*$ be represented as $x = x_1 \dots x_n$ and $y = y_1 \dots y_m$. Let $\mathbf{M}^1 = \{m^1_{ij}\}$ denote a $n \times m$ - matrix defined through*

$$m^1_{ij} = F(x_i, y_j) \tag{25}$$

Let F, H, K and J denote functions to some number field. Let the $(n \times m)$-matrices $\mathbf{M}^k = \{m^k_{ij}\}$ be recursively defined for $k \geq 2$ through

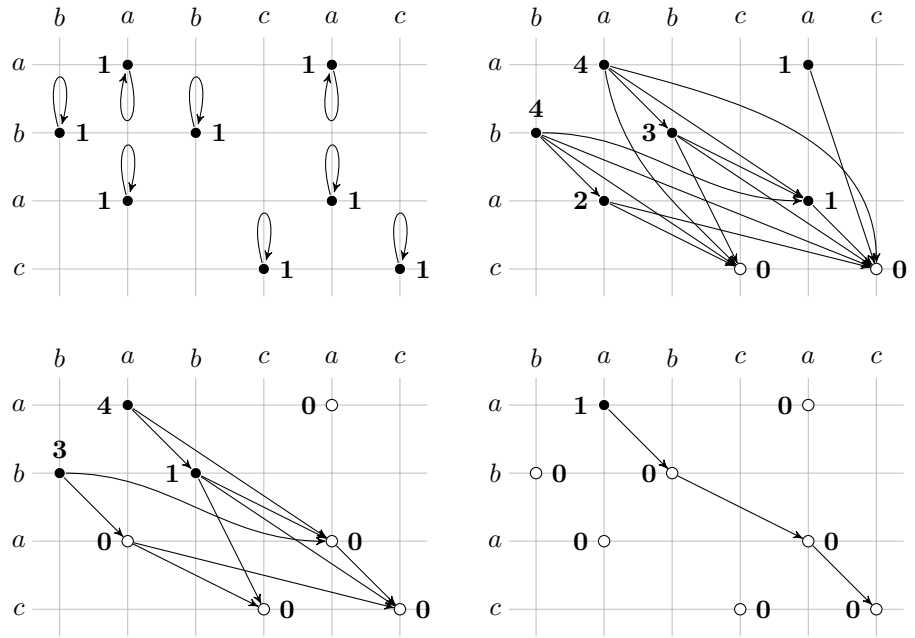
$$m^k_{ij} = m^1_{ij} \sum_{K(i,j)} H(\mathbf{M}^{k-1}). \tag{26}$$

Then $\omega^k(x, y) = \sum_{ij} m^k_{ij}$ and

$$\omega(x, y) = J\left(\{\omega^k(x, y)\}\right). \tag{27}$$

F, H, K and J are functions that need to be specified in practice to take into account specific constraints. Once these functions are specified, the GAF provides for a specific algorithm to calculate the quantity $\omega(x, y)$, which subsequently obtains its specific meaning. Therefore, the GAF provides a framework for designing algorithms to calculate set intersections, i.e. $\omega(x, y)$, allowing for the consideration of different constraints.

Figure 4: The basic Grid-algorithm (Theorem 2) operating on the grid spanned by $r = abac$ and $t = babcac$. The four embeddings show minors of the direct product graph. Upper left panel: all vertices with 1-long paths (8), upper right panel: all vertices with 2-long paths (15), lower-left panel: all 3-long paths (8), lower-right panel: and all 4-long paths (1). Each n-long path represents an n-long common subsequence of r and t.



For example, Theorem 2 can be derived from the GAF with the following specifications:

$$F(x_i, y_i) = \begin{cases} 1 & \text{if } x_i = y_j \\ 0 & \text{otherwise} \end{cases},$$
$$H(\mathbf{M}^{k-1}) = m_{ab}^{k-1},$$
$$K(i,j) = \{(a,b) : (i < a \leq n) \wedge (j < b \leq m)\},$$
$$J\left(\{\omega^k(x,y)\}\right) = \sum_k \omega^k(x_i, y_j)$$

and then ω is interpretable as the number μ of matching nonempty subsequences.

It should be noted that each of these functions can be changed independently from the others and some of these modifications have sensible interpretations. We mention some examples of such modifications, taken from [18]:
Setting

$$F(x_i, y_j) = \begin{cases} 1 & \text{if } x_i = y_j \\ 0 \leq u(x_i, y_j) < 1 & \text{otherwise} \end{cases}$$

changes $\mathbf{M^1}$ from a binary diagonal "hard-matching" matrix to a full "soft-matching" similarity matrix.
Setting

$$H(\mathbf{M}^{k-1}) = \sum_{a>i, b>j} v(a-i, b-j) m_{ab}^{k-1}$$

yields a way for soft-penalizing for gap-width and setting

$$K(i,j) = \{(a,b) : (i < a \leq h < n) \wedge (j < b \leq h < m)\}$$

amounts to hard-penalizing gaps bigger than h. Finally, setting

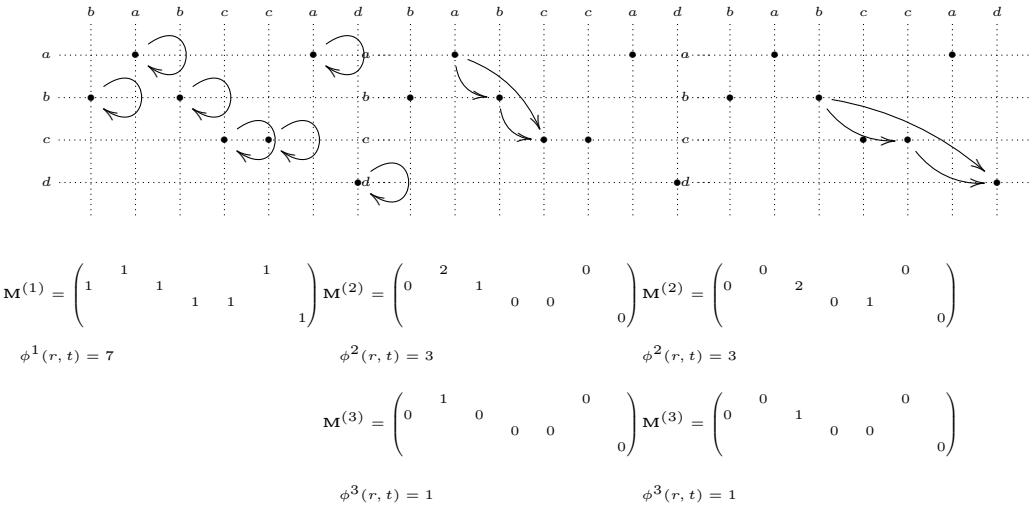
$$J\left(\{\omega^k(x,y)\}\right) = \sum_k w(k)\omega^k(x_i, y_j)$$

allows one to weigh for the length of the common subsequences, probably assigning a bigger weight to longer subsequences.

4.3 Scaling objects: Tuning the Grid-algorithm to warping

Warping is useful when one wants to compare objects that have the same shape but differ in scale. Here we consider linear warping. More specifically, let $u \sqsubseteq x$ with $u = u_1, \ldots, u_{|u|}$ and let $i_x(u) = i_1, \ldots, i_{|u|}$ denote an embedding of u in x such that, for nonnegative integers p and q, $i_j = p + q \times i_{j-1}$. Note that p is a time-origin and q is a time-scale. Then we say that u (or x) is a (p,q)-linear warp of x (or u). For example, let $x = abcdefghijk$. Then $u = abc$ is a subsequence under this constraint, where $p = 1$ and $q = 0$; so is $u = aceg$ where $p = 2$ and $q = 0$. Furthermore, $u = acfj$ is also subsequence under this constraint, where $p = 2$ and $q = 1$.

Matching subsequences under this linear warping constraint can be characterised in the following corollary, which is derived from Theorem 3.



(a) *Initialization* (b) 0-*linear warping* (c) 1-*linear warping*

Figure 5: *Count matching subsequences of* $r = abcd$ *in* $t = babccad$ *under* p*-linear warping as constraint: (a) Initialize the algorithm of Corollary 2 to calculate the matrix* $\mathbf{M}^{(1)}$*. (b) Count* 2*- and* 3*-long matching subsequences under* 0*-linear warping via calculating the matrices* $\mathbf{M}^{(2)}$ *and* $\mathbf{M}^{(3)}$*.* $\mathbf{M}^{(4)}$ *contains only zeros since there are no 4-long common subsequences. (c) Count* 2*- and* 3*-long matching subsequences under* 1*-linear warping.*

Corollary 2. *Let* $x, y \in \Sigma^*$ *be represented as* $x = x_1 \dots x_n$ *and* $y = y_1 \dots y_m$*. Let* $\mathbf{M}^1 = \{m^1_{ij}\}$ *denote a* $n \times m$ *- matrix defined through*

$$m^1_{ij} = \begin{cases} 1 & \textit{iff } x_i = y_j \\ 0 & \textit{otherwise} \end{cases} \tag{28}$$

Let the $(n \times m)$*-matrices* $\mathbf{M}^k = \{m^k_{ij}\}$ *be recursively defined for* $k \geq 2$ *through*

$$m^k_{ij} = m^1_{ij} \sum_{a>i, b=j+(a-i)\times(1+p+q)} m^{k-1}_{ab}. \tag{29}$$

Then $\mu^k(x,y) = \sum_{ij} m^k_{ij}$ *and* $\mu(x,y) = \sum_k \mu^k(x,y)$*.*

An illustration of Corollary 2 is presented in Figure 5. An algorithm for counting matching subsequences under linear warping is presented in Algorithm 1. It is clear the time complexity of the algorithm is $O(|r| \times |t| \times LCS(r,t))$, where $LCS(r,t)$ is the length of a longest common subsequence of r and t. Faster algorithms can be derived for specific types of constraint, making better use of the fact that we only need to check certain cells of the matrices.

Under the linear warping constraint, a special case arises when some position in a sequence can be any symbol. For example, in an image retrieval task, we are interested in BMW X5 and want to retrieve all images that contains X5. We can represent an image by the vertical and horizontal projections (or profiles), and we know the background parts of the projection vectors are useless for recognising X5. Therefore these

parts can be filled by any values. More specifically, consider the example in Figure 5 where $r = a\underline{b}cd$ and $t = t_1t_2\cdots t_7 = babccad$. We want to count the matching subsequences of r in t under the constraint that the second position in the reference may be any symbol but its position is important. This constraint can be coded in by $r = a\_cd$, where the symbol _ means it can be any symbol. If $p = 0$, then one matching subsequence is $t_2t_3t_4 = abc$. If $p = 1$, then one matching subsequence is $t_5t_7 = cd$. We call this constraint *selective linear warping*.

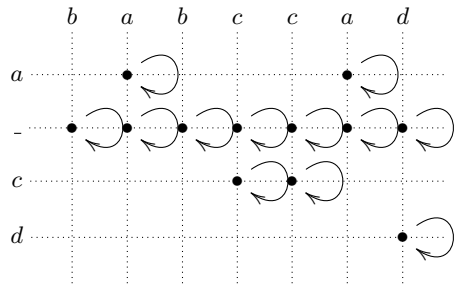
The following corollary provides for a systematic way of counting matching subsequences under this constraint, which is derived from Theorem 3.

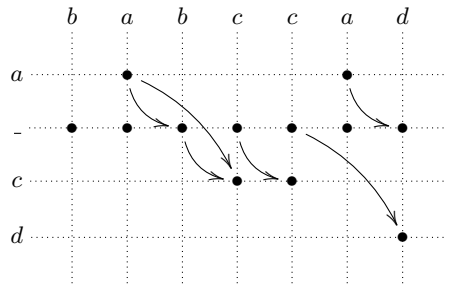
Corollary 3. *The same as Corollary 2 except for the way to construct* $M^1 = (m^1_{ij})$. *We replace Eq.* (28) *by the following:*

$$m^1_{ij} = \begin{cases} 1, \text{ if } r_i = \_ \text{ or } r_i = t_j \\ 0, \text{ otherwise} \end{cases}$$

An illustration of Corollary 3 is presented in Figures 6a and 6b. An algorithm for counting matching subsequences under the selective warping constraint can be obtained by changing the conditional statement 4 in Algorithm 1 to the following:

if $r_i = \_$ or $r_i = t_j$ **then**





$$\mathbf{M}^{(1)} = \begin{pmatrix} & 1 & & & & 1 & \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ & & & 1 & 1 & & \\ & & & & & & 1 \end{pmatrix}, \quad \phi^1(r,t) = 12$$

(a) *Initialization*

$$\mathbf{M}^{(2)} = \begin{pmatrix} & 2 & & & & 1 & \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ & & & 0 & 0 & & \\ & & & & & & 0 \end{pmatrix}, \quad \phi^2(r,t) = 6$$

$$\mathbf{M}^{(3)} = \begin{pmatrix} & 1 & & & & 0 & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & & 0 & 0 & & \\ & & & & & & 0 \end{pmatrix}, \quad \phi^3(r,t) = 1$$

(b) 0-*linear warping*

Figure 6: *Count matching subsequences of* $r = a\_cd$ *in* $t = babccad$ *under selective warping as constraint: (a) Initialize the algorithm of Corollary 3 by computing the matrix* $\mathbf{M}^{(1)}$. *(b) Count* 2- *and* 3*-long matching subsequences via computing matrices* $\mathbf{M}^{(2)}$ *and* $\mathbf{M}^{(3)}$.

Algorithm 1 *An algorithm to count matching subsequences under the (p, q)-linear warping constraint and to find length of longest matching subsequence, which is based on Corollary 2.*

Require: Input $r = r_1 r_2 \cdots r_m$ and $t = t_1 t_2 \cdots t_n$

Ensure: Output the number of all matching subsequences, and the length of the longest matching subsequence

$\delta \leftarrow 0$
for $i = 1$ to m **do**
  for $j = 1$ to n **do**
    if $r_i = t_j$ **then**
      $e^1_{ij} \leftarrow 1$, $e_{ij} \leftarrow 1$, $\delta \leftarrow \delta + 1$
    end if
  end for
end for
if $\delta = 0$ **then**
  return 1
end if
$\phi = \delta$
$k = 1$
while $k \leq m$ **do**
  for $i = 1$ to $m - 1$ **do**
    for $j = n - 1 - p - q$ to 1 **do**
      $ii = i + 1, jj = j + 1 + p + q, e_{i,j} \leftarrow e_{ii,jj}$
    end for
  end for
  $\delta \leftarrow 0$
  for $i = 1$ to m **do**
    for $j = 1$ to n **do**
      $e_{i,j} \leftarrow e_{i,j} \times e^1_{i,j}$
      $\delta \leftarrow \delta + e_{i,j}$
    end for
  end for
  if $\delta = 0$ **then**
    return $\phi + 1$
  end if
  $\phi \leftarrow \phi + \delta$
  $k \leftarrow k + 1$
end while
return $\phi + 1$, and k

5 Set-intersection and run-lengths

In this section we deal with algorithms that calculate the size of the three different forms of set intersection that we discussed previously. Thereto, it is convenient to extend the Grid algorithm such that it can handle a specific form of a quantified property of the the characters in the sequence: their run-lengths. So far, we treated the Grid-algorithm as operating on a (direct-product) graph with vertices that are labeled by pairs from the character set Σ - now we will extend this labeling to the run-lengths.
A run is a substring that consists of a repetition of one and the same character and that cannot be extended in either direction without introducing at least one new, extra character. For example, the substring aaa is a run from the sequence $bbaaac$: it consists of a repetition of the same character and it cannot be extended in either direction without introducing either or both of b and c.

Instead of writing a sequence x as $x = x_1, \ldots, x_{|x|}$, we may write the sequence in so-called run-length notation: we write $x = x_1^{r_1} \ldots x_m^{r_m}$ wherein each $x_i^{r_i}$ denotes an r_i-long substring of x that consists of the same symbol x_i and such that $x_{i-1} \neq x_i \neq x_{i+1}$. For example, we write $x = a^2b^3c^1b^2$ to denote the sequence $x = aabbbcbb$. Encoding an n-long sequence in run-length notation is of complexity $O(n)$ and run-length encoding is widely used in e.g. image-processing [26] and in the construction of holograms [25]. We say that runs $x_i^{r_i}$ from x and $y_j^{s_j}$ from sequence y match, precisely when the characters match, i.e. when $x_i = y_j$.

As there are some useful relations between the runs and the subsequences of a sequence and we discuss these here.

A run $x_i^{r_i}$ consists of other runs and these runs are subsequences of the run and therefore, of the sequence x: $x_i^j \sqsubseteq x_i^{r_i} \sqsubseteq x$ for $\forall j \in [r_i]$. For example, a^3 contains $a^1 = a$, $a^2 = aa$ and $a^3 = aaa$ and all are subsequences of a^3b^2.

If $x_i = x_j$ and both $x_i^{r_i}$ and $x_j^{r_j}$ are distinct runs of x, then x contains $r_i + r_j$ distinct subsequences that only consist of $k \in [r_i + r_j]$ symbols $x_i = x_j$. Hence, if x and y are sequences and $x_i = y_j = \sigma \in \Sigma$, then there exist (at least) $\min\{r_i, r_j\}$ common subsequences that consist of the symbol σ only. If we define $\mathbf{M}^1 = \{m_{ij}^1\}$ with $m_{ij}^1 = \min\{r_i, r_j\}$ if $x_i = y_j$ and $m_{ij}^1 = 0$ otherwise, we have that $\sum_{ij} m_{ij}^1$ equals the sum of all subsequences contained in just one common run.

If $x_i \neq x_j$ and $x_i^{r_i}$ and $x_j^{r_j}$ are runs of x, then x contains at least $r_i \times r_j$ subsequences of the form $x_i^k x_j^\ell$, one for each pair (k, ℓ) with $k \in [r_i]$ and $\ell \in [r_j]$. At least, since x may contain other runs $x_a^{r_a}$ and $x_b^{r_b}$ with $x_i = x_a$ and/or $x_j = x_b$.

5.1 Multiset intersection as matching subsequences

First, we adapt the Grid-algorithm of Theorem 2 to deal with sequences in run-length notation. So we set out to calculate the intersection $\mu(x, y) = \sum_{u \in \mathcal{S}(x,y)} |x|_u \cdot |y|_u$. Thereto, we formulate the next

Theorem 4. *Let the sequences $x, y \in \Sigma^*_{s_{n_y}}$ be represented in run-length notation as $x = x_1^{r_1}, \ldots, x_{n_x}^{r_{n_x}}$ and $y = y_1^{s_1}, \ldots, y_{n_y}^{s_{n_y}}$. Let $\mathbf{M}^1 = \{m_{ij}^1\}$ with $i \in [r_{n_x}]$ and $j \in [s_{n_y}]$ denote an $(r_{n_x} \times s_{n_y})$-matrix defined through*

$$m_{ij}^1 = \begin{cases} \sum_{k=1} \binom{r_i}{k}\binom{s_j}{k} & \text{if } \; x_i = y_j \\ 0 & \text{otherwise} \end{cases} . \tag{30}$$

Furthermore, for $k > 1$, let $\mathbf{M}^k = \{m_{ij}^k\}$ denote an $(r_{n_x} \times s_{n_y})$-matrix, defined by

$$m_{ij}^k = m_{ij}^{k-1} \sum_{\substack{a>i \\ b>j}} \sum_{q=1} \binom{r_a}{q} \binom{s_b}{q}. \tag{31}$$

Then $\mu^k(x,y) = \sum_{ij} m_{ij}^k$ and $\mu(x,y) = \sum_k \mu^k(x,y)$.

The reader notes that the meaning of the number m_{ij}^1 as defined in Eq. (30) differs from the meaning of that number in Eq. (22) of Theorem 2: here, m_{ij}^1 equals the number of matching subsequences that are contained in the pair of common runs $(x_i^{r_i}, y_j^{s_j})$. As the characters of these runs match, each $\binom{r_i}{k}$ k-tuples taken from $x_i^{r_i}$ matches to each of the $\binom{s_j}{k}$ k-tuples taken from $y_j^{s_j}$ hence Eq. (30) follows. Eq. (31) follows from analogous reasoning.

Interestingly, in Theorem 4 we have that the m_{ij}^k do not only, like in Theorem 2, depend on (the matching of) x_i and y_j but also on the the run lengths: $m_{ij}^1 = F(x_i, y_j, r_i, s_j)$. Hence Theorem 4 extends Theorem 3. The reader also notes that, in practice, sequences in run-length notation will be much shorter than the same sequences in a character-by-character notation. Hence, we expect that the runtime of the algorithm implied by Theorem 4 is better than that of the algorithm implied by Theorem 2, provided that the pertaining binomials are effectively evaluated, e.g. by constructing Pascal's triangle on the fly. The reader also notes from the binomials that appear in Theorem 4, that calculating $\mu(x,y)$ may require big integer arithmetic.

5.2 Multiset intersection as shared subsequences

Here, we consider the case in Eq. (14), where the intersection is multiset based and the multiplicity function is

$$\chi_{\mathcal{A}\cap\mathcal{B}}(x) = \min\{\chi_{\mathcal{A}}(x), \chi_{\mathcal{B}}(x)\} \tag{32}$$

In other words, we count shared subsequences, i.e $\psi(x,y) = \sum_{u\in\mathcal{S}(x,y)} \min\{|x|_u, |y|_u\}$.

Theorem 5. *Let $x, y \in \Sigma^*$ be represented in run-length notation as $x = x_1^{r_1} \dots x_n^{r_n}$ and $y = y_1^{s_1} \dots y_m^{s_m}$. Let $\mathbf{M}^1 = \{m_{ij}^1\}$ denote a $(n \times m)$-matrix defined through*

$$m_{ij}^1 = \begin{cases} 2^{min\{r_i, s_j\}} - 1 & \textit{iff } x_i = y_j \\ 0 & \textit{otherwise} \end{cases} \tag{33}$$

Let the $(n \times m)$-matrices $\mathbf{M}^k = \{m_{ij}^k\}$ be recursively defined for $k \geq 2$ through

$$m_{ij}^k = \sum_{a>i, b>j} m_{ab}^{k-1} \cdot (2^{min\{r_a, r_b\}} - 1). \tag{34}$$

Then, for $k > 1$, $\psi_k(x,y) = \sum_{i,j} m_{ij}^k$ and $\psi(x,y) = \sum_{k=1} \psi_k(x,y)$.

In Theorem 5, the numbers m_{ij}^1 store the number of common subsequences, shared by the runs $x_i^{r_i}$ and $y_j^{s_j}$. If the runs do not match, that number equals zero. A run of length r consists of r distinct subsequences that have $\sum_k \binom{r}{k} = 2^r - 1$ different embeddings; this justifies Eq. (33). Analogous reasoning yields Eq. (34).

5.3 Set intersection as common distinct subsequences

Here we discuss an adaptation for the simple, set based intersection, i.e. we just count number of common subsequences as $|\mathcal{S}(x,y)|$.

Corollary 4. *Let $x, y \in \Sigma^*$ be represented in run-length notation as $x = x_1^{r_1} \dots x_n^{r_n}$ and $y = y_1^{s_1} \dots y_m^{s_m}$. Let $\mathbf{M}^1 = \{m_{ij}^1\}$ denote a $(n \times m)$-matrix defined through*

$$m_{ij}^1 = \begin{cases} min\{r_i, s_j\} & \textit{iff } x_i = y_j \\ 0 & \textit{otherwise} \end{cases} \tag{35}$$

Let the $(n \times m)$-matrices $\mathbf{M}^k = \{m_{ij}^k\}$ be recursively defined for $k \geq 2$ through

$$m_{ij}^k = m_{ij}^1 \times \sum_{a>i, b>j} max\{0, (m_{ab}^{k-1} - m_{l_x(a,x_i), l_y(b,x_i)}^{k-1})\} \tag{36}$$

Then, for $k > 1$, $\phi_k(x,y) = \sum_{i,j} m_{ij}^k$ and $\phi(x,y) = \sum_{k=1} \phi_k(x,y)$. Variable $l_x(k,\sigma) = max\{i : (i < k) \wedge (x_i = \sigma)\}$ keeps track of the previous position of a symbol $\sigma \in A$ in a sequence.

A run of length r has only r distinct nonempty subsequences. So m_{ij}^1 in Eq. (35) just counts the number of common distinct subsequences of the common runs. However, several distinct runs may use the same character so we have to discount for the subsequences already counted in previous encounters with the same characters. This reasoning derives from Lemma 6 in [16].

6 Experimental Evaluation of the Algorithms

It is known [16] that Grid-algorithms have an algorithmic complexity of $O(|x| \times |y| \times \min\{|x|, |y|\})$ and that actual runtimes will vary, depending on the similarity of the sequences involved. However, their performance in terms of actual runtimes was never investigated. Therefore, we present a series of experiments to evaluate the performance and limitation of some of the algorithms.

The actual runtime of each of the algorithms is dependent on the lengths of the two strings and how similar they are. We want to find out how each algorithm performs as the length of one string varies under two cases – lower similarity and higher one. Here we use the length of the longest common subsequence as an indication of string similarity.

For this we generated two artificial datasets. Each dataset consists of multiple pairs of strings, each pair being a data object. In both datasets, the first string of every data object consists of 100 characters with each character being selected at random from the English alphabet (26x2 characters – 26 letters with two capitalisations). The two datasets differ in how the second string is constructed, corresponding to the two similarity cases:

Dataset I – low similarity: Each of the second strings has a length between 1 and 60, and consists of characters also randomly selected from the same alphabet. Since both the first and second strings are randomly constructed, their similarity is expected to be low on average.

Dataset II – high similarity: Each of the second strings also has a length between 1 and 60 and is a substring of its first string counterpart. Clearly every pair of strings has high similarity.

Each algorithm was run through both of the datasets and the runtimes were recorded. Each algorithm was run 1,000 times on a standard PC [6] in order to smooth out system level interference (such as from context switches). The runtime average by each algorithm on each pair of strings is plotted on a graph by the length of the second string, along with a trend line (polynomial fit of order 3).

6.1 The Grid algorithm for counting set-based subsequences

The results of running the Grid algorithm for counting set-based subsequences on the two datasets are shown in Figure 7. It is clear that the runtime of the algorithm does not vary linearly with the lengths of the two strings. With r and t being the lengths of the first and second strings respectively, the best case runtime is $O(rt)$ which occurs when the two strings do not share any common elements. The worst case runtime is $O(rtm)$ where $m = max(r, t)$.

6.2 The Grid algorithm for counting multiset-based subsequences

The results of running the Grid algorithm for counting multiset-based subsequences on the two datasets are shown in Figure 8. It should be noted that this algorithm was not run on the full datasets. Due to the large values that the matrix cells can reach some strings would overflow the underlying data types (64bit integer). To overcome this issue the pairs of strings that could not be compared were excluded. The number of compared pairs of strings was approximately 3440 and 3535 for the substring and randomstring datasets respectively. It is clear from the charts that the working tests provided a sufficient runtime profile of this algorithm.

6.3 The Grid algorithm for counting subsequence embeddings

The results of running the Grid algorithm for counting subsequence embeddings on the two datasets can be seen in Figure 9. It is clear that the runtime profile of this algorithm is broadly similar to the Grid (set) variant. In particular the same observed minimum and maximum runtime bounds apply $O(rt)$ and $O(rtm)$.

6.4 The Grid algorithm for counting subsequence embeddings under linear warping

The results of running this algorithm on the two datasets are shown in Figure 10. In particular Figure 10(b) shows that the different lcs values have a distinct effect on the runtime of the algorithm. Additionally, as noted previously the runtime for the algorithm is proportional to $r \times t \times lcs(r, t)$. Therefore it is useful to plot the runtime against the value of $r \times t \times lcs(r, t)$ to help check this. This is done on Figure 11.

We note that once a cell reaches 0 it is 'done' in that it will never be given a different value. We can exploit this observation to optimise the linear warping algorithm – when a cell has a value of 0 we move on to the next cell. The results of running the optimised linear warping algorithm are shown in Figure 12. As can be seen from this figure, the runtime of the optimised algorithm becomes less consistent, but the overall runtime for many cases is reduced. The worst case runtime of the optimised algorithm is under 0.07ms whereas the worst case runtime of the original algorithm is just over 0.1ms.

[6] Intel Core i7 CPU 920 @ 2.67 GHz, with 1 MB L1 cache and 8 MB L3 cache.

6.5 Discussion

Summarizing, we can state that, overall, the algorithms performed well with regards to speed - with most taking less than 50 milliseconds to complete with the longest strings of 60 characters. The exception was the Grid algorithm for counting multiset based common subsequences, which took seconds to run for each pair of strings and encountered, as expected, the integer overflow issue for long strings. The integer overflow issue may provide a challenge to using this algorithm with long strings, and may need to use big integer arithmetic to solve. The main memory requirement is the multiplication of two integers, the lengths of two strings. The memory requirement is unlikely to ever be an issue on even modest systems.

7 Conclusions

In this paper we present work on how to quantify the subsumption relation in the specific context of sequences. Subsumption is commonly used in knowledge representation and ontology; if quantified, it can also be used in data analysis tasks such as pattern recognition where data is usually represented as histograms (i.e. sequences of values) – image analysis to detect objects in an image, and spectral data analysis to detect the presence of a reference pattern in a given spectrum.

We give an axiomatic characterisation of subsumption, suggest a set-based quantification and a sequence-based quantification in particular, present a general algorithmic framework that can be used in computing subsumption in different cases, and present an experimental evaluation of these algorithms.

We conclude the following:

- The generalized Grid-algorithm provides for a very flexible efficient tool to quantify sequential subsumption in a variety of ways.
- Efficient algorithms exist for computing subsumption for various cases – when set (canonical and multiset) is used, when embedding is considered, and when only linear embedding is considered.
- These algorithms are all quadratic in the length of one sequence and linear in the lengths of both sequences and their longest common subsequence.
- The Grid algorithm for multiset-based common subsequence counting returns large integers so big integer arithmetic may be required in its implementation for applications with long sequences.

Future work will include implementing the algorithms using big integer arithmetic and applying the subsumption measure in various practical tasks such as object detection, substance detection, food authentication and spectral archive search. In the case of food authentication [33, 31], a similarity measure is typically used to compare an unknown sample and a known reference based on their spectra, usually after data processing and perhaps feature extraction. This approach works well when the spectral data are carefully collected without much noise so is suitable for use by professionals. It is unfortunately not suitable for non-professionals as data collection conditions can hardly be met. Subsumption could provide an alternative, perhaps better, approach. The reference spectrum can be accurately obtained by professionals and the sample spectrum can be obtained by non-professionals with relatively large degree of noise. The comparison can be done by subsumption – to check if the sample spectrum is a subsumption of the reference spectrum. This approach can thus potentially empower consumers for food authentication.

The concept of subsumption can be used in deep learning to design new loss functions. Instead of using distance (i.e. dis-similarity), which is symmetric, to define the loss function, subsumption may be used, which is asymmetric. This could potentially lead to completely different deep neural networks.

References

[1] Timo Ahonen, Abdenour Hadid, and Matti Pietikäinen. Face description with local binary patterns: Application to face recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(12):2037–2041, Dec 2006.

[2] Steve Awodey. *Category Theory*. Number 52 in Oxford Logic Guides. Oxford University Press, Oxford, UK, 2nd edition, 2009.

[3] Frantz Baader, Ian Horrocks, and Ulrike Sattler. Description logics. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies (2nd Edition)*, International Handbooks on Information Systems, pages 21–44. Springer, New York, 2 edition, 2009.

[4] Sugata Banerji, Atreyee Sinha, and Chengjiun Liu. New image descriptors based on color, texture, shape and wavelets for object and scene image classification. *Neurocomputing*, 117:173–185, 2013.

[5] Wayne Blizard. Multiset theory. *Notre Dame Journal of Formal Logic*, 30(1):36–66, 1988.

[6] Fernando Bobillo, Carlos Bobed, and Eduardo Mena. On the generalization of the discovery of subsumption relationships to the fuzzy case. *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–6, 2017.

[7] Fernando Bobillo and Umberto Straccia. fuzzydl: An expressive fuzzy description logic reasoner. In *Proc IEEE International Conference on Fuzzy Systems*, pages 923 – 930, 07 2008.

[8] Stefan Borgwardt, Marco Cerami, and Rafael Peñaloza. The complexity of subsumption in fuzzy EL. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pages 2812–2818. AAAI Press, 2015.

[9] Stefan Borgwardt and Rafael Peñaloza. Positive subsumption in fuzzy EL with general t-norms. In Francesca Rossi, editor, *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, pages 789–795. AAAI Press, 2013.

[10] Stefan Borgwardt and Rafael Peñaloza. Consistency reasoning in lattice-based fuzzy description logics. *International Journal of Approximate Reasoning.*, 55(9):1917–1938, 2014.

[11] Ronald J. Brachman and Hector J. Levesque. The tractability of subsumption in frame-based description languages. In Ronald J. Brachman, editor, *Proceedings of the National Conference on Artificial Intelligence*, pages 34–37. AAAI Press, 1984.

[12] Hilde Bras, Aart C Liefbroer, and Cees H Elzinga. Standardization of pathways to adulthood? an analysis of dutch cohorts born between 1850 and 1900. *Demography*, 47(4):1013–1034, 2010.

[13] F Carré and M Jacobson. Numerical classification of soil profile data using distance metrics. *Geoderma*, 148(3-4):336–345, 2009.

[14] Reinhard Diestel. *Graph Theory*. Springer, New York (NY), 3rd edition, 2005.

[15] Cees H. Elzinga. Sequence similarity - a non-aligning technique. *Sociological Methods & Research*, 31(4):3–29, 2003.

[16] Cees H. Elzinga, Sven Rahmann, and Hui Wang. Algorithms for subsequence combinatorics. *Theoretical Computer Science*, 409(3):394–404, 2008.

[17] Cees H. Elzinga and Hui Wang. Kernels for acyclic digraphs. *Pattern Recognition Letters*, 33:2239–2244, 2012.

[18] Cees H. Elzinga and Hui Wang. Versatile string kernels. *Theoretical Computer Science*, 495:50–65, 2013.

[19] Cees H. Elzinga, Hui Wang, Zhiwei Lin, and Yash Kumar. Concordance and consensus. *Information Sciences*, 181:2529–2549, 2011.

[20] Gilles Falquet. Subsumption algorithms for description logics. Université de Genève, `http://cui.unige.ch/isi/icle-wiki/_media/cours:sw:subsumption_algos.pdf(Accessed4Feb2017)`.

[21] Stefano Ferilli, Nicola Di Mauro, Teresa M. A. Basile, and Floriana Esposito. A complete subsumption algorithm. In Amedeo Cappelli and Franco Turini, editors, *Advances in Artificial Intelligence: 8th Congress of the Italian Association for Artificial Intelligence, Pisa, Italy*, pages 1–13. Springer, Berlin, Heidelberg, 2003.

[22] Bernhard Ganter, Gerd Stumme, and Rudolf Wille, editors. *Formal Concept Analysis: Foundations and Applications*. Number 3626 in Lecture Notes in Artificial Intelligence. Springer-Verlag, 2005.

[23] Nicola Guarino and Christopher A. Welty. Identity and subsumption. In Rebecca Green, Carola A. Bean, and Sung Hyon Myaeng, editors, *The semantics of Relationships: An Interdisciplinary Perspective*, Information Science and Knowledge Management, chapter 7, pages 111–125. Kluwer, Dordrecht, The Netherlands, 2002.

[24] Richard Hammack, Wilfried Imrich, and Sandi Klavžar. *Handbook of Product Graphs*. Discrete Mathematics and its Applications. CRC Press, Boca Raton, 2nd edition, 2011.

[25] Seung-Cheol Kim and Eun-Soo Kim. Fast computation of hologram patterns of a 3D object using run-length encoding and novel look-up table methods. *Applied Optics*, 48(6):1030–1041, 2009.

[26] Christopher H. Messom, Gourab Sen Gupta, and Serge N. Demidenko. Hough transform run length encoding for real-time image processing. *IEEE Transactions on Instrumentation and Measurement*, 56(3):962–967, 2007.

[27] Ralf Möller and Ralf Haarslev. Tableau-based reasoning. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies (2nd Edition)*, International Handbooks on Information Systems, pages 509–528. Springer, New York, 2 edition, 2009.

[28] Louis Ranjard and Howard A Ross. Unsupervised bird song syllable classification using evolving neural networks. *The Journal of the Acoustical Society of America*, 123(6):4358–4368, 2008.

[29] Achyanta Kumar Sarmah, Shyamanta M. Hazarika, and Smriti Kumar Sinha. Formal concept analysis: current trends and directions. *Artificial Intelligence Review*, 44(1):47–86, 2015.

[30] Taha Sochi. New program with new approach for spectral data analysis. *Measurement*, 46(9):3502 – 3507, 2013.

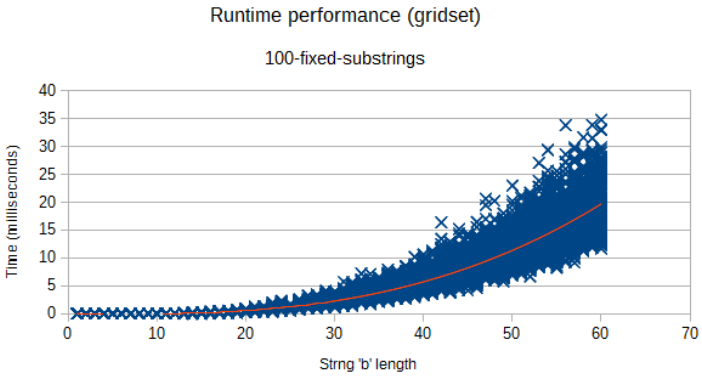
[31] Weiran Song, Hui Wang, Paul Maguire, and Omar Nibouche. Nearest clusters based partial least squares discriminant analysis for the classification of spectral data. *Analytica Chimica Acta*, 1009:27 – 38, 2018.

[32] Apostolos Syropoulos. Mathematics of Multisets. In Christian S. Calude, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Multiset Processing. Mathematical, Computer Science and Molecular Computing Points of View*, number 2235 in Lecture Notes in Computer Science, chapter 17, pages 347–358. Springer, Heidelberg, 2000.

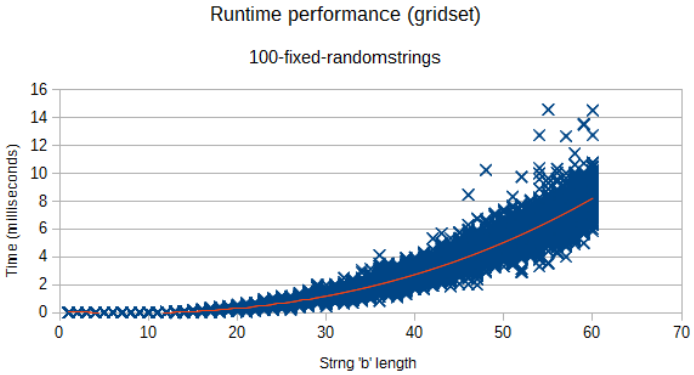
[33] Jordan Vincent, Hui Wang, Omar Nibouche, and Paul Maguire. Differentiation of apple varieties and investigation of organic status using portable visible range reflectance spectroscopy. *Sensors*, 18(6), 2018.

[34] William A Woods. Understanding subsumption and taxonomy: A framework for progress. In John F Sowa, editor, *Principles of Semantic Networks: Explorations in the representation of knowledge*, pages 45–94. Morgan Kaufmann, 2014.

[35] John Yen. Generalizing term subsumption languages to fuzzy logic. In John Mylopoulos and Rick Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 472–477. Kauffman, 1991.

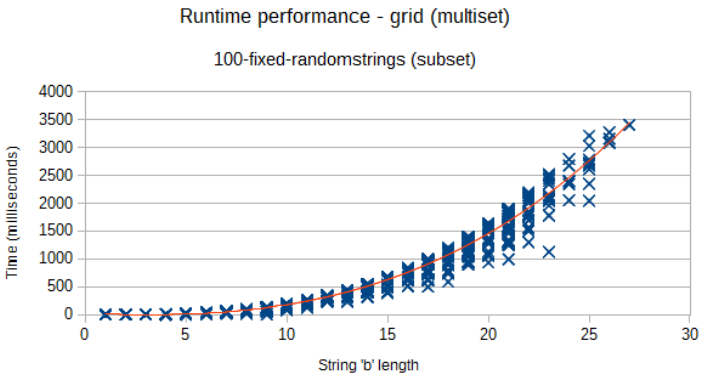


(a) On the 100-fixed-substrings dataset

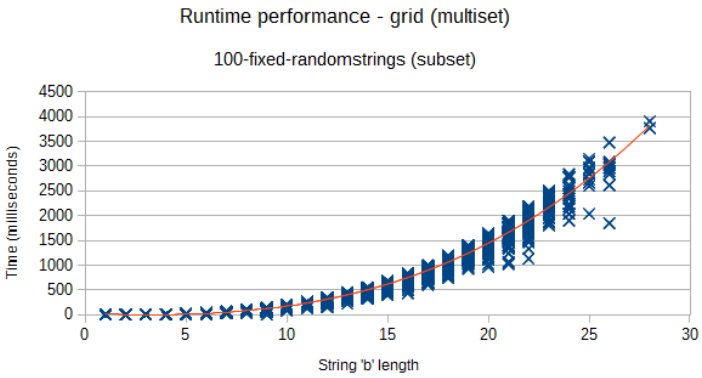


(b) On the 100-fixed-randomstrings dataset

Figure 7: The Grid (set) algorithm

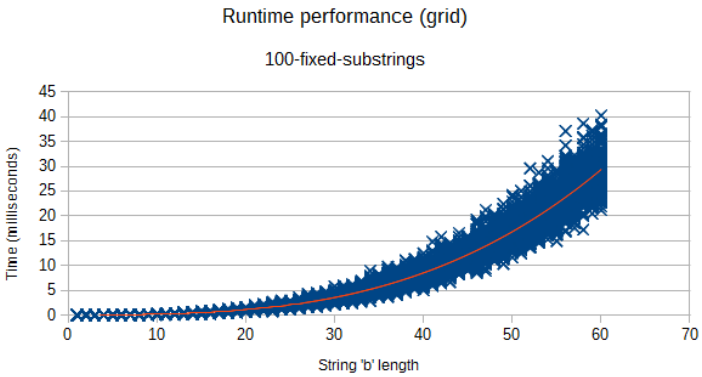


(a) On the 100-fixed-substrings dataset

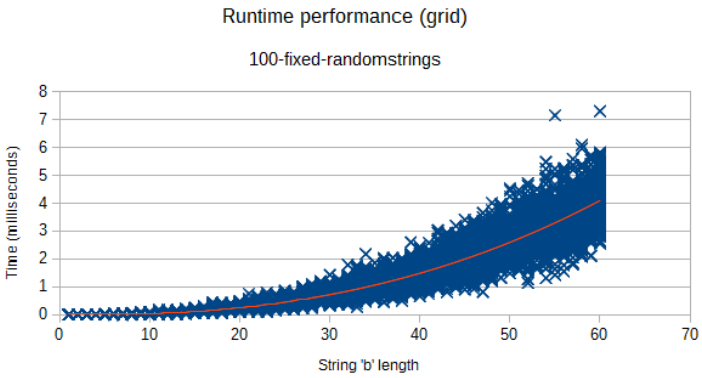


(b) On the 100-fixed-randomstrings dataset

Figure 8: The Grid (multiset) algorithm

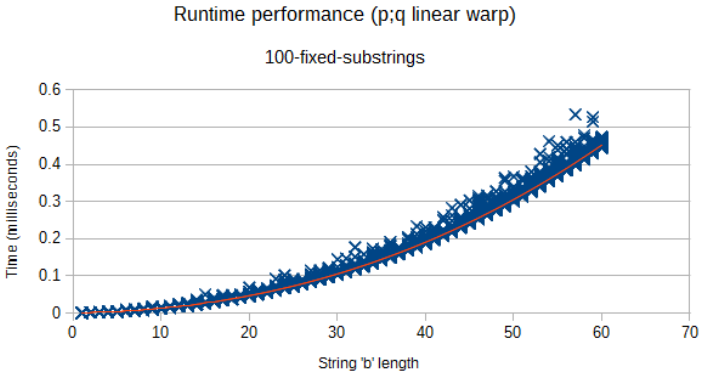


(a) On the 100-fixed-substrings dataset

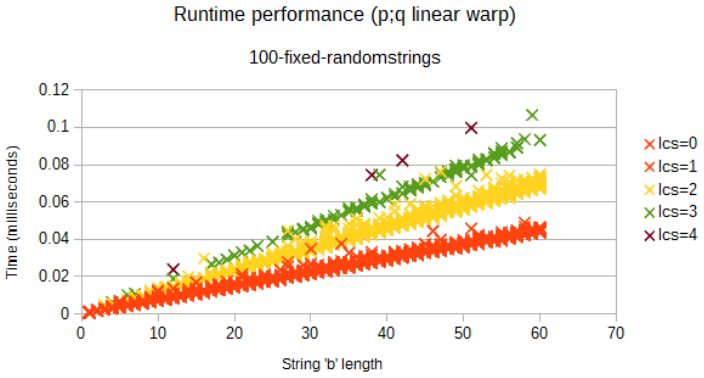


(b) On the 100-fixed-randomstrings dataset

Figure 9: The Grid algorithm



(a) On the 100-fixed-substrings dataset.



(b) On the 100-fixed-randomstrings dataset. The 'lcs' values are the length of the longest common substring for each pair of strings.

Figure 10: The Linear Warping algorithm

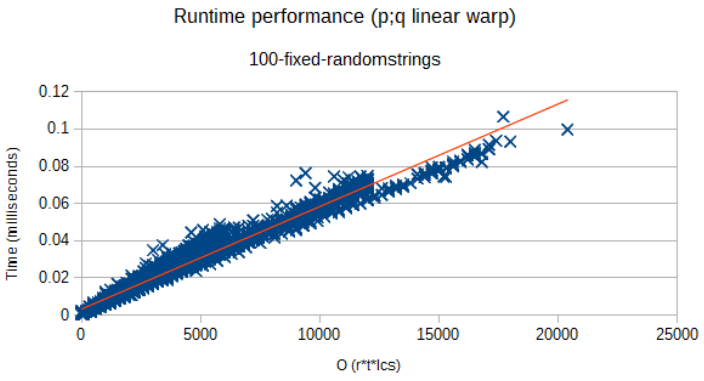
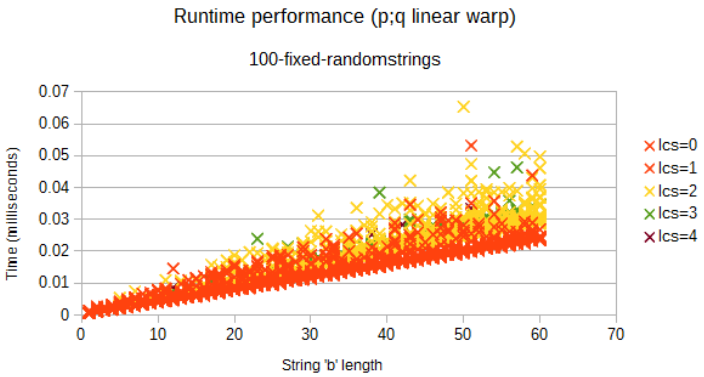
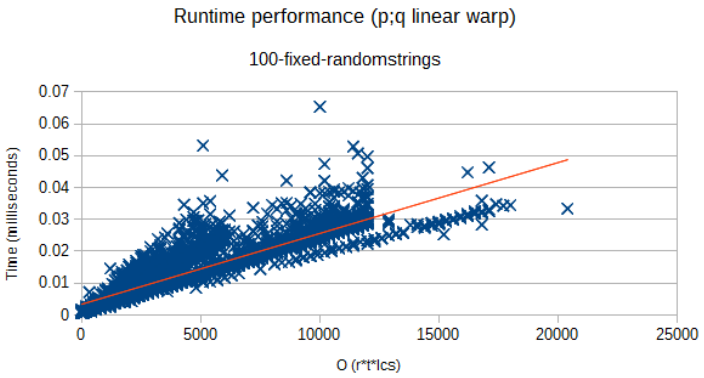


Figure 11: The (p, q)-linear warping algorithm on the 100-fixed-randomstrings dataset, where the X axis is the proposed runtime calculation for the algorithm.



(a) On the 100-fixed-randomstrings dataset



(b) On the 100-fixed-randomstrings dataset

Figure 12: The Optimised Linear Warp Algorithm