Nota et Ignota: Problems and Desiderata in the Constitution of e-Corpora.

Emanuele Dolera, Federico Giusfredi, Alfredo Rizza
Università degli Studi di Pavia

1. Introduction[1]

1.1 Corpora of ancient texts

A corpus of ancient texts can be defined as a set of historical artifacts: we can actually say it is a conventional mapping of historical artifacts into a set, thus, in a sense, it is an "artifact" in itself. The original item(s) of such a set are written texts that are preserved by a direct[2] or indirect tradition: not every element in it is usually well preserved or perfectly survived. Some elements may be damaged (as indeed we know there's normally a large amount of them) so we must cope with a potentially infinite number of mistakes and misprintings (both ancient and modern). A digital version of a corpus of ancient texts should be defined as a set of digital information being isomorphic to the original corpus and satisfying the need for authenticity. Digital information are character data (CDATA) or other Objects, like images or other multi-media files, able to represent the nota and the ignota . We believe that what is known is as important as what is unknown for the constitution and the understanding of any corpora. Our electronic corpus should then guarantee full accessibility and validity. In this paper we will present a particular approach to language corpora constitution and analysis as regards the interaction between human understanding and machine Artificial Intelligence.[3]

1.2 Two Directions

We consider two different approaches to electronic-corpora constitution. With the first one a maximum of descriptive information, provided together with the data to the computer, constitutes an electronic corpus fully capable to offer all basic information for all operations that we would define as 'deductive' (analytical).

1 We would like to express our gratitude for having accepted our paper and given us the occasion to present our projects and idea. We will subdue to your attention our problems and our perspective in the realm of electronic-corpora constitution. We will base our reflections principally on Anatolian texts taking the problematic of cuneiform and hieroglyphic documents as examples and as training material. The work in this project has been undertaken by all three authors with their different competence; in particular, §3 is to be ascribed to Emanuele Dolera; §2, §4, §6.2 to §6.7 to Federico Giusfredi; §1, §5 and §6.1 to Alfredo Rizza; the remaining to the three authors.

2 We consider as direct tradition, for the sake of convenience, also monumental texts and other material culture carrying written evidences.

3 We save for another occasion a discussion about which software should be preferable to realize the project.

A rather different approach requires a more interactive involvement on the side of the machines. Instead of providing a full description of the content, the computer is given a powerful set of computations with which becomes possible the evaluation of minimal and "tentative" heuristic information. The researchers are then allowed to propose different hypothesis at different stages of the "creation" of the electronic corpus. This approach should be mostly interesting in the case of partially or totally unknown languages.

2. A theoretical four-step algorithm

2.1 General structure

For the automatic processing and marking up of a corpus of ancient texts, we propose a four-step Expectation-Maximization to be run by a computer or a net of computers.[4] In some features, our system will show some differences from the classical Expectation-Maximization [Dempster et Al., 1977] and Hidden-Markov-Model[5] protocols. Such differences are due either to the necessity of saving time and space in terms of algorithm complexity and input size or to the mechanical features that, we reckon, lie behind the structure of the linguistic analysis of a complex set of written texts (which may contain different languages, damaged sections, scribal mistakes and so on).

2.2 Overview

The four steps of the algorithm are respectively:

- STEP 1. The software will link all the symbols in the texts to a database with a unique identifier[6] for all the signs of a script; once it has completed the linking phase, it will find out the correct order of reading (that could be right to left, left to right, top down, bottom up or even complex or chaotic).[7]
- STEP 2. Once the signs identity and disposition is recognized, the software finds out the correct value of each sign (in case of multiple values, like in cuneiform and hieroglyphic Anatolian scripts) and then proceeds to separates distinct words.

4 A step 0 is to be postulated: it represents the input phase before the software begins to run. We assume an automatic collection of data to be processed; if it doesn't exist, like in the case of Hittite (as well as Akkadian and Sumerian) cuneiform documents, data will be input in the form of a standard diacritical philological transcription, in order to give the machine all information it needs to start.

5 Cfr. [Y. Ephraim and N. Merhav, 2002].

6 Let us consider for instance Anatolian languages. For a database containing the list of all the signs of Cuneiform Hittite (and Hattic, Palaic, Cun. Luwian), we will base ourselves on the sign list by [Rüster and Neu, HZL]. For Hieroglyphic Luwian scholar in Anatolian will have to compare different works (two, at least: [Marazzi, 1990] and [Hawkins, 2000]).

7 On this problem, see [Shou de Lin et al., 2006].

• STEP 3. Working on a minimal DB of words, the software will identify recognizable segments as spy-words and it will map them all over the documents. It will then proceed dividing the text in conventional sentence-like sections (assuming as a provisional rule that one sentence can only contain a single spy-word).
• STEP 4. Finally, it will attack unknown words with a cluster strategy in order to recognize them and to proceed with learning; all new data will be re-used as new cycle input.

2.3. The algorithm as Coherence Tester

This algorithm is thought to be isomorphic to the process of investigation of a corpus by a scholar. It is not just a mere "storing" device for data scholars already know, but it represents also a heuristic tool. Its application to languages that are still nearly unknown obviously will not converge to optimal results in a reasonable time. Nevertheless it is possible to weight the internal coherence and the likeliness of a set of different theories built upon hypothetical inputs. A very incoherent set of hypothesis would generate a low output set of probabilities within a corpus, while a coherent one will approximate to the values of a well attested data-set.

3. Step 1: About the best choice of linear ordering

This section deals with general processes to discover the writing order for types of two-dimensional ancient scripts. We provide different algorithms, taking care of indicating the computational complexity. The exposition is split into four subsections, in which are described the different stages: the planning of the model, the probabilistic operations with their theoretical justification and the choice of a linear ordering with a final improvement of the previous steps.

3.1 Modeling

The starting point is just algorithmic. First of all, the computer might recognize the various symbols and collect them in a complete data-base. For our aim, knowledge of single symbols is evidently not sufficient, so we focus attention on very short ordered sequences of characters, such as couples or triples. For simplicity, we are going to treat only couples.
Thus, let us think of the text in consideration as an $r \times s$ matrix of (possibly unknown) characters, r being the number of lines and s the number of columns; the graphical elements of the text are indexed in the standard way, following a left-to-right order for the line index and a top-bottom order for the column index. The computer can list all the different couples found in the text by considering, for every symbol, those in its immediate neighborhood, which are at most eight; we can assume that the language in consideration logically allows only these aggregates. The creation of a data-base with all the couples requires an algorithmic complexity of type $O(n)$, with $n = r \cdot s$.

From a probabilistic point of view, this data-base constitutes a finite *sample space* $\Omega = \{\omega_1, \omega_2, \ldots, \omega_N\}$, where each outcome ω_i stands for an ordered couple.
To conclude this preliminary stage, we may think about all the possible linear arrangements of the symbols, disposed in the two dimensions. Gathered here are some fundamental conventions which seem rational to us. Firstly, the starting point is always the symbol in position (1,1), located on the line at the top, on the left column. Then, the sequence of characters must rigorously obey a neighborhood principle, that is the successor of a symbol in position (i, j) can be only one among those in position $(i + 1, j)$, $(i, j+1)$ or $(i+1, j+1)$. After connecting consecutive symbols, the resulting line must not intersect itself. Afterwards, we consider only linear orderings of horizontal or vertical type, with the proviso that the typology can not change from one to another in the middle of the text. Moreover, an arrangement of horizontal-type is organized on couples of lines, that is an element in the i^{th} line cannot be connected with any element in the $(i+2)^{th}$ line before reaching the last column; an analogous statement is in force for vertical-types. We are implicitly assuming that the right linear orders of the symbols share all the above-mentioned properties and, from now on, the terms *ordering* and *arrangement* will designate only those with those features.
One further consideration about orderings is necessary. The number of all possible arrangements increases exponentially with n, so, when n is large, it is useless giving an algorithm which considers all the arrangements. Indeed, let $N(k)$ be the number of ways in which one can read $2k$ symbols equally organized, for example, on two lines; for convention it is not included in $N(k)$ the canonic order, in which every line forms a disjoint sequence from the others, and each line is read from left to right. Then, $N(k)$ satisfies the recurrent relation $N(k) = 2\,[N(k-1) + N(k-2)]$ when $k \geq 3$, together with initial conditions $N(1) = 1$ and $N(2) = 3$. To overcome the difficulty of the exponential complexity which this system yields, we can start fixing a number k, not too small, for which $N(k)$ is reasonable for the computer. Then, we construct global orders starting from groups of $2k$ symbols. For horizontal-type orders we proceed considering all the $N(k)$ possible ones, together with the canonic one, formed with the first $2k$ symbols in the first two lines if $k \leq s$, while, if $k > s$, the arrangements extend to more couples of lines; each arrangement is then repeated periodically to the whole text. The same procedure is applied to the vertical-type orders. In this way, we have defined the class of linear orders we want to investigate in which we hope to find the right one. In every case, this class can be modified while getting more and more information. This point will be clarified in the final remarks.

3.2 The probabilistic algorithm

This probabilistic investigations are inspired to the first Shannon's pioneering works on noisy channels.

Knowledge about the probabilities of couples of symbols in the examined text would lead to discover the correct ordering for the script. Indeed, let us consider the same value of k for which $\mathrm{N}(k)$ is reasonable for the computer and let us list all the arrangements A_h of the first $2k$ symbols, where $h = 1, 2, \ldots, \mathrm{N}(k)$. For each A_h one has an ordered sequence of couple, namely $\omega_{i_1(h)}, \omega_{i_2(h)}, \ldots, \omega_{i_{2k-1}(h)}$, and this gives a reasonable form for the probability of A_h by the formula $\mathsf{P}(A_h) = \prod_{l=1}^{2k-1} p_{il(h)}$, where $p_m := \mathrm{P}(\{\omega_m\})$.

Thus, the correct ordering A_{h^*} can be found by

$$h^* = \mathrm{argmax}_{1 \le h \le \mathrm{N}(k)} \mathsf{P}(A_h) \ . \tag{1}$$

This procedure is then extended to the next group of 2k characters and so on.

When the a-priori probabilities of the couples are unknown we can proceed by conditioning, observing the situation under the hypothesis that A_h is the right ordering. We start defining the *conditional probabilities given an arrangement,* $p_{ih} := \mathsf{P}(\{\omega_i\} \mid A_h)$, simply computing the *frequency of appearance* of the i^{th} couple, following the arrangement A_h. Then, we introduce, for every arrangement, the following *distribution functions*

$$\mathsf{F}_h(x) := \frac{\#\{\mathrm{i} \mid p_{ih} \le x\}}{N} \quad (x \in \Re) \tag{2}$$

where # stands for the cardinality of the relative set. Note that they all have support in the interval [0, 1]. These functions tell us how the probabilities distribute among the possible outcomes. For example, if the p_{ih} uniformly distribute (i.e. $p_{ih} \sim 1/N$), the relative F_h approaches a degenerate distribution with unit mass in $1/N$, which denotes that the values of the probabilities are all concentrated on the mean; on the contrary, F_h is similar to the uniform distribution when the probabilities p_{ih} are variable and "spread" on the various outcomes. The test for the choice of the good ordering is conducted by comparing the F_hs with a fixed probability distribution function G, using probability metrics, such as the *Kolmogorov distance*:

$$\mathrm{d_K}(\mathsf{F}_h; \mathsf{G}) := \sup_{x \in \Re} |\mathsf{F}_h(x) - \mathsf{G}(x)| \ .$$

Other significative choices are the *Lévy distance* or the *Gini-Kantorovich-Wasserstein distance*. See the book of [Rachev, 1991] for their definitions. The choice of the ordering is done by the formula

$$h^* = \mathrm{argmin}_{1 \le h \le \mathrm{N}(k)} \mathrm{d_K}(\mathsf{F}_h; \mathsf{G}) \ . \tag{3}$$

This operation requires a complexity of type $O(N \cdot \mathrm{N}(k))$, which can be reduced to $O(\mathrm{N}(k))$ using smoothing techniques. We noticed that this kind of test, which vaguely recalls a *Kolmogorov-Smirnov test*, is statistically justified by the mean of the

Glivenko-Cantelli Theorem, since we are assuming that the function G is derived by the *real* distribution of the couples.
The conclusive issue is the construction of the function G: we suggest to start with a *Beta distribution,* whose parameters can be estimated with *Bayesian techniques.* We can give different *p-quantile* or an estimation of the *variance,* which says how sure we are of this construction. For example, it is very significant the fraction of couples which are—according to our taste—meaningless. The estimation of the parameter of the Beta can be improved in the course of the study.

3.3 Entropy methods

Another method which can be useful to face the problem of discovering the correct linear order employs entropy. We will give only a generic idea, stating that the above-mentioned test is more powerful than the one we are going to explain. One can define the *entropy* relative to an ordering $\mathsf{H}(A_h) := -\sum_{i=1}^{N} p_{ih} \log(p_{ih})$ and find the arrangement which minimizes it:

$$h^* = \operatorname{argmin}_{1 \le h \le \mathrm{N}(k)} \mathsf{H}(A_h) \ . \tag{4}$$

Such a kind of test is statistically justified by the mean of the *Law of Large Numbers.*

3.4 Final remarks

The methods we have explained can be combined in order to get better results. In these final remarks we discuss how the computer can improve the above-mentioned algorithms. Recall, in fact, that we have treated only *periodic* orderings of period $2k$: we can increase the period after having got information. For example, if one discovers that arrangements which do not contain diagonal connections are better than the others, it can be re-considered the class of all the orderings without those with diagonal connections. In this way, this class shrinks more and more and the value of the period can increase.

4. Step 2: Signs recognition

4.1 General function

Now the system owns a string of signs and it knows in which order it has to process them. It needs, first of all, to find out the correct value of each sign. As minimal Expectation Input (algorithm state 0), the human compiler will give the machine a set of conditioned probabilities to be processed by the Bayesian equation:[8]

8 Or, on a logarithmic scale for computational simplicity,

$$\operatorname{argmax} \mathsf{P}(s_1 \ldots s_n) = \operatorname{argmax} \sum_{i=1}^{n} \mathsf{P}(s_i | s_{i-1}, s_{i-2})$$

$$\text{argmax } \mathsf{P}(s_1 \ldots s_n) = \text{argmax } \prod_{i=1}^{n} \mathsf{P}(s_i|s_{i-1}, s_{i-2}) \quad (5)$$

where the random variable s_n represents the sign value at the n^{th} position, and a cluster analysis is carried out on strings of three signs s_n-2, s_n-1, s_n. This step requires a big amount of starting input, since in case we give instructions about n signs including the space separating words, the amount of data is contained in a n^3 rows DB (for 50 sign values, a good input for Hittite cuneiform, we would have 10^5 data-rows). The advantage is that, in a EM process, after this starting instruction the software will be able to go on building the DB until all the sequences have been studied at least once, during the preceding cycles of the algorithm and specifically of step 2.

4.2 Sequences and ghost space

The reason why we prefer to work on triples of signs is determined by the need to discover word divisions. Single words are not always graphically marked in writing systems such as cuneiform Hittite. Using triples of signs, we can ask the computer to analyze each single sign as hypothetically included between "two spaces",[9] and confront the related probability with the probabilities of strings with a minor or null number of spaces. In this way, using the above equation just once,[10] we will be able to separate words and to recognize the correct value of multiple signs. Clearly, in case spaces are sometimes noted and sometimes ignored within the original script,[11] the sporadic presence of a graphic space will affect the probability for a linguistic boundary to fall in a certain position; in case spaces are always marked, this step can be simply ignored (or run without any consequences, setting the ghost spaces frequency to 0).

4.3 Processing *lacunae*

In ancient documents, a peculiar typology of blank space can also denote a missing part (lacuna) of the text, or a damaged one. Luckily, the direct input in step 0 should instruct the computer how to recognize this accidents from the spaces lying

9 We will call these interruptions "ghost-spaces", because we do not know if linguistic spaces are always marked and if a graphic space represents a linguistic boundary.

10 The minimal supported triple will be space-sign-space, which is necessary in order to process words division.

11 This is for instance the case of cuneiform Hittite, where spaces can either be noted or ignored. Moreover, in some cases a graphic long space is inserted by the scribe for sake of elegance or in order to fill in all the space in a row of the tablet, even though no linguistic boundary exists. In that cases, we should probably distinguish between ghost spaces (hypothesized by the software), good graphical spaces (short ones on the tablet), bad graphical spaces (very long ones on the tablet). This situation represents a clear example of the inconsistency of some writing systems. For graphic inconsistency see [Sproat, 2000, passim].

between signs and words, so we have just to care about the way the software has to treat them. We can isolate two different cases, in which the machine will differently behave.

- The lacuna (or the damage section) is a short one. How short is up to the human compiler to decide; the software will try and integrate it using a reverse probability function on the sign or the word variable. Note that any Bayesian equation of form $P(Z|Y)$ can be reversed respecting the theorem:

$$P(Z|Y) = \frac{P(Z|Y)\,P(Y)}{P(Z)}$$

- The lacuna is a long one. The software will simply renounce to solve it and proceed analyzing from the first recognizable form following the problematic section.

For sake of homogeneity, and in order to let the Markov chain proceed, we have to assign a dummy-value to the probability of elements to occur after a lacuna. In order to limit the damages, we will assign a neutral value to it, so that for each variable X in any step of the algorithm, we define: $P(X|\text{Lost}) = 1$. This will guarantee that the lacuna will not negatively influence the ongoing of the process.

5. Step 3. Collecting words and first tentative attribution of functions

5.1 Overview

After step 2, the software will analyze any sequence between two linguistic boundaries as one word. Step 3 consists in confronting these words with an input database of words and morphemes. The system will try and recognize words belonging to a word class with a high rate of morphological variability (or some other rules), and will hypothesize for this set of words to be VERBS. It will then re-run the text and mark this words (that have become now spy-words) as VERBS in the DB of words and in the text.

5.2 Analyzing the DB

Until now we have failed to talk about one of the principal element in corpora constitution: the so called "tokenization". Our system is a multi-layered one so there might be different levels where to apply the "token-type" scheme. For the definition of these concepts we are now referring to [Barbera - Onesti - Corino, 2007, pp. 35-37 and passim] where correct reference is traced back to semiotic and logic, specifically to the work of Peirce and Quine. At the level under consideration in step 3 a token will be any instance of a graphic word (actual or reconstructed) as devised by the algorithm. Now the system can perform all the analytical operations that we consider important; in what follows I will just mention some of them.

First we need an orderable index of all tokens with sums and frequencies; tokens should be reduced to types and an index of types with sums and frequencies will be built. Then we must consider sequences of tokens together with their relative frequency and estimation of conditional probabilities all gathered within a table of concordance. How to find sequences is a task of minimal effort for the computer. The system will simply start with the first token and will consider all the possible strings of consequent tokens starting with the same initial token and it will note their frequency; when it finds a string with frequency 1 it stops building strings from the first token of the text and starts again in the same way with the second token. So if i is the index indicating the starting point and $f(i,k)$with $k > i$ is the frequency of strings of subsequent tokens $i, i+1, \ldots, k$, every time $f(i,k) = 1$ a new starting point is selected immediately after the one just used. Another set of operations will put forward morphological evaluations[12] on tokens using available information on frequencies and conditional probabilities together with hypothesis introduced by the researchers. These operation will ultimately lead first to the individuation of possible identification of radicals/stems and affixes and, then, to the identification of tokens with a particular function, like a verbal predicative one. The last moment of step 3 consists in marking all tokens (spy-words) with their inferred functions all through the text.

6. Step 4: Sentence boundaries and further word recognition

6.1 Sentence Boundaries

Now that we have marked some tokens as spy-words all over the text (e.g. verbs, like in row 1 and 2 in the following table). We will enucleate homogeneous strings containing only one spy-word. Let's consider the spy-word is a verb:[13] the system will order the surrounding tokens following rules determined by a binary parameter of prominence in the order: verb (marked as V in the table below) - not verb (marked as ? in the table below). With the parameter set to left-prominent (as will become clear this parameter indicates a verb-final syntactic alignment), the first sign to the left of every V will be given value 1, the first on the right value 2 and so on with odd values to the left and even ones to the right; the other way round with the parameter set to right-prominent (verb-initial alignment). In cases of overlapping values, e.g. token 235 with value 4/1, for sake of homogeneity the lower or the odd value will be preferred and so the assignment of the element to the sentence (at the moment a 'sentence' is a cluster of tokens that contains just one V), as you can see in row 3 or 4 in the table below:

12 Such evaluation can be based on simple engines working with classic word similarity functions applied to words showing analogue variations in paradigms or flexional behavior, which may be supposed to belong to a same class.

13 We consider this situation as the best for many languages.

Unmapped Tokens	231	232	233	234	235	236	237	238
Mapped	?	?	V	?	?	V	?	V
Left-prominent	3	1	V	2/3	4/1	V	2/1	V
Right-prominent	4	2	V	1	3/2	V	1/2	V

When the choice between lower vs. odd value is set to 'odd', a verb-final or verb-initial alignment is considered (e.g. a SOV word order); when it is set to 'lower', the word order considered is more of the type SVO (OVS).

6.2 Segments

For the first many cycles of the EM algorithm, we will have to deal with the problem of recognizing words that are not yet owned, with a sufficient range of certainty, by the DB in step 3.[14] The goal of step 4 is to carry out a cluster analysis on strings of words in order to maximize the probabilities of word classes to occur in a certain position and to contain a recognizable morpheme that may be already known by the DB in step 3.

6.3 General function

The core function of this step will run only as long as the DB in step 3 is not able to recognize all words. Hence, it is potentially running forever, surely with lower frequency as the cycles proceed. Its form is a little different from the normal Hidden-Markov-Model and Expectation-Maximization Bayesian engines, since we chose to consider with it two factors that are, in our opinion, very loosely coupled, such as the word form and the word position within a text. Let c_n be the n^{th} word-class in the string, and let m_n be a recognizable morpheme being present in the n^{th} word of the string. The equation to be maximized should have this general form:

$$\text{argmax } \mathsf{P}(c_1 \ldots c_n) = \text{argmax} \prod_{i=1}^{n} \mathsf{P}(c_i|m_i) + \prod_{i=1}^{n} \mathsf{P}(c_i|c_{i-1}) \qquad (6)$$

and the minimal input will be constituted by a set of data about frequencies of words in a given position with respect to the spy-word.[15]

6.4 Log scale

Let us limit ourselves to the two main parts of the equation (call them P1 and P2). Each part is represented by a serial product of conditional probabilities: the

14 For discussion see above 6.1.

15 E.g. a noun, an adverb, an adjective following or preceding the verb or the $i - 1^{th}$ after the verb, and so on until the whole segment is processed.

first peculiarity of this equation is that it is isomorphic to the union of two Markov chains. This simple fact is problematic in terms of computation, since the study of max and min will be very difficult for the computer. In fact, even though we can convert both elements P1 and P2 into a homomorphic logarithmic scale,

$$\sum_{i=1}^{n} \log \mathsf{P}(c_i|m_i) \text{ and } \sum_{i=1}^{n} \log \mathsf{P}(c_i|c_{i-1}),$$

we cannot map on the same scale the whole function, so we have to renounce reducing products to sums.

6.5 Why sums?

By this point of view, it would look better if we chose to multiply P1 and P2 instead of summing them:

$$\text{argmax } \mathsf{P}(c_1 \ldots c_n) = \text{argmax } \prod_{i=1}^{n} \mathsf{P}(c_i|m_i)\mathsf{P}(c_i|c_{i-1})$$

but, on the other hand, such a solution would generate, in our system, a peculiar problem. A low entry for one of the two factors would annihilate the other one, even though its own value is very high.[16] The insertion of the sum helps the system to save hypothesis that output, for instance, a high probability given a position and a low one given the morphematic evidence. The first important consequence regards the processing of problematic *ignota*. If the software meets an adverb for the first time and doesn't recognize any clear adverbial morpheme, it will anyway consider the fact that the word occurs in a convincing position for adverbs (or viceversa for a recognizable word in a position that is processed for the first time). Since the EM is always learning something new, in the first cycles it will not possess all the data about syntaxis and morphology, so this "caution-device" is absolutely necessary. The second consequence regards instead the processing of scribal errors. Sentence "This fish is is not tasty" contains a repetition, while sentence "This fish is not tasty" is a correct one. The repetition would automatically compromise the positional structure of the sentence; in other words the probability will be close to zero (formally, if w = 'verb', $\mathsf{P}(w_{n+1})|\mathsf{P}(w_n) \approx 0$). Any serial product would give a close-to-zero output value. Since these two sentences are semantically equivalent and for many aspects similar to each other, we don't want them to have a very different probability output. The insertion of a sum would limit damages and avoid one of the functions have a close to zero value. We propose to call this property *stability*.

16 In other words, the problem is simply that $\lim_{P1 \to 0} P1 \times P2 = 0$ for each value of P2 and $\lim_{P2 \to 0} P1 \times P2 = 0$ for each value of P1.

6.6 Final arrangements
Let us now go back to equation (6):

$$\text{argmax } \mathsf{P}(c_1 \ldots c_n) = \text{argmax } \prod_{i=1}^{n} \mathsf{P}(c_i|m_i) + \prod_{i=1}^{n} \mathsf{P}(c_i|c_{i-1})$$

For definition:

$$0 \leq \prod_{i=1}^{n} \mathsf{P}(c_i|m_i) \leq 1 \text{ and } 0 \leq \prod_{i=1}^{n} \mathsf{P}(c_i|c_{i-1}) \leq 1.$$

Let us assume they are autonomous and observe their behavior as two separate elements. If no weight intervention is operated, given two possible maximum values fitting $P1$, $P2$ and $P1'$, $P2'$ with $P1 = P2'$ and $P2 = P1'$, we will see that both solutions satisfy equation (6): this fact defines a symmetry.
In other words the maximum is not unique. Operating a choice between two identical (or very similar) values that maximize (6) for a string means privileging the information about the morphematic structure of words or the information about their position within the string itself. Clearly, we must instruct the software about our priorities, and in order to do it we will insert two parameters α and β, summing up to 1, that will affect the output probability values of both P1 and P2. Hence the final form of the equation will be:

$$\text{argmax } \mathsf{P}(c_1 \ldots c_n) = \text{argmax } \alpha \prod_{i=1}^{n} \mathsf{P}(c_i|m_i) + \beta \prod_{i=1}^{n} \mathsf{P}(c_i|c_{i-1})$$

$$\text{where} \quad \alpha + \beta = 1 \tag{7}$$

6.7 Meaning of α and β parameters
The insertion of α and β permits us to operate an important selection among a cluster of values that have a similar output but a very different meaning. Yet, the idea of setting them before cycle 1 and saving them until the corpus is completed would be problematic. In fact, the accuracy of the software with respect to syntax and morphology may vary while the cycles proceed. Probably when the DB in step 3 is nearly completed for all attested words, the morphological data set will be so heavy that parameter α should tend to 1, parameter β to 0. Due to this fact, we suggest to consider α and β as variable at the beginning of every new cycle. Empirical experience will give us details about the engine of this variation.

7. Conclusion

The creation of a software being able to support research, step by step, in both analyzing and marking up ancient texts and corpora would be, we reckon, extremely

important, and the state of the art technologies permit to create such a device. The algorithm presented in this paper is a possible form such a tool can take and, we believe, a very strong and ductile one.

References

Computer Science:

Dempster et al., 1977: A. P. Dempster — N. M. Laird and D.B. Ruby. 1977. Maximum Likelihood from Incomplete Data via the 'EM' Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)* 39/1: 1-38.

Ephraim et al., 2002: Y. Ephraim and N. Merhav. 2002. Hidden Markov processes. *IEEE Transactions on Information Theory* 48: 1518-1569.

Neal et al., 1999: R. Neal and G. Hinton. 1999. A view of the EM algorithm that justifies incremental, sparse, and other variants. In: M. I. Jordan (ed.), *Learning in Graphical Models*. Cambridge (MA): MIT Press, pp. 355-368.

Rachev, S.T. 1991. *Probability metrics and the stability of stochastic models.* Chichester: Wiley series in probability and mathematical statistics.

Shannon, C.E. 1949. *A Mathematical Theory of Communication*. Urbana: University of Illinois Press.

Shou de Lin et al., 2006: Shou de Lin and K. Knight. 2006. Discovering the linear writing order of a two-dimensional ancient hieroglyphic script. Information Sciences Institute, University of Southern California.

Sproat, R. 2000. *A Computational Theory of Writing Systems*. Cambridge: Cambridge University Press.

Winkler, W.E. 1999. *The state of record linkage and current research problems.* Technical report, Statistical Research Division, U.S. Bureau of the Census, Wachington, DC.

Philology:

Barbera - Onesti - Corino, 2007: M. Barbera — E. Corino and C. Onesti (eds.), 2007. *Corpora e linguistica in rete*. Perugia: Guerra Edizioni

Barbera - Onesti - Corino, 2007a : M. Barbera — E. Corino and C. Onesti. 2007. Cosa è un corpus? Per una definizione più rigorosa di corpus, token, markup. In: [Barbera - Corino - Onesti (eds.), 2007], pp. 25-88.

Hawkins, J.D. 2000. *Corpus of Hieroglyphic Luwian Inscriptions*. New York — Berlin: De Gruyter.

Marazzi, M. 1990. *Il geroglifico anatolico: problemi di analisi e prospettive di ricerca.* Roma: Dipartimento di Studi Glottoantropologici Università "La Sapienza".

Rüster and Neu, HZL : C. Ruster — E. Neu. 1989. *Hethitisches Zeichenlexikon.* Wiesbaden: Harrassowitz.