Discovering the Linear Writing Order of a Two-Dimensional Ancient Hieroglyphic Script

Shou-de Lin
Information Sciences Institute
University of Southern California
sdlin@isi.edu

Kevin Knight
Information Sciences Institute
University of Southern California
knight@isi.edu

Abstract

This paper demonstrates how machine learning methods can be applied to deal with a real-world decipherment problem where very little background knowledge is available. The goal is to discover the linear order of a two-dimensional ancient script, Hieroglyphic Luwian. This paper records a complete decipherment process including encoding, modeling, parameter learning, optimization, and evaluation. The experiment shows that the proposed approach is general enough to recover the linear order of various manually generated two-dimensional scripts without needing to know in advance what language they represent and how the two-dimensional scripts were generated. Since the proposed method does not require domain specific knowledge, it can be applied not only to language problems but also order discovery tasks in other domains such as biology and chemistry.

1 Introduction

The Hieroglyphic Luwian script is a two-dimensional script discovered in middle Asia, preserved on rock dating back to 700-1300 BCE [4]. Unlike modern scripts that possess a clear linear order for reading (for example, left-to-right, top-down for English), the Luwian symbols seem not to be arranged regularly enough to indicate any specific writing order (e.g., see the upper line in Figure 1). This paper discusses a general process to discover the writing order (e.g., see the second line in Figure 1) for this type of two-dimensional script.

Figure 1 A snapshot of the two-dimensional Luwian script (upper) and one sample linear order of symbols (lower)

So far, there is no convincing evidence about the precise order of this script. A linguistic authority says "it is a system which may leave in doubt the correct order of reading" [4]. Up to the present day there has not yet been much effort focused on applying machine learning methods for decipherment. Knight and Yamada [5] propose a general framework to discover the text-to-speech relationships for unknown scripts by applying a finite-state transducer model together with the EM (Expectation-Maximization) algorithm to learn parameters. Sproat [7] claims that the orthography of a language represents a consistent level of linguistic representation, and the mapping from this level to surface spelling is a regular relation. He proposes to use a rule-based system (realized by finite state automata) to encode the regularities. Both works aim at finding the relationships among writing symbols and their sounds. We have not yet seen computational approaches for order discovery in any ancient writing system, including Hieroglyphic Luwian.

The challenges of discovering the linear order for unknown scripts are threefold:

1. Modeling: Without any knowledge or examples of linear order, we have to abandon enlisting help from either supervised training or rule writing. This task is as difficult as asking somebody who knows no English to figure out the linear order of a set of English characters

written in two-dimensional manner. The challenge lies in how to model this problem as something that can be handled by machine using unsupervised techniques.

2. Complexity: In general the number of possible orderings increases exponentially with the number of symbols. Hence we face a huge search space.

3. Evaluation: As with all kinds of human discovery problems, there is no easily obtained linear data for this script, which makes it hard to verify the results.

The next section describes how we model the linear-order discovery task as an unsupervised learning problem. We also provide an intuition indicating how the proposed approach resembles what human beings might do for this task. In Section 3, we show how the model described in Section 2 can be applied (and refined) to deal with a real-world problem. Additionally, we will discuss how to reduce the search complexity to polynomial. We describe several evaluation strategies and results in Section 4 and conclude in the last section.

2 Modeling

Our general strategy is as follows: we model the way the Luwians generated the script with Shannon's noisy-channel model [6], which allows us to further translate the problem into a traveling-salesman-like problem. We then apply the EM algorithm to learn the parameters (i.e., a Luwian language model) within the model. Finally we apply these parameters to compute the associated probability for each plausible order in order to extract the one with the highest probability as the result.

2.1 Noisy Channel Model

We start by exploiting the noisy-channel model as shown in Figure 2.

We assume that the ancient people have the original linear script in mind before writing, and then they carve the scripts in a two-dimensional manner on the stone. When time passed (as did the Luwians), the only remains are the two-dimensional scripts on the rocks as observed. Based on Shannon's model, the original linear script is the input X to the noisy

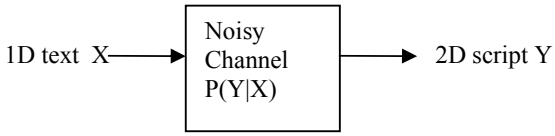


Figure 2 The noisy channel model representing how Luwian people wrote the script

channel. The way the Luwians wrote down the script can be treated as a noisy channel that perturbs the original one-dimensional source text X into a two-dimensional appearance Y. This noisy channel (represented as P(Y|X) here) is a black box to us since we do not know how those two-dimensional scripts were generated. Once the problem is modeled as a noisy channel, it is obvious that this order-discovery task can be represented as finding the X that maximizes P(X|Y).

This formula can further be decomposed into two components, P(X) and P(Y|X), by Bayes' rule (see equation 1)[1]. P(X) can be generated by the language model of the script, which essentially stands for how frequently this script should occur in a large sample of ancient Luwian literature. P(Y|X) can be treated as the noisy channel that represents how the Luwian people wrote down the script. Shannon's model together with Bayes' rule tells us that the desired linear order X should not only appear frequently in the Luwian literature (i.e. high P(X)), but also possess a high chance of producing the observed two-dimensional script Y.

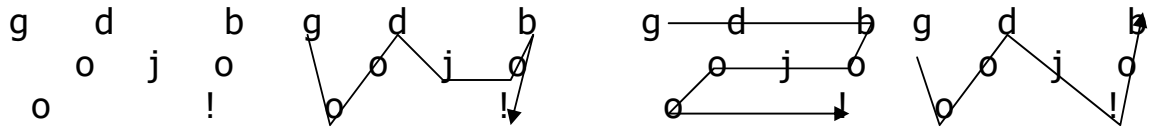
2.2 Linear Order for Known Language

We would like to demonstrate the idea described in Section 2.1 with a toy example: we assume that we are asked to find the linear order of a known two-dimensional language, say the English in Figure 3a, without being told how it was generated.

$$\arg\max_X p(X \mid Y) = \arg\max_X p(X) * P(Y \mid X)$$

Equation 1

[1] In fact there should be an extra term P(Y) in Bayes' rule deduction. However, P(Y) can be ignored in the optimization process since it models the probability of the observed output, which should be identical for all X.



(a) **(b)** **(c)** **(d)**

Figure 3.a: The two-dimensional English. 3.b-3.d: Three arbitrary linear orders

According to Equation 1, we need to search for an order X that maximizes P(X)*P(Y|X). In this example, we ignore P(Y|X), since we assume there is no indication of how this script was generated. Therefore we only need to maximize P(X). In other words, whichever permutation of sequence appears to be valid English is a plausible candidate. People can scramble the characters mentally, and sooner or later they will find out it probably means "good job!" (Figure 3.b), as this permutation occurs more frequently to English speakers than others (e.g., gdbojoo! in 3.c and goodj!ob in 3.d). What happens in the human mind mimics the procedure of maximizing P(X), where meaningful sentences possess higher P(X) than meaningless ones.

This toy example demonstrates that our model, to some extent, reflects how human beings solve the same puzzle. A machine can also perform such an analysis, if it knows which English orders are more probable than others. One way to compute this is to exploit an n-gram English language model, which can be easily obtained by a simple statistical analysis of a large English corpus. Assuming that English letter-bigram probabilities are given, we can program the machine to enumerate all possible orders and compute their associated probabilities (e.g., p(X)=p(goodjob!)=p(g) * p(o|g) * p(o|o) * p(d|o) * p(j|d) * p(b|j) * p(b|o) * p(!|b)). After that, we extract the sequence with the highest probability as the result. The probability of 3.c and 3.d should be much lower than 3.b since p(d|b) in 3.c and p(o|!) in 3.d do not occur too often in English. In this sense, if the bigram language model is known, then the linear-order discovering problem is similar to the traveling salesman problem in a graph (where the

nodes represent letters and the weighted links represent the corresponding bigram probability between letters), except that the salesman does not have to go back to the origin.

2.3 EM for Unknown Scripts

Section 2.2 shows that it is possible to recover the linear order of a known script from its two-dimensional form. However, this approach is not applicable to the decipherment of the Luwian script, whose language model is unknown.

On the other hand, when the order probability P(X) is known, it is possible to apply the method of "fractional counting" to generate the n-gram probabilities by counting relatively how many times each n-gram appears in each possible ordering. For example, if there are three possible orders with associated probability 20%, 30%, and 50%, and the bigram P(e|s) occurs once, once, and twice respectively in these three sequences, the fractional count of P(e|s) will be 1*20%+1*30%+2*50%. Finally, the language model can be constructed by normalizing the fractional count values. Statistically speaking, "learning the language model" and "learning the associated probability for the ordered sequences P(X)" are dual problems, in the sense that one can provide sufficient information to understand the other. Unfortunately we know neither for Luwian. In this case we propose to use the idea of EM [2] to learn both models simultaneously, as shown in Figure 4.

For the initialization, we assign certain probabilities (e.g., a uniform distribution) to all the n-gram parameters in the language model. In the M step of EM, we enumerate all the possible linear orders X and use the imperfect language model learned so far to generate the associated probabilities P(X). In the E step of EM, we count how frequently each n-gram appears for all possible orders X, weighted by their associated probability P(X), to generate a refined language model. Then the new language model can be exploited to create a more precise P(X). It has been proven that if we execute the E step and M step iteratively, EM will ultimately converge on a locally optimal solution (i.e., the parameter settings for the language

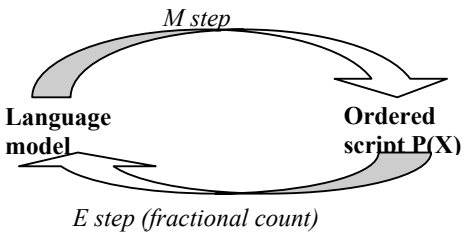


Figure 4: Applying EM to refine the language model and P(X) recursively

model will lead to a locally optimal P(X)). Once the language model is learned, we can finally apply it to compute the associated probability for each order and extract the most probable one.

2.4 Discussion

In fact, what our program does is similar to asking a human to learn the linear order for some script he or she has never seen. Why is it intuitively possible at all? In this section, we would like to provide an insight showing that our proposed method is a generalization of one plausible decipherment process. Thinking of what methods human beings might use to deal with such an ordering-decipherment task, first we might try to group symbols that co-occur frequently in the same neighborhood. Based on the co-occurrence frequencies, we might be able to hypothesize some plausible order in certain less ambiguous areas (e.g., corner areas). Then we could propagate the order information from those less-ambiguous areas to the remaining areas and refine our belief in the co-occurrence frequencies. These steps could be applied repeatedly either until one plausible order is determined or until we learn that the current hypotheses are not applicable to the remaining areas, in which case we would need to backtrack by choosing other hypotheses.

This decipherment process is a simplified version of what our program does. The process of choosing the most plausible direction based on current beliefs (e.g., what symbols co-occur with others frequently) corresponds to the "maximization" step in the expectation-maximization (EM) algorithm. Using the currently plausible order information to modify the

existing beliefs resembles the “estimation” step in EM. The major difference is that human beings can take only a small amount of highly possible choices into account, but the machine is able to take all situations into account (and furthermore, weight them by the possibilities of occurrence) to make a better decision.

The power of our approach comes from its ability to capture hidden subtle regularities. Although the program knows nothing about a language in the beginning, it starts to absorb information and refine its understanding about the language after more and more perturbed scripts are seen.

3 Deciphering

While Section 2 illustrates the basic idea of applying the noisy-channel model and EM to handle the order-discovery problem, in this section we address practical issues we have faced while implementing the deciphering program.

3.1 Assumptions

In order to simplify the computation, we make two assumptions. The first assumption is that the Luwians always write the next symbol in the neighborhood of the current symbol. The definition of neighborhood symbols (say, X and Y) is twofold: the straight line between the center of X and Y cannot touch any other symbol, and X and Y have to be in the same or adjacent columns (e.g., c1, c2, and c3 in Figure 5 are columns). Under this assumption we cannot jump across a symbol to connect to the next symbol. This assumption is plausible since it is the case for most writing systems[2]. With this assumption, complexity can be reduced from O(n!) to $O(b^n)$, where b is the average branching factor (average number of neighbors for Luwian Hieroglyphics per symbol), since for each symbol there are on average b candidates to be its successor.

[2] There are only very few writing systems that violate this assumption, for example, the "honorific inversion" for Egyptian writing.

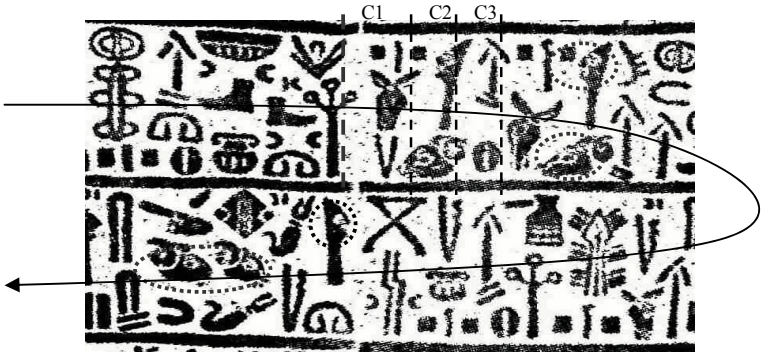


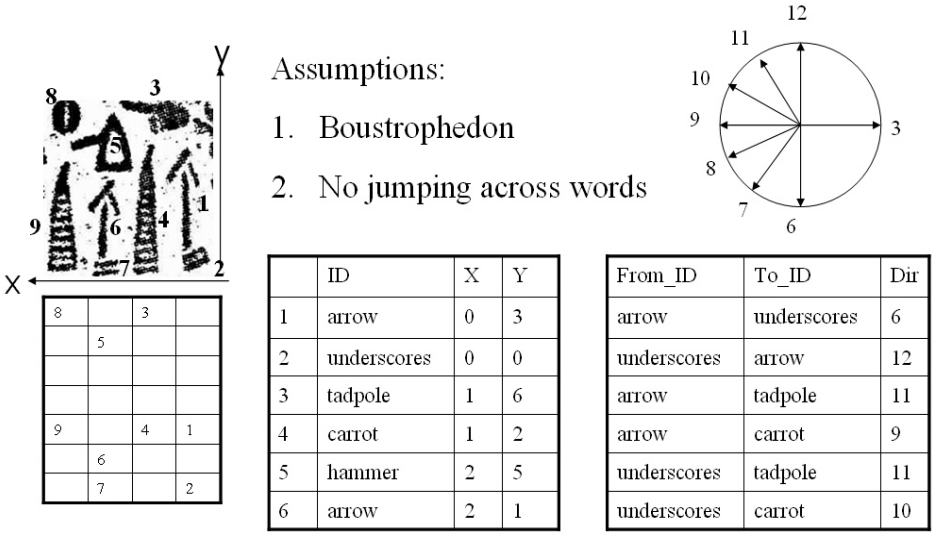
Figure 5: The symbols with faces in adjacent rows always face oppositely

Furthermore, linguists agree that the big picture (external appearance) follows the so-called "boustrophedon", which means that alternate rows progress in opposite directions, as in plowing a field, shown by the dotted line in Figure 5. This hypothesis is convincing, since the symbols with human or animal shapes tend to face different directions in subsequent rows, as circled in Figure 5. This assumption implies that we cannot move backward horizontally within one row. In the boustrophedon writing the first row can be either left-to-right or right-to-left. We choose "right-to-left" in the experiment because in the data we used there seems to be a clear starting point in the upper-right corner of the script.

With these two assumptions, one can further reduce the total number of plausible orders to $O(2^k)$ where k is the total number of columns. Since the external appearance is known, our program has to determine only the precise progress direction locally. The first assumption implies that there are only two choices (top-down or bottom-up) for each column, and thus the plausible number of orders is $O(2^k)$. It is still exponential, but as will be shown in Section 3.4, these two assumptions allow us to apply the forward-backward algorithm for fractional counting, further reducing the overall complexity down to polynomial.

3.2 Encoding

Encoding of the Luwian text reproductions into computer friendly format was done manually. There are 9 rows (separated by rigid horizontal lines) and 753 symbols total (73



8		3	
	5		
9		4	1
	6		
	7		2

	ID	X	Y
1	arrow	0	3
2	underscores	0	0
3	tadpole	1	6
4	carrot	1	2
5	hammer	2	5
6	arrow	2	1

From_ID	To_ID	Dir
arrow	underscores	6
underscores	arrow	12
arrow	tadpole	11
arrow	carrot	9
underscores	tadpole	11
underscores	carrot	10

Figure 6 Digitizing the Luwian script

distinct) in the page we tested on. As will be shown in the evaluation section, 753 symbols are essentially enough for determining the order for many languages. For each symbol, we first recognize it (i.e., assign it a symbol identifier such as “arrow” or “carrot”) and then record its two-dimensional position (as shown in the middle Table in Figure 6).

Once the positions of the symbols are determined, we can generate the relative direction from one symbol to the other automatically (we use the clock direction as shown in the upper right part of Figure 6). The true input to our discovery program is the table with recognized symbol identifiers and their relative directions to neighboring symbols, as demonstrated in Figure 7. The arrows show the plausible directions to proceed. For example, there is a two-way channel between symbols S1(arrow) and S2(underscores). Note that there is no link in between a middle node and a node in the other column (e.g., no connection between S4 and S1) since such progression will inevitably violate the assumptions we make. Also, there is no cycle involving nodes from different columns, or one of our assumptions will again be violated.

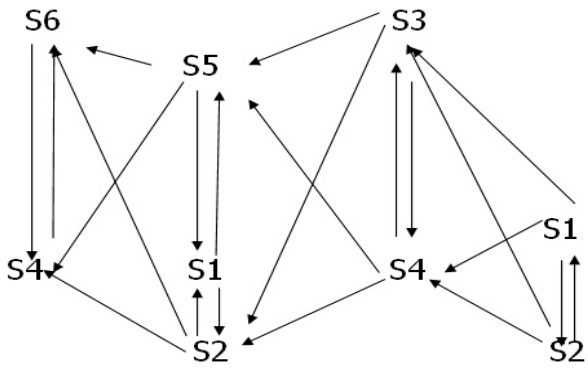


Figure 7 The graph representation of the of the data in Figure 6

3.3 Modeling the Weight of the Links

Once the information is recorded as in Figure 6, the next step is to generate the associated weight automatically for each link such that the sum of the weights in a path reflects the possibility of occurrence for that path. We have two models for the weights from one symbol W_n to its neighbor W_{n+1}, borrowed and modified from what we have described in Section 2:

The first ("model 1") is to model the weight as the bigram Luwian probability $P(W_{n+1}|W_n)$, which is exactly the same as we used in section 2.2. The limitation is that we ignore the noisy channel (i.e., how the Luwians generated the two-dimensional script) and rely completely on how frequently a sequence occurs in the literature.

To compensate for this limitation, in the second model ("model 2") we choose to use $P(W_{n+1}|W_n)*P(D_{n,n+1})$ to represent the weight. $P(D_{n,n+1})$ is the direction from the current symbol to the next one. The idea behind this model is that we assume the ancient Luwian people preferred progressing in some directions more than others, and that this preference is independent of the symbols themselves. From the decipherment point of view, this model uses two types of independent information to compensate for the weakness of each. That is, if several symbols are equally probable as successors, then the model will choose the one in the preferred direction; on the other hand, if several directions are equally probable for progressing, the model will pick the symbol that is most plausible to follow the current one. For both

models, we have to apply the EM algorithm to learn the associated parameters. For initialization, we tried uniform distribution for all weights as well as random assignments. The results show that EM converges to very similar results for various initializations we have tried.

3.4 Complexity

As illustrated in Section 3.1, even under our two assumptions, the complexity of this deciphering task is still exponential. This is because in the EM phase, we need to generate the probabilities for all 2^k different orders before performing the fractional counting. The script used for the experiment has on average 30 columns per row, and unfortunately, 2^{30} is computationally intractable.

However, we can benefit from our assumptions and do much more than ease the complexity from a higher order exponential $O(b^n)$ to a lower level exponential $O(2^k)$. We have found that with these two assumptions, it is possible to apply the forward-backward algorithm [1] for fractional counting, without having to list all plausible orders, which reduces the complexity to polynomial.

The forward-backward algorithm is a dynamic programming algorithm. We propose to use it to store the alpha and beta values for each link in the graph. The alpha value is the aggregated probability from the starting point to the source of the link while the beta value is the aggregated probability from the destination of this link to the ending point. In other words, the alpha value of a link is the sum of all the path probabilities that reach this link from the starting point, and the beta value is the sum of the probabilities of all paths leaving from the link and terminating at the ending point. The fractional count of a link is equal to its own weight times its alpha value times its beta value divided by alpha[end], which is the alpha value of the last node[3].

[3] Note that since there is one long sequence in our computation, the alpha[end] value for every fractional count is the same. Since the goal of fractional count is to calculate the relative occurrence frequency of the links, it is not necessary for us to truly compute alpha[end] during the computation.

There are two types of links in the graph. One is the link that progresses horizontally (e.g., link S3→S5 in Figure 8), the other is the link that progress vertically (e.g,. link S5→S1). For the link traverse horizontally, its alpha value inherits only from the previous vertical link that is in the same column (e.g., Equation 2a shows the alpha value for link S3→S5). Although there are three incoming links to node S3 (i.e, S1→S3, S2→S3, S4→S3), the other two do not contribute to the alpha value of link S3→S5, since coming from S1 or S2 to S3, the next node we have to visit is S4 instead of S5, under our second assumption. In other words, it is illegal to progress from S3 to S5 unless all the other nodes (i.e., S4) in the same column are visited, so the alpha value of S3→S4 can be based on just the alpha value of S4→S3. Similarly the beta value of S3→S5 can be generated from the beta value of S5→S1 link, but not the S5→S6 or S5→S4 link, because the second assumption tells us that after moving from S3 to S5, the next node to visit has to be S1 rather than S4 or S6 (see Equation 2b). Equations 2a and 2b tells us that for horizontal links, the alpha value and beta value are independent of each other, thus the dynamic programming condition holds. Likewise, for the links that progress vertically, such as S5→S1, the alpha value is the weighted sum of all the incoming alpha values from the previous column (Equation 2c). The beta value of S5→S1 can be produced based on the beta value of S1→S2 (Equation 2d). Equation 2c and 2d shows that for a vertical link, its choice of previous route (i.e., alpha value) cannot affect its choice of future route (i.e., beta value), thus forward-backward algorithm can be applied.

$$\text{Alpha}(S3 \rightarrow S5) = \text{Alpha}(S4 \rightarrow S3) * \Pr(S3|S4) \quad \text{…Equation 2a}$$

$$\text{Beta}(S3 \rightarrow S5) = \text{Beta}(S5 \rightarrow S1) * \Pr(S1|S5) \quad \text{…Equation 2b}$$

$$\text{Alpha}(S5 \rightarrow S1) = \text{Alpha}(S3 \rightarrow S5) * P(S5|S3) + \text{Alpha}(S4 \rightarrow S5) * P(S5|S4) \quad \text{…Equation 2c}$$

$$\text{Beta}(S5 \rightarrow S1) = \text{Beta}(S1 \rightarrow S2) * P(S2|S1) \quad \text{…Equation 2d}$$

Equation 2a-2d

The above analysis shows that given our two assumptions, both alpha and beta values can be propagated in a first-order Markov manner without interfering with each other. Once the alpha-beta values are computed and stored, we can calculate the amount each specific link contributes to its associated bigram parameter be multiplying its transition probability with both the alpha and beta values. The ultimate bigram probability is the accumulation of all the identical bigram link probabilities. Similarly, the fractional count for the $P(D_{n,n+1})$ table can be computed by aggregating the $P(D_{n,n+1})$ times alpha and beta.

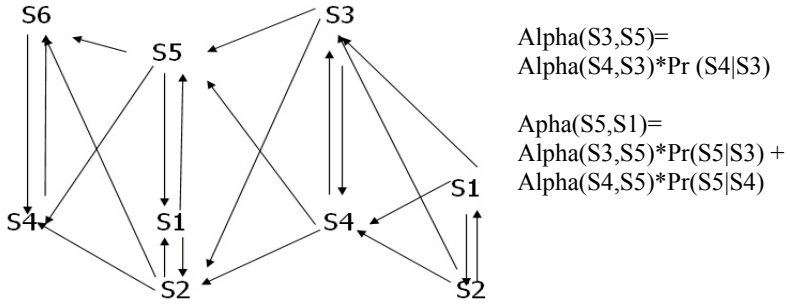


Figure 8 How the alpha value propagates in the graph

3.5 Decoding

As shown in the previous section, an alpha value depends only on its neighbor's alpha values under our assumptions. Therefore, after the EM converges, we can apply dynamic programming again to extract the path of the highest probability. The output order generated by model 2 on our experimental page is shown in Figure 9. In the decoding phase, we modified the dynamic programming algorithm a bit by recording not only the best probability in each step, but also the second-best probability. Therefore, once the dynamic programming algorithm finishes walking through the nodes in the graph, we not only have the order with the highest probability, but also the second-best one, which we will need for further evaluation.

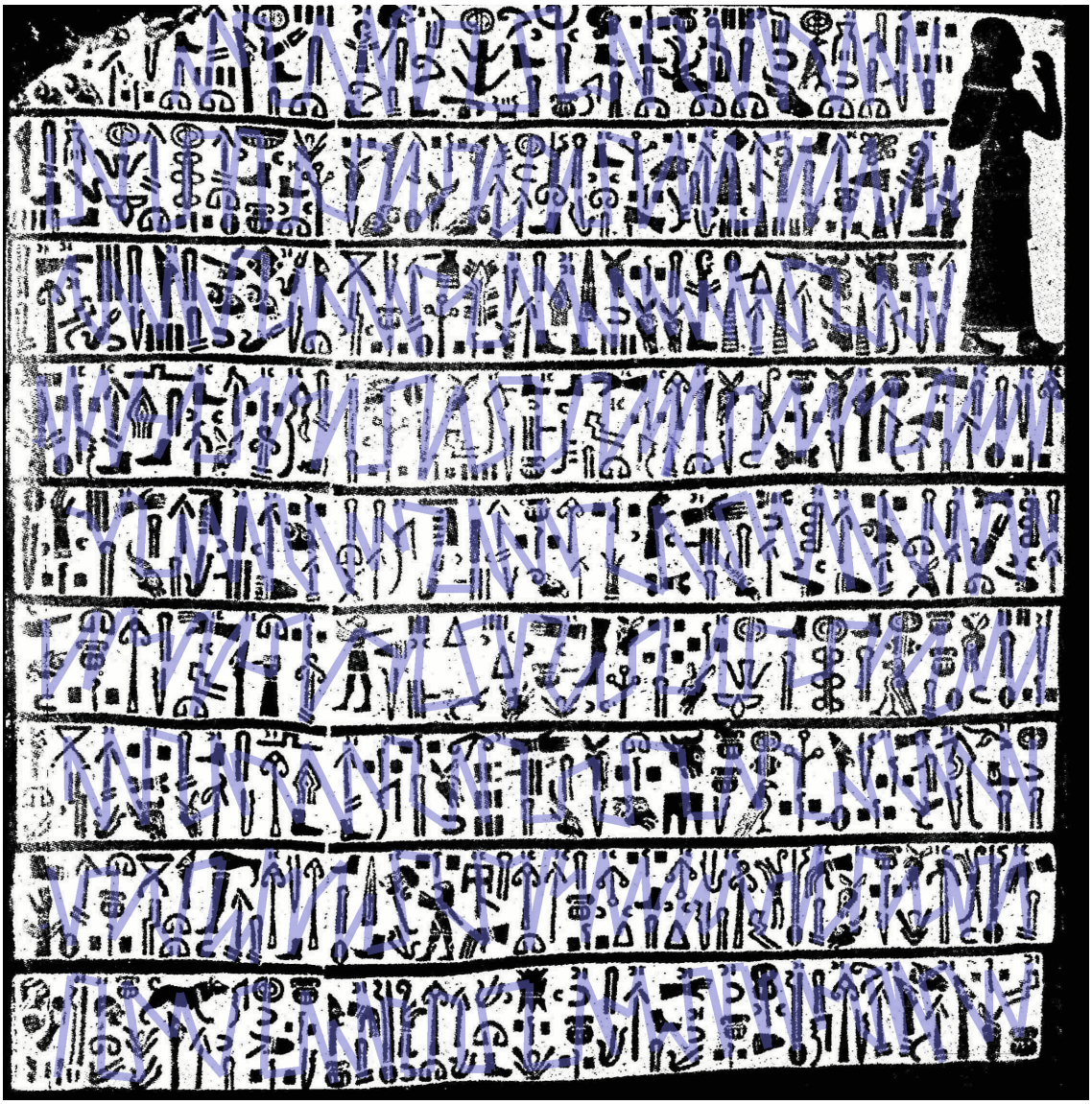


Figure 9 The writing order discovered by our second model (9 rows, 753 symbols)

4 Evaluations

Evaluation is always the trickiest part for any machine discovery problem. The chicken-and-egg paradox arises in the sense that to evaluate the results, we need to know the precise order; on the other hand it is because that we were not aware of the precise order therefore it is necessary to pursue such discovery. In this section we will describe several indirect strategies to evaluate our results.

4.1 Checking the Machine's Confidence

One method of verification is to check how much confidence the machine has toward the order it picks. We propose to measure confidence by comparing the probability between the best and the second-best order returned by the machine. In view of the fact that the machine makes its decision based on probability, closeness between the best and second-best probabilities implies that they might be virtually the same from the machine's point of view. This situation could indicate lack of data or under-fitting of the model. Table 1 records the best and second-best negative log-probability for each row in each model.

From this table one can learn that in the first model (bigram only) only 2/9 rows have distinguishable probabilities while in model 2 (bigram plus independent direction) all the rows have distinguishable probability. This implies that with only bigram information, the system does not feel comfortable with its choice. It gains more confidence when the direction factor is considered. Although this evidence does not necessarily show that the second model can generate accurate results, it does indicate that it provides sufficient information for the machine to make a confident decision.

Models / rows	Model 1		Model 2	
	best	2nd	best	2nd
row1	149	150	**224**	**230**
row2	193	194	**306**	**315**
row3	165	165	**277**	**282**
row4	193	193	**309**	**318**
row5	160	161	**250**	**257**
row6	**150**	**247**	**240**	**251**
row7	**157**	**260**	**253**	**257**
row8	178	179	**288**	**293**
row9	189	190	**284**	**292**

Table 1 The best and second-best log probabilities (bold pairs are distinguishable ones)

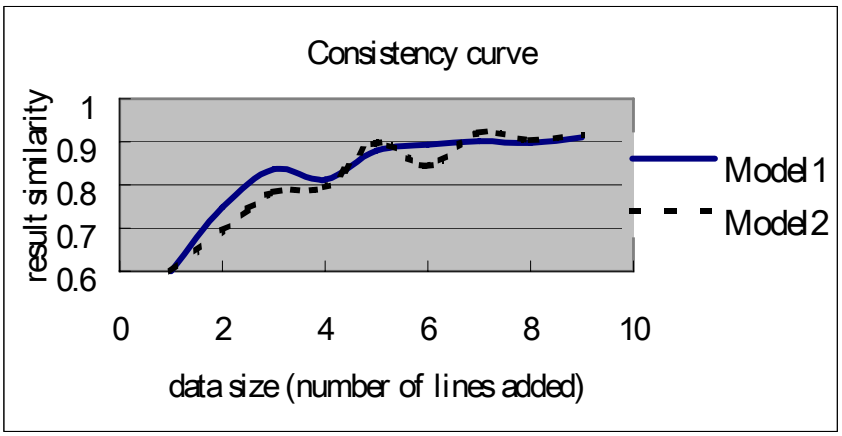


Figure 10 How the system changes while the data increases

4.2 Checking the consistency

One typical way to evaluate a natural language learning program is to check its learning curve over data, i.e., to examine how much the accuracy can be improved when more data is added to training. In our problem, generating the learning curve over data is not possible because the gold standard answer is not available. However, we can draw a similar graph to see how the results change as new data are introduced. Figure 10 shows how much the result alters as data are added line by line. For both models, the similarity of the outputs remains steady and reaches 90% after the 7^{th} line is added, which implies a stabilization of the system.

4.3 Evaluating the Methodology

Instead of verifying the results directly, in this section we propose to verify the methodology by checking whether it works for known languages and writing systems. The basic idea is to first have someone collect a set of meaningful sentences for various known languages, secretly overlay them one by one on top of symbols in the two-dimensional Luwian scripts (without violating the two assumptions), then execute our program again to see if it can recover the original linear order. This time we do know the true order for verification.

We use 5 natural languages (Chinese, English, Arabic, Latin, Spanish) and one programming language (Java) for testing. We collected a paragraph of writing for each language.

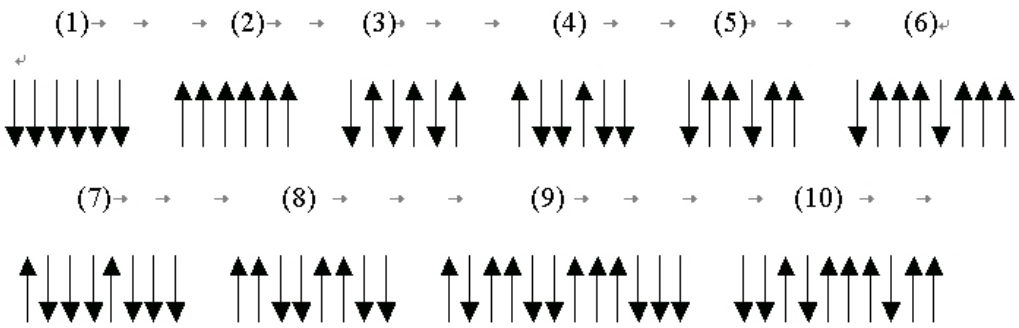


Figure 11 Ten different patterns to generate two-dimensional scripts

We need only 753 letters (753 characters in Chinese) since the tested Luwian script has a total of 753 symbols. To generate two-dimensional scripts, we apply ten different patterns (Figure 11). Note that the tenth pattern is a random one that does not follow any rules. Then we replace the symbols on the Luwian page by the letters in the paragraph, as exemplified in Figure 12.

These newly generated two-dimensional scripts (there are totally 60 different ones since we have 6 different languages and each has 10 different ways of encoding) can then serve as the input to both models to generate the linear results. We evaluate the results by counting how many of the columns returned by our system have the correct order. The results are shown in Table 2. Note that the baseline (randomly guess a direction at each decision point) reaches 50% for all the patterns.

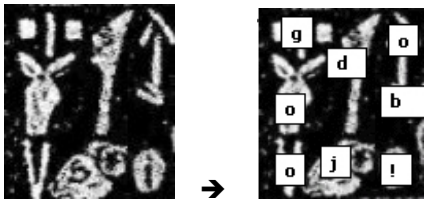


Figure 12 Replacing Luwian symbols with English letters according to pattern 1

Table 2 shows that both models achieve better performance than the baseline, for all writing patterns. For each case, it takes less than 1 minute for our program (running on 1GHz PC) to converge to a stable result. If the script was written according to pattern 1 or 2 (both have

strong tendency for directions), the second model can discover the order almost perfectly except for Chinese. We believe it is not perfect in Chinese because unlike others, Chinese script is logographic, and the number of distinct symbols in logographic writing is much higher than in alphabetic or syllabic systems. The left part of Table 3 describes the $P(D_{n,n+1})$ learned by model 2 for pattern 1. This shows that the program does learn the fact that the system prefers progressing in the 6 o'clock direction much more than 12 o'clock direction. Also, it learns another distinguishable factor for pattern 1, which is that the 10 or 11 o'clock direction is preferable to 7 or 8 o'clock. The right column in Table 3 indicates that for pattern 3, the model learned that pr(6) is almost the same as pr(12), while pr(9) has a much higher probability than any other horizontal directions, which matches the intrinsic characteristic of that pattern. As to the other patterns that have less-strong preference for the directions, model 2 still reaches 80% accuracy for most of the languages. Table 2 also points out that for model 1 the accuracy ranges from 70% to 85% (except Chinese), depending on the language, but not on the writing pattern. This is reasonable, since only the language model is used there, and we did not model the noisy channel. It also tells us that even if the script was written randomly as pattern 10, our approach could still reproduce the linear order with accuracy higher than 75%.

Next we show some linear English (written in pattern 3) recovered by model 2: *"While Section 2 iullartsets the basic ideas of combining the nosihc-y annel model and EM to handle the order-discovery problem, in this section we uowl dleki to dadress mose practical issues we have faced while trying to implement thed icephering program."* as well as the linear English (written in pattern 10) recovered by our model 1, which is the most difficult challenge for our program as well as for human beings: *"While Section 2 illuartsets the basic edias oc fombining thon esihc-yennal moled and EM to handle the odrer-dicsove yrorplb, mein this section we uowl dlike to address some practical isusse we have faced while yrting ti ompleme tnthed icerehping program."* The recovered scripts are not perfect but understandable.

	model \ pattern	1	2	3	4	5	6	7	8	9	10	Avg
753 letters in Latin	1	83%	87%	85%	84%	84%	85%	86%	84%	85%	85%	85%
	2	**100%**	**100%**	**91%**	**80%**	**87%**	**84%**	**89%**	**88%**	**79%**	**77%**	**88%**
753 letters in English	1	76%	79%	76%	78%	77%	78%	79%	76%	77%	78%	77%
	2	**98%**	**98%**	**89%**	**77%**	**82%**	**79%**	**85%**	**87%**	**74%**	**72%**	**84%**
753 letter in Arabic	1	71%	76%	73%	74%	73%	75%	76%	72%	72%	75%	74%
	2	**95%**	**97%**	**84%**	**72%**	**78%**	**78%**	**79%**	**77%**	**68%**	**66%**	**79%**
753 letters in Spanish	1	75%	77%	76%	76%	76%	76%	77%	75%	76%	76%	76%
	2	**99%**	**99%**	**88%**	**72%**	**75%**	**77%**	**80%**	**82%**	**75%**	**73%**	**82%**
753 characters in Chinese	1	60%	72%	65%	61%	67%	69%	66%	66%	66%	67%	70%
	2	**87%**	**82%**	**76%**	**67%**	**68%**	**72%**	**71%**	**74%**	**67%**	**67%**	**73%**
753 letters in Java	1	82%	87%	84%	85%	85%	86%	87%	84%	85%	86%	85%
	2	**100%**	**100%**	**90%**	**84%**	**83%**	**87%**	**88%**	**86%**	**83%**	**84%**	**89%**
Luwian Scripts	1	60%	49%	52%	55%	56%	52%	52%	56%	56%	50%	54%
	2	**80%**	**28%**	**54%**	**57%**	**65%**	**44%**	**45%**	**63%**	**53%**	**50%**	**54%**
Baseline	random guess	50%	50%	50%	50%	50%	50%	50%	50%	50%	50%	50%

Table 2 The accuracy of 10 different writing patterns in model one and two

We perform a similar human study by giving human subjects[4] two-dimensional scripts written in some language they do not know according to the above patterns, and ask them to figure out the linear order of each. The results show that among ten samples we sent out, ninety percent of subjects perform better than 50%. The average accuracy is 58% while the highest is 73%, which is far below our program's 79% average and 97% highest score. The result demonstrates that although people can learn certain regularity from the data and perform slightly better than the baseline, to get better results in this task, the use of EM algorithm together with the powerful computational capability of a machine can be very helpful.

[4] The Subjects' background including computer science, computational linguistics, psychology, and signal processing.

The bottom of Table 2 demonstrates the accuracy of our Luwian result assuming that it was written according to each of the ten previous patterns[5]. For example, if it was written according to pattern 2, then the results our program generates would reach only 28% accuracy (since pattern 2 is written in bottom-up manner, while our discovery shows that most of the lines are top-down). It is clear that pattern 1 possesses a much higher accuracy (80% top-down by model 2) than all the others. This shows that if we trust the validity of our program, then we can conclude that most of the Luwian scripts are written top-down.

Pattern 1	Pattern 3
pr(6)=0.620	pr(6)= 0.312
pr(7)=0.001	pr(7)= 0.020
pr(8)=0.005	pr(8)=0.031
pr(9)=0.030	pr(9)=0.259
pr(10)=0.123	pr(10)=0.048
pr(11)=0.224	pr(11)=0.024
pr(12)=0.001	pr(12)=0.308

Table3 $P(D_{n,n+1})$ learned by model 2 in Latin.

After manually checking the results shown in Figure 9, we found that most of our bottom-up writings come either from the blurred area or because the boundaries of columns are unclear. Our theory is that the Luwians might in general follow a top-down manner during writing. However, every Luwian symbol has a different shape and size, so there are situations now and then (e.g., a desire to make better use of the space) preventing the writer from following the general rule. Dr. Hawkins addressed similar issues by saying "signs arranged in one or more vertical columns from top to bottom….But the signs come in a variety of shapes, long and slender, wide and flat etc….. a problem for us is that it is by no means always clear

[5] Note that the sum of the probabilities for opposite patterns (e.g, pattern 1 and pattern 2) is sometimes more than 100%. It is because that there are a few columns with only single symbol, and for these columns both top-down and bottom-up direction are regarded as correct.

in what order the signs are to be read" [3]. The major contribution of this paper is not only to strengthen the hypothesis that the general writing pattern is top-down, but also that our program can provide a plausible solution for the order in the ambiguous areas. It finds the regularities as well as the exceptions (areas where the writing pattern was abandoned), while the latter is a very hard task for human beings. Table 4 shows the learned parameters for $P(D_{n,n+1})$ and certain bigram probabilities in Luwian Hieroglyphic.

pr(6)=0.498	p(small | triangle)=0.25
pr(7)=0.009	p(four | triangle)=0.249
pr(8)=0.004	p(buffalo | triangle)=0.249
pr(9)=0.146	p(swan | triangle)=0.250
pr(10)=0.086	p(crocodile | three)=0.250
pr(11)=0.138	p(deer | three)=0.249
pr(12)=0.122	p(three | three)=0.499

Table 4 The P(Dir) and bigram probabilities for the Luwian

5 Conclusion

In this paper we describe an unsupervised method for discovering the linear order of the two-dimensional Hieroglyphic Luwian script. We represent the problem as Shannon's noisy-channel model and apply EM to learn the two different parameter sets within the models. Finally, we apply dynamic programming algorithm to extract the most probable order.

We propose several ways to evaluate this discovery problem: We examine whether the machine has confidence about the results it produced by comparing the best and second-best results. The results imply that model 2 might be a better fit. We also generate a consistency curve to examine the sufficiency of the data. Finally, we evaluate our methodology by applying it to various known languages. The results show that our program, having neither any pre-

requisite knowledge about the language nor how the two-dimensional script was generated, can still recover the linear script with more than 80% accuracy. As to the Luwian scripts, our experiment shows that in general it was written top-down, but it also identifies certain exceptional areas.

The lesson we have learned is that with the modern computational power and a proper model, machines are capable of grasping the hidden regularities and dealing with tasks that might be extremely difficult for human beings.

The contributions of our study are not only about finding the writing order of an unknown ancient script, but also to demonstrate how a general, unsupervised approach can be designed to deal with a situation where very little background knowledge is available for learning. Furthermore we propose several indirect but general strategies to evaluate such a discovery system. Since no prior knowledge is necessary in our approach, it has potential to be applied not only to natural language problems but also to tasks related to order discovery in areas such as biology, chemistry, and others.

6 References

1. **L.E. Baum, An Inequality and Associated Maximization in Statistical Estimation for Probabilistic Functions of Markov Processes, Inequalities. 627(3) (1972) 1-8.**
2. **A.D. Dempster, N.M. Laird, and D.B. Rubin, Maximum likelihood for incomplete data via the EM algorithm, Journal of Royal Statistical Society Series B. 39 (1977) 1-38.**
3. **J.D. Hawkins, Corpus of Hieroglyphic Luwian Inscription. Vol. I. 1999: Walter de Gruyter.**
4. **J.D. Hawkins, The Luwians: Scripts and Texts. The Luwians, ed. C. Melchert. 2003: Leiden.**
5. **K. Knight and K. Yamada, A Computational Approach to Deciphering Unknown Scripts. in: Proceedings of the ACL Workshop on Unsupervised Learning in Natural Language Processing, 1999.**
6. **C. Shannon, A mathematical theory of communication, Bell System Technical Journal. 27(3) (1948) 379-423.**
7. **R. Sproat, A Computational Theory of Writing Systems. 2000: Cambridge: Cambridge University Press.**