Article

Statistical models for identifying missing and unclear signs of the Indus script

Varun Venkatesh[1], Ali Farghaly[2]
[1] Dublin High School, Dublin, California
[2] Visiting professor of NLP at the Faculty of Computers and Information at Cairo University, Egypt

SUMMARY

A writing system was developed between 2500 and 1800 BCE in the Indus Valley civilization in the Indian subcontinent and remains undeciphered. Indus script texts found so far in the archeological digs from this civilization are limited in number and include a lot of damaged artifacts with missing and unclear signs. Identifying the missing and unclear signs and extending the Indus text corpus will aid the researchers in deciphering this script. This work aimed to predict the missing and unclear signs using n-gram Markov chain models using the Interactive Corpus of Indus Texts (ICIT) text corpus. First, we analyzed patterns and concordances of the signs, pairs, triplets, and other n-grams and discovered the positional behavior of signs in the Indus texts. With that understanding, we built Markov chain language models based on n-grams, augmented with sign positional probabilities. Since signs could be missing in any location of the texts, we devised and implemented effective sign fill-in algorithms on top of these Markov chain models. Using the language models and the sign fill-in algorithms, we tuned our models and predicted single signs that were deliberately removed from complete texts with about 63% accuracy. Then we used the best model and our tuned parameters to predict missing and unclear single signs in about 100 texts. The statistical methods we described here improve our understanding of the Indus script. Filling in the missing signs makes the corpus more complete and helps contribute to the broader decipherment effort.

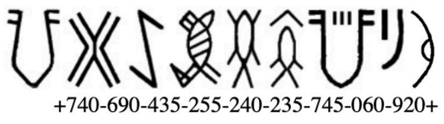
INTRODUCTION

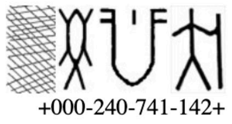
The Indus Valley civilization thrived from about 3300 BCE, northwest of the Indian subcontinent, and is one of the world's oldest civilizations (1). The Indus Valley people developed a writing system between 2500 and 1800 BCE, which flourished during its mature period. After this period, this civilization died out until archeologists excavated Indus archeological sites in the early 1900s. Archaeologists have unearthed about 10% of these sites. They have found many seals, tablets, and pots with writings in the Indus script.

The Indus script bears little resemblance to any of the ancient Indian or Middle Eastern scripts, and researchers have not deciphered it yet. Several researchers have claimed that they have partially or fully deciphered this script, but the scientific community has not accepted these claims (2). The challenge for the decipherment is that there are no inscriptions comparable to the Rosetta stone with a multilingual script that we have found. Additionally, the lengths of the Indus texts are small, making decipherment efforts complex. Furthermore, there needs to be more consensus over what language the script encodes (2,3). Some have even claimed it is not a writing system (4). Most researchers agree, however, that the writing encodes a language that could be one based on a Proto-Dravidian, Indo-Aryan, or Proto-Munda (2,3). There is consensus that the script has too many signs to be alphabetic or pure-syllabic and is likely to be a logo-syllabic script, with logograms representing a concept or a word, along with several syllables (5).

Some Indus artifacts have survived well, while elements and age have damaged many of these Indus artifacts with textual writings (**Figure 1A**). We find that the texts in some of the artifacts are in fragmentary condition, chipped, and broken (1). The Indus text corpus has several texts with no missing signs or some with one or more missing, unclear, and doubtfully restored signs (**Figure 1B-C**). Catalogers have used expert opinion to guess the missing signs (1,3). Making this guess is not a perfect science; therefore, they mark the missing or unclear signs missing or doubtfully restored in the corpus. Manual prediction of signs is complex, and researchers have felt the need for automation based on the statistical distribution of the text and mathematical models to predict these missing and unclear signs. One such mathematical model is the n-gram Markov chain language model.

The n-gram Markov chain language model, also called the Markov chain model or Hidden Markov Model, in this research work helps understand the Indus texts better from a statistical basis and to use it to predict these missing signs. For a given sequence of text in this context, an n-gram is

a)


b)


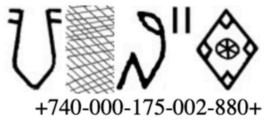
c)


Figure 1: Indus Texts. a) Text from Mohenjo-Daro seal M-671 with no missing signs (left), b) Text from Harappa- H-610 (middle), and c) Text from Lothal- L-95 (right). Texts here are Right to Left, with missing Indus signs represented by a hashed block. ICIT codes run below the text.

a contiguous sequence of ‘n’ signs. A probabilistic statistical model built using n-grams that uses contiguous sequences of signs, their frequencies, and their position in the texts is an n-gram Markov language model (6).

The conditional probability of sign ‘A,’ given signs ‘B,’ ‘C,’ and ‘D’ in that order, is:

$$P(A|\ B\ C\ D) = P(A)\ P(B|C)\ P(C|D) \qquad \textbf{Eq. 1}$$

The Markov model assumes that the probability of a sign can be derived just from the conditional probability of a few of the sign(s) preceding it. Applying the Markov assumption for an n-gram and generalizing it, we need the conditional probabilities of just the n sequence of signs before that.

$$P(w_n|\ w_{1:n-1} \sim P(w_n|w_{n-1}) \qquad \textbf{Eq. 2}$$

w_n is the n_{th} sign. The expression $w_{1:n-1}$ indicates the signs $w_1, w_2, \ldots, w_{n-1}$.

For example, using a bigram model (n = 2, where n is the number of consecutive signs), and where 690, 435, 235, 240 are signs, the conditional probability of sign 690 is : P(690| 435 235 240) = P(690|435)

The same for a trigram model (n = 3), the conditional probability of sign 690 is: P(690| 435 235 240) = P(690|435 235)

We can use the language models to estimate the conditional probability of the missing signs and to predict the next sign given a set of signs.

Some interesting studies have performed statistical analysis of the Indus text and built n-gram Markov models of the Indus texts that have contributed to a greater understanding of the texts’ linguistic nature (7–11). Some researchers have explored optical character recognition and visual recognition algorithms for identifying and understanding the script, which they could use to predict the missing Indus texts (12), as performed for ancient Greek (13). Some researchers have done excellent work in identifying missing and unclear text using the M77 Indus Corpus (11). Some of these studies were done using Markov chain models and have aided researchers in predicting some missing signs (14).

We hypothesized that we could identify missing and unclear signs in the Indus corpus using a more recent and complete ICIT corpus and advanced Markov chain n-gram models, with a newly developed positional probability model and sign-filling algorithms (15). Since the problem with multiple missing signs in a text is much more complex, we focused on restoring single missing signs in the Indus texts. We did statistical analyses of the Indus texts focused on text length, sign clusters, and sign positional distribution and built a positional probability model. We then built various Markov chain language models with different smoothing and interpolation techniques for various n-grams (6). We built the Sign Fill-In Start and the Sign-Fill Full algorithms on top of the various language models. We then passed the training and test datasets through the algorithms and calculated extrinsic model performance using Hit@N, which is the % accuracy of predicting the correct sign among N predicted signs. We used Hit@1, Hit@5, and Hit@10 measures. Others have used similar measures to evaluate extrinsic model performance for missing Babylonian texts to fill in missing signs (16). The Lidstone model with Sign Fill-In Start and n-gram order 4 gave the best results among the various models we tried. We could predict a missing sign at about a 63% rate among the ten predicted signs. We then used that to predict the signs for about 100 texts where single signs were missing and published the predicted signs. Filling these missing and unclear signs makes the Indus text corpus more complete and will help in future decipherment efforts.

RESULTS

The ICIT text corpus has about 4500+ texts, and after data-cleaning to remove unwanted texts, we ended with 2223 unique texts to build our model. We computed the average length of our cleaned-up ICIT corpus and determined it to be between four and five signs. We rarely found the contiguous text of more than ten signs (**Figure 2A**).

Sign Frequencies

Understanding how signs behave in the texts is essential

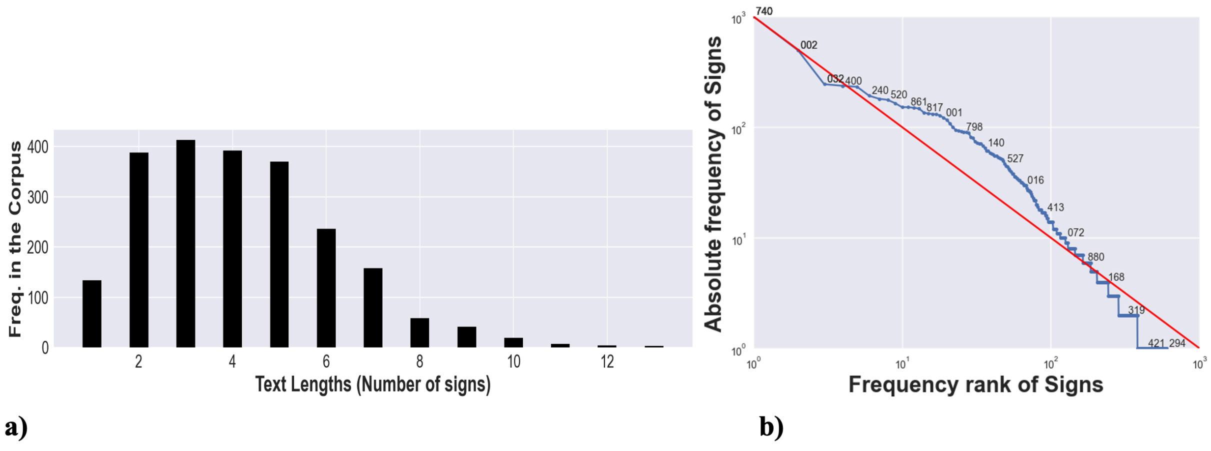
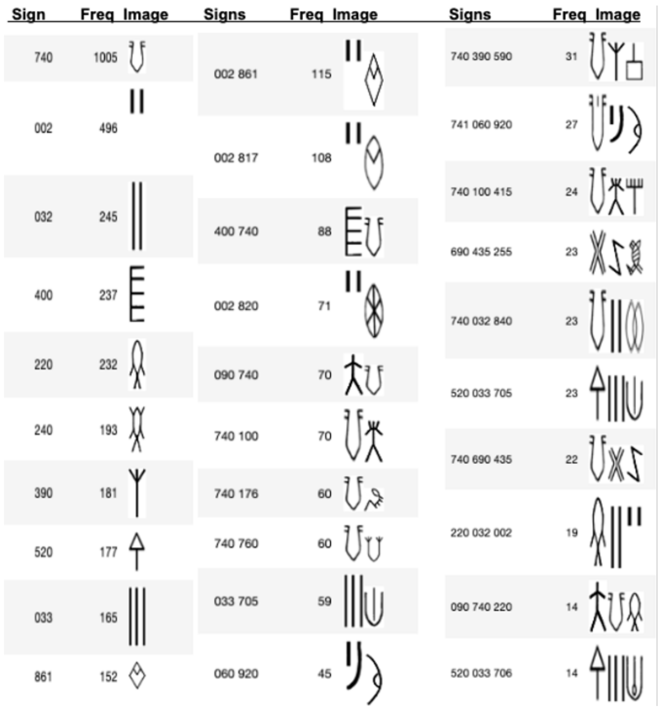


Figure 2: Statistical Analysis of Indus Texts. a) Text Length Distribution of the Indus texts. The length of individual Indus texts (number of signs) in the x-axis is plotted against the count of the number of texts of that length found in the cleaned up ICIT corpus b) Zipf-Mandelbrot Plot for Indus signs (individual characters). $\log_{10}$ of frequency ranks of each sign in the x-axis is plotted against the $\log_{10}$ of absolute frequencies in the y-axis in the cleaned-up ICIT Indus corpus. The straight line (red) indicates ideal Zipf-Madelbrot behavior.

for statistical analysis. So, In the cleaned-up corpus, we calculated the sign frequencies for each sign across the corpus. As expected for a typical language script, we found that a few signs appeared with high frequencies, and there was a long tail of signs with lesser frequencies in the corpus (**Figure 2B**). When plotted in a log-log scale, the frequency ranks of each sign and the absolute frequencies in the cleaned-up ICIT corpus closely resembled the plot for the Zipf-Mandelbrot law. Almost all natural languages follow this law, and the Indus texts following it indicate that the texts represent a language. The curve nearly fits the formula:

$$\log(f)_r = a - b^* \log(r+c) \qquad \textbf{Eq. 3}$$

where f_r is the frequency of sign with rank r when ranked by frequency and a, b, c are constants.



Sign	Freq	Image	Signs	Freq	Image	Signs	Freq	Image
740	1005		002 861	115		740 390 590	31	
002	496		002 817	108		741 060 920	27	
032	245		400 740	88		740 100 415	24	
400	237		002 820	71		690 435 255	23	
220	232		090 740	70		740 032 840	23	
240	193		740 100	70		520 033 705	23	
390	181		740 176	60		740 690 435	22	
520	177		740 760	60		220 032 002	19	
033	165		033 705	59		090 740 220	14	
861	152		060 920	45		520 033 706	14	

Figure 3: Top 10 unigrams, bigrams, and trigrams with high frequencies in the cleaned-up ICIT. Indus corpus (Right to Left n-gram). Sign/Signs represents the ICIT code(s) of the sign(s), Freq. is the frequency of the sign(s) in the ICIT corpus. Image is the visual representation of the sign(s).

This analysis concluded that following the Zipf-Mandelbrot law, the Indus script closely exhibits a language-like behavior. Thus, a language model would be apt to build using the ICIT text corpus.

The N-gram Analysis

Identifying frequencies of sign clusters reveal sign patterns that could be significant. To identify such patterns, we constructed a list of n-grams from the text by finding pairs and triplets of signs that occur next to each other (**Figure 3**). The analysis indicated a high frequency of some signs clustering with each other, and such patterns may show that this was not accidental and that these high-frequency sign clusters likely were used to convey a name, thing, idea, or a concept.

Sign Positional Analysis

We did a positional analysis of several high-frequency signs to get a good idea of whether some signs appeared in some positions of the texts more frequently or not. We found positional patterns for some of the signs but only for some. For example, Sign 740, the Jar sign, the most frequent sign in the corpus, tends to occur more at the end of texts and seldom at the beginning (**Figure 4A**). Sign 861, the house/courtyard sign, tends to occur more at the beginning and rarely at the end of the texts (**Figure 4B**). We analyzed high-frequency signs and found that while there were patterns for some signs, many tended to occur all over the texts. When we analyzed signs by each position, we found that signs 820, 861, 817, and 920 dominated the beginning of the texts (position 1). Signs 032 and 140 are often in the middle (**Figure 5**).

Language Models

With a better understanding that Indus scripts follow patterns and represent a language, we cleaned up and split the Indus texts into training and test sets. We used MLE (Maximum Likelihood Expectation-based language model), KneserNey Interpolated, Lidstone, Stupid Back-off, and Witten Bell Interpolated language models. We used the training set data to build n-gram sign tokens with n = 2 to 7 and train these models to identify the missing signs using the models (17). This was A simple replacement method of replacing the missing signs with every possible sign and selecting the resulting texts with the best model score yielded poor results. It was likely because the overall text model score did not reflect the model score of n-gram snippets formed by the missing text well. Therefore, we devised two algorithms

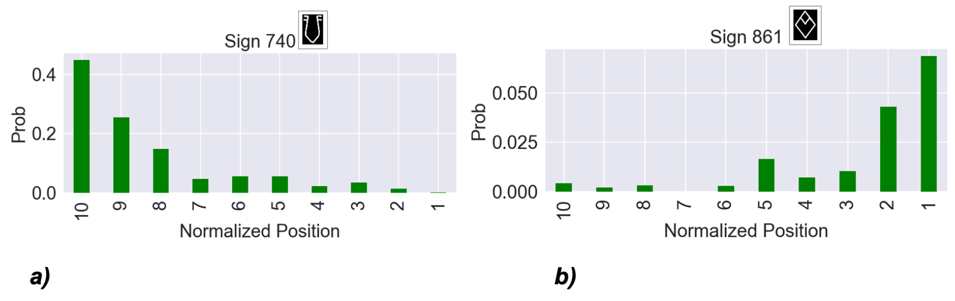


Figure 4: Positional Analysis of some signs. a) Positional Analysis of a sample, sign 740. Normalized position of a sign in text plotted against probability of finding the sign in that position in the corpus - sign 740 b) Positional Analysis of sign 861. Normalized position of a sign plotted against probability of finding that sign in that position in the corpus - sign 861 (Right to Left text).

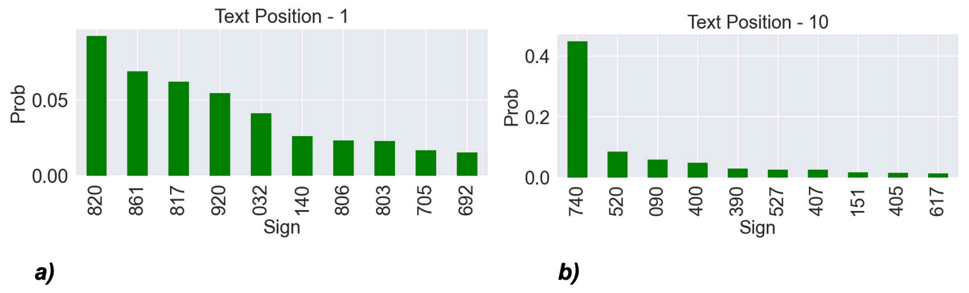


Figure 5: Positional probabilities by position. a) Top 10 ICIT signs with the highest probabilities of being found in normalized position 1. Their probabilities are in the y-axis b) Top 10 ICIT signs with the highest probabilities of being found in normalized position 10. Their probabilities are in the y-axis (Right to Left text).

to fill in the signs. The 'Sign Fill-in Start' algorithm finds signs that make the longest n-grams possible using conditional probabilities of signs before it. The 'Sign Fill-in Full' algorithm finds signs that make the longest n-grams possible with high conditional probability for that n-gram order from the list of n-grams in the corpus.

Missing signs n the 'Training dataset' and predictions

We used the training dataset and removed a sign from each Indus text to check our algorithms and language models' effectiveness. We ran this through all language models and compared the predicted sign with what we removed. We ran it 100 times with signs replaced at random locations each time and averaged the results. All the models did well, as we had trained the models using the data in the training sets. We got excellent hit rates, and the Sign Fill-in Full algorithm did better than the Sign Fill-in Start algorithm (**Table 1**). Increasing the number of texts kept the results the same.

Missing signs in the 'Test dataset' and predictions

We then took the texts from the test dataset, randomly converted a sign to an unclear sign ('000') in each text, and let the models predict the unclear sign. We repeated this 100 times with signs replaced randomly, and the results averaged. Overall, the Sign Fill-in Full algorithm did worse than Sign Fill-in Start for Hit@1 rate but very well at Hit@5 and Hit@10 (**Table 2**). Both these algorithms performed poorly for missing leftmost signs for Right to Left texts (R to L). Sign Fill-in Full using Lidstone language did the best and topped the Hit@5 and Hit@10 rates among all models. The Lidstone model has good smoothing to overcome the sparsity problem of training data by adding or adjusting the probability mass distribution of signs built into it, to account for probabilities of unknown signs and sign clusters (**Table 2**). For the test dataset to predict the missing medial signs, we picked Sign Fill-in Full with an n-gram order of 3 after trying various n values. It did the best with the Lidstone language model. Using the Lidstone language model, the Sign Fill-in Full with an n-gram order of 4 did the best for the rightmost missing signs. The Sign Fill-in Full did the best with the Lidstone language model for the leftmost missing signs, although hit rates were not very high for any n-gram order. Overall, we could predict signs at about

Model Name	MRR	Total Hit@1 (%)	Total Hit Category @1 (%)	Total Hit@5 (%)	Total Hit@10 (%)	Beg Hit@1 (%)	Med Hit@1 (%)	Ter Hit@1 (%)
Sign Fill-in Start:								
KneserNeyInterpolated	0.673	56.7	63.7	82.3	88.8	41.7	72.7	54.8
Lidstone	0.627	52.6	59.4	76.8	84.1	46.5	72.8	33.4
MLE	0.696	59.6	66.9	82.8	89.4	46.5	72.8	59.4
StupidBackoff	0.696	59.6	66.9	82.8	89.4	46.5	72.8	59.4
WittenBellInterpolated	0.701	60.4	68	82.6	88.2	46.3	75.3	60.1
Sign Fill-in Full:								
KneserNeyInterpolated	0.783	68.1	75.5	92.3	97.2	55.3	74.7	76.2
Lidstone	0.825	73.7	80	94.3	97	62.9	83.7	74.1
MLE	0.832	74.9	81.5	93.9	97.3	62.2	80.2	85.1
StupidBackoff	0.832	74.9	81.5	93.9	97.3	62.2	80.2	85.1
WittenBellInterpolated	0.791	68.9	77.3	92.9	96.8	59.1	74.6	74.8

Table 1: Performance of Sign Fill-in Start and Sign Fill-in algorithms on top of each model on the 'training' dataset to find a missing sign, Avg. of 100 runs with random sign replacements. MRR: Mean reciprocal rank. The higher the MRR, the better the total accuracy of prediction is. Total Hit@1%, Hit@5%, and Hit@10% represent the percent probability of predicting a sign correctly out of 1, 5, and 10 predicted signs. Total Hit Category @1% is the percent probability of correctly predicting a sign category (as categorized in ICIT). Beg Hit@1%, Med Hit@1%, and Ter Hit@1% represent the percent probability of predicting a sign correctly at the beginning, middle and terminal locations, respectively, for Left to Right texts out of one predicted sign.

Model Name	MRR	Total Hit@1 (%)	Total Hit Category @1 (%)	Total Hit@5 (%)	Total Hit@10 (%)	Beg Hit@1 (%)	Med Hit@1 (%)	Ter Hit@1 (%)
Sign Fill-in Start:								
KneserNeyInterpolated	0.366	30.9	43.1	44.9	48.8	18.3	42.9	31.8
Lidstone	0.373	29.2	40.7	47.7	53.5	17.8	40.4	29.3
MLE	0.376	31.4	43.8	46.2	49.6	17.8	40.4	34
StupidBackoff	0.376	31.4	43.8	46.2	49.6	17.8	40.4	34
WittenBellInterpolated	0.393	33	44.8	47.3	50.8	19.6	43.1	35.3
Sign Fill-in Full:								
KneserNeyInterpolated	0.371	28.5	42.8	48.5	57.8	11.4	39.5	32.5
Lidstone	**0.443**	**35**	**49.2**	**56.9**	**62.8**	**19.5**	**48**	**36.7**
MLE	0.367	27.8	43.3	49	57.5	11	39.1	31
StupidBackoff	0.367	27.8	43.3	49	57.5	11	39.1	31
WittenBellInterpolated	0.394	30.1	44.8	51.9	59.6	13	42.6	33.4

Table 2: Performance of Sign Fill-in Start and Sign Fill-in algorithms on top of each model on the 'test' dataset to find a missing sign, Avg. of 100 runs with random sign replacements. MRR: Mean reciprocal rank. The higher the MRR, the better the total accuracy of prediction is. Total Hit@1%, Hit@5%, and Hit@10% represent the percent probability of predicting a sign correctly out of 1, 5, and 10 predicted signs. Total Hit Category @1% is the percent probability of correctly predicting a sign category (as categorized in ICIT). Beg Hit@1%, Med Hit@1%, and Ter Hit@1% represent the percent probability of predicting a sign correctly at the beginning, middle and terminal locations, respectively, for Left to Right texts out of one predicted sign.

~ a 35% rate to get a perfect match of the sign. ICIT corpus also groups signs into categories on how similar they are. We could predict the missing sign category at about ~49%. When we predicted a sign amongst five signs, we could predict at ~57% and amongst ten signs at ~63% accuracy rate.

Real missing signs and predictions

Since the actual missing and unclear texts were similar to the test dataset texts, we then applied the above to the actual missing and unclear texts. We used actual missing and unclear texts with one missing sign separated in the corpus before we built the trained models. There are about 290 texts with multiple missing texts and ~100 with one missing sign. We ran this through the algorithms and the language models and documented the signs the model predicted (**Figure 6**).

Using the best model and Sign Fill-in Full algorithm, we could predict the signs for about 100 texts where single signs were missing and publish the predicted signs. If we did this by random guess, it would only match among ten predicted signs at ~1.4%, and with just positional probabilities, it would match among ten predicted signs at ~7%.

Other Models and Observations

We built a positional probability model with a 7% prediction rate using each sign's positional frequency. We used unigram (n=1) positional probability in a separate trial to predict an unclear sign. While it did much better than a random guess (1.4%), it did poorly overall. We used the positional probability as a fallback. We also constructed an Initial-Terminal Model to see any relationships between the initial and terminal signs (long-distance syntax). For this, we constructed bigrams out of initial and terminal signs. The results for predicting terminal signs using this model were not good, indicating that the relationship between initial and terminal symbols is weak.

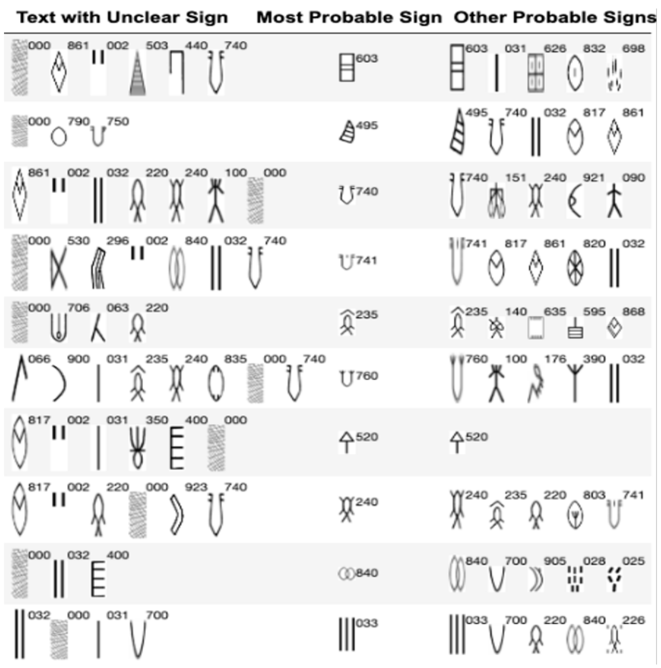


Figure 6: Unclear text with a predicted most probable sign and other probable signs. Up to 5 predictions are shown here for ten texts out of ~100 unclear texts. Text is displayed from Left to Right. The Text with Unclear sign column provides the sign and its ICIT code. '000' represents a missing sign and is denoted by a hashed block. The Most Probable sign value is the most likely predicted sign that fills the missing sign. Other probable signs are signs that can fill the missing sign in the decreasing order of probability.

DISCUSSION

We performed statistical analysis of the Indus texts using sign clusters and sign positional distribution and built a positional probability model. The positional analysis indicates that the texts follow some syntactic structure with text beginners, text enders, and sets of medial signs. n-grams exhibit various frequencies in the text, from bigrams to pentagrams, and as we go beyond pentagrams, that frequency decreases. The positional affinity of some signs indicates that they may be representing some syntax in the language.

Based on the understanding that the Indus script behaves like a language script and the sign cluster behavior, we hypothesized that we could use Markov chain language models to understand the script better. For this, we built various Markov chain language models with various smoothing and interpolation techniques for various orders of n-grams to test all combinations. The problem of filling in unclear and missing signs in texts is complex as signs could be missing at the beginning, end of a text, or in the middle, making just the look-ahead that a language model provides by itself unsuitable. Some have used dynamic algorithms like the Viterbi algorithm to find the most probable state sequence for a given observation in a hidden Markov model (14). We took a different approach to filling in the signs. We built the algorithms that we call Sign-Fill-In Start and Sign-Fill Full on top of the various language models and put our training and test datasets through them. We calculated extrinsic model performance in filling in a missing sign at various locations in the text.

For the Indus text, using the ICIT corpus, we found that a group of three signs in a cluster capture much information when the signs appear in the middle of a text. Four signs are significant when appearing in the rightmost position (in R to L text). Signs appearing in the leftmost positions (in R to L text) are the most difficult to predict. In many texts, the leftmost signs seem separate and may not be related to the signs before them. Many signs tend to appear in the leftmost position, making predicting them difficult.

Our state-of-the-art language models and algorithms will be helpful for any researcher deciphering the Indus script. With a ~63% rate of successfully predicting a missing sign among ten possible signs, this will help in making the corpus more complete and in the deciphering effort. We have shared the code used in this project with the Indus script research community via GitHub so that other researchers in the Indus script research community can expand on the ideas to develop other novel ways to enhance these ideas.

The Indus text corpus is limited, and we do not have long texts either. The language models that use this limited corpus thus have limited information. We are likely trying to learn a language from snippets of texts, names, or measures. Long-distance relationships of signs, such as through medial-terminal or initial-terminal relationships, are not considered in our n-gram models.

We can extend this research to fill in multiple missing signs in a single text, which is a more challenging problem than filling in a single missing sign. We can also use deep learning techniques on the Indus texts with missing texts to see if that improves the prediction rates as was done with other languages (16). Since we have already developed the Markov language models for the ICIT corpus, we can also use these models to find anomalies and outliers. For example, these language models can be used to identify if Indus texts found in West Asia fit well with the Indus texts found in the Indian subcontinent.

MATERIALS AND METHODS

Data Preprocessing and Clean-up

To start designing a statistical model of the Indus script, one needs the corpus of the Indus text to be as complete as possible and digitized. The first significant corpus documenting the Indus texts is M77 (1977), with ~3500 separate lines of text. Most of the previous research works have used this corpus. The ICIT database is one of the corpora that have some 4500+ artifacts with texts. It is a living corpus; the corpus administrators add newly found texts regularly. M77 has 417 distinct signs in M77, and ICIT has 695 distinct signs. The variations in the number of signs are due to differences in opinions about whether a sign is an allograph of a sign or a distinct sign, along with how mirror image signs and repeated double signs are interpreted. As the ICIT corpus is continuously updated, we decided to use that over M77.

The ICIT corpus uses unique numbers for each sign. ICIT represents text beginning and ending with +, missing sign as 000, text separation with a /, and doubtfully restored text with [and]. For example, a seal from Lothal (L-95) has text that in ICIT code can be represented as **+740-000-175-002-880+** and read right to left. As a first step, we converted the ICIT text corpus data into easily workable CSV files. As with any statistical analysis, we need to clean the data, and we did that for the ICIT corpus. We considered repeated texts from the same dig site with the same cult/animal object as duplicate texts to avoid the TAB effect (11) of us re-counting the same tablet. We kept only one copy of such repeated texts. We removed texts with unclear signs and separated them to do further work on them. We discarded texts where the directionality of the text was ambiguous or unknown because the direction of the text is vital for this statistical and positional modeling. We excluded Multi-line texts, as in several of them directionality of the texts was ambiguous. We excluded Multipart texts, where texts spanned multiple faces of the artifacts, as there was ambiguity in the direction of these texts. For us to build n-gram models, working with Left to Right text (L to R) was easier, so we converted all text to L to R text. We then removed the + and - signs to make it easy to process.

Sign Positional Analysis

Since texts are of different lengths, for the positional analysis part, we performed normalization to make it convenient to compare positions. We normalized all texts to the standard size of 10 signs just for sign positional analysis purposes. The positions are from right to left, and we designated the rightmost position 1, similar to what other researchers have done (18, 19). We computed the frequencies for each sign for each position and plotted them to identify patterns. We built a positional probability matrix for each sign and each position from these analyses. We then built a positional probability model on it, which predicted the likely sign given a position in a text.

Language Models

We divided the cleaned-up Indus corpus dataset into 80% train and 20% test sets. We padded the text with a text beginner ‘<s>’ and text ender ‘</s>’ as text beginning and ending add more information to the computed n-grams. We used the training dataset to build language models. We trained all the language models with unigram to hexagram tokens. We built two algorithms, Sign Fill-in Start and Sign Fill-in Full, that we designed on top of the language models.

Sign Fill-in Start Algorithm

As mentioned, this algorithm we built finds signs that make the longest n-grams possible using conditional probabilities from the language model. Here is how this algorithm work for missing signs in various positions in the text:

Missing Sign is in one of the Medial (any of the middle) Positions:

Let us assume an Indus text looks like this: A, B, C, D, E, F, with each English character representing an Indus sign. If sign D is missing, the text looks like A, B, C, #, E, and F, with # representing the missing sign. We find the highest conditional probability of the sign that follows the signs [A, B, C] using L to R language models (language models build with L to R texts). If we do not find a match (when the conditional probability is >= our minimum probability threshold), we back off and find the highest conditional probability of the sign that follows the signs - [B, C]. If we still do not find a match, then the highest conditional probability of the sign follows the signs [C]. If there is no match till the end, we use the sign predicted by a positional probability model as our answer. We then do this for the reverse of the text and find the highest conditional probability of the sign that follows the signs [F, E] (this is the reverse of the text) using the R to L language models (Language models built with R to L texts). If we do not find any probability greater than our threshold, we back off and find the highest conditional probability of the sign that follows the signs [F]. If there is no match till the end, we use the sign predicted by a positional probability model for that position as our answer. We select the sign with the highest model context score as our missing sign prediction.

Missing Sign is in the Rightmost Position:

If the missing sign is in the rightmost position, such as in A, B, C, D, E #, we get the sign predicted by the conditional probability greater than our threshold for [A, B, C, D, E], [B, C, D, E], [C, D, E], [D, E], [E] in that order, using the L to R language models, stopping when we find a match. If there is no match till the end, we use the sign predicted by the positional probability model for that position as our answer.

Missing Sign is in the Leftmost Position:

If the sign that is missing is in the leftmost position, such as in # B, C, D, E, F., We get the sign predicted by conditional probability greater than our threshold for [F, E, D, C, B], [E, D, C, B], [D, C, B], [C, B], [B], using the R to L language models, in that order, stopping when we find a match. If there is no match till the end, we use the sign predicted by a positional probability model for that position as our answer.

Sign Fill-in Full Algorithm

As mentioned before, this algorithm finds signs that make the longest n-grams possible with high conditional probability for that n-gram order from the list of n-grams in the corpus for a language model. Here is how this algorithm work for missing signs in various positions in the text:

Missing Sign is in one of the Medial Positions:

If the Indus text has a missing sign in the middle as in A, B, C, # E, F with # representing the missing sign, starting from the leftmost sign, get *al*l the n-gram of various sizes that includes the missing sign #. Ignore any n-gram that does not include the #, i.e., [A, B, C # E, F], [A, B, C, # F] [A, B, C #]. Then move right one character and do the same thing [B, C, #, E, F], [B, C, #, F] [B, C #]. We do this until we start with # [C, #, E, F], [C, #, E] [C, #, E] [C, #] [#, E, F], [#, E] [#]. These are input n-grams, all the possible n-grams that include the missing sign.

For each of the input n-grams produced above, starting from the order we want to start from, i.e., 'k,' get *al*l the k $_{th}$ order model n-gram from the n-gram model. For each input n-gram above, we do a sign-by-sign match with the model n-gram list for that order and find options for sign candidates for the missing sign that will lead to a full match. For each sign candidate, we get the context model score (how well the text snippet, aka the context, fits the overall language model). Once we have a full match, we do the same thing for the rest of the input n-grams of the same order. We select the answer that has the best score for the highest order. The sign used for the missing sign is our answer. If we do not have a match for k th order, back off and go down one order of input n-gram, do the same for $k_{-1\ th}$ input order, and repeat the same process until we get a match. If we do not get a match when we reach [#], we use the position of the missing sign and get the sign that the positional probability model returns for that position. The Sign fill-in Full algorithm uses the n-gram language models to get the context model scores for the options, but it operates like a simple back-off algorithm.

Missing sign in Rightmost Position:

Let us assume that the sign is missing in the rightmost position, such as A, B, C, D, E, #. The input n-grams that we would try to maximize the context score down the order would be [A, B, C, D, E #], [B, C, D, E, #], [C, D, E, #], [D, E, #], [E, #], [#] in that order. This part of the algorithm is similar to the pure look ahead case of predicting the next sign given a set of signs from the standard n-gram models.

Missing sign in Leftmost Position:

Assume that the sign is missing in the leftmost position, such as in # B C D E F. The input n-grams that we would try to maximize the score would be: [#, B, C, D, E, F], [#, B, C, D, E], [#, B, C, D], [#, B, C, D], [#, B, C], [#, B], [#, B],[#] in that order.

As described, we implemented the Sign Fill-in Start and the Sign Fill-In Full algorithms on top of all the language models.

ACKNOWLEDGEMENTS

We thank Dr. Andreas Fuls for graciously sharing the Indus Text corpus through the ICIT website. Sincere thanks to the Indus research community for inspiring us to contribute to the area of Computational Historical linguistics for deciphering the Indus script.

Received: October 18, 2022
Accepted: January 17, 2023
Published: July 7, 2023

REFERENCES

1. Parpola, Asko, and Jagat Pati Joshi. “Memoirs of the Archeological Survey of India.” vol. 86, *ASI*, 1987.
2. Alex, Bridget. “Why We Still Can’t Read the Writing of the Ancient Indus Civilization?” *Discover Magazine*. www.discovermagazine.com/planet-earth/why-we-still-cant-read-the-writing-of-the-ancient-indus-civilization. Accessed Sep 2021.
3. Bonta, Steven. “The Indus Valley Script: A New Interpretation.*” Penn State University -Altoona College*, 2010.
4. Farmer, Steve, *et al*. “The Collapse of the Indus-Script Thesis: The Myth of a Literare Harappan Civilization.” *Electronic Journal of Vedic Studies*, vol. 11, no. 2, 2004.
5. Fuls, Andreas. “Classifying Undeciphered Writing Systems.” *Journal of Historical Linguistics*, vol. 128, pp. 42-58, 2015.
6. Manning, Christopher, *et al*. “Foundations of Statistical Natural Language Processing.*”* Edited by Hinrich Schutze, MIT Press, 1999.
7. Mahadevan, Iravatham. “The Indus script: Texts, concordance, and tables.*” Memoirs - Archaeological Survey of India*. vol. 77, Archaeological Survey of India, 1977.
8. Yadav, Nisha, *et al*. “A Statistical Approach For Pattern Search In Indus Writing.” *International Journal of Dravidian Linguistics*, vol. 37, pp. 39-52, 2008.
9. Rao, Rajesh P.N, *et al*. “Entropic evidence for linguistic structure in the Indus script.” *Science*, vol. 324, no. 5931, 2009.
10. Daggumati, Shruti, and Peter Revesz. “A method of identifying allographs in undeciphered scripts and its application to the Indus Valley Script.” *Humanities and Social Sciences Communications volume*, vol. 8, no. 50, 2021.
11. Rao, Rajesh P.N, *et al*. “A Markov model of the Indus script.” *PNAS*, vol. 106, no. 33.
12. Palaniappan, Satish, and Ronojoy Adhikari2. “Deep Learning the Indus Script.” *ArXiv*, no. Feb 2017.
13. Assael, Yannis, *et al*. “Restoring and attributing ancient texts using deep neural networks.” *Nature*, no. 603, 2022, pp. 280–283.
14. Yadav, Nisha, *et al*. “Statistical Analysis of the Indus Script Using n-Grams.” *PLOS One*, vol. 5(3), no. e9506, 2010.
15. Wells, Bryan, and Fuls, Andreas. “Online Indus Writing Database.” Berlin 2010, http://www.indus.epigraphica.de/, Accessed 30 Nov 2022.
16. Fetaya, Ethan, *et al*. “Restoration of fragmentary Babylonian texts using recurrent neural networks.” *PNAS*, vol. 117, no. 37, 2020.
17. Jurafsky, Dan, and James H. Martin. “Speech and Language Processing,*”* 2nd Edition. Pearson Prentice Hall, 2008.
18. Wells, Bryan K. “Epigraphic Approaches to Indus Writing.” Oxbow Books, 2011.
19. Fuls, Andreas. “Positional Analysis of Indus Signs.” Вопросы эпиграфики: материалы международной конференции “Вопросы эпиграфики. Выпуск 7, часть 1, Университет Дмитрия Пожарского, 2013