Segmenting Numerical Substitution Ciphers

Nada Aldarrab
Department of Information Technology
King Abdulaziz University
Jeddah, Saudi Arabia
nialdarrab@kau.edu.sa

Jonathan May[†]
Information Sciences Institute
University of Southern California
Marina del Rey, California, USA
jonmay@isi.edu

Abstract

Deciphering historical substitution ciphers is a challenging problem. Example problems that have been previously studied include detecting cipher type, detecting plaintext language, and acquiring the substitution key for segmented ciphers. However, attacking unsegmented ciphers is still a challenging task. Segmentation (i.e. finding substitution units) is essential for cracking those ciphers. In this work, we propose the first automatic methods to segment those ciphers using Byte Pair Encoding (BPE) and unigram language models. Our methods achieve an average segmentation error of 2% on 100 randomly-generated monoalphabetic ciphers and 27% on 3 real historical homophonic ciphers. We also propose a method for solving non-deterministic ciphers with existing keys using a lattice and a pretrained language model. Our method leads to the full solution of the IA cipher; a real historical cipher that has not been fully solved until this work.

1 Introduction

The contents of thousands of historical documents are still unknown to the contemporary age, even though they are encrypted using classical methods. Example documents include books from secret societies, diplomatic correspondences, and pharmacological books. Previous work has been done on collecting historical ciphers from libraries and archives and making them available for researchers (Megyesi et al., 2019, 2020). However, decipherment of classical ciphers is an essential step to reveal the contents of those historical documents.

Methods for deciphering historical substitution ciphers have been proposed by the natural language processing community (Ravi and Knight, 2008; Corlett and Penn, 2010; Nuhn et al., 2013, 2014; Hauer et al., 2014; Aldarrab, 2017; Kambhatla et al., 2018; Aldarrab and May, 2021). However, these methods all assume that cipher elements, or *substitution units*, are clearly segmented (i.e., that token boundaries are well established). Many historical documents, however, are enciphered as continuous sequences of digits that hide token boundaries (Lasry et al., 2020). An example cipher (the IA cipher) is shown in Figure 1 (Megyesi et al., 2019). Solving those ciphers is very challenging since it is not possible to directly search for the key without finding substitution units. We use the term **numerical ciphers** to refer to these unsegmented substitution ciphers.[1]

In this work, we address the problem of segmenting numerical ciphers. The contributions of our work are:

- We propose novel unsupervised methods to segment numerical ciphers **with no existing keys** using Byte Pair Encoding (BPE) (Gage, 1994) and unigram language models (Kudo, 2018).

- We conduct extensive testing of our methods on different cipher types. We report results on synthetic and real historical ciphers and show how performance varies with cipher type and length. Our methods achieve an average segmentation error of 2% on 100 randomly-generated monoalphabetic ciphers and 27% on 3 real homophonic ciphers.

- We propose the first model to segment nondeterministic numerical ciphers **with existing keys** using a segmentation lattice and a pretrained language model. Our method unveils the content of the IA cipher; a letter from the 16th century that has not been fully solved until this work.

[†] Work done prior to JM joining Amazon.

[1] This is because they nearly always use a numerical symbol set, though our proposed methods can be applied to any unsegmented substitution cipher.

Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 706–714
December 7-11, 2022 ©2022 Association for Computational Linguistics

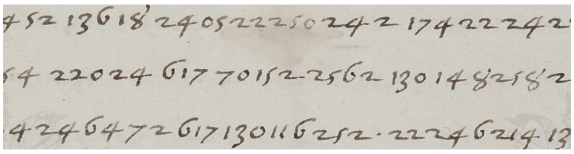


Figure 1: The IA cipher (16th century).[2]

2 Problem Definition

A substitution cipher is a cipher that is created by substituting each plaintext character with another character according to a substitution table called the **key**. We define major terms in the following subsections.

2.1 Substitution types

In this paper, we focus on two types of substitution ciphers: Monoalphabetic and homophonic ciphers. Monoalphabetic ciphers are created by replacing each plaintext character with a unique substitute using a 1→1 substitution key. Homophonic ciphers are created by replacing each plaintext character with one of multiple possible substitutes using a 1→M substitution key.

For example, the key shown in Figure 2 contains a homophonic substitution table (the top part). As shown in the figure, each plaintext character (e.g. i) can be substituted with one of multiple substitutes (e.g. 54 or 74). It is common to encipher vowels with more than one character, which makes homophonic ciphers harder to crack.

2.2 Cipher elements

Cipher elements are substitution units that correspond to plaintext elements according to a cipher key. There are three main types of cipher elements in historical ciphers:

- **Regular elements:** These elements usually encode letters, common syllables, or prepositions. In the example key shown in Figure 2, the top part defines regular cipher elements.

- **Nomenclature elements:** This refers to elements in a key that represent whole words (often proper names). In Figure 2, the second part defines nomenclature elements.

- **Nulls:** These are cipher elements that do **not** correspond to any plaintext word or character. Nulls are usually used in ciphers to confuse cryptanalysts. Sometimes, nulls are used for a purpose. For example, they could be used to mark the beginning of nomenclature elements in numerical ciphers.

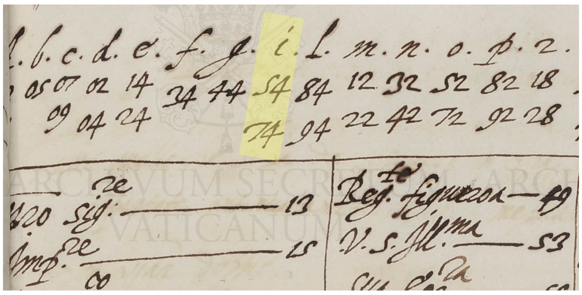


Figure 2: An example homophonic key from the Vatican Secret Archives (16th century).[3] The added highlight in the top section shows that, for example, i can be substituted with 54 or 74. The bottom section contains nomenclature elements.

2.3 Fixed and variable-length ciphers

Numerical ciphers can be classified as fixed or variable-length ciphers. In fixed-length ciphers, regular elements have the same length (i.e. the same number of digits). However, in variable-length ciphers, regular elements can be of different lengths. For example, the letter a might be enciphered as 1, 12, or 121.

2.4 Ciphertext segmentation

Numerical ciphers impose a special challenge, in that they hide cipher element boundaries. For example, in the numerical cipher shown in Figure 1, it is unclear which digits represent substitution units. Identifying substitution units, which we call **segmentation**, is a challenging task that is necessary to solve these ciphers. Another challenge in solving numerical ciphers is that segmentation can be non-deterministic. For example, a cipher can have these substitutions in its key:

Cipher	Plain
2	a
22	n
8	d

which means that the ciphertext 2228 can be segmented as:

[2] Record 189 in the DECODE database: https://de-crypt.org/decrypt-web/RecordsView/189

[3] Record 206 in the DECODE database: https://de-crypt.org/decrypt-web/RecordsView/206

Cipher Segmentation	Plain
2 | 2 | 2 | 8	a a a d
2 | 22 | 8	a n d
22 | 2 | 8	n a d

Such ciphers are called **non-deterministic ciphers.** On the other hand, **deterministic ciphers** have only one possible segmentation according to their keys.

In this paper, we focus on the problem of segmenting numerical ciphers. We look at two cases for numerical cipher segmentation depending on whether or not a key exists for the cipher in hand. The following sections describe our proposed methods for each case.

3 Segmenting Ciphers with no Existing Keys

We start with the first (and the more challenging) case; segmenting a numerical cipher with no existing key. In this case, all we have is a sequence of digits (see, for example, Figure 3). To solve the cipher, we need to segment the ciphertext before trying to find the substitution key. In this section, we describe our proposed methods for segmenting numerical ciphers without using a key.

3.1 Baselines

We first try two baselines: 1-digit and 2-digit segmentation. We remove line breaks and consider the text as one long sequence of digits. In 1-digit segmentation, we split the ciphertext into individual digits. In 2-digit segmentation, we split the ciphertext into two-digit elements (except the last digit if the number of digits in the cipher is odd). The latter is a stronger baseline since we notice that most cipher elements in historical ciphers are two digits long.

3.2 Byte Pair Encoding (BPE)

Our first proposed method for cipher segmentation is Byte Pair Encoding (BPE). BPE is a simple compression algorithm that has been used for many natural language processing tasks (Gage, 1994; Sennrich et al., 2016). In BPE, the most frequent pair of bytes is iteratively replaced with a single, unused byte to represent the replaced pair. The motivation behind using BPE for our problem is that the digits that belong to the same cipher element have high mutual information, so we would like them to be grouped together.

3.3 Unigram language model

One downside of BPE is that it is a greedy algorithm that employs a deterministic symbol replacement strategy. BPE does not provide multiple possible segmentations with probabilities. As we notice from our experiments (Section 5.1), the resulting BPE segmentation leaves many singleton digits unpaired.

To mitigate this problem, we use the subword segmentation algorithm proposed by Kudo (2018), which is based on a unigram language model. This algorithm provides candidate segmentations with probabilities. The unigram language model assumes that each subword occurs independently. Thus, the probability of a subword sequence is the product of the subword probabilities. The probabilities are iteratively estimated using the Expectation Maximization (EM) algorithm. The most probable subword segmentation is then found by the Viterbi algorithm.

We evaluate the two baselines and our proposed methods on synthetic and real historical ciphers. The following sections describe our datasets, experiments, and results.

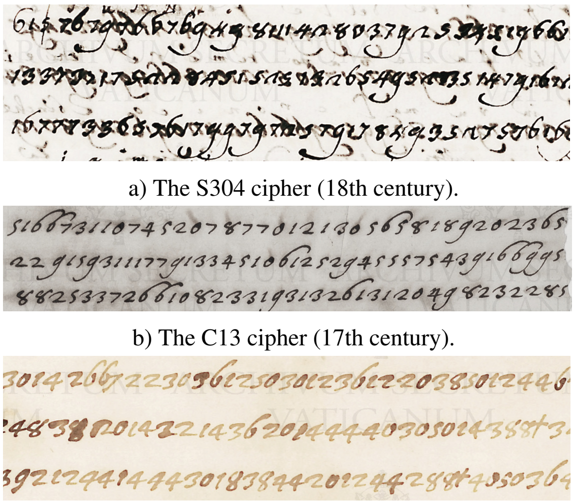
4 Data

To evaluate our methods on monoalphabetic numerical ciphers, we create synthetic ciphers from sample text extracted from English Wikipedia. We lowercase all characters and remove all non-alphabetic and non-space symbols. We notice that most historical ciphers in the DECODE collection are two pages long and contain about 2K characters, so we choose cipher length 2,048 for our experiments. We create 100 English ciphers using randomly generated keys. We use the numbers from 0 to 99 as possible cipher elements. This creates variable-length ciphers when single digits are chosen in the key. We report the average scores of the 100 ciphers for each experiment.

For evaluation on real historical ciphers, we use 3 ciphers from the the Vatican Secret Archives, retrieved from the DECODE database (Megyesi et al., 2020; Lasry et al., 2020). Table 1 shows cipher statistics. We use ciphers C13, S304, and F283 (see Figure 3). For these ciphers, human transcriptions and gold segmentations are available on the DECODE database.

Cipher	Length	Types	Tokens	1-dig Tokens	2-dig Tokens	1+2-dig Tokens
S304	1,258	82	675	150 (22%)	496 (73%)	646 (96%)
C13	1,879	97	917	0 (0%)	872 (95%)	872 (95%)
F283	2,239	50	1,050	1 (0%)	979 (93%)	980 (93%)

Table 1: Statistics of the ciphers obtained from the DECODE database. Length is the number of digits in the cipher. Types and tokens are those of cipher elements, not individual cipher digits.



a) The S304 cipher (18th century).

b) The C13 cipher (17th century).

c) The F283 cipher (16th century).

Figure 3: Three historical ciphers from the Vatican Secret Archives.[4]

5 Experimental Evaluation

We carry out three types of experiments. First, we start with the simplest cipher type; monoalphabetic ciphers with spaces. The existence of spaces indicates word boundaries, which gives some clues on how to segment the ciphertext. Second, we remove spaces and try to segment the same monoalphabetic ciphers, which is expected to be a harder task. Third, we experiment with segmenting homophonic ciphers, which is the most challenging case discussed in this paper.

We apply our proposed segmentation methods on both synthetic and real historical ciphers. We also study the effect of cipher length on segmentation quality.

As an evaluation metric, we use Segmentation Edit Rate (SegER). SegER is the segment-level Levenshtein distance between output and gold segmentations, divided by the number of segments in the gold segmentation. We define SegER as:

$$\text{SegER} = \frac{\text{\# of edits}}{\text{\# of reference segments}} \qquad (1)$$

where possible edits include the insertion, deletion, and substitution of single segments. For example, if the gold segmentation is 65 17 77 71 and the output segmentation is 65 17 7 7 7 1, then recovering the gold segmentation from the output segmentation requires 2 substitutions and 2 deletions (SegER = 4/4 = 100%). The lower the SegER, the closer the output segmentation is to the gold segmentation.

We use the SentencePiece implementation of BPE and unigram language model (Kudo and Richardson, 2018). Segmenting a 2K-token cipher takes about 0.08s on a 3.6 GHz Quad-Core Intel Core i3 processor. For homophonic ciphers, we set the vocabulary size to the maximum number found by the unigram language model. For monoalphabetic ciphers, we set vocabulary size to 36 (26 maximum possible 2-digit elements for 1-1 substitutions + 10 singleton digits).

We use the default settings in SentencePiece, but we set character coverage to 100%. We learn subwords from raw unsegmented ciphertext, represented as continuous sequences of digits. We keep line-breaks as they appear on cipher scans since we notice that line-breaks usually do not cut through a cipher element in historical ciphers.

5.1 Monoalphabetic ciphers with word spaces

We first experiment with monoalphabetic ciphers with spaces. We test our methods on the 100 synthetic ciphers described in Section 4. Table 2 (first column) shows SegER scores for all models. As expected, the 2-digit baseline is much better than the 1-digit baseline, with a SegER score of 23%, as opposed to 181% for the 1-digit baseline.

We find that default BPE does not perform better than the 2-digit baseline, with a SegER score of about 34%. However, as noted in Section 3.1, most cipher elements in historical ciphers are one or two digits long. In our random sample of three historical ciphers, about 95% of cipher tokens are

[4] Records 121, 208, and 72 in the DECODE database: https://de-crypt.org/decrypt-web/RecordsList

Model	**w/ spaces**	**w/o spaces**
1-dig baseline	181.05	181.05
2-dig baseline	23.20	49.89
BPE	34.32	36.74
BPE 2	10.95	13.72
Unigram LM	40.28	41.61
Unigram LM 2	**2.45**	**2.70**

Table 2: Average SegER % (↓) for segmenting 100 synthetic ciphers using different models.

Gold	86	1	17		77	65	39
BPE 2	86	1	1	$\underline{77}$	7	65	39
Unigram LM 2	86	1	17		77	65	39
Gold	65	17	77	71			
BPE 2	65	1	$\underline{77}$	$\underline{77}$	1		
Unigram LM 2	65	17	77	71			

Figure 4: Example BPE segmentation errors (incorrect merges underlined). In both examples, BPE chooses the wrong merges as it goes greedily from left to right. Unigram LM, on the other hand, looks at candidate segmentations and chooses the highest scoring candidate based on segmentation probabilities.

one and two-digit (as shown in Table 1). Longer elements appear less often (less than 5% of tokens in our test ciphers). Thus, we limit BPE piece length to a maximum of 2 digits. This improves the SegER score by reducing it to about 11%. We call this model “BPE 2” in Table 2.

We then apply the unigram language model of Kudo (2018). We notice that the default unigram language model is subpar to default BPE. However, adding the 2-digit heuristic, we get the best result of all models with a SegER of 2.45% (Called “Unigram LM 2” in Table 2).

To better explain the motivation behind using the unigram language model, we show example BPE 2 errors in Figure 4. These two examples come from the same cipher. For this cipher, BPE learned the right vocabulary elements of 17, 71, and 77. However, since 77 is the most frequent of all, BPE always prefers to merge the two 7s first. This early merge results in two unmerged single digits (1 and 7 in the first example and two 1s in the second example). This way, BPE 2 misses the correct merges of 17 and 71. The unigram language model, on the other hand, looks at the overall score of segmentation candidates and chooses the most probable one according to unigram frequencies. In both examples, Unigram LM 2 does a better job at segmenting ciphertext.

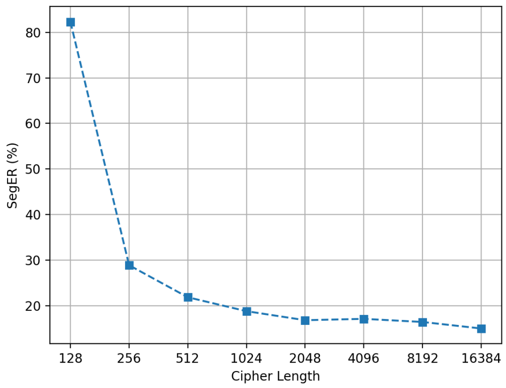


Figure 5: SegER % (↓) for segmentation of different cipher lengths.

To study the effect of cipher length on segmentation quality, we create a monoalphabetic substitution cipher with variable-length cipher elements from English text. The cipher’s length is 16,384 characters. We start by testing our model on the first 128 characters of the text, then we increase the length by a power of 2 until we reach 16,384. Figure 5 shows segmentation results for different cipher lengths. As expected, segmentation quality improves as cipher length increases.

5.2 Monoalphabetic ciphers without word spaces

We test our methods on the same set of 100 synthetic ciphers after removing spaces. To resemble real historical ciphers, we break the ciphertext into 43-character lines. The number of characters per line varies from one cipher to another, but as an approximation, we choose the average number of characters per line in a random sample of real ciphers.

As shown in Table 2 (second column), SegER scores for no-space monoalphabetic ciphers are generally slightly worse than ciphers with spaces. Our best performing model (Unigram LM 2) achieves a SegER of 2.7% on no-space monoalphabetic ciphers, which is very close to the 2.45% on the same ciphers with spaces.

5.3 Homophonic ciphers

We test our segmentation methods on three real homophonic ciphers: S304, C13, and F283 (Table 1). Note that S304 is the shortest and F283 is the longest of these ciphers (F283 is almost twice as long as S304).

Model	Cipher Name		
	S304	C13	F283
1-dig baseline	164.15	204.91	213.14
2-dig baseline	60.00	41.11	64.19
BPE	78.07	51.80	50.10
BPE 2	63.11	46.02	38.29
Unigram LM	84.59	72.85	38.19
Unigram LM 2	**46.67**	**20.83**	**14.95**

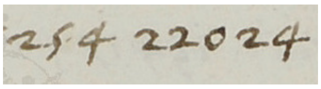
Table 3: SegER % (↓) for segmenting three real homophonic ciphers using different models.

Table 3 shows SegER scores for different models. We notice that the 2-digit baseline is a strong baseline since most cipher elements are 2-digit in these historical ciphers. The 2-digit baseline is much better than the 1-digit baseline on these historical ciphers, with an average improvement of more than 70%. As we have seen in our synthetic, monoalphabetic cipher experiments, restricting piece length to a maximum of 2 improves performance for BPE and Unigram LM. With the 2-digit heuristic, SegER improves by an average of 18% and 58% for BPE and Unigram LM, respectively.

While we could not find previously published work on this problem, we can see that our best method (Unigram LM 2) achieves an average SegER of 27% on the three real homophonic ciphers, with the best score of 15% on the longest, 2,239-digit F283 cipher.

6 Segmenting Non-Deterministic Ciphers with an Existing Key

We now consider the second case: Suppose we have a cipher and a key, but the cipher is non-deterministic. This case can arise in practice when the key of the cipher is found while combing through historical archives, for example. Alternatively, the key could have been found by a cryptanalyst by solving a part of the cipher. Although the cipher key exists in these scenarios, the non-deterministic segmentation makes it impossible to directly apply the key to recover the plaintext; recall the ambiguous segmentation example of the word and from Section 2.4. In this case, it is very challenging to manually recover the whole plaintext, especially when the cipher is very long.



```
2   5   4   2   2   0   2   4
25      4   2   2   0   2   4
2   5   4   22      0   2   4
2   5   4   2   2   0   24
25      4   22      0   2   4
25      4   2   2   0   24
2   5   4   22      0   24
25      4   22      0   24
```

Figure 6: Example segmentation ambiguity for the IA cipher. A short 8-digit part of the cipher produces 8 possible segmentations according to the key. The number of candidate segmentations increases exponentially with respect to cipher length.

6.1 Lattice segmentation

We take as an example the IA cipher (Figure 1), which we retrieved from the DECODE database (Megyesi et al., 2020). The first few lines of this 16th-century cipher were deciphered in 2019. However, since the cipher is non-deterministic, the remaining ciphertext (more than 200 lines) has not previously been deciphered.

This is a real use case for our proposed method; a real historical cipher with an existing key but with a non-deterministic segmentation. For example, consider this part of the IA cipher key:

Cipher	Plain	Cipher	Plain
0	e	22	p
2	o	24	r
4	a	25	t
5	s		

Figure 6 shows a short 8-digit part of the IA cipher. As shown in the figure, this part can be segmented in 8 possible ways according to the key. The number of candidate segmentations increases exponentially with respect to cipher length.

To solve this problem, we create a lattice to model all possible segmenations of the cipher using the existing key. Then we use a pretrained language model to choose the best possible segmentation (i.e. the segmentation that gives the most probable plaintext according to the language model).

For the segmentation lattice, we create a Finite-State Transducer (FST) that models the possible merges of cipher symbols. Figure 7 shows part of the FST. The shown transitions model the ambigu-

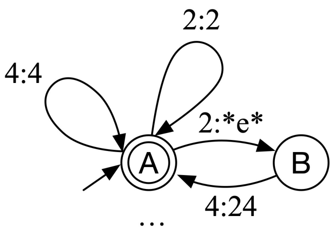


Figure 7: Part of the segmentation FST for the IA cipher. This part models the possiblity of merging the digits 2 and 4 to become 24 (corresponds to letter r in the key) vs. keeping them unmerged (letters o and a in the key). *e* is used to indicate the empty string.

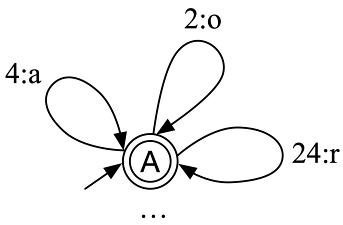


Figure 8: Part of the key FST for the IA cipher. This part models the substitutions: 2→o, 4→a, and 24→r.

ity of segmenting the digits 2 and 4. According to the key, these two digits can be merged to become 24 (plaintext r) or stay unmerged (plaintext letters o and a, respectively). We create another FST to model the key (shown in Figure 8).

We train a 5-gram character Italian language model on the historical data released by Aldarrab and May (2021). Composing the language model, key FST, and segmentation FST creates a lattice of all possible decipherments of the text. We use the Carmel finite-state toolkit to find the most probable plaintext according to the language model (Graehl, 2010).

We follow Aldarrab and May (2021) and use character-level Translation Edit Rate (TER) as our evaluation metric. TER is the character-level Levenshtein distance between system output and gold solution, divided by the number of characters in the gold solution. Verifying the resulting plaintext by a native Italian speaker, our method achieves a TER of 1.12%, which means that our model's output is almost 99% correct.

6.2 The IA cipher

The IA cipher is 11,026 characters long and dates back to 1536 CE. The key from DECODE included 21 cipher elements. However, decoding the rest of the cipher revealed 2 more cipher elements (shown in Table 4). Cipher element 19 seems to encode a nomenclature element, while cipher element 26 encodes the letter x.

Cipher	Plain
19	(possibly a nomenclature element)
26	x

Table 4: Additions to the IA cipher key discovered by our approach.

We determined that there are human transcription errors in the transcription from DECODE. In total, we corrected 30 transcription errors in this cipher.

There also seem to be some errors in the original manuscript. Such errors can result from spelling mistakes or substitution mistakes during encipherment, for example. For those errors, we do not change the original ciphertext and consider the text as is.

7 Related Work

Previous decipherment work has mainly been focused on solving substitution ciphers with clearly segmented cipher elements, e.g. (Hart, 1994; Olson, 2007; Ravi and Knight, 2008; Corlett and Penn, 2010; Nuhn et al., 2013, 2014; Hauer et al., 2014; Aldarrab, 2017). Early decipherment approaches search for the substitution table that gives a highly probable plaintext according to a character LM. More recent approaches incorporate neural models. Kambhatla et al. (2018) use beam search and a neural LM to score candidate plaintext hypotheses from the search space for substitution ciphers. Aldarrab and May (2021) view decipherment as a sequence-to-sequence translation problem. However, all of these works only deal with ciphers that have clear substitution units.

Lasry et al. (2020) present an extensive study on papal ciphers from the 16th to 18th century. Those ciphers are numerical substitution ciphers that need to be segmented. For segmentation, Lasry et al. (2020) create a set of segmenters (called "parsers" in that work) from a collection of known cipher keys. Then they test the cipher in hand to see if any of the previously created segmenters can be a good fit. Our method, by contrast, is not limited to existing keys. In fact, our method is completely unsupervised and only uses ciphertext as input. A similar task to learning cipher segments is learning linguistic units (words and morphemes) from unsegmented text, e.g. (De Marcken, 1996; Goldsmith, 2001; Xu et al., 2018).

8 Conclusion

In this work, we present automatic methods for segmenting numerical substitution ciphers. We propose a novel approach to segment numerical ciphers with no existing keys using subword segmentation algorithms. We use BPE and unigram language models as unsupervised methods to learn substitution units. We add a 2-digit heuristic based on historical cipher analysis. Our best method is able to segment 100 randomly generated monoalphabetic ciphers with an average SegER of less than 3%, while still being robust to removing spaces. We test our methods on 3 real homophonic ciphers from the 16th-18th centuries. Our best method achieves an average SegER of 27%, with a SegER of 15% on the F283 cipher. To the best of our knowledge, this is the first work on automatically segmenting numerical substitution ciphers.

We also propose a method for solving non-deterministic substitution ciphers with existing keys using a lattice and a pretrained language model. Our method achieves a TER of 1.12% on the IA cipher, a real historical cipher that has not been fully solved until this work.

Limitations

In this work, we present automatic methods for segmenting numerical ciphers. We specifically target unsegmented **substitution** ciphers, which are the most common types of ciphers in the 16th-18th centuries. Thus, the proposed methods might be less effective for other encipherment techniques. For example, in transposition ciphers, letters are rearranged to create ciphertext. This can affect the learned vocabulary using BPE or unigram LM, especially when we deal with a variable-length cipher. Those encipherment methods are out of the scope of this paper.

Another challenge for ciphertext segmentation is the **length** of the ciphertext. In general, the shorter the cipher, the harder it can get to find the correct segmentation (Figure 5). However, since our main goal is to segment real historical ciphers, we test our methods on real homophonic ciphers of 675 characters or more. These are very common lengths in historical cipher collections as described in Section 4.

Acknowledgments

We would like to thank Beáta Megyesi for her assistance and Milena Anfosso for verifying our automatic solution of the IA cipher. The first author is grateful for King Abdulaziz University, Saudi Arabia for supporting this research with a scholarship award. The second author is supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR0011-15-C-0115.

References

Nada Aldarrab. 2017. Decipherment of historical manuscripts. Master's thesis, University of Southern California.

Nada Aldarrab and Jonathan May. 2021. Can sequence-to-sequence models crack substitution ciphers? In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7226–7235, Online. Association for Computational Linguistics.

Eric Corlett and Gerald Penn. 2010. An exact A* method for deciphering letter-substitution ciphers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1040–1047, Uppsala, Sweden. Association for Computational Linguistics.

Carl G. De Marcken. 1996. *Unsupervised Language Acquisition*. Ph.D. thesis, Massachusetts Institute of Technology, USA.

Philip Gage. 1994. A new algorithm for data compression. *C Users J.*, 12(2):23–38.

John Goldsmith. 2001. Unsupervised learning of the morphology of a natural language. *Computational Linguistics*, 27(2):153–198.

Jonathan Graehl. 2010. Carmel finite-state toolkit.

George W. Hart. 1994. To decode short cryptograms. *Commun. ACM*, 37(9):102–108.

Bradley Hauer, Ryan Hayward, and Grzegorz Kondrak. 2014. Solving substitution ciphers with combined language models. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 2314–2325, Dublin, Ireland. Dublin City University and Association for Computational Linguistics.

Nishant Kambhatla, Anahita Mansouri Bigvand, and Anoop Sarkar. 2018. Decipherment of substitution ciphers with neural language models. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 869–874, Brussels, Belgium. Association for Computational Linguistics.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia. Association for Computational Linguistics.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

George Lasry, Beáta Megyesi, and Nils Kopal. 2020. Deciphering papal ciphers from the 16th to the 18th century. *Cryptologia*, 0(0):1–62.

Beáta Megyesi, Nils Blomqvist, and Eva Pettersson. 2019. The DECODE database collection of historical ciphers and keys. In *Proceedings of the 2nd International Conference on Historical Cryptology, HistoCrypt 2019, Mons, Belgium, June 23-26, 2019*, volume 158 of *Linköping Electronic Conference Proceedings*, page 158:008. Linköping University Electronic Press.

Beáta Megyesi, Bernhard Esslinger, Alicia Fornés, Nils Kopal, Benedek Láng, George Lasry, Karl de Leeuw, Eva Pettersson, Arno Wacker, and Michelle Waldispühl. 2020. Decryption of historical manuscripts: the decrypt project. *Cryptologia*, 44(6):545–559.

Malte Nuhn, Julian Schamper, and Hermann Ney. 2013. Beam search for solving substitution ciphers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1568–1576, Sofia, Bulgaria. Association for Computational Linguistics.

Malte Nuhn, Julian Schamper, and Hermann Ney. 2014. Improved decipherment of homophonic ciphers. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1764–1768, Doha, Qatar. Association for Computational Linguistics.

Edwin Olson. 2007. Robust dictionary attack of short simple substitution ciphers. *Cryptologia*, 31(4):332–342.

Sujith Ravi and Kevin Knight. 2008. Attacking decipherment problems optimally with low-order N-gram models. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 812–819, Honolulu, Hawaii. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Hongzhi Xu, Mitchell Marcus, Charles Yang, and Lyle Ungar. 2018. Unsupervised morphology learning with statistical paradigms. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 44–54, Santa Fe, New Mexico, USA. Association for Computational Linguistics.