Computational Decipherment of Unknown Scripts

by

Bradley Hauer

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Algorithmic decipherment is a prime example of a truly unsupervised problem. This thesis presents several algorithms developed for the purpose of decrypting unknown alphabetic scripts representing unknown languages. We assume that symbols in scripts which contain no more than a few dozen unique characters roughly correspond to the phonemes of a language, and model such scripts as monoalphabetic substitution ciphers. We further allow that an unknown transposition scheme could have been applied to the enciphered text, resulting in arbitrary scrambling of letters within words (anagramming). We also consider the possibility that the underlying script is an abjad, in which only consonants are explicitly represented.

Our decryption system is composed of three steps. The first step in the decipherment process is the identification of the encrypted language. We propose three methods for determining the source language of a document enciphered with a monoalphabetic substitution cipher. The best method achieves 97% accuracy on 380 languages. The second step is to map each symbol of the ciphertext to the corresponding letter in the identified language. We propose a novel approach to deciphering short monoalphabetic substitution ciphers which combines both character-level and word-level language models. Our method achieves a significant improvement over the state of the art on a benchmark suite of short ciphers. The third step is to decode the resulting anagrams into readable text, which may involve the recovery of unwritten vowels. Our approach obtains an average decryption word accuracy of 93% on a set of 50 ciphertexts in 5 languages. Finally, we apply our new techniques to the Voynich manuscript, a centuries-old document written in an unknown script, which has resisted decipherment despite decades of study.

Preface

The work presented in Chapter 3 of this thesis was published as Bradley Hauer, Ryan Hayward, and Grzegorz Kondrak, "Solving Substitution Ciphers with Combined Language Models", proceedings of The 25th International Conference on Computational Linguistics (COLING 2014), pages 2314-2325. This work was a collaborative effort between all three authors; I completed all implementation and experiments.

The work presented in Chapter 4 has been accepted for publication in the Transactions of the Association for Computational Linguistics as Bradley Hauer and Grzegorz Kondrak, "Decoding Anagrammed Texts Written in Unknown Language and Script". Again, this work was a collaborative effort between the two authors, with all implementation experiments being done by me.

In both of the above cases, the original manuscripts were altered as needed, to improve formatting, readability, and consistency. Additional material which was removed from the manuscripts due to space constraints has also been added to this thesis, and some material from the manuscripts has been relocated elsewhere in the thesis to improve readability.

Acknowledgements

I am grateful for the assistance and support of my supervisor, Greg Kondrak. I would also like to acknowledge Ryan Hayward for his contributions to this thesis.

Thanks to Prof. Moshe Koppel for the assessment of the Hebrew examples.

I would also like to thank the reviewers of the two papers which comprise this thesis, for their insightful and helpful comments.

This thesis was completed with funding from Alberta Innovates – Technology Futures and Alberta Innovation & Advanced Education, as well as the Natural Sciences and Engineering Research Council of Canada.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Encipherment refers to any process which conceals the meaning of a text, called the *plaintext*, by systematically operating on the symbols within the text, resulting in an unreadable *ciphertext*. Typically, an encipherment algorithm is not entirely fixed, but allows some decisions to be made during the encipherment process. Such algorithms typically require a second input, in addition to the plaintext to be enciphered, which specifies how these decisions are to be made. This input is called the *key*. The idea is that, if the key is known, reversing the decipherment process should be easy; that is, if we fully understand what was done to the plaintext to produce the ciphertext, we should be able to reverse the process to recover the plaintext. However, it is equally important that if the key is not known, recovering the plaintext should be as difficult as possible. Indeed, this is the entire purpose of encipherment – to conceal the plaintext from all parties except the intended recipients (who have been given the key beforehand). Encipherment is used when two or more parties must communicate over an insecure channel, such as radio or email, where it is possible for some third party to intercept any message sent.

The design of encipherment algorithms typically follows *Kerckhoffs's assumption*; any messages sent can be intercepted, and all that is needed to recover the plaintext for any ciphertext is the key. In particular, the encipherment algorithm itself (i.e. the method by which the key is used) is assumed to be public knowledge. Despite this assumption, as long as the key is kept secret, an intercepted plaintext alone is, in theory, meaningless, since, as stated above, it should not be possible to recover the plaintext from the ciphertext without the key.

Decipherment seeks to subvert the security provided by an encipherment algorithm. In a decipherment task, one plays the role of a third party who has obtained a ciphertext, and knows the encipherment algorithm used to produce it (Kerckhoffs's assumption), but has no knowledge of the key. The task is to discover the key, at which point the plaintext can be recovered, exactly as the intended recipient would be expected to do, thus breaking the security the encipherment algorithm provides. Decipherment algorithms are typically tailored to a particular encipherment algorithm, by finding and exploiting some property of the encipherment.

The *monoalphabetic substitution cipher* (often shortened simply to substitution cipher) is an example of an encipherment algorithm. It enciphers the plaintext using a bijective function between the set of symbols which appear in the text, called the *plaintext alphabet*, and another set of symbols, called the *ciphertext alphabet*. Without loss of generality, the set of all possible such functions can be viewed as the set of all permutations of the plaintext alphabet (in practice, for ease of reading, we typically write plaintext and ciphertext symbols in lowercase and uppercase, respectively), giving $n!$ possible functions for an n-letter alphabet ($26!$, roughly 10^{26}, for English). This one-to-one mapping between the plaintext and ciphertext alphabets is the key to this cipher. Given such a key, the plaintext is converted to a ciphertext by replacing each symbol in the plaintext with the output of the key given that symbol. An example is shown in Figure 1.1.

If the key and ciphertext are both known, reversing the encipherment to recover the plaintext is straightforward: simply invert the key, and replace each ciphertext symbol with the corresponding plaintext symbol. We want to consider the task of deciphering a ciphertext produced using a substitution cipher. Due to the extremely large number of possible keys, a brute force approach to decipherment (that is, simply trying every possible key) is not feasible. This is not true of all encipherment algorithms: the Caesar cipher, for example, has only n possible keys for an n-letter alphabet, making decipherment easy: the decipherer can simply try them all. Since using the wrong key will almost certainly result in obvious nonsense, the decipher can simply look for the one decipherment which is meaningful, and safely assume that to be the correct plaintext. The Caesar cipher is therefore a very weak

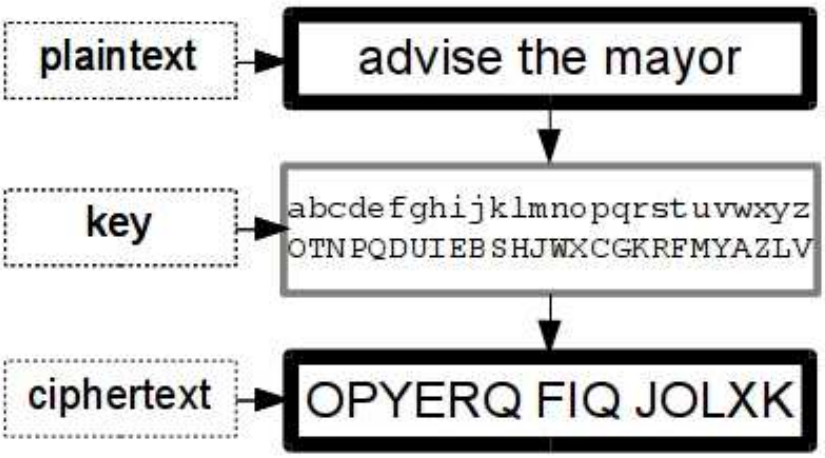

Figure 1.1: An example of encryption with a substitution cipher. The key dictates that each instance of ‘a’ should be replaced by an ‘O’, each ‘d’ with a ‘P’, and so on, until the message is fully encrypted. Reversing the process is obviously easy if the key is known, but non-trivial if it is not.

encipherment, doing little to provide security when communicating over an insecure channel. Under Kerckhoffs’s assumption, a third party needs only the key to obtain the plaintext, and this can be easily obtained by brute force. For a substitution cipher, however, this will not work, as the number of keys is too large to simply try them all. Therefore, more complex decipherment methods are needed.

Methods for the manual decipherment of monoalphabetic substitution ciphers have been known for centuries; for a detailed discussion of these methods, see, for example, Singh (1999). Such methods, however, rely on intuition, creativity, and guesswork from experienced cryptanalysts, and so cannot be automated. Modern work on decipherment typically seeks a fully automated approach, in the form of a computer program which takes a ciphertext as input, and produces the correct plaintext as output. This is motivated by the time and cost advantages an automated decipherment, performed by a computer, provides over manual decipherment, as well as the numerous applications decipherment has to other tasks. This thesis presents new methods for a variety of decipherment problems, including the fully automated decipherment of substitution ciphers, and more difficult variations of this task, as well as several of the aforementioned applications.

In Chapter 2, we review some relevant prior work on decipherment. This includes discussion of previous methods for solving substitution ciphers, as well as some published work on related tasks. The main contribution of this chapter is to provide an overview of the work that has been done in computational decipherment, and to introduce the methods that we will be comparing our own decipherment algorithm against.

In Chapter 3 we present an algorithm which completely automates the process of recovering the key to a monoalphabetic substitution cipher, allowing for recovery of the plaintext given nothing more than the ciphertext and a corpus of text in the source language. As with many prior decipherment algorithms, our method uses an objective function based on language models.

A language model is simply a statistical machine learning method which uses a large sample of text in a language, called a corpus, to learn to estimate the probability of a randomly chosen word or character from a text being a particular word or character. A character language model might tell us, for example, that the probability of a randomly chosen letter in an English text being 'e' is about 10%, while a word language model might tell us that a random word has a roughly 5% chance of being 'the'. Such simple models are rarely useful, as they completely ignore context – we call these models *unigram* models, as they consider a single word or character in isolation. In general, an n-gram word (character) language model estimates the probability of a random word (character) taking on a particular value given the previous $n-1$ words (characters). For example, a trigram character language model might estimate that the probability of a random character being 'e', given that the previous two characters are 'q' and 'u' (in that order) is about 31%, while a bigram word language model might tell us that the probability of a random word being 'am', given that the previous word was 'you', is extremely small (which could help inform us that "you am" is generally not grammatical in English). See, for example, Jurafsky et al. (2000) for a more in-depth look at language modeling techniques, theory and applications.

Language models are important to decipherment, since they can be employed to guide an algorithm to a solution which looks like it might come from a text in

a particular language – which is exactly what we are seeking. Using the probabilities from one or more language models as an objective function therefore simply means that we try to find the key which gives the most probable, or "language-like", decipherment (which we generally assume will be the correct plaintext).

Our algorithm uses a novel method of combining word and character language models with a variety of context sizes, creating a comprehensive key evaluation function. We also develop a key mutation function which is based on the relationship of *pattern-equivalence* (which we will formally define in the description of our algorithm). This function generates a tree of keys which we search to find the highest-scoring key according to our evaluation function. We test our algorithm on two sets of 400 short ciphers, using an experimental setup comparable to that of prior work on this task, and show that our solver outperforms the previous state of the art. We then explore some variations of the task, obtaining good results which demonstrate that our method is applicable to the tasks of unsupervised transliteration and deniable encryption.

Chapter 4 builds upon the previous chapter, motivated by our attempts to analyze a long-standing decipherment problem, an undeciphered 15^{th} century document known as the Voynich manuscript (VMS). The foremost obstacle to attempts to decipher the VMS is that the underlying language is not known, and prior research has yielded some conflicting results. This is a problem which is not often addressed in prior work on decipherment, which often simply assumes that the language of the cipher is known, while in practice (see, for example, Knight et al. (2011)), identifying the original language of the text is often a necessary precursor to decipherment. We present three new methods for this task of *ciphertext language identification*, and show that all three outperform a previously published method on the same task on a 380-language test set. We then consider a second complication when investigating the VMS, the re-ordering of letters within words. Evidence from prior work, and new evidence presented in this thesis, suggests that, in addition to the unique script, the text of the VMS has been further secured by re-arranging the letters within each word, a process called *anagramming*. We extend the algorithm from Chapter 3 to be able to function even when letter order is not assumed to be

reliable, and combine this with a Viterbi decoder to create a method for solving ciphers which combine substitution with anagramming, the first known solver for this task. After evaluating this system on ciphers from five languages, we apply all the tools we have created to the VMS, and present new evidence for the identity of the underlying language, providing new insight into this centuries-old mystery.

Chapter 5 then concludes the thesis. It summarizes the contributions and results of this thesis, and discusses potential future work.

Chapter 2

Prior Work on Decipherment

This chapter provides an overview of recent notable work pertaining to the application of computational techniques, specifically techniques from natural language processing (NLP), to decipherment problems. The focus will be on work related to substitution ciphers; additional prior work will be covered as needed throughout this thesis.

Kevin Knight has been a leading proponent of attacking decipherment problems with NLP techniques, as well as framing NLP problems as decipherment. Knight and Yamada (1999) introduce the topic to the natural language processing (NLP) community by demonstrating how to decode unfamiliar writing scripts using phonetic models of known languages. They present a statistical method based on the expectation-maximization (EM) algorithm to learn correspondences between written symbols and phonemes, treating text as a cipher for speech. They apply their method to Spanish and Japanese (to test their method on both phonetic and syllabic scripts), in each case proceeding as though the script in question was unknown, and could only be read by first deciphering the symbols into phonemes. The results of these decipherments, when read aloud using a speech synthesizer, were comprehensible to a speaker of the language, representing a successful decipherment. Experiments on Chinese were not as successful, however they still obtain a syllable accuracy of 22%. This EM-based method is now outperformed by newer algorithms, to be discussed below; we will compare our results to modern, state-of-the-art methods in Section 3.4.1.

Knight et al. (2006) propose a method for deciphering substitution ciphers which

is based on Viterbi decoding with mapping probabilities computed with the EM algorithm. Their method seeks to find a maximum-probability decipherment as determined by a character language model. The method correctly deciphers 90% of symbols in a 400-letter ciphertext when a trigram character language model is used. As with Knight and Yamada (1999), this method is now outperformed by more recently published solvers. The authors also apply their method to ciphertext language identification, the task of determining, given a ciphertext, the language in which the plaintext was written. This is done by applying the method repeatedly, each time using a language model trained on text in a different language. The language which yields the most probable decipherment is identified as the most likely language of the cipher. Using models from 80 different languages, they report successful classifications on three ciphers that represent English, Spanish, and Spanish written without vowels. We will present our own methods for this task in Section 4.2, including a method which uses a similar framework of quickly deciphering the ciphertext into each candidate language.

Ravi and Knight (2008) present a more complex but slower method for solving substitution ciphers, which incorporates constraints that model the 1-to-1 property of the key. They model the problem as optimally solving an integer linear program, a well-known NP-hard problem. The method is therefore slow, precluding the use of higher order language models. As with Knight et al. (2006), the objective function is the probability of the decipherment relative to an n-gram character language model. They test their approach on a set of 400 ciphers, 50 each of length 2, 4, 8, 16, 32, 64, 128, and 258, with separate experiments using unigram, bigram, and trigram language models (note, however, that for the experiment using a trigram model, results are not given for cipher lengths greater than 64). While we do not have access to their data set, we follow their experimental procedure as closely as possible, and compare directly to their best reported result (the trigram solver) in Section 3.4.1.

Ravi and Knight (2009) formulate the problem of unsupervised transliteration as decipherment in order to reconstruct cross-lingual phoneme mapping tables. The task is to back-transliterate names written in the Japanese Katakana script, that is,

to restore names transliterated from the Latin script into Katakana to their original form. They apply an EM-based method augmented with additional information, such as initializing favorable mappings with higher probabilities, and re-ranking results using a monolingual English corpus. The final method obtains over 50% character accuracy, as determined by normalized minimum edit distance. We apply our own decipherment system to unsupervised transliteration between Latin and Cyrillic scripts (a task which can more readily be modeled as a substitution cipher) in Section 3.4.5.

Knight et al. (2011) relate a successful decipherment of a nineteenth-century cipher known as the Copiale Cipher. The decipherment presented was achieved by combining both manual and computational techniques. In this thesis, in contrast, we pursue fully automated decipherment techniques. The method of Knight et al. (2006) was applied as a method of ciphertext language identification, as described above, and correctly identified German as the underlying language of the cipher. More accurate methods, such as that of Ravi and Knight (2008), were presumably too slow or insufficiently general for this purpose.

Moving on from Knight's work, Olson (2007) presents a method of solving monoalphabetic substitution ciphers, the same task we consider in Chapter 3. This method improves upon previous dictionary-based approaches by employing an array of selection heuristics, attempting to match ciphertext words against a word list, producing candidate solutions which are then ranked by "trigram probabilities". It is unclear how these probabilities are computed, but the resulting language model seems deficient. For example, given a ciphertext for plaintext *"it was a bright cold day in april*" (the opening of George Orwell's novel *Nineteen Eighty-Four*) the online solver[1] produces *"us far a youngs with had up about*". This instance of common words arranged in a nonsensical way is typical of the errors we saw when using this solver. The paper does not report exact error rates, only claiming that "only minor errors" are made on a test set of 21 ciphers. This method does address the same task as our own work, but we do not compare to it directly as no suitable implementation of the method was available.

[1] `http://www.quipqiup.com/index.php` (accessed December 11, 2015)

Norvig (2009) describes a method for solving substitution ciphers where word boundaries (i.e. spaces) are not preserved, using a hill-climbing method that involves both word and character language models. The two model types are only loosely combined; specifically, the word model is used to select the best solution from a small number of candidates generated using the character model, using a hill climbing algorithm with random restarts. They demonstrate near perfect results on four ciphers with lengths between 128 and 256 characters. An implementation of this solver is provided by the author, and we compare our own method to it in Section 3.4.4.

Corlett and Penn (2010) use fast heuristic A* search for decipherment, which can handle much longer ciphers than the method of Ravi and Knight (2008), while still finding the optimal solution. While the task is the same as that of Ravi and Knight (2008) (which is the same task we address in Chapter 3), the authors report results only on ciphers of at least 6000 characters, which are much easier to break than short ciphers. The ability to break shorter ciphers implies the ability to break longer ones, as a long substitution cipher can always be shortened, but the converse is not true. Our approach to this problem achieves a near-zero error rate for ciphers as short as 64 characters.

Nuhn et al. (2013) set a new state of the art for automated decipherment of substitution ciphers. Their method constructs solutions one symbol at a time, developing a tree structure where the n^{th} level of the tree deciphers the n most frequent symbols. Since this tree grows exponentially with the number of iterations, they employ beam search to prune all but the k most promising partial solutions, as determined by the probabilities of the partial decipherments assigned by a character n-gram model, for a tunable parameter k. Their method is inexact but fast, allowing them to incorporate higher-order (up to 6-gram) character language models. Their experimental setup is similar to that of Ravi and Knight (2008), with a set of 400 ciphers, 50 each of length 2, 4, 8, ..., 256. As with Ravi and Knight (2008), they do not make their data available, but do outline the procedure by which it was generated (including, importantly, the source of the text to be enciphered), allowing us to create a comparable data set. This allows us to compare our results to theirs (as

well as to Ravi and Knight (2008)), which we do in Section 3.4.1. The work presented in this thesis improves decipherment accuracy by incorporating word-level information for the generation and scoring of candidate keys.

Related to the task of identifying the language of a cipher is the task of identifying the encryption algorithm which generated the ciphertext. In this thesis, we assume that the cipher system is known to be a monoalphabetic substitution cipher, potentially with complications such as anagramming, removal of word boundaries, or the addition of noise. Nuhn and Knight (2014) work to remove such assumptions, developing a classifier which can predict what kind of cipher a given ciphertext is, using features such as symbol repetition.

Chapter 3

Solving Substitution Ciphers with Combined Language Models[1]

Monoalphabetic substitution is a well-known method of enciphering a *plaintext* by converting it into a *ciphertext* of the same length using a *key*, which is equivalent to a permutation of the alphabet. The method is elegant and easy to use, requiring only the knowledge of a key whose length is no longer than the size of the alphabet. There are over 10^{26} possible 26-letter keys, so brute-force decryption is infeasible. Manual decipherment of substitution ciphers typically starts with frequency analysis, provided that the ciphertext is sufficiently long, followed by various heuristics (Singh, 1999).

In this chapter, we investigate the task of automatically solving substitution ciphers. This task consists of recovering the plaintext from the ciphertext given only the ciphertext and a corpus representing the language of the plaintext, and is an active area of research (Ravi and Knight, 2008; Corlett and Penn, 2010; Nuhn et al., 2013). The key is a 1-1 mapping between plaintext and ciphertext alphabets, which are assumed to be of equal length (see Figure 1.1 for an example). Accurate and efficient automated decipherment can be applied to other problems, such as optical character recognition (Nagy et al., 1987), decoding web pages that utilize an unknown encoding scheme (Corlett and Penn, 2010), cognate identification (Berg-Kirkpatrick and Klein, 2011), bilingual lexicon induction (Nuhn et al., 2012), machine translation without parallel training data (Ravi and Knight, 2011),

[1] This chapter is a modified and expanded version of Hauer et al. (2014).

and archaeological decipherment of lost languages (Snyder et al., 2010).

The contribution of this chapter is a novel approach to the problem that combines both character-level and word-level language models. Previous methods often suffer from a lack of word-level information. When given a ciphertext enciphering the phrase *"it was a bright cold day in april*" from George Orwell's *Nineteen Eighty-Four*, for example, a re-implementation of the method of Ravi and Knight (2008) using a bigram character language model produces the output *"ae cor o blathe wind dof as oulan*". This is typical of the decipherments this solver produces – many common bigrams, but few words, producing a completely unintelligible solution. Even higher-order language models do not fully remedy this issue; our re-implementation of the 6-gram solver presented by Nuhn et al. (2013) produces *"it mad a knight owes say if arnie*", which consists entirely of in-vocabulary words (perhaps due to the fact that no word is longer than six letters), but the output is not a plausible decipherment, and only the words "it" and "a" are correct. Incorporating word information in a separate model does not resolve this issue either. As described in Chapter 2, the solver of Olson (2007) uses a dictionary method to generate solutions (incorporating word information), and uses a character language model as part of the ranking method, but still produces *"us far a youngs with had up about*"; the method of Norvig (2009) reverses the approach, using a character language model to generate solutions, and a word language model to select one, and produces *"ache red tab scoville magenta i*" (this solver does not use given word boundaries). Both of these have the same problem as the Nuhn et al. (2013) solver (these problems occur on shorter ciphers in general, not just on this example), producing nonsense solutions which consist entirely of in-vocabulary words. We aim to develop a solver which fully combines word and character language models in a single evaluation function, with the goal of being able to correctly solve shorter ciphers.

We formulate decipherment as a tree search problem, and find solutions with beam search, which has previously been applied to decipherment by Nuhn et al. (2013), or Monte Carlo Tree Search (MCTS), an algorithm originally designed for games, which can provide accurate solutions in less time. We compare the speed

and accuracy of both approaches. On a benchmark set of variable-length ciphers, we achieve significant improvement in terms of accuracy over the state of the art. Additional experiments demonstrate that our approach is robust with respect to the lack of word boundaries and the presence of noise. In particular, we use it to recover transliteration mappings between different scripts without parallel data, and to solve the *Gold Bug* riddle, a classic example of a substitution cipher. Finally, we investigate the feasibility of deniable encryption with monoalphabetic substitution ciphers.

This chapter is organized as follows: we describe our approach to combining character-level and word-level language models with respect to key scoring (Section 3.1), and key generation (Section 3.2). In Section 3.3, we introduce Monte Carlo Tree Search and its adaptation to decipherment. In Section 3.4, we discuss several evaluation experiments and their results. Section 3.5 is devoted to experiments in deniable encryption. Section 3.6 summarizes and concludes the chapter.

3.1 Key Scoring

Previous work tend to employ either character-level language models or dictionary-type word lists. However, word-level language models have the potential to improve the accuracy and speed of decipherment. The information gained from word n-gram frequency is often implicitly used in manual decipherment. For example, a 150-year old cipher of Edgar Allan Poe was solved only after three-letter ciphertext words were replaced with high-frequency unigrams *the*, *and*, and *not*.[2] Similarly, a skilled cryptographer might guess that a repeated 'XQ YWZ' sequence deciphers as the high-frequency bigram *"of the*". We incorporate this insight into our candidate key scoring function.

On the other hand, our character-level language model helps guide the initial stages of the search process, when few or no words are discernible, towards English-like letter sequences. In addition, if the plaintext contains out-of-vocabulary (OOV) words, which do not occur in the training corpus, the character model will favor

[2] `http://www.newswise.com/articles/edgar-allen-poe-cipher-solved`

pronounceable letter sequences. For example, having identified most of the words in plaintext *"village of XeYoviY and burned it*", our solver selects *pecovic* as the highest scoring word that fits the pattern, which in fact is the correct solution.

In order to assign a score to a candidate key, we apply the key to the ciphertext, and compute the probability of the resulting letter sequence using a combined language model that incorporates both character-level and word-level information. With unigram, bigram, and trigram language models over both words and characters trained on a large corpus, n-gram models of different orders are combined by deleted interpolation (Jelinek and Mercer, 1980). The smoothed word trigram probability $\hat{P}$ is:

$$\hat{P}(w_k|w_{k-2}w_{k-1}) = \lambda_1 P(w_k) + \lambda_2 P(w_k|w_{k-1}) + \lambda_3 P(w_k|w_{k-2}w_{k-1}),$$

such that the λs sum to 1. The linear coefficients are determined by successively deleting each trigram from the training corpus and maximizing the likelihood of the rest of the corpus (Brants, 2000). The probability of text $s = w_1, w_2, \ldots, w_n$ according to the smoothed word language model is:

$$P_W(s) = P(w_1^n) = \prod_{k=1}^{n} \hat{P}(w_k|w_{k-2}w_{k-1}).$$

The unigram, bigram, and trigram character language models are combined in a similar manner to yield $P_C(s)$. The final score is then computed as a linear combination of the log probabilities returned by both character and word components:

$$\text{score}(s) = \chi \log P_C(s) + (1 - \chi) \log P_W(s),$$

with the value of χ optimized on a development set. The score of a key is taken to be the score of the decipherment that it produces.

The handling of the OOV words is an important feature of the key scoring algorithm. An incomplete decipherment typically contains many OOV words, which according to the above equations would result in probability $P_W(s)$ being zero. In order to avoid this problem, we replace all OOV words in a decipherment with a special UNKNOWN token for the computation of $P_W(s)$. Prior to deriving the word language models, a sentence consisting of a single UNKNOWN token is appended

to the training corpus. As a result, word n-grams that include an UNKNOWN token are assigned very low probability, encouraging the solver to favor decipherments containing fewer OOV words.

3.2 Key Mutation

The process of generating candidate keys can be viewed as constructing a search tree, where a modified key is represented as a child of an earlier key. The root of the tree contains the initial key, which is generated according to simple frequency analysis (i.e., by mapping the nth most common ciphertext character to the nth most common character in the training corpus). We repeatedly spawn new tree leaves by modifying the keys of current leaves, while ensuring that each node in the tree has a unique key. The fitness of each new key is evaluated by scoring the resulting decipherment, as described in Section 3.1. At the end of computation, we return the key with the highest score as the solution.

There are an exponential number of possible keys, so it is important to generate new keys that are likely to achieve a higher score than the current key. We exploit this observation: any word n-gram can be represented as a pattern, or sequence, of repeated letters (Table 3.1). We identify the pattern represented by each word n-gram in the ciphertext, and find a set of *pattern-equivalent* n-grams from the training corpus. For each such n-gram, we generate a corresponding new key from the current key by performing a sequence of transpositions.

Pattern	p-equivalent n-grams
ABCD	said, from, have
ABCC	will, jazz, tree
ABCA	that, says, high
ABCD EFG	from you, said the
ABCA ABD	that the, says sam
ABC DEEFGBCHICG	the bookshelves

Table 3.1: Examples of pattern-equivalent n-grams.

Pattern-equivalence (abbreviated as p-equivalence) induces an equivalence relation between n-grams (Moore et al., 1999). Formally, two n-grams u and v are

p-equivalent ($u \overset{p}{\equiv} v$) if and only if they satisfy the following three conditions, where ␣ stands for the space character:

1. $|u| = |v|$
2. $\forall i$: $u_i = \text{␣} \Leftrightarrow v_i = \text{␣}$
3. $\forall i, j$: $u_i = u_j \Leftrightarrow v_i = v_j$

For example, consider ciphertext 'ZXCZ ZXV'. Adopting *"that"*, which is p-equivalent to 'ZXCZ', as a temporary decipherment of the first word, we generate a new key in which Z maps to *t*, X to *h*, and C to *a*. This is accomplished by three letter-pair transpositions in the parent key, producing a child key where 'ZXCZ' deciphers to *"that"*. Further keys are generated by matching 'ZXCZ' to other p-equivalent words, such as *"says"* and *"high"*. The process is repeated for the second word 'ZXV', and then for the entire bigram 'ZXCZ ZXV'. Each such match induces a series of modifications to the leaf, resulting in a new key. This example is illustrated in Figure 3.1. The general leaf expansion algorithm is summarized in Figure 3.2.

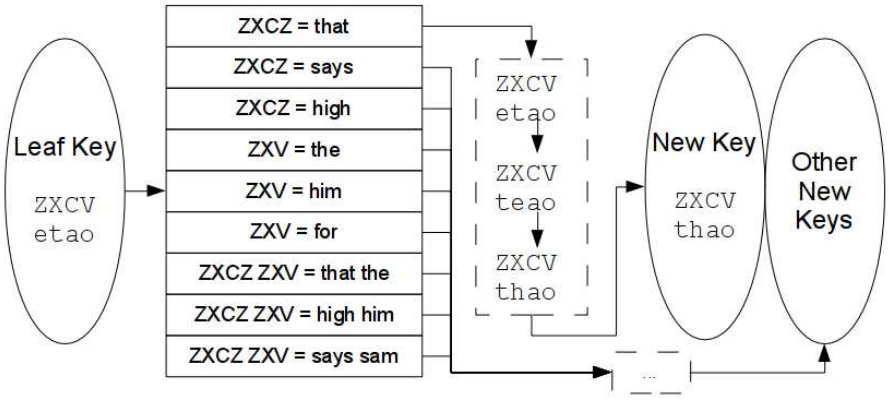

Figure 3.1: Expanding a leaf through substitution of p-equivalent n-grams.

In order to avoid spending too much time expanding a single node, we limit the number of replacements for each n-gram in the current decipherment to the k most promising candidates, where k is a parameter optimized on a development set. Note that n-grams excluded in this way may still be included as part of a higher-order

```
1: function EXPAND(Leaf, CipherText)
2:     for all word n-grams w in CipherText do
3:         for k best w' s.t. w' ≡^p w do
4:             NewLeaf = Modify(Leaf, w ↦ w')
5:             if NewLeaf not in the tree then
6:                 add NewLeaf as a child of Leaf
7:             if score(NewLeaf) > score(BestLeaf)
   then
8:                 BestLeaf = NewLeaf
9:     return BestLeaf
```

Figure 3.2: Leaf expansion.

n-gram. For example, if the word *birddog* is omitted in favor of more promising candidates, it might be considered as a part of the bigram *struggling birddog*.

Two distinct modes of ranking the candidate n-grams are used throughout the solving process. In the initial stage, n-grams are ranked according to the score computed using the method described in Section 3.1. Thus, the potential replacements for a given ciphertext n-gram are the highest scoring p-equivalent n-grams from the training corpus regardless of the form of the decipherment implied by the current key. Afterwards, candidates are ranked according to their Hamming distance to the current decipherment, with score used only to break ties. This two-stage approach is designed to exploit the fact that the solver typically gets closer to the correct decipherment as the search progresses.

3.3 Tree Search

Nuhn and Ney (2013) show that finding the optimal decipherment with respect to a character bigram model is NP-hard. Since our scoring function incorporates a language model score, choosing an appropriate tree search technique is crucial in order to minimize the number of search errors, where the score of the returned solution is lower than the score of the actual plaintext. In this section we describe two search algorithms: an adaptation of Monte Carlo Tree Search (MCTS), and a version of beam search.

3.3.1 Monte Carlo Tree Search

MCTS is a search algorithm for heuristic decision making. Starting from an initial state that acts as the root node, MCTS repeats these four steps: (1) **selection** – starting from the root, recursively pick a child until a leaf is reached; (2) **expansion** – add a set of child nodes to the leaf; (3) **simulation** – simulate the evaluation of the leaf node state; (4) **backpropagation** – recursively ascend to the root, updating the simulation result at all nodes on this path. This process continues until a state is found which passes a success threshold, or time runs out.

```
1: Root contains InitialKey
2: for m iterations do
3:     recursively select optimal Path from Root
4:     Leaf = last node of Path
5:     BestLeaf = EXPAND(Leaf, CipherText)
6:     append BestLeaf to Path
7:     Max = Path node with the highest score
8:     assign score of Max to all nodes in Path
```

Figure 3.3: MCTS for decipherment.

Previous work with MCTS has focused on board games, including Hex (Arneson et al., 2010) and Go (Enzenberger et al., 2010), but it has also been employed for problems unrelated to game playing (Previti et al., 2011). Although originally designed for two-player games, MCTS has also been applied to single-agent search (Browne et al., 2012). Inspired by such single-agent MCTS methods (Schadd et al., 2008; Matsumoto et al., 2010; Méhat and Cazenave, 2010), we frame decipherment as a single-player game with a large branching factor, in which the simulation step is replaced with a heuristic scoring function. Since we have no way of verifying that the current decipherment is correct, we stop after performing m iterations. The value of m is determined on a development set.

The function commonly used for comparing nodes in the tree is the upper-confidence bound (UCB) formula for single-player MCTS (Kocsis and Szepesvári, 2006). The formula augments our scoring function from Section 3.1 with an additional term:

$$UCB(n) = \text{score}(n) + C\sqrt{\frac{ln(v(p(n)))}{v(n)}}$$

where $p(n)$ is the parent of node n, and $v(n)$ is the number of times that n has been visited. The second term favors nodes that have been visited relatively infrequently in comparison with their parents. The value of C is set on a development set.

Figure 3.3 summarizes our implementation. Each iteration begins by finding a path through the tree that is currently optimal according to the UCB. The path begins at the root, includes a locally optimal child at each level, and ends with a leaf. The leaf is expanded using the function EXPAND shown in Figure 3.2. The highest-scoring of the generated children is then appended to the optimal path. If the score of the new leaf (*not* the UCB) is higher than the score of its parent, we backpropagate that score to all nodes along the path leading from the root. This encourages further exploration along all or part of this path.

3.3.2 Beam Search

Beam search is a tree search algorithm that uses a size-limited list of nodes currently under consideration, which is referred to as the beam. If the beam is full, a new node can be added to it only if it has a higher score than at least one node currently in the beam. In such a case, the lowest-scoring node is removed from the beam and any further consideration.

Nuhn et al. (2013) use beam search for decipherment in their character-based approach. Starting from an empty root node, a partial key is extended by one character in each iteration, so that each level of the search tree corresponds to a unique ciphertext symbol. The search ends when the key covers the entire ciphertext.

By contrast, we apply beam search at the word n-gram level. The EXPAND subroutine defined in Figure 3.2 is repeatedly invoked for a specified number of iterations (a tunable parameter). In each iteration, the algorithm analyzes a set of word n-gram substitutions, which may involve multiple characters, as described in Section 3.2. The search stops early if the beam becomes empty. On short ciphers (32 characters or less), the best solution is typically found within the first five iterations, but this can only be confirmed after the search process is completed.

3.4 Experiments

In order to evaluate our approach and compare it to previous work, we conducted several experiments. We created three test sets of variable-length ciphers: (1) with spaces, (2) without spaces, and (3) with spaces and added encipherment noise. In addition, we tested our system on Serbian Cyrillic, and the Gold Bug cipher.

We derive our English language models from a subset of the New York Times corpus (LDC2003T05) containing 17M words. From the same subset, we obtain letter-frequency statistics, as well as the lists of p-equivalent n-grams. For comparison, Ravi and Knight (2008) use 50M words, while Nuhn et al. (2013) state that they train on a subset of the Gigaword corpus without specifying its size.

3.4.1 Substitution Ciphers

Following Ravi and Knight (2008) and Nuhn et al. (2013), we test our approach on a benchmark set of ciphers of lengths, 2, 4, 8, ..., 256, where each length is represented by 50 ciphers. The plaintexts are randomly extracted from the Wikipedia article on *History*, which is quite different from our NYT training corpus. For in-domain testing, we also extract a second set of ciphers from a held-out portion of the NYT corpus. Spaces are preserved, and the boundaries of the ciphers match word boundaries.

Figure 3.4 shows the decipherment error rate of the beam-search version of our algorithm vs. the published results of the best-performing variants of Ravi and Knight (2008) and Nuhn et al. (2013): letter 3-gram and 6-gram, respectively. The decipherment error rate is defined as the ratio of the number of incorrectly deciphered characters to the length of the plaintext. Our approach achieves a statistically significant improvement on ciphers of length 8 and 16. Shorter ciphers are inherently hard to solve, while the error rates on longer ciphers are close to zero. Unfortunately, Nuhn et al. (2013) only provide a graph of their error rates, which in some cases prevents us from confirming the statistical significance of the improvements (c.f. Table 3.2).

Examples of decipherment errors are shown in Table 3.3. As can be seen, the

proposed plaintexts are often perfectly reasonable given the cipher letter pattern. The solutions proposed for very short ciphers are usually high-frequency words; for example, the 2-letter ciphers matching the pattern 'AB' are invariably deciphered as *"of"*. The errors in ciphers of length 32 or more tend to be confined to individual words, which are often OOV names.

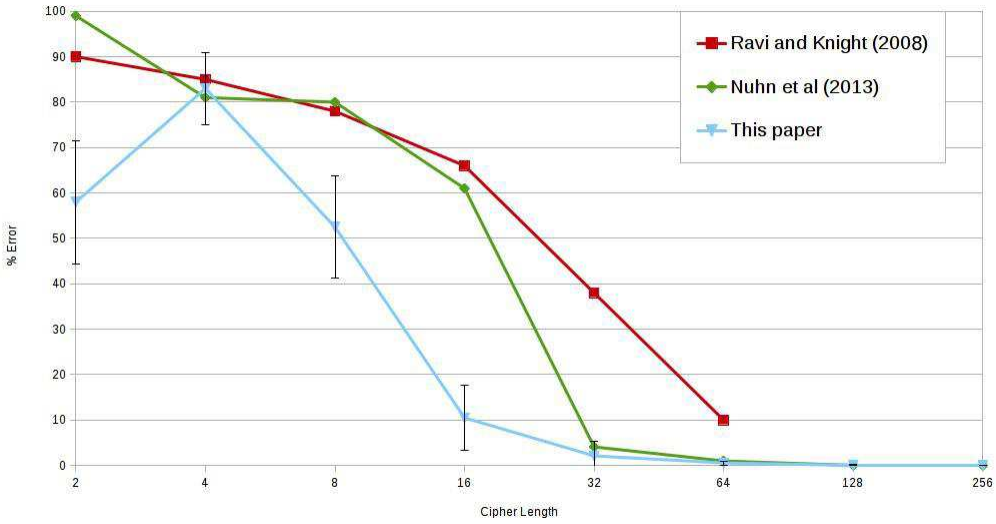

Figure 3.4: Average decipherment error rate as a function of cipher length on the Wikipedia test set.

	Wikipedia			NYT				
	with spaces			with spaces		no spaces	noisy	
	Beam	MCTS	Greedy	Beam	MCTS	MCTS	Beam	MCTS
2	58.00	58.00	58.00	81.00	81.00	75.00	83.00	83.00
4	83.00	83.00	83.00	66.00	66.00	77.50	83.50	83.50
8	52.50	52.50	52.50	49.00	49.00	55.71	73.50	73.50
16	10.50	12.62	18.50	13.50	14.50	55.00	69.75	69.38
32	2.12	6.12	10.88	0.88	0.94	28.57	46.81	50.44
64	0.56	0.72	2.50	0.03	0.03	7.85	16.66	25.47
128	0.14	0.16	0.16	0.00	1.61	0.87	5.20	5.41
256	0.00	0.00	0.10	0.02	0.02	0.00	2.73	2.75

Table 3.2: Average decipherment error rate of our solver as a function of cipher length on the Wikipedia and the NYT test sets.

Cipher length	Cipher pattern	Actual plaintext	Decipherment
2	AB	to	of
4	ABCD	from	said
4	ABBC	look	been
8	ABCDCEFG	slobodan	original
8	ABCDE FG	filed by	would be
16	ABCCDEE BFG HBCI	jarrett and mark	carroll and part
16	ABCDE FGCHA IJKL	group along with	drugs would make

Table 3.3: Examples of decipherment errors.

3.4.2 Beam Search vs. MCTS

The error rates of the two versions of our algorithm are very close, with a few exceptions (Table 3.2). Out of 400 ciphers with spaces in the Wikipedia test set, the MCTS variant correctly solves 260 out of 400 ciphers, compared to 262 when beam search is used. In 9 MCTS solutions and 3 beam search solutions, the score of the proposed decipherment is lower than the score of the actual plaintext, which indicates a search error.

By setting the beam size to one, or the value of C in MCTS to zero, the two search techniques are reduced to *greedy* search. As shown in Table 3.2, in terms of accuracy, greedy search is worse than MCTS on the lengths of 16, 32, and 64, and roughly equal on other lengths. This suggests that an intelligent search strategy is important for obtaining the best results.

In terms of speed, the MCTS version outperforms beam search, thanks to a smaller number of expanded nodes in the search tree. For example, it takes on average 9 minutes to solve a cipher of length 256, compared to 41 minutes for the beam search version. Direct comparison of the execution times with the previous work is difficult because of variable computing configurations, as well as the unavailability of the implementations. However, on ciphers of the length of 128, our MCTS version takes on average 197 seconds, which is comparable to 152 seconds reported by Nuhn et al. (2013), and faster than our reimplementation of the bigram solver of Ravi and Knight (2008) which takes on average 563 seconds. The trigram solver of Ravi and Knight (2008) is even slower, as evidenced by the fact that they report no corresponding results on ciphers longer than 64 letters.

3.4.3 Noisy Ciphers

Previous work has generally focused on noise-free ciphers. However, in real-life applications, we may encounter cases of imperfect encipherment, in which some characters are incorrectly mapped. Corlett and Penn (2010) identify the issue of noisy ciphers as a worthwhile future direction. Adding noise also increases a cipher's security, as it alters the pattern of letter repetitions in words. In this section, we evaluate the robustness of our approach in the presence of noise.

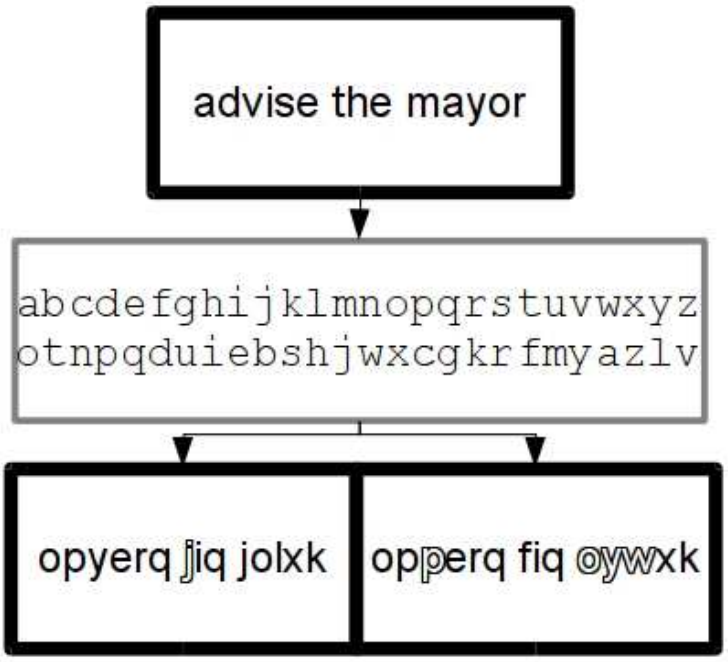

Figure 3.5: Encrypting a plaintext to create two ciphers with varying levels of noise. The highlighted letters deviate randomly from the key, indicating noise in the encipherment process.

In order to quantify the effect of adding noise to ciphers, we randomly corrupt $log_2(n)$ of the ciphertext letters, where n is the length of the cipher. Our results on such ciphers are shown numerically in full in Table 3.2 and summarized graphically in Figure 3.6. As expected, adding noise to the ciphertexts increases the error rate in comparison with ciphers without noise. However, our algorithm is still able to break most of the ciphers of length 64 and longer, and makes only occasional mistakes

on ciphers of length 256. Beam search is substantially better than MCTS only on lengths of 32 and 64. These results indicate that our word-oriented approach is reasonably robust with respect to the presence of noise.

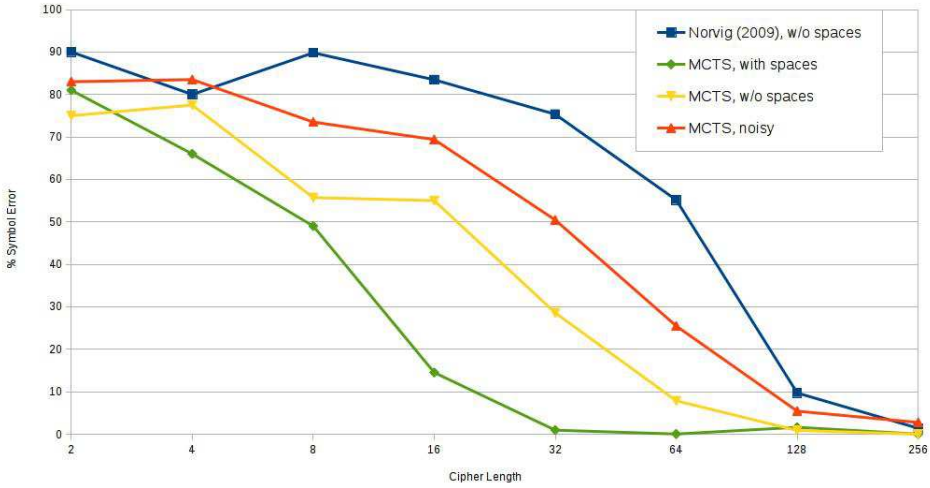

Figure 3.6: Selected average decipherment error rates as a function of cipher length on noisy ciphers and ciphers without spaces from the NYT test set.

3.4.4 Ciphers Without Spaces

Removing spaces that separate words prior to encipherment is another way of increasing the security of a cipher. The assumption is that the intended recipient, after applying the key, will still be able to guess the location of word boundaries, and recover the meaning of the message. We are interested in testing our approach on such ciphers, but since it is dependent on word language models, we need to first modify it to identify word boundaries. In particular, the two components that require word boundaries are the scoring function (Section 3.1), and the search tree node expansion (Section 3.2).

In order to compute the scoring function, we try to infer word boundaries in the current decipherment using the following simple greedy algorithm. The current decipherment is scanned repeatedly from left to right in a search for words of length L, where L gradually decreases from the length of the longest word in the training corpus, down to the minimal value of 2. If a word is found, the process is applied recursively to both remaining parts of the ciphertext. We use a fast greedy search

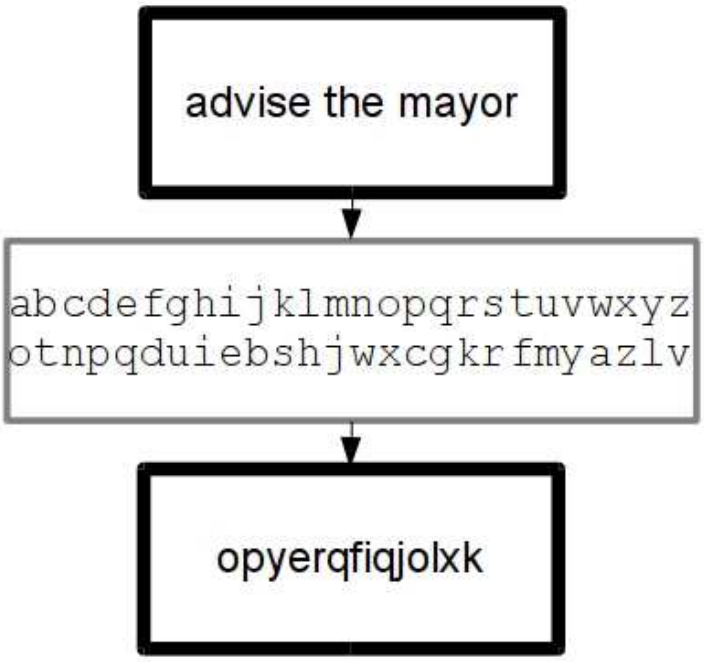

Figure 3.7: Encrypting a plaintext to create a cipher without spaces.

instead of a slower but more accurate dynamic programming approach as this search must be executed each time a key is evaluated.

In the search tree node expansion step, for each substring of length at least 2 in the current decipherment, we attempt to replace it with all pattern-equivalent n-grams (with spaces removed), for n from 1 to 3. Substrings up to the length of the longest such n-gram in the training corpus are considered. As a result, each key spawns a large number of children, increasing both time and memory usage. Overall, the modified algorithm is as much as a hundred times slower than the original algorithm. However, when MCTS is used as the search method, we are still able to perform the decipherment in reasonable time.

For testing, we remove spaces from both the plaintexts and ciphertexts, and reduce the number of ciphers to 10 for each cipher length. Our results, shown graphically in Figure 3.6 and numerically in Table 3.2 compare favorably to the solver of (Norvig, 2009), which is designed to work on ciphers without spaces.

The final test of our decipherment algorithm is the cipher from *The Gold Bug* by

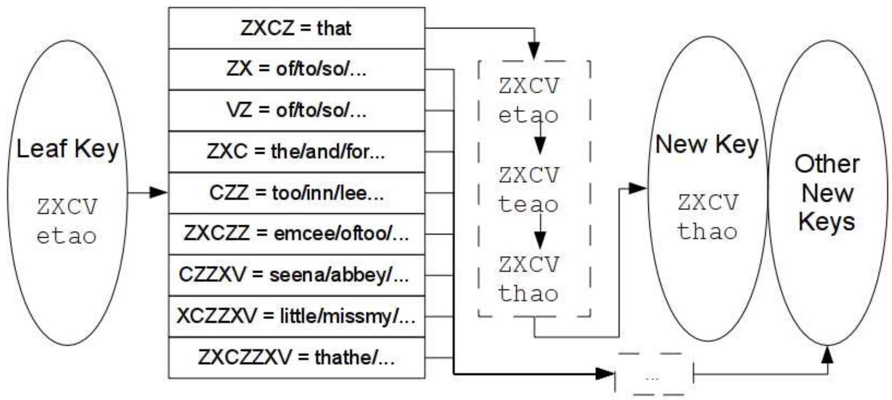

Figure 3.8: Expanding a leaf when the ciphertext has no spaces. The ciphertext is ZXCZZXV. Note that 2-grams and 3-grams also have spaces removed, i.e "of too" becomes "oftoo".

Edgar Alan Poe. In that story, the 204-character cipher gives the location of hidden treasure. Our implementation finds a completely correct solution, the beginning of which is shown in Table 3.4. Both experiments reported in this section confirm that our word-based approach works well even when spaces are removed from ciphers.

5 3 ‡ ‡ † 3 0 5)) 6 * ; 4 8 2 6) 4 ‡ .) 4 ‡) ; 8 0 6 * ; 4 8 † 8 ¶ 6 0)) 8 5 ;	
a g o o d g l a s s i n t h e b i s h o p s h o s t e l i n t h e d e v i l s s e a t	

Table 3.4: The beginning of the Gold Bug cipher and its decipherment.

3.4.5 Unsupervised Transliteration

We further test the robustness of our approach by performing an experiment in which unsupervised transliteration – the phonetic conversion of a name or other untranslated word into a different writing system- - is framed as decipherment of an unknown script. Supervised transliteration, where pairs of words are provided in both scripts, has been the subject of substantial prior work (see, for example, Bhargava et al. (2011) or Nicolai et al. (2015)). In unsupervised transliteration, we are given only a corpus of text written in the language of the target script, with no transliteration pairs given.

For this experiment, we selected Croatian and Serbian, two closely related languages that are written in different scripts (Latin and Cyrillic). The correspondence

between the two script alphabets is not exactly one-to-one: Serbian Cyrillic uses 30 symbols, while Croatian Latin uses 27. In particular, the Cyrillic characters љ, њ, and џ are represented in the Latin script as digraphs *lj*, *nj*, and *dž*. In addition, there are differences in lexicon and grammar between the two languages, which make this task a challenging case of noisy encipherment.

In the experiment, we treat a short text in Serbian as enciphered Croatian and attempt to recover the key, which in this case is the mapping between the characters in the two writing scripts. Each letter with a diacritic is considered as different from the same letter with no diacritic. We derive the word and character language models from the Croatian part of the ECI Multilingual Corpus, which contains approximately 720K word tokens. For testing, we use a 250-word, 1583-character sample from the Serbian version of the Universal Declaration of Human Rights.

Table 3.5: Serbian Cyrillic deciphered as Croatian. The decipherment errors are shown in boldface.	

The decipherment error rate on the Serbian ciphertext drops quickly, leveling at about 3% at the length of 50 words (Figure 3.9). The residual error rate reflects the lack of correct mapping for the three Serbian letters mentioned above. As can be seen in Table 3.5, the actual decipherment of a 30-word ciphertext contains only a handful of isolated errors. On the other hand, a pure frequency-based approach fails on this task with a mapping error rate close to 90%.

3.5 Deniable Encryption

In one of Stanisław Lem's novels, military cryptographers encipher messages in such a way that the ciphertext appears to be plain text (Lem, 1973). Canetti et al. (1997) investigate a related idea, in which the ciphertext "looks like" an encryption of a plaintext that is different from the real message. In the context of monoalphabetic substitution ciphers, we define the task as follows: given a message, find an encipherment key yielding a ciphertext that resembles natural language text. For

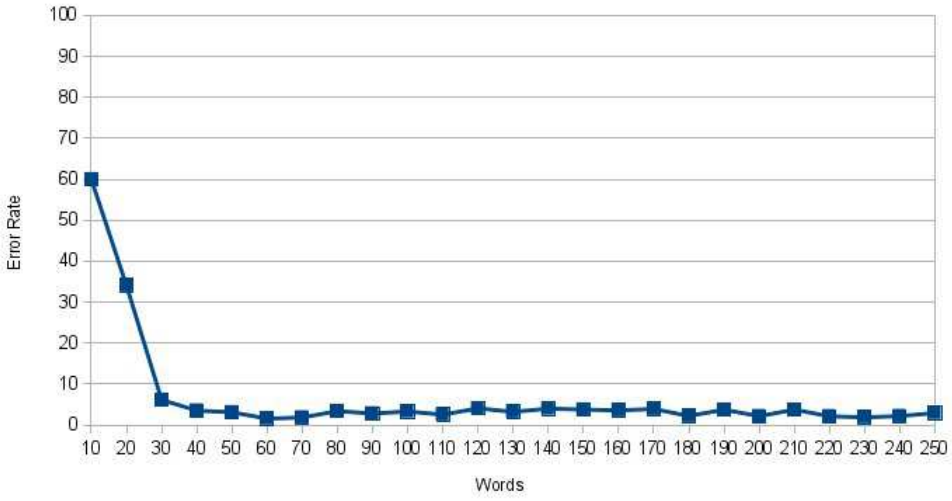

Figure 3.9: Decipherment error rate on a Serbian sample text as a function of the ciphertext length.

example, *"game with planes"* is a deniable encryption of the message *"take your places"* (the two texts are p-equivalent).

We applied our solver to a set of sentences from the text of *Nineteen Eighty-Four*, treating each sentence as a ciphertext. In order to ensure that the alternative plaintexts are distinct from the original sentences, we modified our solver to disregard candidate keys that yield a solution containing a content word from the input. For example, *"fine hours"* was not deemed an acceptable deniable encryption of *"five hours"*. With this condition added, alternative plaintexts were produced for all 6531 sentences. Of these, 1464 (22.4%) were determined to be composed entirely of words seen in training. However, most of these deniable encryptions were either non-grammatical or differed only slightly from the actual plaintexts. It appears that substitution ciphers that preserve spaces fail to offer sufficient flexibility for finding deniable encryptions.

In the second experiment, we applied our solver to a subset of 757 original sentences of length 32 or less, with spaces removed. The lack of spaces allows for more flexibility in finding deniable encryptions. For example, the program finds *"draft a compromise"* as a deniable encryption of *"zeal was not enough"*. None of the produced texts contained out-of-vocabulary words, but most were still ungrammat-

ical or nonsensical. Allowing for some noise to be introduced into the one-to-one letter mapping would likely result in more acceptable deniable encryptions, but our current implementation can handle noise only on the input side.

3.6 Summary

We have presented a novel approach to the decipherment of monoalphabetic substitution ciphers that combines character and word-level language models. We have proposed Monte Carlo Tree Search as a fast alternative to beam search on the decipherment task. Our experiments demonstrate significant improvement over the current state of the art. Additional experiments show that our approach is robust in handling ciphers without spaces, and ciphers with noise, including the practical application of recovering transliteration mappings between Serbian and Croatian.

Chapter 4

Decoding Anagrammed Texts Written in an Unknown Language and Script[1]

The Voynich manuscript is a medieval codex[2] consisting of 240 pages written in a unique script, which has been referred to as the world's most important unsolved cipher (Schmeh, 2013). The type of cipher that was used to generate the text is unknown; a number of theories have been proposed, including substitution and transposition ciphers, an *abjad* (a writing system in which vowels are not written), steganography, semi-random schemes, and an elaborate hoax. However, the biggest obstacle to deciphering the manuscript is the lack of knowledge of what language it represents.

Identification of the underlying language has been crucial for the decipherment of ancient scripts, including Egyptian hieroglyphics (Coptic), Linear B (Greek), and Mayan glyphs (Ch'olti'). On the other hand, the languages of many undeciphered scripts, such as Linear A, the Indus script, and the Phaistos Disc, remain unknown (Robinson, 2002). Even the order of characters within text may be in doubt; in Egyptian hieroglyphic inscriptions, for instance, the symbols were sometimes rearranged within a word in order to create a more aesthetically-pleasing inscription (Singh, 1999). Another complicating factor is the omission of vowels in some writing systems.

[1]This chapter is a modified and expanded version of Hauer and Kondrak (2016).

[2]The manuscript was radiocarbon dated to 1404-1438 AD in the Arizona Accelerator Mass Spectrometry Laboratory (*http://www.arizona.edu/crack-voynich-code*, accessed Nov. 20, 2015).

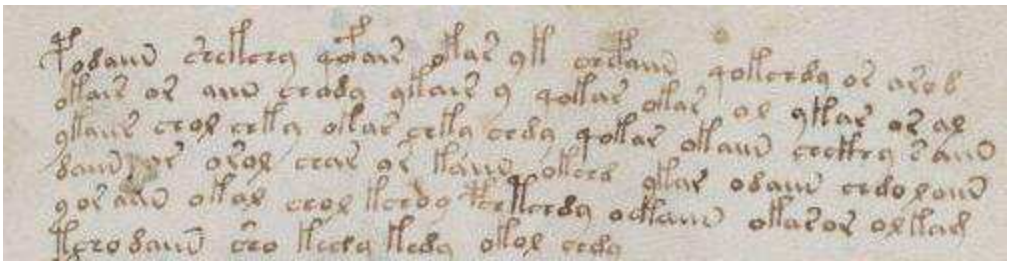

Figure 4.1: A sample from the Voynich manuscript.

Applications of ciphertext language identification extend beyond secret ciphers and ancient scripts. Nagy et al. (1987) frame optical character recognition as a decipherment task. Knight et al. (2006) note that for some languages, such as Hindi, there exist many different and incompatible encoding schemes for digital storage of text; the task of analyzing such an arbitrary encoding scheme can be viewed as a decipherment of a substitution cipher in an unknown language. Similarly, the unsupervised derivation of transliteration mappings between different writing scripts lends itself to a cipher formulation (Ravi and Knight, 2009).

The Voynich manuscript is written in an unknown script that encodes an unknown language, which is considered the most challenging type of a decipherment problem (Robinson, 2002, p. 46). Inspired by the mystery of both the Voynich manuscript and the undeciphered ancient scripts, we develop a series of algorithms for the purpose of decrypting unknown alphabetic scripts representing unknown languages. We assume that symbols in scripts which contain no more than a few dozen unique characters roughly correspond to phonemes of a language, and model them as monoalphabetic substitution ciphers. We further allow that an unknown transposition scheme could have been applied to the enciphered text, resulting in arbitrary scrambling of letters within words (*anagramming*). Finally, we consider the possibility that the ciphertext script is an abjad, in which only consonants are represented in writing.

Our decryption system is composed of three steps. The first task is to identify the language of a ciphertext, by comparing it to samples representing known languages. The second task is to map each symbol of the ciphertext to the corresponding letter in the identified language. The third task is to decode the resulting

anagrams into readable text, which may involve the recovery of unwritten vowels.

The chapter is structured as follows: We discuss prior work on the VMS in Section 4.1. In Section 4.2, we propose three methods for the source language identification of texts enciphered with a monoalphabetic substitution cipher. Section 4.3 presents and evaluate our approach to the decryption of texts composed of enciphered anagrams. Section 4.4 is dedicated to applying our new techniques to the Voynich manuscript. Section 4.5 summarizes and concludes the chapter.

4.1 Prior Work on The Voynich Manuscript

Since the discovery of the Voynich manuscript (henceforth referred to as the VMS), there have been a number of decipherments claims. Newbold and Kent (1928) proposed an interpretation based on microscopic details in the text, which was subsequently refuted by Manly (1931). Other claimed decipherments by Feely (1943) and Strong (1945) have also been refuted (Tiltman, 1968). A detailed study of the manuscript by d'Imperio (1978) details various other proposed solutions and the arguments against them. A general theme across claimed decipherments is the extreme flexibility of the methods, both in how they are produced, and in the interpretation of the results. As noted by Manly (1931) and d'Imperio (1978), the method of Newbold and Kent (1928) is so flexible and allows for so many decisions to be made arbitrarily, that numerous possible decipherments, including rhyming poems, can be produced depending according whims of the decipherer, making the claimed decipherment completely meaningless. Tiltman (1968) notes that the decipherment of Feely (1943) "produced text in unacceptable medieval Latin, in unauthentic abbreviated forms." Such decipherments are not falsifiable, and cannot be systematically reproduced. We do not claim to have deciphered any part of the VMS, instead focusing primarily on analysis of the text, with the goal of identifying the underlying language of the text. We will work from a specific, well-defined set of assumptions, which restrict what operations may be carried out on the text, and which we will justify prior to our analysis.

Numerous languages have been proposed to underlie the VMS. The properties

and the dating of the manuscript suggest Latin and Italian as potential candidates. Jaskiewicz (2011) presents a method for ciphertext language identification using character frequency statistics and a specially-designed weighted distribution distance function. A list of the five most likely candidates according to this method includes Moldavian and Thai. We will compare this method to our own work in Section 4.2.4, and show that our own methods, which point to completely different languages, are more accurate at ciphertext language identification.

Reddy and Knight (2011) give a comprehensive overview of the current state of VMS research, as well as analyzing the VMS using multiple statistical techniques. Different methods of analysis reveal similarities to different languages. For example, they discover an excellent match between the VMS and Quranic Arabic in the distribution of word type lengths, but find that the predictability of letters within words is much higher in the VMS than in Arabic. Chinese Pinyin exhibits the opposite trend, with very similar letter predictability, but a radically different distribution of word lengths. The question of what language the VMS was written in, prior to encipherment, is a significant motivation for the work in this chapter, which we intend to explore using the ciphertext language identification methods we present.

It has been suggested previously that some anagramming scheme may alter the sequence order of characters within words in the VMS. Rugg (2004) notes the apparent similarity of the VMS to a text in which each word has had its letters sorted into alphabetical order, producing a text consisting of *alphagrams* (but qualifies this by saying that such a text would lack properties the VMS possesses), referencing the highly repetitive nature of the text, and the usually (for a natural language text) regular word structure. Reddy and Knight (2011) show that the letter sequences are generally more predictable than in natural languages, which could easily be explained (along with the aforementioned repetition and regularity) by a systematic anagramming scheme.

Some researchers have argued that the VMS may be an elaborate hoax created to only appear as a meaningful text. Rugg (2004) suggests a tabular method, similar to the sixteenth century technique of the Cardan grille, although recent dating of

the manuscript to the fifteenth century, well before the Cardan grille method was invented, provides evidence to the contrary. Schinner (2007) uses analysis of random walk techniques and textual statistics to support the hoax hypothesis.

This leads to a vital question: does the text of the VMS originate from some natural language, enciphered or otherwise? Attempts to identify the language behind the text must naturally assume that there is a language to identify, and similarly, attempts to decipher the document must assume that there is some concealed plaintext to recover. There is substantial evidence to support this assumption. Landini (2001) identifies in the VMS language-like statistical properties, such as Zipf's law, which were only discovered in the last century, while carbon dating has shown that the VMS was created in the 15^{th} century, making it unlikely that such features could have been generated artificially. Reddy and Knight (2011) report that the VMS exhibits other properties characteristic of natural language text, such as evidence that the text contains morphological structure, and that the pages and sections of the document appear to have consistent topics. Montemurro and Zanette (2013) find further evidence of linguistic information and structure in the VMS. They use information theoretic techniques to find long-range relationships between words and sections of the manuscript, as well as relationships between the text and the illustrations. This is strong evidence for the claim that the VMS text represents an enciphered natural language, as it is unlikely that 15^{th} century scribes would even be aware of these properties (information theory having only emerged in the $20th$ century), much less be able to work them consistently into the text. That same paper finds relationships between the text and the figures in the VMS. Given this evidence, the premise that the VMS's unique script enciphers a text written in a natural language, is entirely reasonable.

4.2 Source Language Identification

In this section, we propose and evaluate three methods for determining the source language of a document enciphered with a monoalphabetic substitution cipher. We frame it as a classification task, with the classes corresponding to the candidate

languages, which are represented by short sample texts. The methods are based on:

1. relative character frequencies,
2. patterns of repeated symbols within words,
3. the outcome of a trial decipherment.

4.2.1 Character Frequency

An intuitive way of guessing the source language of a ciphertext is by character frequency analysis. The key observation is that the relative frequencies of symbols in the text are unchanged after encipherment with a 1-to-1 substitution cipher. The idea is to order the ciphertext symbols by frequency, normalize these frequencies to create a probability distribution, and choose the closest matching distribution from the set of candidate languages.

More formally, let P_T be a discrete probability distribution where $P_T(i)$ is the probability of a randomly selected symbol in a text T being the i^{th} most frequent symbol. We define the distance between two texts U and V to be the Bhattacharyya (1943) distance between the probability distributions P_U and P_V:

$$d(U, V) = -\ln \sum_i \sqrt{P_U(i) \cdot P_V(i)}$$

The advantages of this distance metric include its symmetry, and the ability to account for events that have a zero probability (in this case, due to different alphabet sizes). The language of the closest sample text to the ciphertext is considered to be the most likely source language. This method is not only fast but also robust against letter reordering and the lack of word boundaries.

4.2.2 Decomposition Pattern Frequency

Our second method expands on the character frequency method by incorporating the notion of decomposition patterns. This method uses multiple occurrences of individual symbols within a word as a clue to the language of the ciphertext. For example, the word *seems* contains two instances of 's' and 'e', and one instance of

'm'. We are interested in capturing the relative frequency of such patterns in texts, independent of the symbols used.

Formally, we define a function f that maps a word to an ordered n-tuple $(t_1, t_2, \ldots t_n)$, where $t_i \geq t_j$ if $i < j$. Each t_i is the number of occurrences of the i^{th} most frequent character in the word. For example, $f(\text{seems}) = (2, 2, 1)$, while $f(\text{beams}) = (1, 1, 1, 1, 1)$. We refer to the resulting tuple as the *decomposition pattern* of the word. The decomposition pattern is unaffected by monoalphabetic letter substitution or anagramming. As with the character frequency method, we define the distance between two texts as the Bhattacharyya distance between their decomposition pattern distributions, and classify the language of a ciphertext as the language of the nearest sample text.

It is worth noting that this method requires word separators to be preserved in the ciphertext. In fact, the effectiveness of the method comes partly from capturing the distribution of word lengths in a text. On the other hand, the decomposition patterns are independent of the ordering of characters within words. We will take advantage of this property in Section 4.3.

4.2.3 Trial Decipherment

The final method that we present involves deciphering the document in question into each candidate language. The decipherment is performed with a fast *greedy-swap* algorithm, which is related to the algorithms of Ravi and Knight (2008) and Norvig (2009). It attempts to find the key that maximizes the probability of the decipherment according to a bigram character language model derived from a sample document in a given language. The decipherment with the highest probability indicates the most likely plaintext language of the document.

The greedy-swap algorithm is shown in Figure 4.2. The initial key is created by pairing the ciphertext and plaintext symbols in the order of decreasing frequency, with null symbols appended to the shorter of the two alphabets. The algorithm repeatedly attempts to improve the current key k by considering the "best" swaps of ciphertext symbol pairs within the key (if the key is viewed as a permutation of the alphabet, such a swap is a transposition). The best swaps are defined as those

```
1: k_max ← InitialKey
2: for m iterations do
3:     k ← k_max
4:     S ← best swaps for k
5:     for each {c_1, c_2} ∈ S do
6:         k' ← k(c_1↔c_2)
7:         if p(k') > p(k_max) then k_max ← k'
8:     if k_max = k then return k_max
9: return k_max
```

Figure 4.2: Greedy-swap decipherment algorithm.

that involve a symbol occurring among the 10 least common bigrams in the decipherment induced by the current key. If any such swap yields a more probable decipherment, it is incorporated in the current key; otherwise, the algorithm terminates. The total number of iterations is bounded by m, which is set to 5 times the size of the alphabet. After the initial run, the algorithm is restarted 20 times with a randomly generated initial key, which often results in a better decipherment. All parameters were established on a development set.

4.2.4 Evaluation

We now directly evaluate the three methods described above by applying them to a set of ciphertexts from different languages. We adapted the dataset created by Emerson et al. (2014) from the text of the Universal Declaration of Human Rights (UDHR) in 380 languages.[3] The average length of the texts is 1710 words and 11073 characters. We divided the text in each language into 66% training, 17% development, and 17% test. The training part was used to derive character bigram models for each language. The development and test parts were separately enciphered with a random substitution cipher.

Table 4.1 shows the results of the language identification methods on both the development and the test set. We report the average top-1 accuracy on the task of identifying the source language of 380 enciphered test samples. The differences between methods are statistically significant according to McNemar's test

[3]Eight languages from the original set were excluded because of formatting issues.

Method	Dev	Test
Random Selection	0.3	0.3
Jaskiewicz (2011)	54.2	47.6
Character Frequency	72.4	67.9
Decomposition Pattern	90.5	85.5
Trial Decipherment	94.2	97.1
Oracle Decipherment	98.2	98.4

Table 4.1: Language identification accuracy (in % correct) on ciphers representing 380 languages.

with $p < 0.0001$. The random baseline of 0.3% indicates the difficulty of the task. The "oracle" decipherment assumes a perfect decipherment of the text, which effectively reduces the task to standard language identification.

All three of our methods perform well, with the accuracy gains reflecting their increasing complexity. Between the two character frequency methods, our approach based on Bhattacharyya distance is significantly more accurate than the method of Jaskiewicz (2011), which uses a specially-designed distribution distance function. The decomposition pattern method makes many fewer errors, with the correct language ranked second in roughly half of those cases. Trial decipherment yields the best results, which are close to the upper bound for the character bigram probability approach to language identification. The average decipherment error rate into the correct language is only 2.5%. In 4 out of 11 identification errors made on the test set, the error rate is above the average; the other 7 errors involve closely related languages, such as Serbian and Bosnian.

The trial decipherment approach is much slower than the frequency distribution methods, requiring roughly one hour of CPU time in order to classify each ciphertext. More complex decipherment algorithms are even slower, which precludes their application to this test set. Our re-implementations of the dynamic programming algorithm of Knight et al. (2006) and the integer programming solver of Ravi and Knight (2008) average 53 and 7000 seconds of CPU time, respectively, to solve a single 256 character cipher, compared to 2.6 seconds with our greedy-swap method. The dynamic programming algorithm improves decipherment accuracy over our method by only 4% on a benchmark set of 50 ciphers of 256 characters. We con-

(a) `organized compositions through improvisational music into genres`
(b) `fyovicstu dfnrfecpcfie pbyfzob cnryfgcevpcfivm nzecd cipf otiyte`
(c) `otvfusyci cpifenfercfd bopbfzy fgyiemcpfcvrcnv nczed fpic etotyi`
(d) `adegiknor ciimnooopsst ghhortu aaiiilmnooprstv cimsu inot eegnrs`
(e) `adegiknor compositions through aaiiilmnooprstv music into greens`

Figure 4.3: An example of the encryption and decryption process: (a) plaintext; (b) after applying a substitution cipher; (c) ciphertext after random anagramming; (d) after substitution decipherment (in the alphagram representation); (e) final decipherment after anagram decoding (errors are underlined).

clude that our greedy-swap algorithm strikes the right balance between accuracy and speed required for the task of cipher language identification.

4.3 Anagram Decryption

In this section, we address the challenging task of deciphering a text in an unknown language written using an unknown script, and in which the letters within words have been randomly scrambled. The task is designed to emulate the decipherment problem posed by the VMS, with the assumption that its unusual ordering of characters within words reflects some kind of a transposition cipher. We restrict the source language to be one of the candidate languages for which we have sample texts; we model an unknown script with a substitution cipher; and we impose no constraints on the letter transposition method. The encipherment process is illustrated in Figure 4.3. The goal in this instance is to recover the plaintext in (a) given the ciphertext in (c) without the knowledge of the plaintext language. We also consider an additional encipherment step that removes all vowels from the plaintext.

Our solution is composed of a sequence of three modules that address the following tasks: language identification, script decipherment, and anagram decoding. For the first task we use the decomposition pattern frequency method described in Section 4.2.2, which is applicable to anagrammed ciphers. After identifying the plaintext language, we proceed to reverse the substitution cipher using a heuristic search algorithm guided by a combination of word and character language models. Finally, we unscramble the anagrammed words into readable text by framing the decoding as a tagging task, which is efficiently solved with a Viterbi decoder. Our

modular approach makes it easy to perform different levels of analysis on unsolved ciphers.

4.3.1 Script Decipherment

For the decipherment step, we adapt the state-of-the-art solver summarized in Chapter 3. The original method crucially depends on the notion of *pattern equivalence* within words. For example, *MZXCX* is pattern-equivalent with *there* and *bases*, but not with *otter*. In order to make the method work on anagrams, we relax the definition of pattern equivalence to include strings that have the same decomposition pattern, as defined in Section 4.2.2. Under the new definition, the order of the letters within a word has no effect on pattern equivalence. For example, *MZXCX* is equivalent not only with *there* and *bases*, but also with *three* and *otter*, because all these words map to the $(2, 1, 1, 1)$ pattern.

This modified solver repeatedly attempts to substitute word n-grams from the language sample into ciphertext n-grams that have the same decomposition pattern, while updating the current decipherment key accordingly. This key mutation procedure generates a tree structure, which is searched for the best-scoring decipherment. Internally, the solver represents all words as alphagrams, in which letters are reshuffled into the alphabetical order (Figure 4.3d). The trigram language models over both words and characters are derived by converting each word in the training sample into its alphagram. Like the original method, our modified solver limits the number of substitutions for a given n-gram. In order to handle the increased ambiguity, we use a letter-frequency heuristic to select the most likely mapping of letters within an n-gram. On a benchmark set of 50 ciphers of length 256, the average error rate of the modified solver is 2.6%, with only a small increase in time and space usage.

4.3.2 Anagram Decoder

The output of the script decipherment step is generally unreadable (see Figure 4.3d). The words might be composed of the right letters but their order is unlikely to be correct. We proceed to decode the sequence of anagrams by framing it as a simple

hidden Markov model, in which the hidden states correspond to plaintext words, and the observed sequence is composed of their anagrams. Without loss of generality, we convert anagrams into alphagrams, so that the emission probabilities are always equal to 1. Any alphagrams that correspond to unseen words are replaced with a single 'unknown' type. We then use a modified Viterbi decoder to determine the most likely word sequence according to a word trigram language model, which is derived from the training corpus, and smoothed using deleted interpolation (Jelinek and Mercer, 1980).

4.3.3 Vowel Recovery

Many writing systems, including Arabic and Hebrew, are abjads that do not explicitly represent vowels. Reddy and Knight (2011) provide evidence that the VMS may encode an abjad. The removal of vowels represents a substantial loss of information, and appears to dramatically increase the difficulty of solving a cipher.

In order to apply our system to abjads, we remove all vowels in the corpora prior to deriving the language models used by the script decipherment step. We assume the ability to partition the plaintext symbols into disjoint sets of vowels and consonants for each candidate language. The anagram decoder is trained to recover complete in-vocabulary words from sequences of anagrams containing only consonants. At test time, we remove the vowels from the input to the decipherment step of the pipeline. In contrast with Knight et al. (2006), our approach is able not only to attack abjad ciphers, but also to restore the vowels, producing fully readable text.

4.3.4 Evaluation

In order to test our anagram decryption pipeline on out-of-domain ciphertexts, the corpora for deriving language models need to be much larger than the UDHR samples used in the previous section. We selected five diverse European languages from Europarl (Koehn, 2005): English, Bulgarian, German, Greek, and Spanish. The corresponding corpora contain about 50 million words each, with the exception of Bulgarian which has only 9 million words. We remove punctuation and

	Step 2	Step 3	Both	Ceiling
English	99.5	98.2	97.7	98.7
Bulgarian	97.0	94.7	91.9	95.3
German	97.3	90.6	88.7	91.8
Greek	95.7	96.6	92.7	97.2
Spanish	99.1	98.0	97.1	99.0
Average	97.7	95.7	93.8	96.5

Table 4.2: Word accuracy on the anagram decryption task.

numbers, and lowercase all text.

We test on texts extracted from Wikipedia articles on art, Earth, Europe, film, history, language, music, science, technology, and Wikipedia. The texts are first enciphered using a substitution cipher, and then anagrammed (Figure 4.3a-c). Each of the five languages is represented by 10 ciphertexts, which are decrypted independently. In order to keep the running time reasonable, the length of the ciphertexts is set to 500 characters.

The first step is language identification. Our decomposition pattern method, which is resistant to both anagramming and substitution, correctly identifies the source language of 49 out of 50 ciphertexts. The lone exception is the German article on technology, for which German is the second ranked language after Greek. This error could be easily detected by noticing that most of the Greek words "deciphered" by the subsequent steps are out of vocabulary. We proceed to evaluate the following steps assuming that the source language is known.

The results in Table 4.2 show that our system is able to effectively break the anagrammed ciphers in all five languages. For Step 2 (script decipherment), we count as correct all word tokens that contain the right characters, disregarding their order. Step 3 (anagram decoding) is evaluated under the assumption that it has received a perfect decipherment from Step 2. On average, the accuracy of each individual step exceeds 95%. The values in the column denoted as *Both* are the actual results of the pipeline composed of Steps 2 and 3. Our system correctly recovers 93.8% of word tokens, which corresponds to over 97% of the in-vocabulary words within the test files, The percentage of the in-vocabulary words, which are shown in the *Ceiling* column, constitute the effective accuracy limits for each language.

	Step 2	Step 3	Both	Ceiling
English	99.9	84.6	84.5	98.7
Bulgarian	99.1	71.1	70.4	95.0
German	98.5	73.7	72.9	91.8
Greek	97.7	65.4	63.6	97.0
Spanish	99.8	73.7	73.3	99.0
Average	99.0	73.8	73.1	96.4

Table 4.3: Word accuracy on the abjad anagram decryption task.

The errors fall into three categories, as illustrated in Figure 4.3e. Step 2 introduces *decipherment errors* (e.g., deciphering 's' as 'k' instead of 'z' in "organized"), which typically preclude the word from being recovered in the next step. A *decoding error* in Step 3 may occur when an alphagram corresponds to multiple words (e.g. "greens" instead of "genres"), although most such ambiguities are resolved correctly. However, the majority of errors are caused by *out-of-vocabulary* (OOV) words in the plaintext (e.g., "improvisational"). Since the decoder can only produce words found in the training corpus, an OOV word almost always results in an error. The German ciphers stand out as having the largest percentage of OOV words (8.2%), which may be attributed to frequent compounding.

Table 4.3 shows the results of the analogous experiments on abjads (Section 4.3.3). Surprisingly, the removal of vowels from the plaintext actually improves the average decipherment step accuracy to 99%. This is due not only to the reduced number of distinct symbols, but also to the fewer possible anagramming permutations in the shortened words. On the other hand, the loss of vowel information makes the anagram decoding step much harder. However, more than three quarters of in-vocabulary tokens are still correctly recovered, including the original vowels.[4] In general, this is sufficient for a human reader to understand the meaning of the document, and deduce the remaining words.

[4]The differences in the Ceiling numbers between Tables 4.2 and 4.3 are due to words that are composed entirely of vowels.

4.4 Voynich Experiments

In this section, we present the results of our experiments on the VMS. We attempt to identify the source language with the methods described in Section 4.2; we quantify the similarity of the Voynich words to alphagrams; and we apply our anagram decryption algorithm from Section 4.3 to the text.

4.4.1 Data

Unless otherwise noted, the VMS text used in our experiments corresponds to 43 pages of the manuscript in the "type B" handwriting (VMS-B), investigated by Reddy and Knight (2011), which we obtained directly from the authors. It contains 17,597 words and 95,465 characters, transcribed into 35 characters of the Currier alphabet (d'Imperio, 1978).

Language	Text	Words	Characters
English	Bible	804,875	4,097,508
Italian	Bible	758,854	4,246,663
Latin	Bible	650,232	4,150,533
Hebrew	Tanach	309,934	1,562,591
Arabic	Quran	78,245	411,082

Table 4.4: Language corpora.

For the comparison experiments, we selected five languages shown in Table 4.4, which have been suggested in the past as the language of the VMS (Kennedy and Churchill, 2006). Considering the age of the manuscript, we attempt to use corpora that correspond to older versions of the languages, including the King James Bible, Bibbia di Gerusalemme, and Vulgate.

4.4.2 Source Language

In this section, we present the results of our ciphertext language identification methods from Section 4.2 on the VMS text.

The closest language according to the letter frequency method is Mazatec, a native American language from southern Mexico. Since the VMS was created before the voyage of Columbus, a New World language is an unlikely candidate. The top

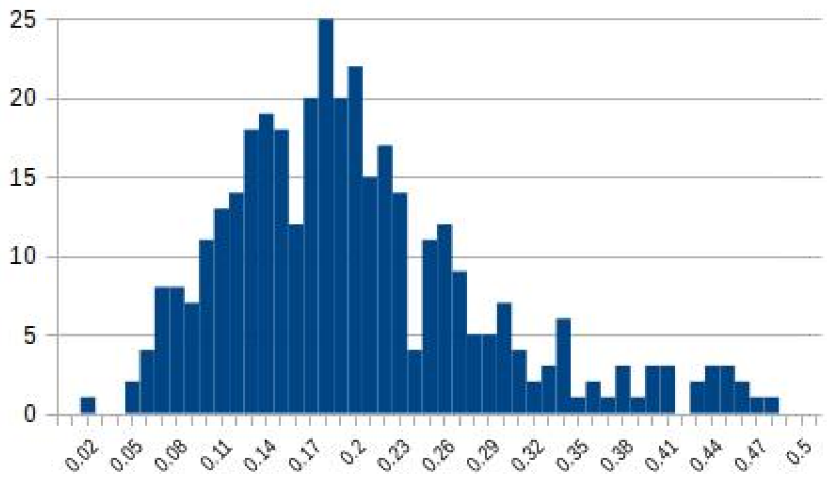

Figure 4.4: Histogram of distances between the VMS and samples of 380 other languages, as determined by the decomposition pattern method. The single outlier on the left is Hebrew.

ten languages also include Mozarabic (3), Italian (8), and Ladino (10), all of which are plausible guesses. However, the experiments in Section 4.2.4 demonstrate that the frequency analysis is much less reliable than the other two methods.

The top-ranking languages according to the decomposition pattern method are Hebrew, Malay (in Arabic script), Standard Arabic, and Amharic, in this order. We note that three of these belong to the Semitic family. The similarity of decomposition patterns between Hebrew and the VMS is striking. The Bhattacharyya distance between the respective distributions is 0.020, compared to 0.048 for the second-ranking Malay. The histogram in Figure 4.4 shows Hebrew as a single outlier in the leftmost bin. In fact, Hebrew is closer to a sample of the VMS of a similar length than to any of the remaining 379 UDHR samples.

The ranking produced by the the trial decipherment method is sensitive to parameter changes; however, the two languages that consistently appear near the top of the list are Hebrew and Esperanto. The high rank of Hebrew corroborates the outcome of the decomposition pattern method. Being a relatively recent creation, Esperanto itself can be excluded as the ciphertext language, but its high score is remarkable in view of the well-known theory that the VMS text represents a constructed language.[5] We hypothesize that the extreme morphological regularity of

[5]The theory was first presented in the form of an anagram (Friedman, 1959). See also a more

Esperanto (e.g., all plural nouns contain the bigram 'oj') yields an unusual bigram character language model which fits the repetitive nature of the VMS words.

In summary, while there is no complete agreement between the three methods about the most likely underlying source language, there appears to be a strong statistical support for Hebrew from the two most accurate methods, one of which is robust against anagramming. In addition, the language is a plausible candidate on historical grounds, being widely-used for writing in the Middle Ages. In fact, a number of cipher techniques, including anagramming, can be traced to the Jewish Cabala (Kennedy and Churchill, 2006).

4.4.3 Finite-state model

In this section, we construct a finite-state automation (FSA) that models the structure of the VMS words.

Elmar Vogt manually developed a "grammar" with the purpose of generating most of the words in the VMS-B while being as simple as possible.[6] The grammar is presented as a diagram comprising 25 states and 90 transitions, some of which rely on memory of prior characters within the word. Since the diagram cannot be easily converted into an FSA, we implement it instead as a computer program.

We approach the same task in a more principled manner by applying the ALERGIA algorithm as implemented by the Treba toolkit (Hulden, 2012) to the set of VMS-B words. In order to simplify the resulting automaton, we prune out all states and transitions that are traversed fewer than 100 times, and then remove all non-accepting states with no outgoing transitions. The final automaton contains 26 states and 96 transitions. It is illustrated in Figure 4.5 [7].

In comparison to Vogt's grammar, our FSA is not only conceptually simpler, but also more general, accepting a greater percentage of VMS word tokens (62% vs. 56%). Since a trivial one-state automaton would accept all words, we also measure the specificity of the solutions. We create a set of 10,000 OOV words by altering a single character in randomly selected VMS tokens. Our FSA accepts only

recent proposal by Balandin and Averyanov (2014).

[6] https://voynichthoughts.wordpress.com/grammar/

[7] The image was created using the OpenFst library (`http://www.openfst.org`)

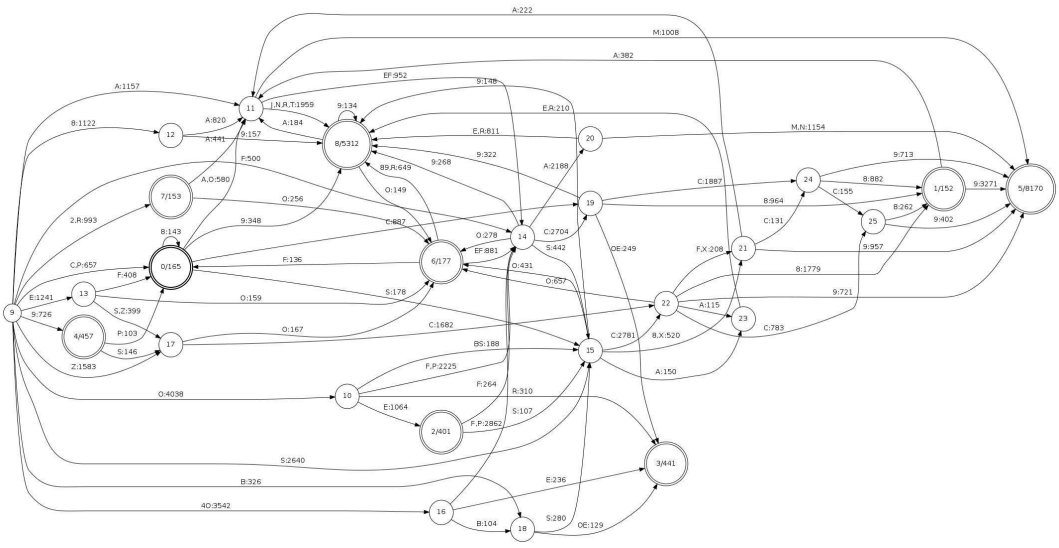

Figure 4.5: Finite state automaton for VMS-B. State 9 is the start state.

75 of those words, compared to 292 for Vogt's grammar.

The simplicity of the constructed FSA corroborate previous observations on the unusually regular structure of VMS words.

4.4.4 Alphagrams

In this section, we quantify the peculiarity of the VMS lexicon by modeling the words as alphagrams. We introduce the notion of the *alphagram distance*, and compute it for the VMS and for natural language samples.

We define a word's alphagram distance with respect to an ordering of the alphabet as the number of letter pairs that are in the wrong order. For example, with respect to the QWERTY keyboard order, the word *rye* has an alphagram distance of 2 because it contains two letter pairs that violate the order: (r, e) and (y, e). A word is an alphagram if and only if its alphagram distance is zero. The maximum alphagram distance for a word of length n is equal to the number of its distinct letter pairs.

In order to quantify how strongly the words in a language resemble alphagrams, we first need to identify the order of the alphabet that minimizes the total alphagram distance of a representative text sample. The decision version of this problem is NP-complete, which can be demonstrated by a reduction from the path variant of

the traveling salesman problem. Instead, we find an approximate solution with the following greedy search algorithm. Starting from an initial order in which the letters first occur in the text, we repeatedly consider all possible new positions for a letter within the current order, and choose the one that yields the lowest total alphagram distance of the text. This process is repeated until no better order is found for 10 iterations, with 100 random restarts.

When applied to a random sample of 10,000 word tokens from the VMS, our algorithm yields the order `4BZOVPEFSXQYWC28ARUTIJ3*GHK69MDLN5`, which corresponds to the average alphagram distance of 0.996 (i.e., slightly less than one pair of letters per word). Under this order, 61% of word tokens in the VMS-B are alphagrams. The corresponding result on English is `jzbqwxcpathofvurim-slkengdy`, with an average alphagram distance of 2.454. Note that the letters at the beginning of the sequence tend to have low frequency, while the ones at the end occur in popular morphological suffixes, such as $-ed$ and $-ly$. For example, the beginning of the first article of the UDHR with the letters transposed to follow this order becomes: *"All ahumn biseng are born free and qaule in tiingdy and thrisg."*

To estimate how close the solution produced by our greedy algorithm is to the actual optimal solution, we also calculate a lower bound for the total alphagram distance with any character order. The lower bound is $\sum_{x,y} \min(b_{xy}, b_{yx})$, where b_{xy} is the number of times character x occurs before character y within words in the text.

Figure 4.6 shows the average alphagram distances for the VMS and five comparison languages, each represented by a random sample of 10,000 word tokens which exclude single-letter words. The *Expected* values correspond to a completely random intra-word letter order. The *Lexicographic* values correspond to the standard alphabetic order in each language. The actual minimum alphagram distance is between the *Lower Bound* and the *Computed Minimum* obtained by our greedy algorithm.

The results in Figure 4.6 show that while the expected alphagram distance for the VMS falls within the range exhibited by natural languages, its minimum alphagram distance is exceptionally low. In absolute terms, the VMS minimum is less

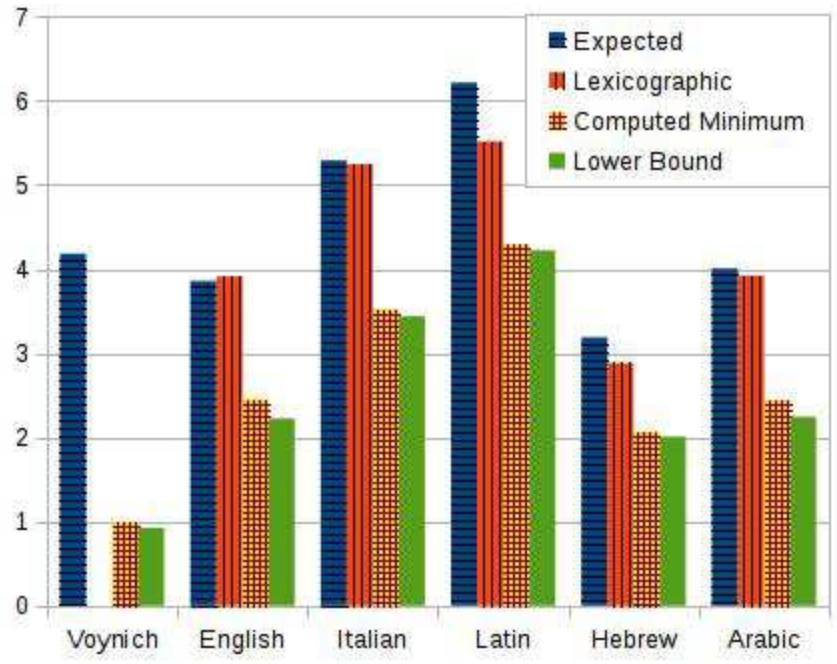

Figure 4.6: Average word alphagram distances.

than half the corresponding number for Hebrew. In relative terms, the ratio of the expected distance to the minimum distance is below 2 for any of the five languages, but above 4 for the VMS. These results suggest that, if the VMS encodes a natural language text, the letters within the words may have been reordered during the encryption process.

4.4.5 Decipherment Experiments

In this section, we discuss the results of applying our anagram decryption system described in Section 4.3 to the VMS text.

We decipher each of the first 10 pages of the VMS-B using the five language models derived from the corpora described in Section 4.4.1. The pages contain between 292 and 556 words, 3726 in total. Figure 4.7 shows the average percentage of in-vocabulary words in the 10 decipherments. The percentage is significantly higher for Hebrew than for the other languages, which suggests a better match with the VMS. Although the abjad versions of English, Italian, and Latin yield similar levels of in-vocabulary words, their distances to the VMS language according to the decomposition pattern method are 0.159, 0.176, and 0.245 respectively, well

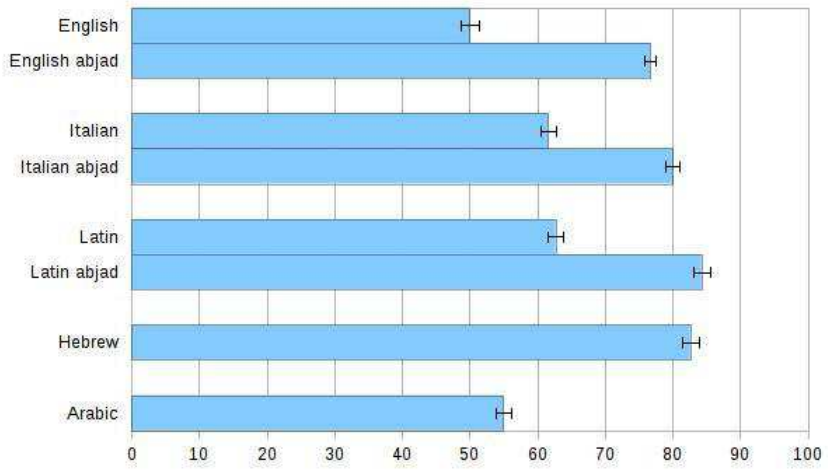

Figure 4.7: Average percentage of in-vocabulary words in the decipherments of the first ten pages of the VMS.

above Hebrew's 0.020; they would not even rank in the top 100 candidates of our 380 languages. The trial decipherment method similarly ranks the abjad versions of English, Italian, and Latin well below Hebrew.

None of the decipherments appear to be syntactically correct or semantically consistent. This is expected because our system is designed for pure monoalphabetic substitution ciphers. If the VMS indeed represents one of the five languages, the amount of noise inherent in the orthography and the transcription would prevent the system from producing a correct decipherment. For example, in a hypothetical non-standard orthography of Hebrew, some prepositions or determiners could be written as separate one-letter words, or a single phoneme could have two different representations. In addition, because of the age of the manuscript and the variety of its hand-writing styles, any transcription requires a great deal of guesswork regarding the separation of individual words into distinct symbols (Figure 4.1). Finally, the decipherments necessarily reflect the corpora that underlie the language model, which may correspond to a different domain and historical period.

Nevertheless, it is interesting to take a closer look at specific examples of the system output. The first line of the VMS (VAS92 9FAE AR APAM ZOE ZOR9 QOR92 9 FOR ZOE89) is deciphered into Hebrew as **איש אליו לביחו ו עלי אנשיו המצות ועשה לה הכהן**.[8] According to a native speaker of the language, this is not

[8]Hebrew is written from right to left.

quite a coherent sentence. However, after making a couple of spelling corrections, Google Translate is able to convert it into passable English: *"She made recommendations to the priest, man of the house and me and people."* [9]

Even though the input ciphertext is certainly too noisy to result in a fluent output, the system might still manage to correctly decrypt individual words in a longer passage. In order to limit the influence of context in the decipherment, we restrict the word language model to unigrams, and apply our system to the first 72 words (241 characters)[10] from the "Herbal" section of the VMS, which contains drawings of plants. An inspection of the output reveals several words that would not be out of place in a medieval herbal, such as הצר '*narrow*', איכר '*farmer*', אור '*light*', אויר '*air*', אש '*fire*'.

The results presented in this section could be interpreted either as tantalizing clues for Hebrew as the source language of the VMS, or simply as artifacts of the combinatorial power of anagramming and language models. We note that the VMS decipherment claims in the past have typically been limited to short passages, without ever producing a full solution. In any case, the output of an algorithmic decipherment of a noisy input can only be a starting point for scholars that are well-versed in the given language and historical period.

4.5 Summary

We have presented a multi-stage system for solving ciphers that combine monoalphabetic letter substitution and unconstrained intra-word letter transposition to encode messages in an unknown language.[11] We have evaluated three methods of ciphertext language identification that are based on letter frequency, decomposition patterns, and trial decipherment, respectively. We have demonstrated that our language-independent approach can effectively break anagrammed substitution ciphers, even when vowels are removed from the input. The application of our methods to the Voynich manuscript suggests that it may represent Hebrew, or another

[9] https://translate.google.com/ (accessed Nov. 20, 2015).

[10] The length of the passage was chosen to match the number of symbols in the Phaistos Disc inscription.

[11] Software at https://www.cs.ualberta.ca/~kondrak/.

abjad script, with the letters rearranged to follow a fixed order.

Chapter 5

Conclusion

This thesis has presented a suite of state-of-the-art algorithms for the analysis and decipherment of substitution ciphers. We have demonstrated the efficacy of a new cipher-breaking algorithm based on combining language models over words and characters with varying context sizes, successive key mutation, and heuristic search, and have shown that it exceeds the performance of previously published top-performing methods on the same task on a comparable data set. This algorithm can be directly applied to unsupervised transliteration between Serbian and Croatian and deniable encryption, with good results on both tasks.

We have further shown how this algorithm can be extended to work on anagrammed ciphers, removing any assumptions regarding symbol order within words, and have also shown that a Viterbi decoder can efficiently decode unenciphered anagrams. Taken together, this results in a decipherment algorithm which is resistant to any anagramming scheme, ranging from accidental letter transpositions to anagramming employed as a further form of encryption. We have presented novel evidence that the Voynich Manuscript features such an anagramming scheme, specifically with a method similar to alphagramming where symbols within each word are roughly arranged according to some lexical order. Applying our algorithm to the VMS reveals new evidence that it may be a cipher of Hebrew, providing new information on a long-standing cryptological, linguistic, and historical problem.

To further augment our decipherment system, we have also provided three new methods of identifying the source language of a cipher, based on character frequency distributions, patterns of symbol repetition within words, and the results of a

fast, preliminary decipherment into each language. These methods allow trade-offs between generality and classification accuracy, and work to remove the common assumption that the language of the plaintext is known. All three methods perform well on a 380-language test set. When we apply these methods to the VMS, Hebrew once again stands out, especially when applying the symbol repetition method, our most accurate method which is resistant to anagramming.

It appears that computational decipherment is quickly approaching the limit of what can be achieved on the task of solving monoalphabetic substitution ciphers, as evidenced by the near-zero error rates we achieve on ciphers as short as 32 characters. Performance on shorter ciphers will always be limited by the existence of "reasonable-but-wrong" solutions, cases where a ciphertext could reasonably have been generated from multiple distinct plaintexts. Faced with such a ciphertext, any solver, human or machine, must inevitably make a guess as to which plaintext is the intended solution – should "ABCDE FG" decipher to "filed by" or "would be"? Thus an information-theoretic limit exists on what can be achieved on this task.

Nevertheless, this thesis opens several possibilities for further research. Potential future work includes extending our approach to handle homophonic ciphers, in which the one-to-one mapping restriction is relaxed, greatly increasing the generality of the solver.

Future decipherment work could also focus on unsupervised transliteration, to reduce the dependence on linguistic similarity so that transliteration could be performed between languages less closely related than Serbian and Croatian, and to allow transliteration between scripts with no approximate 1-to-1 mapping, such as the Japanese Katakana-to-English transliteration pursued by Ravi and Knight (2009). Unsupervised transliteration is well-suited to being framed as a decipherment task, as it directly applies the idea of deciphering an unknown script into one which can be readily understood.

The task of deniable encryption is also one which could be explored further, with the goal of generating syntactically correct and meaningful ciphertexts. Concealing the meaning of a text in such a way that the resulting ciphertext appears to be simply an ordinary text, rather than the nonsense text that encryption typically

produces, is interesting from a linguistic as well as computational standpoint.

The pipeline approach presented in Chapter 4 might be outperformed by a unified generative model. The techniques could be made more resistant to noise; for example, by softening the emission model in the anagram decoding phase. It would also be interesting to jointly identify both the language and the *type* of the cipher (Nuhn and Knight, 2014), which could lead to the development of methods to handle more complex ciphers. Finally, the anagram decoding task could be extended to account for the transposition of words within lines, in addition to the transposition of symbols within words. This, combined with our methods for handing anagrammed ciphers, would create a solver which would be completely resistant to the direction of writing (left-to-right versus right-to-left), as well as the intentional use of word permutation as an encryption technique.

The Voynich Manuscript is not the only outstanding script decipherment problem. Robinson (2002) describes several "lost languages", ancient writing which can no longer be read by any living person. Future work could focus on using decipherment techniques to provide new insights into these scripts. Of course, the VMS itself remains unsolved, and with the field of computational decipherment rapidly producing increasingly effective and resilient methods, further analysis of this centuries-old text using these new tools could certainly motivate future work.

References

Broderick Arneson, Ryan B Hayward, and Philip Henderson. 2010. Monte Carlo Tree Search in Hex. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):251–258.

Arcady Balandin and Sergey Averyanov. 2014. The Voynich manuscript: New approaches to deciphering via a constructed logical language.

Taylor Berg-Kirkpatrick and Dan Klein. 2011. Simple effective decipherment via combinatorial optimization. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 313–321.

Aditya Bhargava, Bradley Hauer, and Grzegorz Kondrak. 2011. Leveraging transliterations from multiple languages. In *Proceedings of the 3rd Named Entities Workshop (NEWS 2011)*, pages 36–40, Chiang Mai, Thailand, November. Asian Federation of Natural Language Processing.

A. Bhattacharyya. 1943. On a measure of divergence between two statistical populations defined by their probability distributions. *Bull. Calcutta Math. Soc.*, 35:99–109.

Thorsten Brants. 2000. TnT – a statistical part-of-speech tagger. In *Proceedings of the Sixth Conference on Applied Natural Language Processing*, pages 224–231.

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43.

Ran Canetti, Cynthia Dwork, Moni Naor, and Rafi Ostrovsky. 1997. Deniable encryption. In *Advances in Cryptology–CRYPTO'97*, pages 90–104.

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.

Mary E. d'Imperio. 1978. The Voynich manuscript: an elegant enigma. Technical report, DTIC Document.

Guy Emerson, Liling Tan, Susanne Fertmann, Alexis Palmer, and Michaela Regneri. 2014. Seedling: Building and using a seed corpus for the human language project. In *Proceedings of the 2014 Workshop on the Use of Computational Methods in the Study of Endangered Languages*, pages 77–85, Baltimore, Maryland, USA, June. Association for Computational Linguistics.

Markus Enzenberger, Martin Muller, Broderick Arneson, and Richard Segal. 2010. Fuego – an open-source framework for board games and go engine based on Monte Carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):259–270.

Joseph Martin Feely. 1943. *Roger Bacon's Cypher. The Right Key Found*. Rochester, NY.

Elizebeth S Friedman. 1959. Acrostics, anagrams, and chaucer. *Philological Quarterly*, 38:1.

Bradley Hauer and Grzegorz Kondrak. 2016. Decoding anagrammed texts written in an unknown language and script. *Transactions of the Association for Computational Linguistics*, 4:75–86.

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, August. Dublin City University and Association for Computational Linguistics.

Mans Hulden. 2012. Treba: Efficient numerically stable EM for PFA. In *Proceedings of The 11th International Conference on Grammatical Inference (ICGI)*, pages 249–253.

Grzegorz Jaskiewicz. 2011. Analysis of letter frequency distribution in the Voynich manuscript. In *Proceedings of the International Workshop (CS&P'11)*, pages 250–261.

F. Jelinek and R.L. Mercer. 1980. Interpolated estimation of Markov source parameters from sparse data. *Pattern recognition in practice*.

Dan Jurafsky, James H Martin, Andrew Kehler, Keith Vander Linden, and Nigel Ward. 2000. *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*, volume 2. MIT Press.

Gerry Kennedy and Rob Churchill. 2006. *The Voynich manuscript: The mysterious code that has defied interpretation for centuries*. Inner Traditions/Bear & Co.

Kevin Knight and Kenji Yamada. 1999. A computational approach to deciphering unknown scripts. In *Proceedings of the Association for Computational Linguistics (ACL) Workshop on Unsupervised Learning in Natural Language Processing*, pages 37–44.

Kevin Knight, Anish Nair, Nishit Rathod, and Kenji Yamada. 2006. Unsupervised analysis for decipherment problems. In *Proceedings of the International*

Conference on Computational Linguistics(COLING)/Association for Computational Linguistics (ACL) 2006 Main Conference Poster Sessions, pages 499–506.

Kevin Knight, Beáta Megyesi, and Christiane Schaefer. 2011. The Copiale cipher. In *the 4th Workshop on Building and Using Comparable Corpora: Comparable Corpora and the Web*, pages 2–9.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based Monte-Carlo Planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Euro. Conf. Mach. Learn.*, pages 282–293, Berlin, Germany. Springer.

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5, pages 79–86.

Gabriel Landini. 2001. Evidence of linguistic structure in the Voynich manuscript using spectral analysis. *Cryptologia*, 25(4):275–295.

Stanisław Lem. 1973. *Memoirs found in a bathtub*. The Seabury Press.

John Matthews Manly. 1931. Roger Bacon and the Voynich MS. *Speculum*, 6(03):345–391.

Shimpei Matsumoto, Noriaki Hirosue, Kyohei Itonaga, Kazuma Yokoo, and Hisatomo Futahashi. 2010. Evaluation of simulation strategy on single-player Monte-Carlo tree search and its discussion for a practical scheduling problem. In *the International MultiConference of Engineers and Computer Scientists*, volume 3, pages 2086–2091.

Jean Méhat and Tristan Cazenave. 2010. Combining UCT and nested Monte Carlo search for single-player general game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):271–277.

Marcelo A Montemurro and Damián H Zanette. 2013. Keywords and co-occurrence patterns in the Voynich manuscript: an information-theoretic analysis. *PloS one*, 8(6):e66344.

Dennis Moore, W.F. Smyth, and Dianne Miller. 1999. Counting distinct strings. *Algorithmica*, 23(1):1–13.

George Nagy, Sharad Seth, and Kent Einspahr. 1987. Decoding substitution ciphers by means of word matching with application to ocr. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(5):710–715.

William Romaine Newbold and Roland Grubb Kent. 1928. *The Cipher of Roger Bacon*. University of Pennsylvania Press.

Garrett Nicolai, Bradley Hauer, Mohammad Salameh, Adam St Arnaud, Ying Xu, Lei Yao, and Grzegorz Kondrak. 2015. Multiple system combination for transliteration. In *Proceedings of the Fifth Named Entity Workshop*, pages 72–77, Beijing, China, July. Association for Computational Linguistics.

Peter Norvig. 2009. Natural language corpus data. In Toby Segaran and Jeff Hammerbacher, editors, *Beautiful data: the stories behind elegant data solutions*. O'Reilly.

Malte Nuhn and Kevin Knight. 2014. Cipher type detection. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1769–1773, Doha, Qatar, October. Association for Computational Linguistics.

Malte Nuhn and Hermann Ney. 2013. Decipherment complexity in 1:1 substitution ciphers. In *the 51st Annual Meeting of the Association for Computational Linguistics*, pages 615–621.

Malte Nuhn, Arne Mauser, and Hermann Ney. 2012. Deciphering foreign language by combining language models and context vectors. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, pages 156–164.

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*, pages 1568–1576.

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

Alessandro Previti, Raghuram Ramanujan, Marco Schaerf, and Bart Selman. 2011. Applying UCT to Boolean satisfiability. In *Theory and Applications of Satisfiability Testing-SAT 2011*, pages 373–374. Springer.

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 (EMNLP)*, pages 812–819.

Sujith Ravi and Kevin Knight. 2009. Learning phoneme mappings for transliteration without parallel data. In *Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 37–45.

Sujith Ravi and Kevin Knight. 2011. Deciphering foreign language. In *the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 12–21.

Sravana Reddy and Kevin Knight. 2011. What we know about the Voynich manuscript. In *Proceedings of the 5th Association for Computational Linguistics – Human Language Technologies Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities*, pages 78–86. Association for Computational Linguistics.

Andrew Robinson. 2002. *Lost languages: the enigma of the world's undeciphered scripts*. McGraw-Hill New York.

Gordon Rugg. 2004. An elegant hoax? A possible solution to the Voynich manuscript. *Cryptologia*, 28(1):31–46.

Maarten PD Schadd, Mark HM Winands, H Jaap Van Den Herik, Guillaume MJ-B Chaslot, and Jos WHM Uiterwijk. 2008. Single-player Monte-Carlo tree search. In *Computers and Games*, pages 1–12. Springer.

Andreas Schinner. 2007. The Voynich manuscript: evidence of the hoax hypothesis. *Cryptologia*, 31(2):95–107.

Klaus Schmeh. 2013. A milestone in Voynich manuscript research: Voynich 100 conference in Monte Porzio Catone, Italy. *Cryptologia*, 37(3):193–203.

Simon Singh. 1999. *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*. Random House.

Benjamin Snyder, Regina Barzilay, and Kevin Knight. 2010. A statistical model for lost language decipherment. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1048–1057.

Leonell C Strong. 1945. Anthony Askham, the author of the Voynich manuscript. *Science*, 101(2633):608–609.

John Tiltman. 1968. *The Voynich Manuscript, The Most Mysterious Manuscript in the World*. Baltimore Bibliophiles.