Solving Substitution Ciphers with Combined Language Models

Bradley Hauer **Ryan Hayward** **Grzegorz Kondrak**
Department of Computing Science
University of Alberta
Edmonton, AB, Canada
{bmhauer,hayward,gkondrak}@ualberta.ca

Abstract

We propose a novel approach to deciphering short monoalphabetic ciphers that combines both character-level and word-level language models. We formulate decipherment as tree search, and use Monte Carlo Tree Search (MCTS) as a fast alternative to beam search. Our experiments show a significant improvement over the state of the art on a benchmark suite of short ciphers. Our approach can also handle ciphers without spaces and ciphers with noise, which allows us to explore its applications to unsupervised transliteration and deniable encryption.

1 Introduction

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 (Figure 1). 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 paper, we investigate the task of automatically solving substitution ciphers. Complete automation of the key discovery process remains an active area of research (Ravi and Knight, 2008; Corlett and Penn, 2010; Nuhn et al., 2013). The task is to recover the plaintext from the ciphertext without the key, given only a corpus representing the language of the plaintext. The key is a 1-1 mapping between plaintext and ciphertext alphabets, which are assumed to be of equal length. Without loss of generality, we assume that both alphabets are composed of the same set of symbols, so that the key is equivalent to a permutation of the alphabet. 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), and archaeological decipherment of lost languages (Snyder et al., 2010).

Our contribution is a novel approach to the problem that combines both character-level and word-level language models. 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 work is licensed under a Creative Commons Attribution 4.0 International Licence. Page numbers and proceedings footer are added by the organizers. Licence details: http://creativecommons.org/licenses/by/4.0/

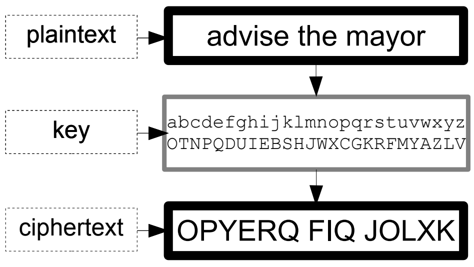

Figure 1: An example of encryption with a substitution cipher.

The paper is organized as follows. After reviewing previous work on automated decipherment in Section 2, we describe our approach to combining character-level and word-level language models with respect to key scoring (Section 3), and key generation (Section 4). In Section 5, we introduce Monte Carlo Tree Search and its adaptation to decipherment. In Section 6, we discuss several evaluation experiments and their results. Section 7 is devoted to experiments in deniable encryption.

2 Related Work

Kevin Knight has been the 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 NLP community by demonstrating how to decode unfamiliar writing scripts using phonetic models of known languages. Knight et al. (2006) explore unsupervised learning methods, including the expectation-maximization (EM) algorithm, for a variety of decipherment problems. Ravi and Knight (2009) formulate the problem of unsupervised transliteration as decipherment in order to reconstruct cross-lingual phoneme mapping tables, achieving approximately 50% character accuracy on U.S. names written in the Japanese Katakana script. Reddy and Knight (2011) apply various computational techniques to analyze an undeciphered medieval document. Knight et al. (2011) relate a successful decipherment of a nineteenth-century cipher, which was achieved by combining both manual and computational techniques.

In the remainder of this section, we focus on the work specifically aimed at solving monoalphabetic substitution ciphers. Olson (2007) presents a method that improves on previous dictionary-based approaches by employing an array of selection heuristics. The solver attempts 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 solver[1] produces *"us far a youngs with had up about*". Our new approach, which employs word-level language models, correctly solves this cipher.

Ravi and Knight (2008) formulate decipherment as an integer programming problem in which the objective function is defined by a low-order character language model; an integer program solver then finds the solution that is optimal with respect to the objective function. This method is slow, precluding the use of higher order language models. Our reimplementation of their 2-gram solver deciphers *"it was a bright cold day in april*" as *"ae cor o blathe wind dof as oulan*". By contrast, our approach incorporates word-level information and so tends to avoid out-of-vocabulary words.

Norvig (2009) describes a hill-climbing method that involves both word and character language models, but the models are only loosely combined; specifically, the word model is used to select the best solution from a small number of candidates identified by the character model. When applied to the cipher that corresponds to our example sentence from Orwell, the solver[2] returns *"ache red tab scoville magenta i*".

[1]`http://www.blisstonia.com/software/Decrypto` (accessed August 1, 2013)
[2]`http://norvig.com/ngrams` (accessed June 2, 2014)

Corlett and Penn (2010) use fast heuristic A* search, which can handle much longer ciphers than the method of Ravi and Knight (2008), while still finding the optimal solution. 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, but the converse is not true. Our approach achieves a near-zero error rate for ciphers as short as 128 characters.

Nuhn et al. (2013) set the state of the art by employing beam search to solve substitution ciphers. Their method is inexact but fast, allowing them to incorporate higher-order (up to 6-gram) character language models. Our work differs in incorporating word-level information for the generation and scoring of candidate keys, which improves decipherment accuracy.

3 Key Scoring

Previous work tend to employ either character-level language models or dictionary-type word lists. However, word-level language models have a potential of improving 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*.[3] 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 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]`http://www.newswise.com/articles/edgar-allen-poe-cipher-solved`

4 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. 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 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 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. $\forall i$: $u_i = ␣ \Leftrightarrow v_i = ␣$
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 transpositions resulting in a new key. Leaf expansion is summarized in Figure 3.

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 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. 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.

```
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 2: MCTS for decipherment.

5 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.

5.1 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.

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 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 2 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. 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.

5.2 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

```
1: function EXPAND(Leaf, CipherText)
2:     for all word $n$-grams $w$ in CipherText do
3:         for $k$ best $w'$ s.t. $w' \overset{p}{\equiv} w$ do
4:             NewLeaf = Modify(Leaf, $w \mapsto w'$)
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: Leaf expansion.

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 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 4. 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.

6 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.

6.1 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. Spaces are preserved, and the boundaries of the ciphers match word boundaries.

Figure 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 2).

Examples of decipherment errors are shown in Table 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'

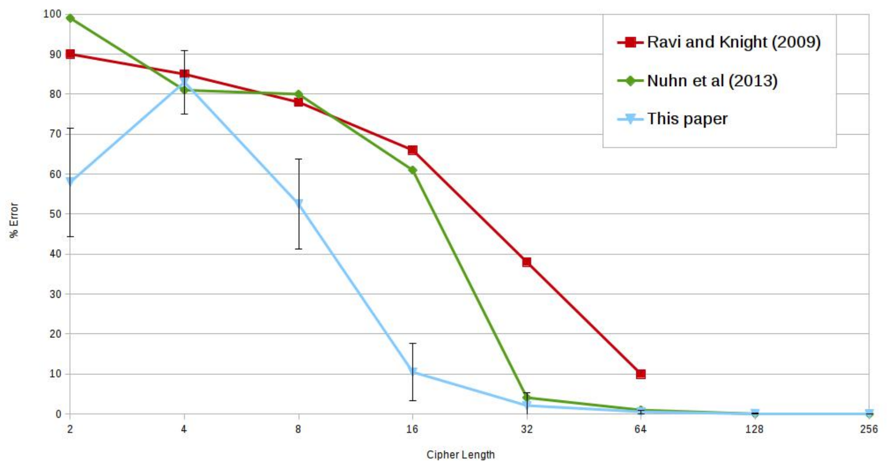

Figure 4: Average decipherment error rate as a function of cipher length on the Wikipedia test set.

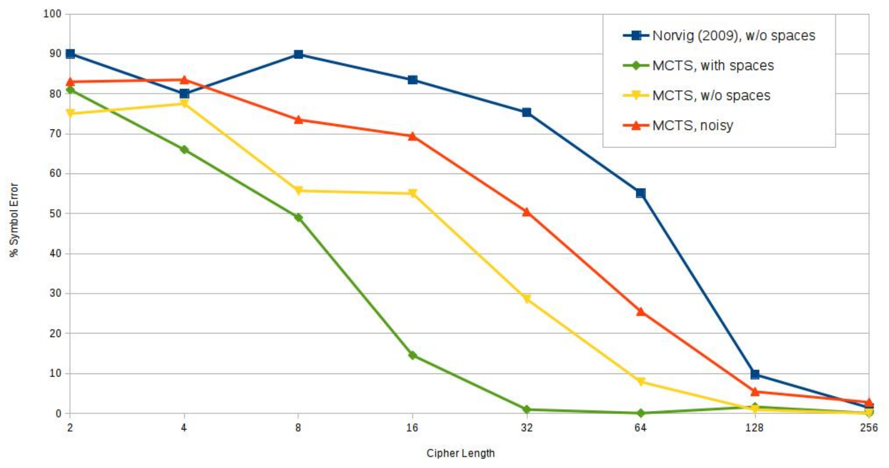

Figure 5: Average decipherment error rate as a function of cipher length on the NYT test set.

	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 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: Examples of decipherment errors.

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.

6.2 Beam Search vs. MCTS

The error rates of the two versions of our algorithm are very close, with a few exceptions (Table 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 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.

6.3 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.

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 in Table 2. 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.

6.4 Croatian and Serbian

We further test the robustness of our approach by performing an experiment on decipherment of an unknown script. For this experiment, we selected Croatian and Serbian, two closely related languages

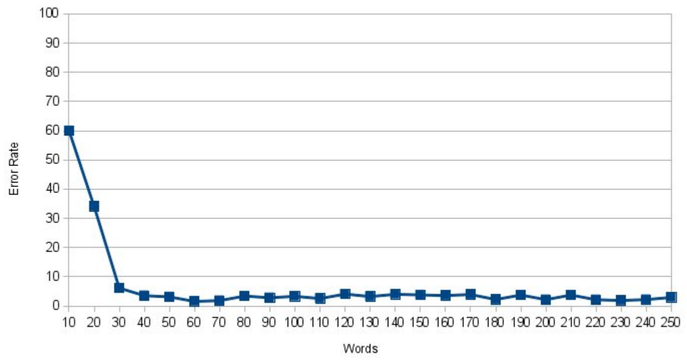

Figure 6: The decipherment error rate on a Serbian sample text as a function of the ciphertext length.

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.

сва људска бића рађају се слободна и једнака у достојанству и правима она су обдарена разумом и свешћу и треба једни према другима

sva **š**udska bi**h**a ra**l**aju se s**ž**obodna i jednaka u dostojanstvu i pravima ona su obdarena ra**č**umom i sve**ch**u i treba jedni prema dru**z**ima

Table 4: 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 6). The residual error rate reflects the lack of correct mapping for the three Serbian letters mentioned above. As can be seen in Table 4, 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%.

6.5 Ciphers Without Spaces

Removing spaces that separate words 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), and the search tree node expansion (Section 5).

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 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 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). As a result,

53‡‡†305))6*;4826)4‡.)4‡);806*;48†8¶60))85;1‡(;:‡*8†83(88)5*†;46(;88*96*?;8)*‡(;485);5*†2:*‡(;4956*2(5*−4)8¶8*;4069285);)6†8)4
agoodglassinthebishopshostelinthedevilsseatfortyonedegreesandthirteenminutesnortheastandbynorthmainbranchseventhlimbeastsidesh

Table 5: The beginning of the Gold Bug cipher and its decipherment.

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 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 in Figure 5, 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 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 5. Both experiments reported in this section confirm that our word-based approach works well even when spaces are removed from ciphers.

7 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 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 ungrammatical 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.

8 Conclusion

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.

In the future, we would like to extend our approach to handle homophonic ciphers, in which the one-to-one mapping restriction is relaxed. Another interesting direction is developing algorithms to generate syntactically correct and meaningful deniable encryptions.

Acknowledgements

This research was supported by the Natural Sciences and Engineering Research Council of Canada and the Alberta Innovates Technology Futures.

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.

Taylor Berg-Kirkpatrick and Dan Klein. 2011. Simple effective decipherment via combinatorial optimization. In *Empirical Methods in Natural Language Processing*, pages 313–321.

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 *the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1040–1047.

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.

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

Kevin Knight and Kenji Yamada. 1999. A computational approach to deciphering unknown scripts. In *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 *the COLING/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.

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

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.

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.

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 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 *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 *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 *Empirical Methods in Natural Language Processing*, pages 812–819.

Sujith Ravi and Kevin Knight. 2009. Learning phoneme mappings for transliteration without parallel data. In *NAACL*, 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 *the 5th ACL-HLT Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities*, pages 78–86.

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.

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 *the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1048–1057.