

Classic cryptanalysis using hidden Markov models

Rohit Vobbilisetty, Fabio Di Troia, Richard M. Low, Corrado Aaron Visaggio & Mark Stamp

To cite this article: Rohit Vobbilisetty, Fabio Di Troia, Richard M. Low, Corrado Aaron Visaggio & Mark Stamp (2016): Classic cryptanalysis using hidden Markov models, Cryptologia, DOI: [10.1080/01611194.2015.1126660](https://doi.org/10.1080/01611194.2015.1126660)

To link to this article: <http://dx.doi.org/10.1080/01611194.2015.1126660>



Published online: 16 Mar 2016.



Submit your article to this journal [↗](#)



Article views: 19



View related articles [↗](#)



View Crossmark data [↗](#)

Classic cryptanalysis using hidden Markov models

Rohit Vobbilisetty, Fabio Di Troia, Richard M. Low, Corrado Aaron Visaggio and Mark Stamp

ABSTRACT

In this article, the authors present a detailed introduction to hidden Markov models (HMM). They then apply HMMs to the problem of solving simple substitution ciphers, and they empirically determine the accuracy as a function of the ciphertext length and the number of random restarts. Application to homophonic substitutions and other classic ciphers is briefly considered.

KEYWORDS

cryptanalysis; hidden Markov models; Jakobsen's algorithm; Purple cipher; simple substitution

1. Introduction

A hidden Markov model (HMM) can be viewed as a machine learning technique that employs a discrete hill climb. An HMM includes a Markov process that is “hidden” in the sense that we cannot directly observe the process. However, we do have access to a series of observations that are related to the hidden Markov process by discrete probability distributions.

Given a set of observations, we can train an HMM to fit the observations. That is, we can perform a discrete hill climb to determine the parameters of an HMM, specifically, the matrix of probabilities that drives the hidden Markov process, as well as the matrix of probabilities that relates the hidden states to the observations. For training, we only need to specify the dimensions of these probability matrices. This is the sense in which an HMM is a machine learning technique; we make virtually no assumptions, yet through the training process, significant statistical properties can be revealed.

Once we have trained an HMM, the resulting model can be used to score sequences. Given an observation sequence, the higher the score we obtain from a given HMM, the more closely the sequence matches the observations used to train the HMM. HMMs have proven invaluable in a wide array of fields, including speech recognition (Rabiner 1989), natural language processing (Cave and Neuwirth 1980), malware detection (Wong and Stamp 2006), and a variety of problems in bioinformatics.

In this article, we present a relatively detailed introduction to HMMs and illustrate their utility by considering classic cryptanalysis problems. In particular, we focus on the simple substitution cipher, and we also briefly consider

homophonic substitutions, among other classic ciphers. We are particularly interested in the effect of multiple random restarts on the success of the HMM hill climb. This work was inspired by the Berg-Kirkpatrick and Klein's (2013) paper, in which an expectation maximization algorithm (essentially, a restricted case of the HMM) is applied to homophonic substitutions with the goal of analyzing the unsolved Zodiac 340 cipher.

The remainder of this article is organized as follows. In Section 2, we provide an introduction to HMMs. In Section 3, we give results from various experiments. Finally, Section 4 contains our conclusions and suggestions for future work.

2. HMMs

In this section, we provide a detailed introduction to HMMs. Then, in Section 3, we analyze the effect of multiple random restarts in the context of simple substitution cryptanalysis.

A generic illustration of a HMM is given in Figure 1. In this figure, the X_i represent the state sequence of the underlying Markov process while the \mathcal{O}_i are the observations (i.e., outcomes that we can observe). The Markov process is hidden in the sense that we cannot directly observe what is happening above the dashed line in Figure 1. However, the state transitions are determined by the probabilities contained in the $N \times N$ matrix A , while the probabilities in the $N \times M$ matrix B relate the (hidden) states of the Markov process to the observations \mathcal{O}_i .

The standard HMM notation is summarized in Table 1. Note that the observations are assumed to come from the set $\{0, 1, \dots, M-1\}$, which simplifies the notation with no loss of generality. That is, we simply associate each distinct observation with one of the elements $0, 1, \dots, M-1$, so that $\mathcal{O}_i \in V = \{0, 1, \dots, M-1\}$ for $i = 0, 1, \dots, T-1$.

The matrix $A = \{a_{ij}\}$ is $N \times N$ and is defined as

$$a_{ij} = P(\text{state } q_j \text{ at } t+1 \mid \text{state } q_i \text{ at } t).$$

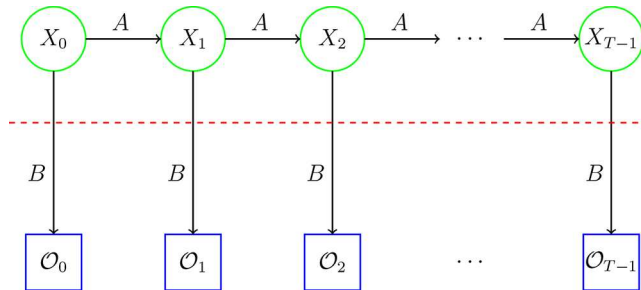


Figure 1. Hidden Markov model (HMM).

Table 1. Hidden Markov model (HMM) notation.

Notation	Explanation
T	Length of the observation sequence
N	Number of states in the model
M	Number of observation symbols
Q	Distinct states of the Markov process, q_0, q_1, \dots, q_{N-1}
V	Possible observations, assumed to be $0, 1, \dots, M-1$
A	State transition probabilities
B	Observation probability matrix
π	Initial state distribution
\mathcal{O}	Observation sequence, $\mathcal{O}_0, \mathcal{O}_1, \dots, \mathcal{O}_{T-1}$

Also, the matrix A is row stochastic, that is, each row of A satisfies the conditions of a discrete probability distribution. Note that the probabilities a_{ij} are independent of t .

The matrix $B = \{b_j(k)\}$ is $N \times M$ and is defined by

$$b_j(k) = P(\text{observation } k \text{ at } t \mid \text{state } q_j \text{ at } t).$$

As with A , the matrix B is row stochastic and the probabilities $b_j(k)$ are independent of t . The slightly unusual notation $b_j(k)$ is chosen to simplify some of the expressions that appear below.

An HMM is defined by A , B and the initial state distribution π (and, implicitly, by the dimensions N and M). Therefore, we denote an HMM by $\lambda = (A, B, \pi)$.

Suppose that we are given the HMM $\lambda = (A, B, \pi)$, which was obtained by training on the observation sequence

$$\mathcal{O} = (\mathcal{O}_0, \mathcal{O}_1, \mathcal{O}_2, \dots, \mathcal{O}_{T-1}).$$

Then, we can compute the probability of any given state sequence

$$X = (x_0, x_1, x_2, \dots, x_{T-1})$$

as follows. From the definition of the π matrix, we see that π_{x_0} is the probability of starting in state x_0 . Also, $b_{x_0}(\mathcal{O}_0)$ is the probability of observing \mathcal{O}_0 when the Markov process is in state x_0 , and a_{x_0, x_1} is the probability of transiting from state x_0 to state x_1 . Continuing, the probability of the state sequence X is given by

$$P(X) = \pi_{x_0} b_{x_0}(\mathcal{O}_0) a_{x_0, x_1} b_{x_1}(\mathcal{O}_1) a_{x_1, x_2} b_{x_2}(\mathcal{O}_2) \cdots a_{x_{T-2}, x_{T-1}} b_{x_{T-1}}(\mathcal{O}_{T-1}). \quad (1)$$

To uncover the “best” hidden state sequence, we could simply compute $P(X)$ for each possible state sequence and deem the path X that yields the highest probability as best. In fact, dynamic programming provides an efficient way to determine the “best” state sequence in this sense. That is, in dynamic programming, we choose the best path.

In contrast, for an HMM, we define “best” as the X that maximizes the expected number of correct states x_i . This is one sense in which an HMM

is an expectation maximization (EM) algorithm. Below, we discuss the algorithms used in HMMs, but as an aside, we note that the algorithm used for dynamic programming and that used to train an HMM are very similar.¹ Also, the “best” X obtained via dynamic programming need not agree with that obtained by the HMM approach; see Stamp (2012) for such an example. In fact, the HMM solution might not correspond to a valid path since it could include a transition probability of 0. This is not to imply that dynamic programming is superior to HMMs or vice versa; both approaches can be extremely useful.

2.1. Three problems

There are three fundamental problems that we can solve using HMMs. Here, we briefly describe these three problems. Then, in the next section, we give efficient algorithms for the solution of each.

2.1.1. Problem 1

Given the model $\lambda = (A, B, \pi)$ and a sequence of observations \mathcal{O} , determine $P(\mathcal{O} | \lambda)$, that is, we compute the likelihood of an observed sequence \mathcal{O} , given the model λ . This can be viewed as scoring a given observation sequence against a given model.

2.1.2. Problem 2

Given $\lambda = (A, B, \pi)$ and an observation sequence \mathcal{O} , find an optimal state sequence for the underlying Markov process. This corresponds to “uncovering” the hidden part of the HMM. This problem was briefly discussed in the previous section.

2.1.3. Problem 3

Given an observation sequence \mathcal{O} and N and M , find the model $\lambda = (A, B, \pi)$ that maximizes the probability of \mathcal{O} . This is the training process where we determine a model that best fits the observed data. This process involves a discrete hill climb on the parameter space represented by A , B , and π .

2.1.4. Discussion

Suppose that we are given several samples of malware code, all of which are known to belong to the same malware family. Further, suppose that we extract the opcode sequences from all of these samples, ignoring operands, directives, labels, and so on, and we concatenate these sequences. If we can solve Problem 3, then we can train an HMM on this opcode sequence to obtain an HMM λ . Then, if we later obtain a suspect sample, we can extract its opcode sequence,

¹Specifically, in a dynamic program, we use “max” to update the scores at each step, whereas in an HMM, a sum over the same set of values is used instead.

then use the solution to Problem 1 to compute a score based on the model λ . The resulting score can be used to classify the sample as either belonging to the malware family that was used to generate the model λ , or not. Note that in this case, we did not use Problem 2; this is the situation that prevails in many practical applications.

The utility of HMMs derives largely from the fact that there are efficient algorithms to solve the three problems outlined above. In the next section, we present these algorithms in some detail. Then, we turn our attention to an example that nicely illustrates the strength of the technique and leads us directly into classic cryptanalysis problems.

2.2. Three solutions

2.2.1. Solution to Problem 1

Let $\lambda = (A, B, \pi)$ be a given model, and let $\mathcal{O} = (\mathcal{O}_0, \mathcal{O}_1, \dots, \mathcal{O}_{T-1})$ be a series of observations. We want to find $P(\mathcal{O} | \lambda)$.

Let $X = (x_0, x_1, \dots, x_{T-1})$ be a state sequence. Then, by the definition of B , we have

$$P(\mathcal{O} | X, \lambda) = b_{x_0}(\mathcal{O}_0) b_{x_1}(\mathcal{O}_1) \cdots b_{x_{T-1}}(\mathcal{O}_{T-1})$$

and by the definitions of π and A , it follows that

$$P(X | \lambda) = \pi_{x_0} a_{x_0, x_1} a_{x_1, x_2} \cdots a_{x_{T-2}, x_{T-1}}.$$

Since

$$P(\mathcal{O}, X | \lambda) = \frac{P(\mathcal{O} \cap X \cap \lambda)}{P(\lambda)}$$

and

$$P(\mathcal{O} | X, \lambda) P(X | \lambda) = \frac{P(\mathcal{O} \cap X \cap \lambda)}{P(X \cap \lambda)} \cdot \frac{P(X \cap \lambda)}{P(\lambda)} = \frac{P(\mathcal{O} \cap X \cap \lambda)}{P(\lambda)},$$

we have

$$P(\mathcal{O}, X | \lambda) = P(\mathcal{O} | X, \lambda) P(X | \lambda).$$

By summing over all possible state sequences, we obtain

$$\begin{aligned} P(\mathcal{O} | \lambda) &= \sum_X P(\mathcal{O}, X | \lambda) \\ &= \sum_X P(\mathcal{O} | X, \lambda) P(X | \lambda) \\ &= \sum_X \pi_{x_0} b_{x_0}(\mathcal{O}_0) a_{x_0, x_1} b_{x_1}(\mathcal{O}_1) \cdots a_{x_{T-2}, x_{T-1}} b_{x_{T-1}}(\mathcal{O}_{T-1}). \end{aligned} \tag{2}$$

The direct probability calculation in (2) is generally infeasible, since it requires about $2TN^T$ multiplications, that is, it is exponential in the length of the observation sequence. To efficiently compute $P(\mathcal{O} | \lambda)$, the *forward algorithm* (also known as the α -pass) can be used. For $t=0, 1, \dots, T-1$ and $i=0, 1, \dots, N-1$, define

$$\alpha_t(i) = P(\mathcal{O}_0, \mathcal{O}_1, \dots, \mathcal{O}_t, x_t = q_i | \lambda). \quad (3)$$

Then, $\alpha_t(i)$ is the probability of the partial observation sequence up to time t , where the underlying Markov process is in state q_i at time t .

The crucial insight is that the $\alpha_t(i)$ can be computed recursively as follows.

- (1) Let $\alpha_0(i) = \pi_i b_i(\mathcal{O}_0)$, for $i=0, 1, \dots, N-1$.
- (2) For $t=1, 2, \dots, T-1$ and $i=0, 1, \dots, N-1$, compute

$$\alpha_t(i) = \left(\sum_{j=0}^{N-1} \alpha_{t-1}(j) a_{ji} \right) b_i(\mathcal{O}_t).$$

- (3) Then from (3), it is clear that

$$P(\mathcal{O} | \lambda) = \sum_{i=0}^{N-1} \alpha_{T-1}(i).$$

The forward algorithm only requires about N^2T multiplications. In particular, the forward algorithm is linear in the length of the observation sequence.

2.2.2. Solution to Problem 2

Given the model $\lambda = (A, B, \pi)$ and a sequence of observations \mathcal{O} , our goal is to find the most likely state sequence. As mentioned above, there are different possible interpretations of “most likely.” For HMMs, we want to maximize the expected number of correct states (in contrast to a dynamic program, which finds the highest scoring overall path).

First, we define the *backward algorithm*, or β -pass. This is analogous to the α -pass discussed above, except that it starts at the end and works back toward the beginning.

For $t=0, 1, \dots, T-1$ and $i=0, 1, \dots, N-1$, define

$$\beta_t(i) = P(\mathcal{O}_{t+1}, \mathcal{O}_{t+2}, \dots, \mathcal{O}_{T-1} | x_t = q_i, \lambda).$$

Then, the $\beta_t(i)$ can be computed recursively (and efficiently) as follows.

- (1) Let $\beta_{T-1}(i) = 1$, for $i=0, 1, \dots, N-1$.
- (2) For $t=T-2, T-3, \dots, 0$ and $i=0, 1, \dots, N-1$ compute

$$\beta_t(i) = \sum_{j=0}^{N-1} a_{ij} b_j(\mathcal{O}_{t+1}) \beta_{t+1}(j).$$

For $t=0, 1, \dots, T-1$ and $i=0, 1, \dots, N-1$, define

$$\gamma_t(i) = P(x_t = q_i | \mathcal{O}, \lambda).$$

Since $\alpha_t(i)$ measures the relevant probability up to time t , and $\beta_t(i)$ measures the relevant probability after time t , we use a meet-in-the-middle approach to compute

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(\mathcal{O} | \lambda)}$$

for $t=0, 1, \dots, T-1$. Recall that the denominator $P(\mathcal{O} | \lambda)$ is obtained by summing $\alpha_{T-1}(i)$ over i . From the definition of $\gamma_t(i)$, it follows that the most likely state at time t is the state q_i for which $\gamma_t(i)$ is maximum, where this maximum is taken over i .

2.2.3. Solution to Problem 3

Here, we want to adjust the model parameters to best fit the observations. The sizes of the matrices (N and M) are fixed, and a training sequence \mathcal{O} is given; the elements of A , B and π are to be determined, subject to row stochastic conditions. The fact that we can efficiently re-estimate the model itself is perhaps the most impressive aspect of HMMs.

For $t=0, 1, \dots, T-2$ and $i, j \in \{0, 1, \dots, N-1\}$, define “di-gammas” as

$$\gamma_t(i, j) = P(x_t = q_i, x_{t+1} = q_j | \mathcal{O}, \lambda).$$

Then $\gamma_t(i, j)$ is the probability of being in state q_i at time t and transiting to state q_j at time $t+1$. The di-gammas can be written in terms of α , β , A and B as

$$\gamma_t(i, j) = \frac{\alpha_t(i)a_{ij}b_j(\mathcal{O}_{t+1})\beta_{t+1}(j)}{P(\mathcal{O} | \lambda)}.$$

For $t=0, 1, \dots, T-2$, the $\gamma_t(i)$ and $\gamma_t(i, j)$ are related by

$$\gamma_t(i) = \sum_{j=0}^{N-1} \gamma_t(i, j).$$

Given the γ and di-gamma, the model $\lambda = (A, B, \pi)$ can be re-estimated as follows.

(1) For $i=0, 1, \dots, N-1$, let

$$\pi_i = \gamma_0(i). \tag{4}$$

(2) For $i=0, 1, \dots, N-1$ and $j=0, 1, \dots, N-1$, compute

$$a_{ij} = \sum_{t=0}^{T-2} \gamma_t(i, j) / \sum_{t=0}^{T-2} \gamma_t(i). \tag{5}$$

(3) For $j = 0, 1, \dots, N - 1$ and $k = 0, 1, \dots, M - 1$, compute

$$b_j(k) = \sum_{\substack{t \in \{0, 1, \dots, T-1\} \\ \mathcal{O}_t = k}} \gamma_t(j) / \sum_{t=0}^{T-1} \gamma_t(j). \quad (6)$$

The numerator of each re-estimated a_{ij} gives the expected number of transitions from state q_i to state q_j , while the denominator is the expected number of transitions from q_i to any state. The ratio is the probability of transiting from state q_i to state q_j , which is the desired value of a_{ij} .

The numerator of the re-estimated $b_j(k)$ is the expected number of times the model is in state q_j with observation k , while the denominator is the expected number of times the model is in state q_j . The ratio is the probability of observing symbol k , given that the model is in state q_j , which is the desired value of $b_j(k)$.

Re-estimation is an iterative process. First, we initialize $\lambda = (A, B, \pi)$ with a best guess or, if no reasonable guess is available, we choose random values such that $\pi_i \approx 1/N$, and $a_{ij} \approx 1/N$, and $b_j(k) \approx 1/M$. It is essential that A , B , and π be randomized since precisely uniform values will result in a local maximum from which the model cannot climb. As always, π , A , and B must be row stochastic.

The solution to Problem 3 can be summarized as follows.

- (1) Initialize $\lambda = (A, B, \pi)$.
- (2) Compute $\alpha_t(i)$, $\beta_t(i)$, $\gamma_t(i, j)$, and $\gamma_t(i)$.
- (3) Re-estimate the model $\lambda = (A, B, \pi)$.
- (4) If $P(\mathcal{O} | \lambda)$ increases, go to 2.

Of course, it might be desirable to stop when $P(\mathcal{O} | \lambda)$ does not increase by at least some predetermined threshold and/or to set a maximum number of iterations (in practice, we use a fixed number of iterations). The process as described here is known as Baum-Welch re-estimation (Rabiner 1989).

These three HMM solutions all require computations involving products of probabilities. It is easy to see, for example, that $\alpha_t(i)$ tends to 0 exponentially as t increases. Therefore, any attempt to implement the formulae as given above will inevitably result in underflow. The solution to this underflow problem is to scale the numbers. Scaling is absolutely essential in practice, and it is not quite as straightforward as it might seem. For the sake of brevity, we do not further discuss scaling here; see Stamp (2012) for additional details including pseudocode.

2.3. English text example

In this section, we discuss a classic application of HMMs, which appears to have originated with Cave and Neuwirth (1980). This application nicely illustrates the strength of HMMs and has the additional advantage that it

requires no background in any specialized field such as speech processing. It will also lead us directly into the realm of classic cryptanalysis.

Suppose that Marvin the Martian (IMDB) obtains a large body of English text, such as the Brown Corpus, which totals about 1,000,000 words. Marvin, who has a working knowledge of HMMs but no knowledge of English, would like to determine basic properties of this mysterious writing system. A reasonable question he might ask is whether the characters can be partitioned into sets so that the characters in each set are “different” in some significant way.

Marvin might consider attempting the following. First, remove all punctuation, numbers, and so on, and convert all letters to lowercase. This leaves 26 distinct letters and word-space, for a total of 27 symbols. He could then test the hypothesis that there is an underlying Markov process (of order one) with two states. For each of these two hidden states, he assumes that the 27 symbols are observed according to fixed probability distributions.

This defines an HMM with $N=2$ and $M=27$, where the state transition probabilities of the A matrix and the observation probabilities contained in the B matrix are unknown, while the observations are the series of characters found in the text. To find the A and B matrices, Marvin must solve Problem 3, as discussed above in Section 2.2.3.

We conducted this experiment using $T=50,000$ observations (letters converted to lowercase, plus word-spaces) extracted from the Brown Corpus. We initialized each element of π and A randomly to approximately $1/2$. The precise values used were

$$\pi = (0.51316 \quad 0.48684)$$

and

$$A = \begin{pmatrix} 0.47468 & 0.52532 \\ 0.51656 & 0.48344 \end{pmatrix}.$$

Each element of B was initialized to approximately $1/27$. The precise values in the initial B matrix (actually, the transpose of B) appear in the second and third columns of [Table 2](#).

After the initial iteration, we have

$$\log(P(\mathcal{O} | \lambda)) = -165,097.29.$$

After 100 iterations, we have

$$\log(P(\mathcal{O} | \lambda)) = -137,305.28,$$

which shows that the model has improved significantly.

After 100 iterations, the model $\lambda = (A, B, \pi)$ has converged to

$$\pi = (0.00000 \quad 1.00000)$$

Table 2. Initial and final B transpose.

Letter	Initial		Final	
A	0.03735	0.03909	0.13845	0.00075
B	0.03408	0.03537	0.00000	0.02311
C	0.03455	0.03537	0.00062	0.05614
D	0.03828	0.03909	0.00000	0.06937
E	0.03782	0.03583	0.21404	0.00000
F	0.03922	0.03630	0.00000	0.03559
G	0.03688	0.04048	0.00081	0.02724
H	0.03408	0.03537	0.00066	0.07278
I	0.03875	0.03816	0.12275	0.00000
J	0.04062	0.03909	0.00000	0.00365
K	0.03735	0.03490	0.00182	0.00703
L	0.03968	0.03723	0.00049	0.07231
M	0.03548	0.03537	0.00000	0.03889
N	0.03735	0.03909	0.00000	0.11461
O	0.04062	0.03397	0.13156	0.00000
P	0.03595	0.03397	0.00040	0.03674
Q	0.03641	0.03816	0.00000	0.00153
R	0.03408	0.03676	0.00000	0.10225
S	0.04062	0.04048	0.00000	0.11042
T	0.03548	0.03443	0.01102	0.14392
U	0.03922	0.03537	0.04508	0.00000
V	0.04062	0.03955	0.00000	0.01621
W	0.03455	0.03816	0.00000	0.02303
X	0.03595	0.03723	0.00000	0.00447
Y	0.03408	0.03769	0.00019	0.02587
Z	0.03408	0.03955	0.00000	0.00110
space	0.03688	0.03397	0.33211	0.01298

and

$$A = \begin{pmatrix} 0.25596 & 0.74404 \\ 0.71571 & 0.28429 \end{pmatrix}$$

with the transpose of B appearing in the last two columns in [Table 2](#).²

The most interesting part of this result is the B matrix. Without having made any assumption about the two hidden states, the B matrix tells us that one of the two hidden states corresponds to vowels, while the other hidden state corresponds to consonants. Curiously, word-space acts more like a vowel, while “Y” is not even sometimes a vowel. Anyone familiar with English would not be surprised that there is a clear distinction between vowels and consonants, but the HMM results show us that this distinction is a statistically significant feature inherent in the language. Thanks to HMMs, this could easily be deduced by someone, such as Marvin, who has no background knowledge of the language.

Cave and Neuwirth (1980) obtained further interesting results when considering more than two hidden states. In fact, they were able to sensibly interpret the results for models with up to $N = 12$ hidden states.

²It is worth noting that the model need not converge in every case since we are only assured of finding a local maximum. This topic is addressed in more detail in Section 3.2, where we consider the effect of multiple random restarts.

The generalization to a simple substitution cipher is straightforward. We consider this topic in the next section, and we show that by using multiple random restarts, we can obtain good results using far fewer observations as compared to a more straightforward attack.

3. Cryptanalytic results

In this section, we focus on the application of HMMs to simple substitution cryptanalysis. Our emphasis here is on the effectiveness of multiple random restarts in the HMM hill climb algorithm. First, we discuss Jakobsen's algorithm (1995), which provides a fast and effective hill climb attack on a simple substitution, and will serve as a benchmark for our HMM results.

3.1. Jakobsen's algorithm

A simple substitution cipher uses a fixed one-to-one substitution to encrypt a given message. For example, to encrypt an English message, where the plaintext consists of only, say, uppercase A through Z, we could use a fixed permutation of the alphabet as the key.

In the case of English text, we denote the key as

$$K = k_1, k_2, k_3, \dots, k_{26} \quad (7)$$

where each k_i maps one letter of plaintext to one letter of ciphertext. For example, the key

$$K = (\text{DEFGHIJKLMNOPQRSTUVWXYZABC})$$

maps each plaintext letter to the letter three positions ahead in the alphabet, which is the well-known Caesar's cipher.

It is a standard textbook problem to show that a simple substitution can be broken based on frequency analysis. The distribution of plaintext letters in English is highly non-uniform, and since the mapping from plaintext to ciphertext is fixed, we can deduce information about the key directly from the ciphertext. For example, the letter E is by far the most common letter in English, accounting for more than 12% of the characters in typical text. Hence, the most common letter in a simple substitution ciphertext likely corresponds to E in plaintext.

When attempting to develop an automated hill-climb attack on a simple substitution, the following issues must be addressed:

- (1) How do we select an initial putative key?
- (2) Given a putative key, how do we compute a score?
- (3) How do we systematically modify the current putative key?

For the initial key, the best available information comes from the mono-graph statistics of the ciphertext. Therefore, we map the most frequent letter

There are many possible ways to systematically modify the putative key. Since a simple substitution key is a permutation, swapping elements of the key will always yield another valid key. In Jakobsen’s algorithm, only swaps are used to modify the key. Specifically, given a putative key K as in (7), Jakobsen’s algorithm modifies the key by swapping elements according to the “swap schedule”

where “|” indicates a swap. That is, we swap adjacent elements, then elements at distance 2, then elements at distance 3, and so on. For all experiments considered below, we ignore punctuation, word-space, and case, so that $N=26$ in (8).

Finally, we must specify a scoring function. Let $E = \{e_{ij}\}$ be a 26×26 matrix containing digraph frequencies of English text. For example, $e_{1,1}$ contains the frequency of the digraph AA in a large sample of English text, while $e_{20,8}$ is the frequency of the digraph TH. Figure 2 contains a heatmap of expected digraph frequencies in English. Jakobsen’s algorithm relies on the highly non-uniform distribution of these digraphs for its success and simplicity.

$$\text{score}(D, E) = \sum_{i,j} |d_{i,j} - e_{i,j}|. \quad (9)$$

letters	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A		0.19	0.38	0.33		0.10	0.19	0.04	0.29	0.01	0.09	0.85	0.29	0.59	0.29	0.17	0.00	0.98	0.82	0.15	0.10	0.18	0.08	0.02	0.26	0.01
B	0.15		0.01	0.00	0.18	0.01	0.01	0.00	0.00	0.01	0.00	0.19	0.00	0.00	0.18	0.00	0.00	0.00	0.00	0.01	0.17	0.00	0.00	0.00	0.33	0.00
C	0.43	0.01		0.06	0.01	0.49	0.00	0.00	0.52	0.21	0.00	0.12	0.12	0.01	0.00	0.64	0.01	0.00	0.12	0.01	0.30	0.00	0.00	0.01	0.00	0.00
D	0.39	0.14	0.08		0.08	0.63	0.09	0.06	0.10	0.47	0.02	0.01	0.07	0.11	0.06	0.30	0.07	0.01	0.13	0.23	0.38	0.12	0.03	0.11	0.00	0.06
E	0.01	0.22	0.61	0.02		0.46	0.30	0.18	0.19	0.30	0.03	0.06	0.53	0.48	0.25	0.33	0.36	0.04	0.75	0.30	0.77	0.09	0.24	0.36	0.14	0.15
F	0.23	0.02	0.05	0.00	0.19		0.13	0.02	0.01	0.25	0.01	0.01	0.06	0.01	0.02	0.42	0.00	0.18	0.05	0.36	0.08	0.00	0.03	0.03	0.01	0.00
G	0.21	0.02	0.03	0.00	0.31	0.01		0.03	0.22	0.18	0.00	0.00	0.06	0.03	0.06	0.19	0.02	0.00	0.19	0.07	0.15	0.06	0.00	0.03	0.03	0.01
H	0.84	0.02	0.04	0.01	2.42	0.02	0.01		0.03	0.66	0.00	0.00	0.02	0.03	0.04	0.48	0.02	0.00	0.10	0.05	0.21	0.08	0.01	0.04	0.03	0.03
I	0.23	0.08	0.56	0.27	0.30	0.13	0.21	0.01		0.01	0.00	0.04	0.40	0.22	1.81	0.56	0.07	0.01	0.26	0.94	0.89	0.01	0.22	0.01	0.01	0.00
J	0.03	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.01		0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
K	0.01	0.01	0.01	0.01	0.21	0.01	0.00	0.00	0.02	0.09	0.00	0.00	0.02	0.01	0.04	0.00	0.01	0.00	0.01	0.06	0.00	0.00	0.00	0.00	0.00	0.00
L	0.50	0.06	0.06	0.27	0.70	0.07	0.02	0.00	0.54	0.00	0.02		0.58	0.06	0.02	0.33	0.06	0.00	0.04	0.17	0.15	0.10	0.03	0.04	0.00	0.36
M	0.50	0.11	0.00	0.00	0.63	0.00	0.00	0.00	0.30	0.00	0.00	0.11		0.00	0.32	0.17	0.00	0.07	0.00	0.07	0.11	0.00	0.02	0.00	0.00	0.00
N	0.53	0.08	0.35	0.00	0.64	0.11	0.80	0.10	0.44	0.02	0.05	0.09	0.09	0.13	0.49	0.07	0.01	0.05	0.50	1.39	0.09	0.05	0.12	0.00	0.10	0.00
O	0.14	0.14	0.16	0.18	0.00	0.86	0.09	0.07	0.10	0.01	0.06	0.34	0.49	1.36	0.22	0.22	0.00	0.03	0.31	0.46	0.70	0.17	0.30	0.01	0.04	0.00
P	0.27	0.01	0.01	0.00	0.35	0.01	0.00	0.07	0.12	0.00	0.00	0.20	0.12	0.00	0.28	0.11	0.00	0.36	0.00	0.09	0.08	0.00	0.00	0.00	0.00	0.00
Q	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
R	0.65	0.06	0.16	0.21	1.43	0.08	0.10	0.08	0.64	0.01	0.10	0.13	0.19	0.17	0.64	0.09	0.00	0.12	0.50	0.51	0.13	0.06	0.08	0.00	0.20	0.00
S	0.66	0.14	0.24	0.09	0.73	0.13	0.05	0.36	0.65	0.02	0.05	0.12	0.16	0.10	0.56	0.26	0.01	0.07	0.48	1.31	0.24	0.03	0.21	0.03	0.05	0.00
T	0.63	0.09	0.11	0.00	0.97	0.08	0.03	2.89	1.19	0.01	0.01	0.14	0.10	0.05	1.93	0.07	0.00	0.35	0.40	0.49	0.20	0.01	0.19	0.00	0.20	0.00
U	0.09	0.08	0.13	0.08	0.11	0.00	0.10	0.00	0.07	0.00	0.00	0.00	0.26	0.10	0.37	0.01	0.11	0.00	0.38	0.37	0.34	0.00	0.00	0.00	0.00	0.01
V	0.09	0.00	0.00	0.00	0.65	0.00	0.00	0.00	0.22	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
W	0.32	0.01	0.01	0.01	0.31	0.01	0.00	0.32	0.33	0.00	0.00	0.01	0.02	0.06	0.21	0.01	0.00	0.03	0.04	0.03	0.00	0.00	0.00	0.00	0.02	0.00
X	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Y	0.19	0.07	0.07	0.03	0.14	0.06	0.02	0.07	0.12	0.01	0.01	0.04	0.07	0.04	0.18	0.05	0.00	0.04	0.17	0.20	0.01	0.01	0.09	0.00	0.01	0.00
Z	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Figure 2. English digraph relative frequencies (as percentages).

Note that $\text{score}(D, E) \geq 0$, and a perfect match yields $\text{score}(D, E) = 0$. Consequently, the lower the score, the better.

The beauty of Jakobsen's algorithm derives from the fact that swaps of elements of the putative key K correspond to swaps in the digraph distribution matrix D . Hence, we only have to decrypt the message once; all subsequent scoring computations require nothing more than elementary matrix manipulations. We now illustrate this swapping process using a small simple substitution example.

Consider an alphabet consisting of only the eight letters

E H I K L R S T

Suppose that we are given the ciphertext

$$\text{HTHEIHEILIRKSHEIRLHKTISRKRSIIKLEHTTRLHKTIS} \quad (10)$$

which was encrypted using a simple substitution on this eight-letter alphabet. The frequency counts for the ciphertext in (10) are given by

E	H	I	K	L	R	S	T
4	7	9	5	4	5	4	5

Assuming that these letters appear in plaintext at the same relative frequencies as in ordinary English, listing the letters from most frequent to least frequent, we have

E T I S H R L K

Matching the expected frequencies of the plaintext to the observed frequencies of the ciphertext, we obtain the initial putative key

plaintext	H T E I K S L R	(11)
ciphertext	E H I K L R S T	

Using this putative key, the ciphertext in (10) yields the putative plaintext

TRTHETHEKESILTHESKTIRELSSILEEIKEHTRRSKTIREL

From this putative plaintext, we obtain D , the digram frequency matrix

	E	H	I	K	L	R	S	T
E	1	1	1	1	2	0	2	1
H	3	0	0	0	0	0	0	1
I	0	0	0	1	2	2	0	0
K	2	0	0	0	0	0	0	2
L	1	0	0	0	0	0	1	1
R	2	0	0	0	0	1	1	1
S	0	0	2	2	0	0	1	0
T	0	3	2	0	0	2	0	0

(12)

The next step in Jakobsen’s algorithm is to perform a swap on the putative key. In the first step, we swap the roles of E and H in the “plaintext” line of (11) to obtain the putative key

plaintext	E T H I K S L R
ciphertext	E H I K L R S T

Using this putative key, the ciphertext decrypts to

TRTEHTEHKHSILTEHSKTIRHLSSILHHIKHETRRSKTIRHL

and this putative plaintext yields the digram frequency matrix D' given by

	E	H	I	K	L	R	S	T
E	0	3	0	0	0	0	0	1
H	1	1	1	1	2	0	2	1
I	0	0	0	1	2	2	0	0
K	0	2	0	0	0	0	0	2
L	0	1	0	0	0	0	1	1
R	0	2	0	0	0	1	1	1
S	0	0	2	2	0	0	1	0
T	3	0	2	0	0	2	0	0

(13)

Observe that the updated D' matrix in (13) can be obtained from the D matrix in (12) by simply swapping the first two rows and the first two columns. Therefore, it is not necessary to decrypt the message to determine the updated matrix D' since it can be obtained directly from D .

In general, if we swap elements i and j of the putative key, we can swap rows i and j , and columns i and j of the D matrix to obtain the digraph

distribution matrix for the corresponding putative decryption. This accounts for the efficiency of Jakobsen's algorithm: We decrypt once, and the remainder of the attack only involves matrix manipulations and score computations.

Jakobsen's algorithm is given in Table 3. Note that in Table 3, we use the notation $\text{swap}_K(k_i, k_j)$ to denote the putative key obtained by swapping elements in the i^{th} and j^{th} positions of K , whereas we use $\text{swap}_D(i, j)$ to denote the matrix obtained by swapping the i^{th} and j^{th} rows and columns of D .

Results obtained using Jakobsen's algorithm are given in Figure 3. Each result in Figure 3 is an average of 1,000 test cases, with plaintext selected from the Brown Corpus and keys generated at random.

We have observed that when about 80% of the characters have been recovered, the remaining 20% are easily determined from context. Therefore, we consider accuracy of 80% or more as a success. From Figure 3, we see that for this level of accuracy, Jakobsen's algorithm requires about 400 characters of ciphertext

3.2. HMMs with random restarts

Our goal in this section is to illustrate the effectiveness of multiple random restarts on HMMs, within the context of solving simple substitution ciphers. In the next section, we briefly consider a few other classic cryptanalysis problems where this technique can play a role.

We claim that a simple substitution cipher can be solved using an analogous approach as that used in the English text example of Section 2.3.

Table 3. Jakobsen's algorithm.

```
//
// Given: Matrix  $E$  of digraph statistics and ciphertext  $C$ 
//
let  $N = 26$ 
let  $K = (k_1, k_2, \dots, k_N)$  // initial key obtained from  $C$ 
decrypt  $C$  with  $K$  and compute digraph matrix  $D$ 
let  $s = \text{score}(D, E)$  // score as defined in (9)
let  $a = 1, b = 1$ 
while  $b < N$  do
   $i = a, j = a + b$ 
   $K' = \text{swap}_K(k_i, k_j), D' = \text{swap}_D(i, j), s' = \text{score}(D', E)$ 
  if  $s' < s$  then // score improved
     $s = s', K = K', D = D'$  // update
     $a = 1, b = 1$  // reset swapping
  else // get next key swap
     $a = a + 1$ 
    if  $a + b > N$  then
       $a = 1, b = b + 1$  // next row of (8)
    end if
  end if
end while
return  $K$ 
```

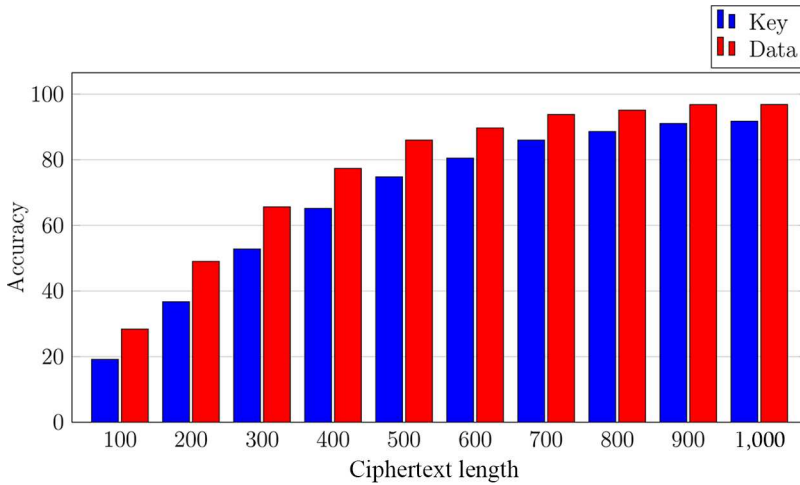


Figure 3. Jakobsen's algorithm.

For example, if we train an HMM with two hidden states using ciphertext from a simple substitution as the observation sequence, then we obtain a B matrix that effectively maps the ciphertext letters to consonants and vowels. This would certainly be an aid to solving a simple substitution, but we can do much better.

Instead of using $N=2$ hidden states in our HMM, we might try $N=n$ hidden states, where n is the number of distinct plaintext symbols. It seems reasonable that in this case, the underlying Markov process would correspond to the transitions between individual letters in the (hidden) plaintext. If this is correct, then the resulting B matrix would, in a probabilistic sense, reveal the connection between the plaintext symbols and the ciphertext symbols, that is, the B matrix would effectively give us the key. Furthermore, if we know the plaintext language, then we could fix the A matrix from the start, reducing the amount of data required by the HMM hill climb.

3.2.1. A simplified example

As with the presentation of Jakobsen's algorithm in Section 3.1, before giving our simple substitution results, we present an example using a restricted alphabet. From the Brown Corpus, we extracted words that have all of their letters among the eight-letter alphabet

$$E T I S H R L K \quad (14)$$

We appended all of these words to obtain a sequence of about 370,000 letters. Then, we initialize each digraph count to 5 (i.e., we use a pseudocount of 5) before computing digraph frequencies from this sequence. Finally, we normalize each element by its corresponding row sum to obtain the A matrix

	E	H	I	K	L	R	S	T
E	0.0202	0.1234	0.1613	0.0018	0.0265	0.0624	0.0607	0.5436
H	0.8784	0.0003	0.1145	0.0001	0.0001	0.0056	0.0001	0.0009
I	0.0073	0.0139	0.0362	0.0303	0.0283	0.0643	0.5191	0.3006
K	0.7582	0.0149	0.1364	0.0056	0.0050	0.0037	0.0143	0.0620
L	0.2764	0.0324	0.3412	0.0041	0.1939	0.0012	0.0407	0.1102
R	0.3780	0.0953	0.1085	0.0027	0.0149	0.0040	0.0824	0.3142
S	0.0955	0.2044	0.1464	0.0047	0.0181	0.0016	0.0403	0.4889
T	0.0106	0.8513	0.0379	0.0002	0.0116	0.0047	0.0251	0.0586

(15)

For example, based on the digraphs in this eight-letter “language,” we see that HE is the most common digraph beginning with H, while HI is the next most common, and the others are all very uncommon. Note that in this formulation, the A matrix is essentially the same as the E matrix in Jakobsen’s algorithm, except that A is subject to the row stochastic condition, as required in an HMM.

We conducted an experiment using a plaintext “message” composed of words from the eight-letter alphabet in (14). For this experiment, the plaintext has 1,000 characters, and we have $M = 8$ since the eight letters in the restricted alphabet are the observations. We choose $N = 8$ hidden states, and we assume that the hidden states correspond to the plaintext letters. Therefore, we initialize A using the matrix in (15), and A is not re-estimated during training.

Recall that A and B are row stochastic. We initialize B to be approximately uniform; in this case, each element of B is approximately $1/8$, subject to the row-stochastic condition. For this particular example, we have initialized the transpose of B as

		plaintext (hidden)							
		E	H	I	K	L	R	S	T
ciphertext	E	0.1262	0.1347	0.1182	0.1188	0.1327	0.1196	0.1216	0.1151
	H	0.1308	0.1173	0.1353	0.1236	0.1312	0.1276	0.1246	0.1278
	I	0.1185	0.1173	0.1151	0.1380	0.1296	0.1180	0.1276	0.1309
	K	0.1231	0.1189	0.1198	0.1252	0.1219	0.1324	0.1306	0.1262
	L	0.1308	0.1363	0.1244	0.1172	0.1127	0.1308	0.1201	0.1262
	R	0.1215	0.1268	0.1306	0.1396	0.1235	0.1212	0.1216	0.1230
	S	0.1308	0.1268	0.1260	0.1188	0.1235	0.1340	0.1231	0.1246
	T	0.1185	0.1220	0.1306	0.1188	0.1250	0.1164	0.1306	0.1262

After 200 iterations of the Baum-Welch re-estimation algorithm (see Section 2.2.3), we find that B transpose converges to

		plaintext (hidden)							
		E	H	I	K	L	R	S	T
ciphertext	E	0.0277	0.0041	0.0000	0.0000	0.0000	0.4793	0.0000	0.0000
	H	0.0000	0.0000	0.0000	0.0000	0.0559	0.2330	0.7789	0.0000
	I	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.2211	1.0000
	K	0.9723	0.0000	0.0000	0.0000	0.0464	0.2877	0.0000	0.0000
	L	0.0000	0.9850	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	R	0.0000	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	S	0.0000	0.0000	0.0000	0.7968	0.0000	0.0000	0.0000	0.0000
	T	0.0000	0.0109	0.0000	0.2032	0.8978	0.0000	0.0000	0.0000

(16)

Recall that we denote the element in row i and column j of B as $b_i(j)$ and that $b_i(j)$ gives the probability of observation j when the model is in hidden state i . By our choice of A , we assume that the hidden states correspond to the plaintext letters E, H, I, K, L, R, S, T, while (as noted above,) the probabilities in B relate the observations (i.e., ciphertext) to the plaintext. From the definition of $b_i(j)$, we see that $\operatorname{argmax}_j b_i(j)$ gives us the most likely mapping of ciphertext i to plaintext. That is, the largest probability in row i of B (equivalently, column i in B transpose) gives us the (encryption) mapping of the plaintext symbol corresponding to i . Hence, we can deduce for this example, the most likely key is given by the boxed entries in (16), that is

plaintext	E H I K L R S T
ciphertext	K L R S T E H I

In this example, a Caesar’s cipher was indeed used.

Among other factors, convergence depends on the initial values selected for the HMM matrices, and hence the model may not converge in every case. In the next section, we discuss extensive experiments using HMMs for simple substitution cryptanalysis, and we show that multiple random restarts can greatly improve the performance. Then, in the subsequent section, we mention a few other classic cryptographic applications where this same approach may be useful.

3.2.2. *Simple substitutions*

Recall that the HMM training algorithm is a hill climb technique. As with any hill climb, an HMM can only find a local maximum, and it will only find a global maximum if we start sufficiently close to a global maximum. Consequently, we might find a better solution by simply running the algorithm again with a different initialization. That is, by making multiple random restarts, we will likely climb different hills, and the “highest” of those hills will provide our best solution. As mentioned above, this strategy is explored in Berg-Kirkpatrick and Klein (2013) as a method to analyze the unsolved Zodiac 340 cipher.

All experiments in this section were performed on English text selected from the Brown Corpus, using 26 letters, (i.e., no word space, case, or punctuation). The encryption permutations were selected at random.

For simplicity, when training the HMMs, we perform a fixed number of iterations, and We experimented with various numbers of iterations, and the results for 100, 200, and 500 iterations are given in Tables 4, 5, and 6, respectively. In general, 200 iterations seems to give us nearly optimal results, with 500 offering little or no improvement. Therefore, in the remainder of the experiments, we use 200 iterations unless otherwise stated.

Table 4. Accuracy with 100 iterations.

Restarts	Data size							
	100	200	300	400	600	800	1,000	1,200
1	2.86	3.47	5.09	6.01	13.64	18.12	25.59	31.06
10	12.70	15.34	23.69	34.22	61.89	72.06	80.75	86.41
100	20.58	31.88	58.25	71.22	85.02	87.69	90.41	92.32
1,000	23.14	44.37	81.94	87.35	91.87	92.01	94.55	98.61
10,000	24.20	52.45	91.57	92.63	94.65	94.61	96.67	99.75
100,000	26.00	63.00	95.00	96.75	99.50	96.88	98.70	99.75

For the experiments discussed in this section, the A matrix is initialized based on English digraph statistics, and it remains fixed throughout. The elements of the B matrix are initialized at random so that the row stochastic condition holds. The distribution of the initial $b_j(k)$ has mean $1/26 \approx 0.0385$ with a standard deviation of about 0.002, and hence the initial $b_j(k)$ tend to cluster closely around $1/26$.

In Figure 4, we plot accuracy versus the length of the ciphertext, where the number of random restarts varies from 1 (i.e., a single iteration) to 10^5 random restarts. We observe a rapid improvement in the results up to about 1,000 restarts, beyond which there is relatively modest improvement. Nevertheless, for very short messages, it might be worth testing 10^5 or more restarts.

As mentioned in Section 3.1, an accuracy of 80% is sufficient to recover a message. From Figure 4, we see that this level of accuracy can be exceeded with 1,000 restarts on a message of length 300. Furthermore, a message of length 200 can nearly be solved with 10^5 random restarts. This is in contrast to Jakobsen's algorithm, where 80% accuracy requires a ciphertext of length greater than 400. The comparison between Jakobsen's algorithm and the HMM with 10^5 random restarts (and 200 iterations) is given in Figure 5. These results clearly show that the HMM is able to obtain considerably more statistical information from the data than Jakobsen's algorithm. It is worth noting that Jakobsen's and the HMM both rely only on digraph statistics.

In Figure 6, we give a 3-D plot of accuracy as a function of ciphertext length and number of restarts. Several additional experiments are presented in Vobbilisetty (2015).

Table 5. Accuracy with 200 iterations.

Restarts	Data size							
	100	200	300	400	600	800	1,000	1,200
1	2.80	3.56	5.27	6.70	14.42	18.68	28.20	33.28
10	12.62	16.22	24.70	38.25	62.93	71.90	84.23	86.85
100	20.36	34.67	61.76	77.99	83.39	84.92	90.32	91.81
1,000	23.56	48.21	83.85	90.55	89.29	89.65	94.91	98.07
10,000	27.00	60.10	90.53	93.58	94.88	93.10	97.83	99.74
100,000	37.00	69.00	94.33	94.50	99.50	94.38	99.70	99.75

Table 6. Accuracy with 500 iterations.

Restarts	Data size						
	100	200	300	400	600	800	1,000
1	2.83	3.67	5.55	7.52	15.24	19.71	29.11
10	12.76	17.22	26.86	42.52	66.18	72.71	84.24
100	20.65	38.00	69.29	82.41	82.73	84.69	88.71
1,000	23.42	53.93	88.18	92.19	85.69	89.13	92.97
10,000	25.30	68.75	92.00	94.35	91.60	92.16	96.18
100,000	32.00	74.00	95.00	96.00	95.17	94.50	98.70

Of course, the improvement obtained with large numbers of random restarts does not come for free. Each iteration of the HMM is comparable in cost to an entire run of Jakobsen’s algorithm. Therefore, using an HMM with large numbers of random restarts is only sensible for the most challenging cases, for which a fast algorithm (such as Jakobsen’s) is not sufficient. Our results indicate that simple substitution ciphertexts with fewer than about 400 characters are unlikely to be solved effectively by Jakobsen’s algorithm, but if we use enough random restarts, an HMM can solve such a problem with slightly more than 200 characters.

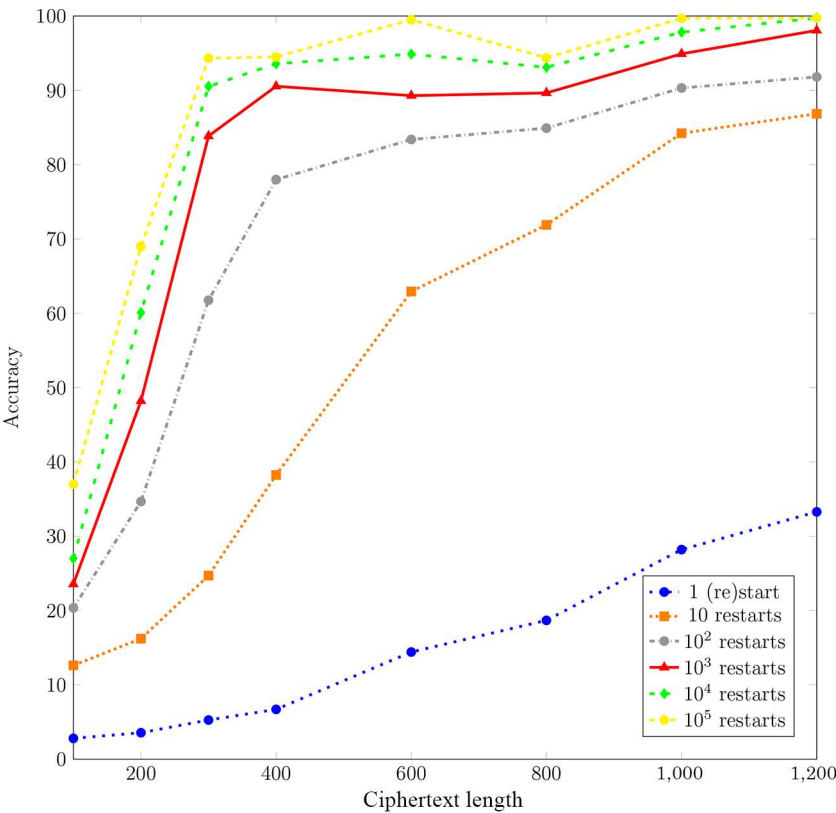


Figure 4. Accuracy vs. data size (200 iterations).

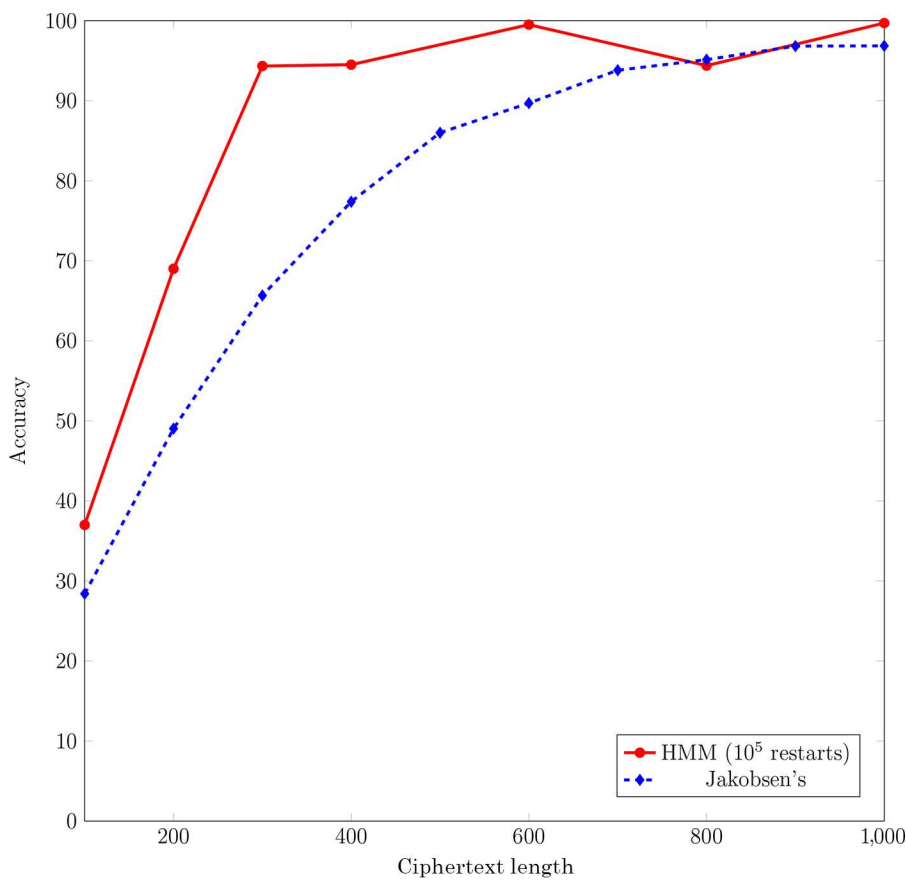


Figure 5. Jakobsen's vs. HMM.

3.2.3. Related topics

In this section, we briefly consider a few topics related to the HMM attack discussed in the previous section. While the list of topics we could consider is large, here we only mention a few examples.

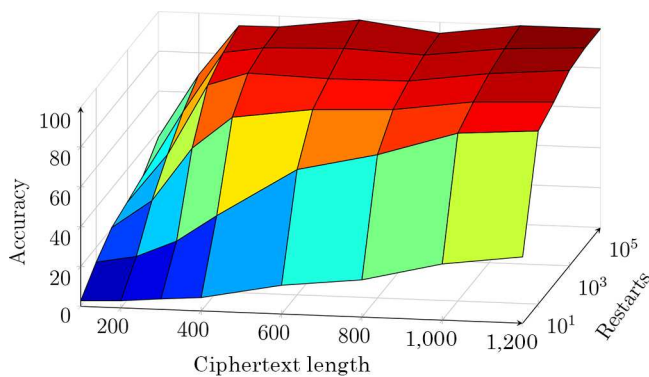


Figure 6. Accuracy vs. data size vs. restarts (200 iterations).

For the results presented in the previous section, the initial values of the B matrix were packed tightly around $1/26$ (standard deviation of 0.002). It would be reasonable to experiment with values that have a greater variance about the mean. Alternatively, we could attempt to mimic Jakobsen's algorithm and initialize the B matrix so that the entries are weighted based on the monograph statistics of the ciphertext. For example, suppose that the most frequent letter in the ciphertext is G. Then, we could initialize the element in the G row and E column of B to, say, $1/2$ and set each of the remaining 25 elements of the same row to approximately $1/50$, while maintaining the row stochastic property of B . We could also weight other rows in an analogous manner.

In Jakobsen's algorithm, the E matrix contains digraph statistics for the plaintext language. In the previous section, we used these same statistics to specify the A matrix in the HMM, and then the A matrix remained fixed throughout the attack. However, when training an HMM, the A matrix can be part of the hill climb. Consequently, in cases where we do not know the plaintext language, an HMM can still work; see the English text example in Section 2.3. In contrast, Jakobsen's algorithm is not applicable to such cases. Of course, more data will generally be required to obtain convergence of the HMM when A must be re-estimated.

Even in cases where the plaintext language is known, a particular message might not follow those statistics closely. In such cases, we could initialize the A matrix based on language statistics and then re-estimate A as part of the hill climb. This would enable the A matrix to be adjusted to better match the specific statistics of the underlying "language" as opposed to forcing the message to fit a generic language model. This approach is likely to be most valuable on short messages, which are precisely the cases where the HMM with multiple random restarts is most beneficial.

A homophonic substitution cipher is a generalization of a simple substitution where multiple ciphertext symbols can map to a single plaintext letter. For example, if several ciphertext symbols map to the letter E, then the English language statistics (both monograph and digraph) would be flattened, making a statistical attack much more difficult. In Dhavare and colleagues (2013), Jakobsen's algorithm is generalized to a homophonic substitution, using a nested hill climb: an outer hill climb on the distribution of plaintext letters within the ciphertext symbols, coupled with an inner hill climb consisting of Jakobsen's algorithm with multiple restarts for various mappings of plaintext letter, based on the assumed distribution from the outer layer. In general, the difficulty of a homophonic substitution depends on both the ciphertext length and the number of ciphertext symbols. For the Jakobsen-like approach in Dhavare and colleagues (2013), good results are obtained for a ciphertext alphabet of size 28 when the ciphertext is of length 500, but messages with a ciphertext alphabet of size 45 require messages in excess of 1,000 characters for any reasonable chance of success.

As previously mentioned, in Berg-Kirkpatrick and Klein (2013), a technique that is essentially equivalent to an HMM with random restarts (and fixed A matrix) is applied to the Zodiac 340 cipher. The solved Zodiac 420 cipher was a homophonic substitution with more than 60 symbols. The work presented in Berg-Kirkpatrick and Klein (2013) provides strong evidence that the Zodiac 340 is not a homophonic substitution (which is not too surprising given that it has remained unsolved for more than 40 years), while the Zodiac 420 was broken in a few days.

The HMM method discussed in Section 3.2 easily generalizes to the case of a homophonic substitution. The A matrix remains the same, that is, a 26×26 matrix of English language digraph statistics. The B matrix is $26 \times m$, where m is the number of distinct ciphertext symbols. As in the simple substitution, after training, the B matrix contains a probabilistic representation of the key. However, deducing the key from the B matrix is somewhat more challenging than in the simple substitution case since more than one element can be selected from each row.

An example of the final B transpose for a homophonic substitution is given in Table 7. As with previous examples, the English plaintext consists of the 26 uppercase letters. In this case, there are 29 distinct ciphertext symbols, which are denoted 0 through 28. From the results in Table 7, most of the key is obvious: The letter A maps to 3, and B maps to 4, and so on. Also, we can use the fact that uncommon letters are more likely to give ambiguous results to select the most likely key. For example, the numbers in the E column in Table 7 are likely to be far more meaningful than the numbers in the J, Q, or Z columns. For the example in Table 7, the actual key is

plaintext	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
ciphertext	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	0	1	2
					26															27						
					28																					

In this case, the key corresponds to a Caesar’s cipher, except when encrypting E, in which case we randomly choose from ciphertext symbols 7, 26, and 28, and when encrypting T, we randomly select either 22 or 27.

As a final example, we consider the WWII Japanese Purple cipher, which is illustrated in Figure 7. A detailed discussion of the Purple cipher is beyond the scope of this article; see Freeman and colleagues (2005) or Stamp and Low (2007) for additional information. For our purposes, the important points are that Purple employs a keyboard substitution, as represented by the plugboards in Figure 7, and time-varying “switched” permutations. The input plugboard effects a simple substitution, the output of which is split into a group of six letters and the remaining 20 letters. The so-called sixes pass through a time-varying permutation of period 25, and the twenties pass

Table 7. Example of B transpose for homophonic substitution.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.82	0.00	0.00
1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.98	0.00
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.32
3	0.97	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	0.00	0.87	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.17
6	0.00	0.00	0.00	1.00	0.00	0.00	0.08	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00
7	0.01	0.00	0.00	0.00	0.25	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	0.00	0.00	0.00	0.00	0.00	0.90	0.00	0.00	0.00	0.37	0.00	0.00	0.00	0.00	0.00	0.00	0.28	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.08
9	0.00	0.00	0.01	0.00	0.00	0.00	0.78	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.11	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.89	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00
11	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
12	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.63	0.00	0.00	0.00	0.00	0.00	0.00	0.41	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.43	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.09	0.00	0.00	0.00	0.00
14	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.99	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.01	0.00	0.00
15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.84	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.00	0.00
16	0.00	0.00	0.00	0.00	0.00	0.09	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.90	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
17	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
18	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.05	0.00	0.00	1.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.11	0.00	0.00
19	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.29	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	0.00	0.09	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.05	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
21	0.00	0.03	0.00	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.27
22	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.26	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.52	0.00	0.05	0.00	0.00	0.00	0.16
23	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
24	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.72	0.02	0.00	0.00	0.00
25	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.90	0.00	0.02	0.00	0.00
26	0.00	0.00	0.00	0.00	0.52	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
27	0.00	0.00	0.01	0.00	0.00	0.00	0.08	0.00	0.00	0.00	0.32	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.48	0.00	0.00	0.00	0.00	0.00	0.00
28	0.02	0.00	0.00	0.00	0.23	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

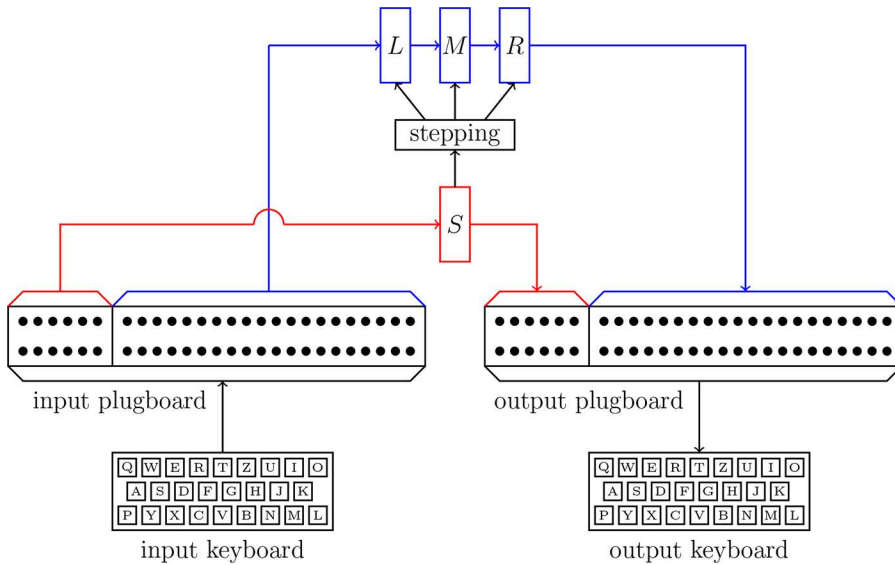


Figure 7. Purple cipher.

through a time-varying permutation of period about 25^3 . For a given ciphertext, the output plugboard uses the same simple substitution as the input plugboard.³ The key consists of the keyboard substitution and the initial positions in the sixes and twenties permutation sequences. The time-varying permutations are generated by switches (we omit the details of the switch stepping) as opposed to the more familiar rotors of Enigma and many other cipher machines of the same era. The crucial observation is that the vast majority of the Purple keyspace is contained in the keyboard substitution, which is a simple substitution.

To simplify somewhat, suppose we have a cipher, call it Lavender, that encrypts using a series of known permutations of the 26 letters, denoted as P_1, P_2, \dots, P_n . Suppose the Lavender cipher key consists of the choice of the initial point in the permutation sequence (i.e., 1 thru n) together with a keyboard permutation. As with Purple, this keyboard permutation is a simple substitution of the alphabet that is applied to the input before it is fed into the variable permutation P_i . The permutation sequence is stepped for each letter, that is, if plaintext p_i is encrypted using permutation P_m , then p_{i+1} is encrypted using permutation P_{m+1} . This Lavender cipher is essentially the Purple cipher, without the infamous 6-20 split.

Let A be the 26×26 matrix of English digraph statistics as used in the HMM attack on a simple substitution. Let A_i be the matrix A with its rows permuted according to P_i and its columns permuted by P_{i+1} , with A_n having its rows permuted by P_n and its columns permuted by P_1 .

³In the Purple cipher machine, the input and output plugboard permutations could differ, but in practice they were always chosen to be the same (Freeman et al. 2005).

Suppose that we know that, say, m is the initial point in the permutation sequence. Then, we can perform the HMM attack as in the simple substitution case, but instead of using a fixed A matrix, we use $A_m, A_{m+1}, A_{m+2}, \dots$, at observations $\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \dots$, respectively, where the observations are the ciphertext characters. As with the simple substitution attack, we obtain a B matrix that contains a probabilistic representation of the key—in this case, the keyboard permutation. If n is not too large (as in Purple) we could simply repeat this attack for each initial position 1 through n , thereby recovering the entire key in the (realistic) case where the initial point in the permutation sequence is not known.

For Purple, we have the 6-20 split, which can only make things easier. For example, analogous to the English text example in Section 2.3, we can likely use an HMM with two hidden states to determine the 6-20 split. Then, the attack discussed in the previous paragraphs would be applicable (with minor modification). It would be interesting to see if this attack could improve on a straightforward hill climb (Dao 2005; Freeman et al. 2005), in the sense of reducing the amount of ciphertext required. It would also be interesting to try to incorporate the search for the initial point in the permutation sequence into the HMM itself.

4. Conclusion and future work

In this article, we discussed HMMs in some detail. We then analyzed the potential benefit of making multiple random restarts in the HMM, in the context of simple substitution cryptanalysis. We also mentioned a few other classic cryptanalysis problems where an HMM-based technique could be applicable.

Related future work could consist of applying HMMs to various other classic cryptanalysis problems. Another interesting line of research is the development of faster implementations of the HMM algorithms so that large numbers of random restarts can be tested more easily. For example, in Vobbilisetty (2015), a GPU-based (Owens et al. 2008) implementation using CUDA (Zeller 2011) is discussed. Various other parallel and distributed approaches could be considered (Bhandarkar 2010; Hymel 2011).

About the authors

Rohit Vobbilisetty received his Masters in Computer Science from San Jose State University in 2015. His research interests include information security and machine learning. He currently works as a Software Engineer at Intuit.

Fabio Di Troia is a lecturer in the Department of Computer Science at San Jose University. His research interests are focused on malware detection, cryptography, and information security.

Richard M. Low, PhD, is a Lecturer in the Department of Mathematics at San Jose State University. His current research interests include cryptography, combinatorics, group theory and combinatorial game theory.

Corrado Aaron Visaggio is assistant professor of Software Security of the MsC in Computer Engineering at the University of Sannio, Italy. He obtained the PhD in computer engineering at University of Sannio (Italy). His research interests include: empirical software engineering, software security, malware analysis and data privacy. He is author of more than 70 papers published on international journals, international and national conference's proceedings, and books. He serves in many program committees and editorial boards of international conferences and journals.

He is responsible for the CINI CyberSecurity Lab at University of Sannio. He founded a Software House. He collaborates as consultant with SMEs and Large Enterprises for innovation and research projects in the field of security and software engineering. He is advisor of several PhD students at University of Sannio and he was invited to several international PhD committees.

Mark Stamp can neither confirm nor deny that he spent more than seven years working as a cryptologic mathematician at the National Security Agency. But, he can confirm that he spent two years at a small Silicon Valley startup, developing a security-related product. For the past several years, Mark has been employed as Professor of Computer Science at San Jose State University, where he teaches courses in information security, conducts research on topics in security and machine learning, writes security-related textbooks, and supervises ridiculously large numbers of Masters student projects. Perhaps not surprisingly, most of his students' projects are security-related.

References

- Berg-Kirkpatrick, T., and D. Klein. 2013. Decipherment with a million random restarts. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 18–21 October, Seattle, Washington, 874–878.
- Bhandarkar, M. 2010. MapReduce programming with apache Hadoop. *Parallel and Distributed Processing (IPDPS), 2010 IEEE International Symposium*, 1–1.
- The Brown Corpus of standard American English. <http://www.cs.toronto.edu/~gpenn/csc401/a1res.html>.
- Cave, R. L., and L. P. Neuwirth. 1980. Hidden Markov models for English. in *Hidden Markov models for speech*, ed. J. D. Ferguson Princeton, NJ, IDA-CRD, October <http://cs.sjsu.edu/stamp/RUA/CaveNeuwirth/>.
- Dao, T. 2005. Purple cipher: Simulation and improved hill-climb attack, Undergraduate research report, Department of Computer Science, San Jose State University. <http://cs.sjsu.edu/~stamp/papers/180H.pdf>.
- Dhavare, A., R. M. Low, and M. Stamp. 2013. Efficient cryptanalysis of homophonic substitution ciphers. *Cryptologia* 37(3):250–81.
- Freeman, W., G. Sullivan, and F. Weierud. 2005. Purple revealed: Simulation and computer-aided cryptanalysis of Angooki Taipu B. *Cryptologia* 29:193–232.
- Hymel, S. R. 2011. Massively parallel hidden Markov models for wireless applications. Master's thesis, Department of Electrical Engineering, Virginia Polytechnic Institute and State University, http://scholar.lib.vt.edu/theses/available/etd-12082011-204951/unrestricted/Hymel_SR_T_2011.pdf.

- IMDB. Marvin the Martian. <http://www.imdb.com/character/ch0030547/>.
- Jakobsen, T. 1995. A fast method for the cryptanalysis of substitution ciphers. *Cryptologia* 19 (3):265–74.
- Owens, J. D., M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips. 2008. GPU computing. *Proceedings of the IEEE* 96(5):879–99.
- Rabiner, L. R. February 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE* 77(2):257–86, <http://www.cs.ubc.ca/~murphyk/Bayes/rabiner.pdf>.
- Stamp, M. 2012. A revealing introduction to hidden Markov models, <http://www.cs.sjsu.edu/~stamp/RUA/HMM.pdf>.
- Stamp, M., and R. M. Low. 2007. *Applied cryptanalysis: Breaking ciphers in the real world*. Hoboken, NJ: John Wiley and Sons.
- Vobbilisetty, R. 2015. Cryptanalysis of classic ciphers using hidden Markov models. Master's report, Department of Computer Science, San Jose State University. http://scholarworks.sjsu.edu/etd_projects/407/.
- Wong, W., and M. Stamp. 2006. Hunting for metamorphic engines. *Journal of Computer Virology* 2(3):211–29.
- Zeller, C. 2011. CUDA C/C++ Basics. NVIDIA Corporation. <http://www.nvidia.com/docs/IO/116711/sc11-cuda-c-basics.pdf>.