Attacking Decipherment Problems Optimally with Low-Order N-gram Models

Sujith Ravi and Kevin Knight
University of Southern California
Information Sciences Institute
Marina del Rey, California 90292
{sravi,knight}@isi.edu

Abstract

We introduce a method for solving substitution ciphers using low-order letter n-gram models. This method enforces global constraints using integer programming, and it guarantees that no decipherment key is overlooked. We carry out extensive empirical experiments showing how decipherment accuracy varies as a function of cipher length and n-gram order. We also make an empirical investigation of Shannon's (1949) theory of uncertainty in decipherment.

1 Introduction

A number of papers have explored algorithms for automatically solving letter-substitution ciphers. Some use heuristic methods to search for the best deterministic key (Peleg and Rosenfeld, 1979; Ganesan and Sherman, 1993; Jakobsen, 1995; Olson, 2007), often using word dictionaries to guide that search. Others use expectation-maximization (EM) to search for the best probabilistic key using letter n-gram models (Knight et al., 2006). In this paper, we introduce an exact decipherment method based on integer programming. We carry out extensive decipherment experiments using letter n-gram models, and we find that our accuracy rates far exceed those of EM-based methods.

We also empirically explore the concepts in Shannon's (1949) paper on information theory as applied to cipher systems. We provide quantitative plots for uncertainty in decipherment, including the famous *unicity distance*, which estimates how long a cipher must be to virtually eliminate such uncertainty.

We find the ideas in Shannon's (1949) paper relevant to problems of statistical machine translation and transliteration. When first exposed to the idea of statistical machine translation, many people naturally ask: (1) how much data is needed to get a good result, and (2) can translation systems be trained without parallel data? These are tough questions by any stretch, and it is remarkable that Shannon was already in the 1940s tackling such questions in the realm of code-breaking, creating analytic formulas to estimate answers.

Our novel contributions are as follows:

- We outline an exact letter-substitution decipherment method which:
  - guarantees that no key is overlooked, and
  - can be executed with standard integer programming solvers
- We present empirical results for decipherment which:
  - plot search-error-free decipherment results at various cipher lengths, and
  - demonstrate accuracy rates superior to EM-based methods
- We carry out empirical testing of Shannon's formulas for decipherment uncertainty

2 Language Models

We work on letter substitution ciphers with spaces. We look for the key (among 26! possible ones) that, when applied to the ciphertext, yields the most English-like result. We take "English-like" to mean

most probable according to some statistical language model, whose job is to assign some probability to any sequence of letters. According to a 1-gram model of English, the probability of a plaintext $p_1...p_n$ is given by:

$$P(p_1...p_n) = P(p_1) \cdot P(p_2) \cdot ... \cdot P(p_n)$$

That is, we obtain the probability of a sequence by multiplying together the probabilities of the individual letters that make it up. This model assigns a probability to any letter sequence, and the probabilities of all letter sequences sum to one. We collect letter probabilities (including space) from 50 million words of text available from the Linguistic Data Consortium (Graff and Finch, 1994). We also estimate 2- and 3-gram models using the same resources:

$$\begin{aligned} P(p_1...p_n) &= P(p_1 \mid START) \cdot P(p_2 \mid p_1) \cdot P(p_3 \mid p_2) \cdot \\ &\quad ... \cdot P(p_n \mid p_{n-1}) \cdot P(END \mid p_n) \end{aligned}$$

$$\begin{aligned} P(p_1...p_n) &= P(p_1 \mid START) \cdot P(p_2 \mid START\, p_1) \cdot \\ &\quad P(p_3 \mid p_1\, p_2) \cdot ... \cdot P(p_n \mid p_{n-2}\, p_{n-1}) \cdot \\ &\quad P(END \mid p_{n-1}\, p_n) \end{aligned}$$

Unlike the 1-gram model, the 2-gram model will assign a low probability to the sequence "ae" because the probability $P(e \mid a)$ is low. Of course, all these models are fairly weak, as already known by (Shannon, 1949). When we stochastically generate text according to these models, we get, for example:

1-gram: ... thdo detusar ii c ibt deg irn toihytrsen ...
2-gram: ... itariaris s oriorcupunond rke uth ...
3-gram: ... ind thnowelf jusision thad inat of ...
4-gram: ... rece bence on but ther servier ...
5-gram: ... mrs earned age im on d the perious ...
6-gram: ... a party to possible upon rest of ...
7-gram: ... t our general through approve the ...

We can further estimate the probability of a whole English sentence or phrase. For example, the probabilities of two plaintext phrases "het oxf" and "the fox" (which have the same letter frequency distribution) is shown below. The 1-gram model which counts only the frequency of occurrence of each letter in the phrase, estimates the same probability for both the phrases "het oxf" and "the fox", since the same letters occur in both phrases. On the other hand, the 2-gram and 3-gram models, which take context into account, are able to distinguish between the English and non-English phrases better, and hence assign a higher probability to the English phrase "the fox".

Model	P(het oxf)	P(the fox)
1-gram	1.83×10^{-9}	1.83×10^{-9}
2-gram	3.26×10^{-11}	1.18×10^{-7}
3-gram	1.89×10^{-13}	1.04×10^{-6}

Over a longer sequence X of length N, we can also calculate $-log_2(P(X))/N$, which (per Shannon) gives the compression rate permitted by the model, in bits per character. In our case, we get:[1]

1-gram: 4.19
2-gram: 3.51
3-gram: 2.93

3 Decipherment

Given a ciphertext $c_1...c_n$, we search for the key that yields the most probable plaintext $p_1...p_n$. There are 26! possible keys, too many to enumerate. However, we can still find the best one in a guaranteed fashion. We do this by taking our most-probable-plaintext problem and casting it as an integer programming problem.[2]

Here is a sample integer programming problem:

variables: x, y
minimize:
$$2x + y$$
subject to:
$$x + y < 6.9$$
$$y - x < 2.5$$
$$y > 1.1$$

We require that x and y take on integer values. A solution can be obtained by typing this integer program into the publicly available lp_solve program,

[1] Because spacing is fixed in our letter substitution ciphers, we normalize $P(X)$ by the sum of probabilities of all English strings that match the spacing pattern of X.

[2] For an overview of integer and linear programming, see for example (Schrijver, 1998).

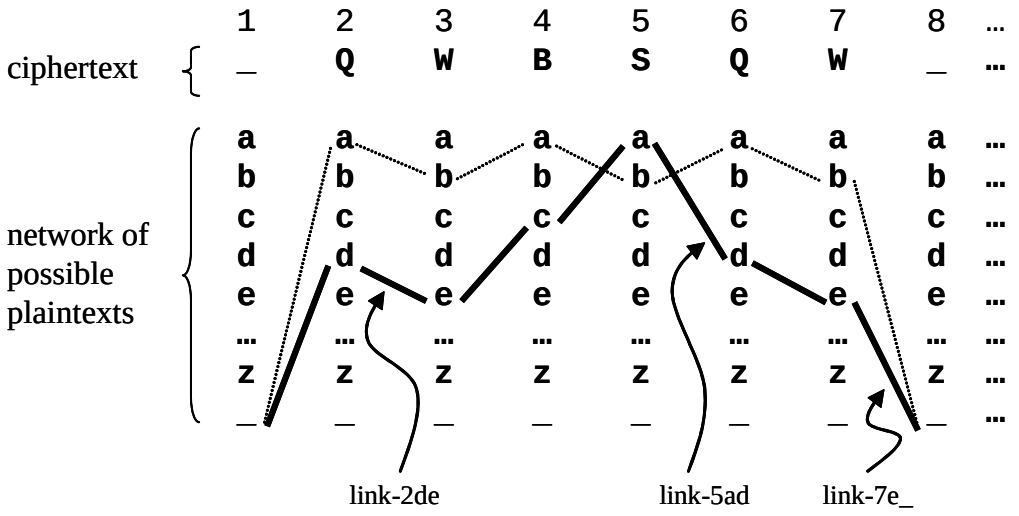

Figure 1: A decipherment network. The beginning of the ciphertext is shown at the top of the figure (underscores represent spaces). Any left-to-right path through the network constitutes a potential decipherment. The bold path corresponds to the decipherment "decade". The dotted path corresponds to the decipherment "ababab". Given a cipher length of n, the network has $27 \cdot 27 \cdot (n-1)$ links and 27^n paths. Each link corresponds to a named variable in our integer program. Three links are shown with their names in the figure.

or the commercially available $CPLEX$ program, which yields the result: $x = 4, y = 2$.

Suppose we want to decipher with a 2-gram language model, i.e., we want to find the key that yields the plaintext of highest 2-gram probability. Given the ciphertext $c_1 ... c_n$, we create an integer programming problem as follows. First, we set up a network of possible decipherments (Figure 1). Each of the $27 \cdot 27 \cdot (n-1)$ links in the network is a binary variable in the integer program—it must be assigned a value of either 0 or 1. We name these variables $link_{XYZ}$, where X indicates the column of the link's source, and Y and Z represent the rows of the link's source and destination (e.g. variables $link_{1aa}$, $link_{1ab}$, $link_{5qu}$, ...).

Each distinct left-to-right path through the network corresponds to a different decipherment. For example, the bold path in Figure 1 corresponds to the decipherment "decade". Decipherment amounts to turning some links "on" (assigning value 1 to the link variable) and others "off" (assigning value 0). Not all assignments of 0's and 1's to link variables result in a coherent left-to-right path, so we must place some "subject to" constraints in our integer program.

We observe that a set of variables forms a path if, for every node in columns 2 through $n-1$ of the network, the following property holds: the sum of the values of the link variables entering the node equals the sum of the link variables leaving the node. For nodes along a chosen decipherment path, this sum will be 1, and for others, it will be 0.[3] Therefore, we create one "subject to" constraint for each node ("_" stands for space). For example, for the node in column 2, row e we have:

subject to:

$$link_{1ae} + link_{1be} + link_{1ce} + ... + link_{1\_e} = link_{2ea} + link_{2eb} + link_{2ec} + ... + link_{2e\_}$$

Now we set up an expression for the "minimize" part of the integer program. Recall that we want to select the plaintext $p_1 ... p_n$ of highest probability. For the 2-gram language model, the following are equivalent:

(a) Maximize $P(p_1 ... p_n)$

(b) Maximize $log_2 P(p_1 ... p_n)$

(c) Minimize $-log_2 P(p_1 ... p_n)$

(d) Minimize $-log_2 [\ P(p_1 \mid START)$

[3] Strictly speaking, this constraint over nodes still allows multiple decipherment paths to be active, but we can rely on the rest of our integer program to select only one.

$$\cdot P(p_2 \mid p_1)$$
$$\cdot \ ...$$
$$\cdot P(p_n \mid p_{n-1})$$
$$\cdot P(END \mid p_n)\,]$$

(e) Minimize
$$-log_2\, P(p_1 \mid START)$$
$$-log_2\, P(p_2 \mid p_1)$$
$$- \ ...$$
$$-log_2\, P(p_n \mid p_{n-1})$$
$$-log_2\, P(END \mid p_n)$$

We can guarantee this last outcome if we construct our minimization function as a sum of $27 \cdot 27 \cdot (n-1)$ terms, each of which is a $link_{XYZ}$ variable multiplied by $-log_2 P(Z|Y)$:

$$\begin{aligned}\text{Minimize} \quad & link_{1aa} \cdot -log_2\, P(a \mid a) \\ & + link_{1ab} \cdot -log_2\, P(b \mid a) \\ & + link_{1ac} \cdot -log_2\, P(c \mid a) \\ & + ... \\ & + link_{5qu} \cdot -log_2\, P(u \mid q) \\ & + ...\end{aligned}$$

When we assign value 1 to link variables along some decipherment path, and 0 to all others, this function computes the negative log probability of that path.

We must still add a few more "subject to" constraints. We need to ensure that the chosen path imitates the repetition pattern of the ciphertext. While the bold path in Figure 1 represents the fine plaintext choice "decade", the dotted path represents the choice "ababab", which is not consistent with the repetition pattern of the cipher "QWBSQW". To make sure our substitutions obey a consistent key, we set up $27 \cdot 27 = 729$ new key_{xy} variables to represent the choice of key. These new variables are also binary, taking on values 0 or 1. If variable key_{aQ} = 1, that means the key maps plaintext a to ciphertext Q. Clearly, not all assignments to these 729 variables represent valid keys, so we augment the "subject to" part of our integer program by requiring that for any letter x,

subject to:

$$key_{xA} + key_{xB} + ... + key_{xZ} + key_{x\_} = 1$$
$$key_{Ax} + key_{Bx} + ... + key_{Zx} + key_{\_x} = 1$$

That is, every plaintext letter must map to exactly one ciphertext letter, and every ciphertext letter must map to exactly one plaintext letter. We also add a constraint to ensure that the ciphertext space character maps to the plaintext space character:

subject to:

$$key_{\_\_} = 1$$

Finally, we ensure that any chosen decipherment path of $link_{XYZ}$ variables is consistent with the chosen key. We know that for every node A along the decipherment path, exactly one active link has A as its destination. For all other nodes, zero active links lead in. Suppose node A represents the decipherment of ciphertext letter c_i as plaintext letter p_j—for all such nodes, we stipulate that the sum of values for $link_{(i-1)xp_j}$ (for all x) equals the value of $key_{p_j c_i}$. In other words, whether a node lies along the chosen decipherment path or not, the chosen key must support that decision.

Figure 2 summarizes the integer program that we construct from a given ciphertext $c_1 ... c_n$. The computer code that transforms any given cipher into a corresponding integer program runs to about one page. Variations on the decipherment network yield 1-gram and 3-gram decipherment capabilities. Once an integer program is generated by machine, we ask the commercially-available $CPLEX$ software to solve it, and then we note which key_{XY} variables are assigned value 1. Because $CPLEX$ computes the optimal key, the method is not fast—for ciphers of length 32, the number of variables and constraints encoded in the integer program (IP) along with average running times are shown below. It is possible to obtain less-than-optimal keys faster by interrupting the solver.

Model	# of IP variables	# of IP constraints	Average running time
1-gram	1,755	1,083	0.01 seconds
2-gram	27,700	2,054	50 seconds
3-gram	211,600	27,326	450 seconds

4 Decipherment Experiments

We create 50 ciphers each of lengths $2, 4, 8, ..., 256$. We solve these with 1-gram, 2-gram, and 3-gram language models. We record the average percentage of ciphertext tokens decoded incorrectly. 50% error means half of the ciphertext tokens are deciphered wrong, while 0% means perfect decipherment. Here

variables:

$link_{ipr}$ 1 if the *i*th cipher letter is deciphered as plaintext letter *p* AND the *(i+1)*th cipher letter is deciphered as plaintext letter *r*
0 otherwise

key_{pq} 1 if decipherment key maps plaintext letter *p* to ciphertext letter *q*
0 otherwise

minimize:

$\sum_{i=1}^{n-1} \sum_{p,r} link_{ipr} \cdot -log\ P(r|p)$ (2-gram probability of chosen plaintext)

subject to:

for all *p*: $\sum_r key_{pr} = 1$ (each plaintext letter maps to exactly one ciphertext letter)

for all *p*: $\sum_r key_{rp} = 1$ (each ciphertext letter maps to exactly one plaintext letter)

$key_{\_\ \_} = 1$ (cipher space character maps to plain space character)

for (*i=1...n-2*), for all *r*: $[\ \sum_p link_{ipr} = \sum_p link_{(i+1)rp}\ ]$ (chosen links form a left-to-right path)

for (*i=1...n-1*), for all *p*: $\sum_r link_{irp} = key_{pc_{i+1}}$ (chosen links are consistent with chosen key)

Figure 2: Summary of how to build an integer program for any given ciphertext $c_1 ... c_n$. Solving the integer program will yield the decipherment of highest probability.

we illustrate some automatic decipherments with error rates:

42% error: the avelage ongrichman hal cy wiof a sevesonme qus antizexty that he buprk lathes we blung than soment - fotes mmasthes

11% error: the average englishman has so week a reference for antiality that he would rather be prong than recent - deter quarteur

2% error: the average englishman has so keep a reference for antiquity that he would rather be wrong than recent - peter mcarthur

0% error: the average englishman has so deep a reverence for antiquity that he would rather be wrong than recent - peter mcarthur

Figure 3 shows our automatic decipherment results. We note that the solution method is exact, not heuristic, so that decipherment error is not due to search error. Our use of global key constraints also leads to accuracy that is superior to (Knight et al., 2006). With a 2-gram model, their EM algorithm gives 10% error for a 414-letter cipher, while our method provides a solution with only 0.5% error. At shorter cipher lengths, we observe much higher improvements when using our method. For example, on a 52-letter textbook cipher, using a 2-gram model, the solution from our method resulted in 21% error as compared to 85% error given by the EM solution.

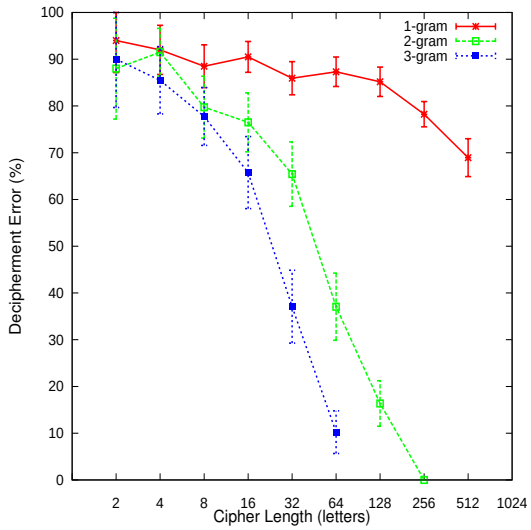

Figure 3: Average decipherment error using integer programming vs. cipher length, for 1-gram, 2-gram and 3-gram models of English. Error bars indicate 95% confidence intervals.

We see that deciphering with 3-grams works well on ciphers of length 64 or more. This confirms

that such ciphers can be attacked with very limited knowledge of English (no words or grammar) and little custom programming.

The 1-gram model works badly in this scenario, which is consistent with Bauer's (2006) observation that for short texts, mechanical decryption on the basis of individual letter frequencies does not work. If we had infinite amounts of ciphertext and plaintext drawn from the same stochastic source, we would expect the plain and cipher frequencies to eventually line up, allowing us to read off a correct key from the frequency tables. The upper curve in Figure 3 shows that convergence to this end is slow.

5 Shannon Equivocation and Unicity Distance

Very short ciphers are hard to solve accurately. Shannon (1949) pinpointed an inherent difficulty with short ciphers, one that is independent of the solution method or language model used; the cipher itself may not contain enough information for its proper solution. For example, given a short cipher like $XYYX$, we can never be sure if the answer is *peep*, *noon*, *anna*, etc. Shannon defined a mathematical measure of our decipherment uncertainty, which he called *equivocation* (now called entropy).

Let C be a cipher, M be the plaintext message it encodes, and K be the key by which the encoding takes place. Before even seeing C, we can compute our uncertainty about the key K by noting that there are 26! equiprobable keys:[4]

$$\begin{aligned} H(K) &= -(26!) \cdot (1/26!) \cdot log_2\,(1/26!) \\ &= 88.4 \text{ bits} \end{aligned}$$

That is, any secret key can be revealed in 89 bits. When we actually receive a cipher C, our uncertainty about the key and the plaintext message is reduced. Shannon described our uncertainty about the plaintext message, letting m range over all decipherments:

$$\begin{aligned} H(M|C) &= \text{equivocation of plaintext message} \\ &= -\sum_m P(m|C) \cdot log_2\, P(m|C) \end{aligned}$$

[4](Shannon, 1948) The entropy associated with a set of possible events whose probabilities of occurrence are $p_1, p_2, ..., p_n$ is given by $H = -\sum_{i=1}^{n} p_i \cdot log_2(p_i)$.

$P(m|C)$ is probability of plaintext m (according to the language model) divided by the sum of probabilities of all plaintext messages that obey the repetition pattern of C. While integer programming gives us a method to find the most probable decipherment without enumerating all keys, we do not know of a similar method to compute a full equivocation without enumerating all keys. Therefore, we sample up to 100,000 plaintext messages in the neighborhood of the most probably decipherment[5] and compute $H(M|C)$ over that subset.[6]

Shannon also described $H(K|C)$, the *equivocation of key*. This uncertainty is typically larger than $H(M|C)$, because a given message M may be derived from C via more than one key, in case C does not contain all 26 letters of the alphabet.

We compute $H(K|C)$ by letting $r(C)$ be the number of distinct letters in C, and letting $q(C)$ be $(26 - r(C))!$. Letting i range over our sample of plaintext messages, we get:

$$\begin{aligned} H(K|C) &= \text{equivocation of key} \\ &= -\sum_i q(C) \cdot (P(i)/q(C)) \cdot log_2\,(P(i)/q(C)) \\ &= -\sum_i P(i) \cdot log_2\,(P(i)/q(C)) \\ &= -\sum_i P(i) \cdot (log_2\, P(i) - log_2\, q(C)) \\ &= -\sum_i P(i) \cdot log_2\, P(i) + \sum_i P(i) \cdot log_2\, q(C) \\ &= H(M|C) + log_2\, q(C) \end{aligned}$$

Shannon (1949) used analytic means to roughly sketch the curves for $H(K|C)$ and $H(M|C)$, which we reproduce in Figure 4. Shannon's curve is drawn for a human-level language model, and the y-axis is given in "decimal digits" instead of bits.

[5]The sampling used to compute $H(M|C)$ starts with the optimal key and expands out a frontier, by swapping letters in the key, and recursing to generate new keys (and corresponding plaintext message decipherments). The plaintext messages are remembered so that the frontier expands efficiently. The sampling stops if 100,000 different messages are found.

[6]Interestingly, as we grow our sample out from the most probable plaintext, we do not guarantee that any intermediate result is a lower bound on the equivocation. An example is provided by the growing sample (0.12, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01), whose entropy steadily increases. However, if we add a 14th item whose $P(m)$ is 0.12, the entropy suddenly decreases from 2.79 to 2.78.

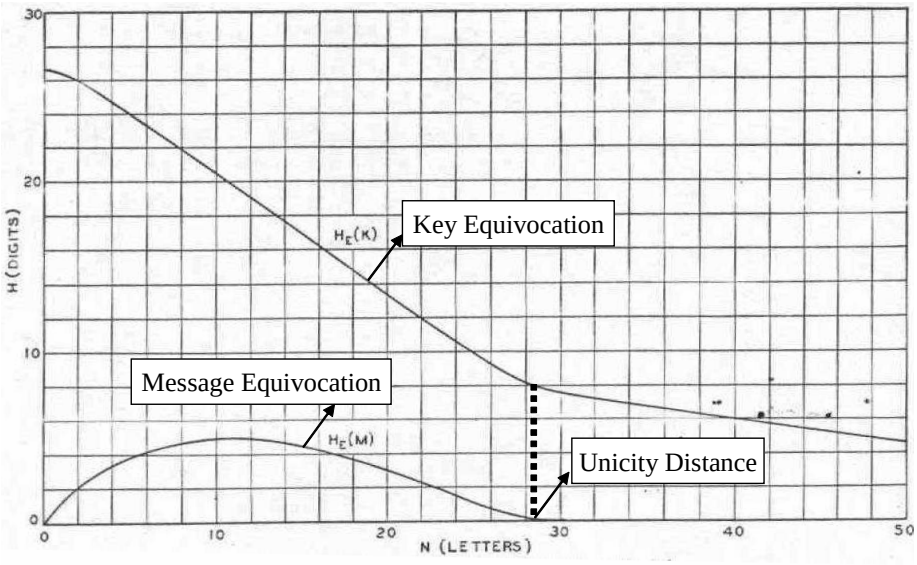

Figure 4: Equivocation for simple substitution on English (Shannon, 1949).

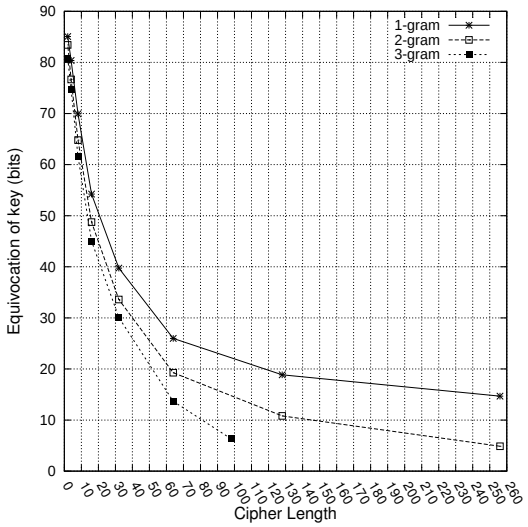

Figure 5: Average key equivocation observed (bits) vs. cipher length (letters), for 1-gram, 2-gram and 3-gram models of English.

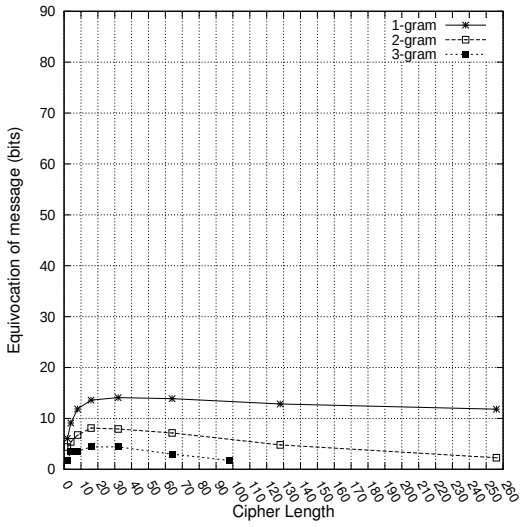

Figure 6: Average message equivocation observed (bits) vs. cipher length (letters), for 1-gram, 2-gram and 3-gram models of English.

For comparison, we plot in Figures 5 and 6 the average equivocations as we empirically observe them using our 1-, 2-, and 3-gram language models.

The shape of the key equivocation curve follows Shannon, except that it is curved from the start, rather than straight.

The message equivocation curve follows Shannon's prediction, rising then falling. Because very short ciphers have relatively few solutions (for example, a one-letter cipher has only 26), the overall uncertainty is not that great.[7] As the cipher gets longer, message equivocation rises. At some point, it then decreases, as the cipher begins to reveal its secret through patterns of repetition.

Shannon's analytic model also predicts a sharp decline of message equivocation towards zero. He

[7] Uncertainty is only loosely related to accuracy—even if we are quite certain about a solution, it may still be wrong.

defines the *unicity distance* (U) as the cipher length at which we have virtually no more uncertainty about the plaintext. Using analytic means (and various approximations), he gives:

$$U = H(K)/(A - B)$$

where:

A = bits per character of a 0-gram model (4.7)

B = bits per character of the model used to decipher

For a human-level language model ($B \sim 1.2$), he concludes $U \sim 25$, which is confirmed by practice. For our language models, the formula gives:

$$U = 173 \text{ (1-gram)}$$
$$U = 74 \text{ (2-gram)}$$
$$U = 50 \text{ (3-gram)}$$

These numbers are in the same ballpark as Bauer (2006), who gives 167, 74, and 59. We note that these predicted unicity distances are a bit too rosy, according to our empirical message equivocation curves. Our experience confirms this as well, as 1-gram frequency counts over a 173-letter cipher are generally insufficient to pin down a solution.

6 Conclusion

We provide a method for deciphering letter substitution ciphers with low-order models of English. This method, based on integer programming, requires very little coding and can perform an optimal search over the key space. We conclude by noting that English language models currently used in speech recognition (Chelba and Jelinek, 1999) and automated language translation (Brants et al., 2007) are much more powerful, employing, for example, 7-gram word models (not letter models) trained on trillions of words. Obtaining optimal keys according to such models will permit the automatic decipherment of shorter ciphers, but this requires more specialized search than what is provided by general integer programming solvers. Methods such as these should also be useful for natural language decipherment problems such as character code conversion, phonetic decipherment, and word substitution ciphers with applications in machine translation (Knight et al., 2006).

7 Acknowledgements

The authors wish to gratefully acknowledge Jonathan Graehl, for providing a proof to support the argument that taking a larger number of samples does not necessarily increase the equivocation. This research was supported by the Defense Advanced Research Projects Agency under SRI International's prime Contract Number NBCHD040058.

References

Friedrich L. Bauer. 2006. *Decrypted Secrets: Methods and Maxims of Cryptology*. Springer-Verlag.

Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. 2007. Large language models in machine translation. In *Proceedings of EMNLP-CoNLL*.

Ciprian Chelba and Frederick Jelinek. 1999. Structured language modeling for speech recognition. In *Proceedings of NLDB: 4th International Conference on Applications of Natural Language to Information Systems*.

Ravi Ganesan and Alan T. Sherman. 1993. Statistical techniques for language recognition: An introduction and guide for cryptanalysts. *Cryptologia*, 17(4):321–366.

David Graff and Rebecca Finch. 1994. Multilingual text resources at the linguistic data consortium. In *Proceedings of the HLT Workshop on Human Language Technology*.

Thomas Jakobsen. 1995. A fast method for cryptanalysis of substitution ciphers. *Cryptologia*, 19(3):265–274.

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

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

Shmuel Peleg and Azriel Rosenfeld. 1979. Breaking substitution ciphers using a relaxation algorithm. *Comm. ACM*, 22(11):598–605.

Alexander Schrijver. 1998. *Theory of Linear and Integer Programming*. John Wiley & Sons.

Claude E. Shannon. 1948. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656.

Claude E. Shannon. 1949. Communication theory of secrecy systems. *Bell System Technical Journal*, 28:656–715.