AUTOMATED ANALYSIS OF CRYPTOGRAMS

Bruce R. Schatz

Introduction

The solution of simple ciphers has long been a popular activity among amateurs. Many approaches are known and most such problems can be solved in a short period. Although a number of computer programs have been written to assist human cryptanalysts in solving these (by collecting statistics, providing up-to-date accounting, and so on), there have been few attempts to automate this process. And these seem to operate in a very statistical, brute-force fashion. This paper describes a program which attempts to solve a simple class of ciphers, cryptograms, by itself in a more semantic manner. The basic method is to deduce vowels, and then guess other letters by filling in words. The approach throughout was to find a reasonable and somewhat human-like set of methods which seemed to work fairly well.

Cryptograms

A cipher is a method for disguising a message by transforming it in some manner. This is typically accomplished by rearranging the letters (transposition) or replacing the letters by a set of symbols (substitution). Cryptanalysis is the process of "breaking" the cipher, i.e. understanding the message by deducing the transformation. The type of ciphers that will be considered are mono-alphabetic simple substitution ciphers with word divisions, i.e. the enciphered message is obtained from the original by making a direct substitution for the letters of the original alphabet from a single cipher alphabet (preserving word boundaries). These will be called "cryptograms", after their designation in crossword puzzle magazines, their common place of occurence. (The American Cryptogram Association calls them "Aristocrats".) The messages will be assumed to be in normal English and of reasonable length (say 100 letters). (Accordingly, the usual letter and word characteristics will not be too maliciously maladjusted.) Cryptograms thus essentially consist of substituting the letters of a permutation of the alphabet for the letters

of the alphabet to produce an enciphered message. An example (a Caesar cipher) is:

original	a b c . . . x y z
cipher	B C D . . . Y Z A

Then "intelligent" would be represented by "JOUFMMJHFOU" and "SPCPU" would mean "robot". The conventions used here will be used throughout, namely the original message (the plaintext) will appear in lower case while the enciphered message (the ciphertext) will appear in upper.

Possible approaches to the solution process

Solving a cryptogram is non-trivial although many people can learn to do it quickly and easily. Only the presence of regularities in the encrypted message makes cryptanalysis possible. Fortunately, for cryptograms the enciphering transformation is simple and does not hide the regularities of the original message much. In addition, there is a large amount of information available on the characteristics of English. (The reference literature on cryptanalysis available and unclassified is somewhat sparse. [10] is the only fairly comprehensive and advanced text in English. [30] is more elementary. There are several good publications sponsored by the American Cryptogram Association of which [25] and [17] are particularly helpful with cryptograms as defined here. (They and [10] contain much useful data on characteristics of English.) A recent cryptographic series contains a number of useful reprints for more advanced cryptanalysis, e.g. [18] and [8].

A solution can be conveniently broken into two stages: entry and development [25]. The first consists of an initial break by guessing what three or four enciphered letters are; the second of filling in the rest by recognizing partially deciphered patterns.

There are a number of known techniques for making an entry. Some of these deal directly with letter frequencies with perhaps some positional constraints (e.g. let the most frequent letter be "e" if it also occurs frequently as a final letter). (See [10, p. 72ff], [25], [17].) Others

attempt to deduce the transformation (e.g. using specific techniques for special types such as linear transformations [30, ch. 1] or attempting to deduce the keyword generating the cipher alphabet [10, pp. 70-72]. One can also look for common words, suffixes, and prefixes. Perhaps the most effective technique is trying to determine the vowels. The four common vowels (e,a,i,o) are among the most common letters (a,e,i,o,u,y make up 40% of normal English letters) and every word contains a vowel (except for oddities such as crwth).

Much is known about the characteristics of vowels. Typical facts include: high frequency (for the four), frequent contacting of low-frequency letters, little contact among themselves but a wide variety of contacts, reversals (e.g. re,er) usually are a vowel and consonant, and so on. Possible methods for finding them thus include contact tables (what letter touches what) (see [10, p. 74ff]) and use of positional data (e.g. in four letter words vowels usually occur second) [1, p. 12ff]. Another useful technique is Lamb's vowel-line shortcut [10, pp. 88-92] which examines the variety of contacts and attempts to isolate the vowels (which have a wide variety). While all of these are fairly effective, the method used here will be a different one, based on singular value decomposition (a technique of numerical linear algebra), which may be of general interest to cryptanalysts. Although it is extremely unlikely that humans use this, it is quite effective and more human-like techniques could be used instead (as above).

There are also several possible approaches to development. One can attempt to match cipher words against pattern words [13]. These are occurences of letter sequences in words such as aba for mom, tot; abbc for book, look; and abccbccbddb for Mississippi. Another possibility is non-pattern words [1] where one searches for sequences of consonants (c) and vowels (v). For example, ccvc for bled, drop; ccvcvc for dragon. However, as the author has some interest in investigating human problem solving, a method most likely used by humans will be used, i.e. guessing words. (Even if one is not interested in how humans

"work", mimicking of human behavior is often a helpful heuristic in constructing complicated computer programs (particularly those which do problem solving where humans provide the only known examples of successful solvers).) The program will attempt to deduce letters by matching partially known words to a vocabulary of common words. For example, if one has QtheT, it is likely that Q=o and W=r (producing "other").

Previous work

There has been little published work on automated cryptanalysis. [21] and [12] are nice surveys of some of this. There is some general mathematical analysis of the encryption problem [28] and a number of solutions to specialized ciphers [32,33], [22], [16]. There is also peripheral mention in the growing literature on cryptography (enciphering) for security in operating systems and data banks [6, 7]. [19] is a scholarly, wide-ranging history of cryptanalysis with many technical details.

[29] described a system to solve cryptograms which attempted to find the permutation for which the digram frequencies of the corresponding "deciphered" message most closely matched the normal English frequencies. This made an initial guess of a permutation (e.g. direct match of English to cipher letter frequencies) then successively hill climbed to more likely neighboring permutations until a peak was reached. A neighboring permutation (i.e. differing by at most 2 letters) was considered more likely if the standard frequencies of the digrams (considered as probabilities in a Markov process) for its corresponding message was higher. [5] reported on a SNOBOL-like programming language specially designed for cryptanalysis.

Probably the closest approach to the work here is that of [24] who describe a system for doing cryptarithmetic (a simple substitution cipher for the digits 0-9 in an arithmetic problem) based on a detailed model of a human solver. The modelling here was much more intuitive in nature, and as cryptarithmetic is a somewhat easier and different prob-

lem than deciphering of a cryptogram, the author's work was essentially independent.

This paper will proceed by explaining the vowel-finding and word guessing procedures, explaining the program operation (control structure and other pieces), and finally working through an example in detail.

Singular Value Decomposition (SVD)

As mentioned above, the approach of entry chosen is to identify the vowels. [23] (to appear) has developed a surprisingly effective method of picking out the vowels using a standard technique of numerical linear algebra, singular value decomposition or SVD. (The entire method will be referred to as SVD.)

It is based on the observation that in English (and several other languages such as Russian), vowels more often follow consonants than other vowels. (In fact vowels contact consonants some 85% of the time [1, p. 4] So the following rule usually holds:

$$\frac{\text{number of vowel-vowel pairs}}{\text{number of vowels}} < \frac{\text{number of consonant-vowel pairs}}{\text{number of consonants}}$$

This essentially says that vowel-vowel pairs should occur less frequently than consonant-vowel pairs. Thus a partition of the alphabet into two classes which satisfied the above rule would likely distinguish between vowels and consonants. [23] gives one such procedure.

First tabulate the digram matrix. (This considers letters two at a time in the cryptogram, i.e. A is a 26 by 26 matrix where a_{ij} is the number of occurences of the i-th letter followed by the j-th letter with blanks and punctuation ignored.) Performing a singular value decomposition finds orthogonal matrices X and Y such that X^TAY is a diagonal matrix whose diagonal elements are the singular values σ_i.

[15] gives a detailed description of singular value decomposition and its computation. [14] gives an algorithm for computing it. For a general exposition, see [31, pp. 317-326].

Now consider a second order approximation of the digram matrix (the simplest that accounts for correlations between pairs of letters). It can be proved that a partition based on the signs of the elements of the second column vectors of X and Y divides the alphabet into two classes satisfying the above vowel-follows-consonant rule. So if the classes are labelled vowels and consonants, the vowels have been identified. (One usually also obtains a third class of "neuter" letters, e.g. h.) It should be noted that this is merely another systematic way of finding letters which satisfy vowel characteristics (here their propensity for contacting mostly consonants). The appendix contains precise details and a proof of the theorem.

In practice this method works fairly well at identifying the vowels. For example, running it on the titles and first five stanzas of "The Hunting of the Snark" by Lewis Carroll [4] (705 letters) produced:

vowel:	aeijo
consonant:	bcdglmnprstvw
neuter:	fhkquxyz

Running it on a ten line portion (448 letters) of a speech on matrix computation by George Forsythe [31], probably much more typical English, produced:

vowel:	aeiou
consonant:	bcfgjlmrst
neuter:	dhkpqvwxyz

Overview of the solver

While the approach used is basically semantic, there will be no use of semantic knowledge beyond words (e.g. of sentences or phrases) nor any knowledge of specialized cipher types (e.g. Caesar ciphers or more general linear transformations). Thus the only attempt is to guess what the cipher letters represent, not to discover the keyword or method generating the transformation.

Essentially, as previously mentioned, the solver consists of an entry and a development. The entry is made by attempting to identify the vowels (in addition to looking for a few special clues). Development continues by a number of routines which communicate in a heterarchical fashion (i.e. they exist as co-routines where each calls several others). The major such are the fill-in and the word search. Minor ones include individual letter procedures and frequency playoff.

There is also a database which contains such empirical data as standard frequencies for letters, initial letters, final letters, and digrams. (See figures 1 and 2). A word vocabulary and some miscellaneous information (e.g. standard common suffixes and prefixes, contact data) are also available.

Fig. 1. Standard letter frequencies with percent occurence in English (from [25, p. 25]).

e	12.51	c	2.75
t	9.50	m	2.37
a	8.06	w	2.06
o	8.00	p	1.97
n	7.12	g	1.80
i	6.85	y	1.70
r	6.20	b	1.49
s	6.11	v	0.97
h	5.99	k	0.68
d	4.09	x	0.21
l	3.68	j	0.17
f	2.79	q	0.09
u	2.77	z	0.07

Fig. 2. Standard initial and final letter frequency ordering (from [30, pp. 178-79]).

Initial	Final	Initial	Final
t	e	d	m
a	s	e	w
s	d	n	k
o	n	l	c
i	t	g	p
c	r	u	i
w	y	y	x
p	o	v	u
b	f	j	b
f	l	k	v
h	a	q	j
m	g	x	z
r	h	z	q

Along the path to solution, a number of "notes" are made. There is one of these for each cipher and plaintext letter. They contain partial possibility information (e.g. e is either Q or Y; W is either a or o) along with a record of which place in which routine provided that information (e.g. word search level 1). Thus there is a log of what decisions were made and why ; this is often useful in deducing letters by process of elimination (e.g. W is either a or o and o has just been identified so W = a). The notes are also helpful in choosing alternate paths when backup is necessary.

A description of the program execution follows. (See figure 3 for a schematic picture of the control flow.)

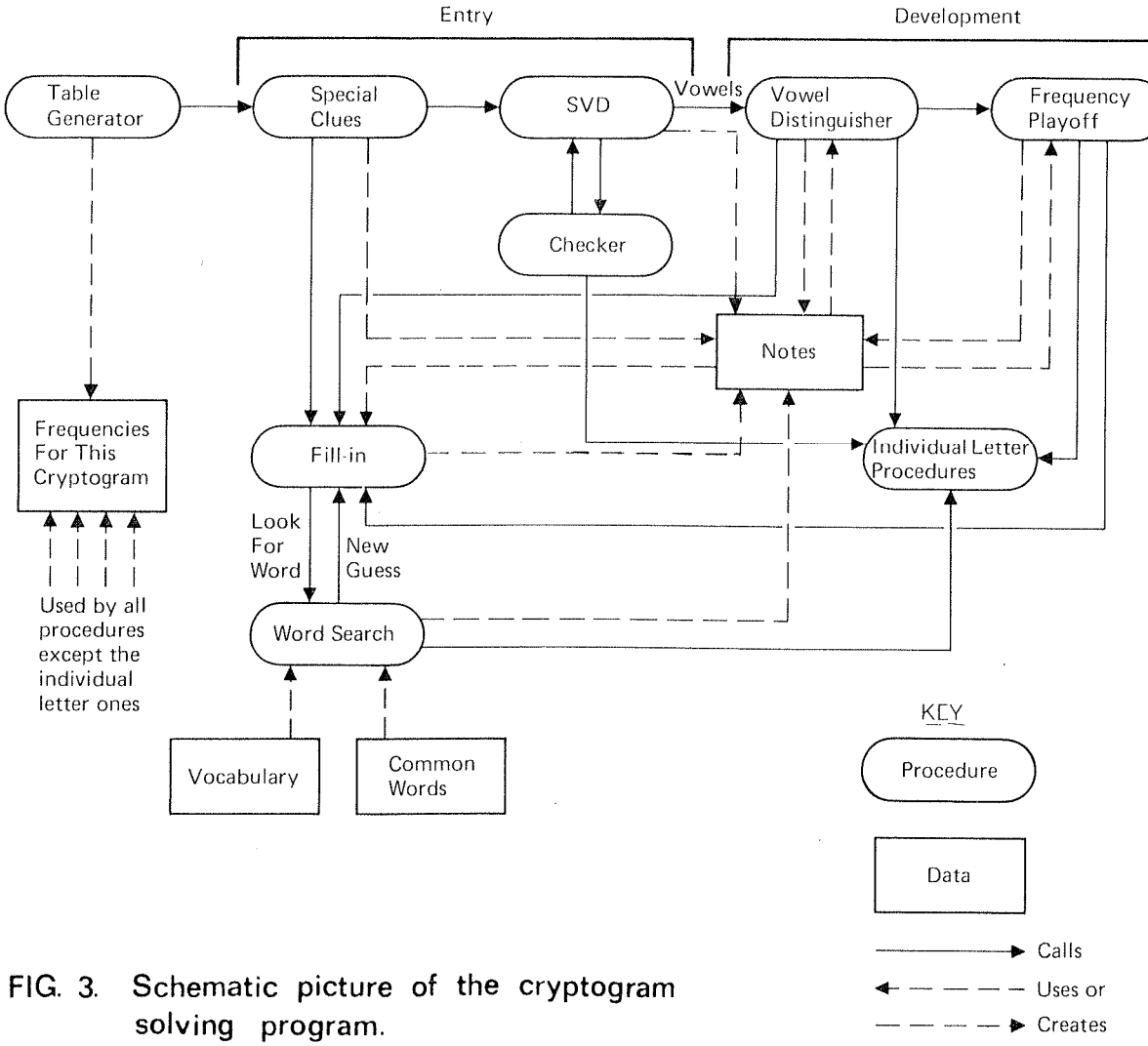

FIG. 3. Schematic picture of the cryptogram solving program.

First the table generator is called. Various frequency tables from the cryptogram are computed: letter frequency, initial and final letter frequencies, a digram matrix, and a contact count (a list of each letter together with the two letters that flanked it on the left and right each time it was used). These are sorted into descending order for subsequent utilization.

Then a special clues procedure is used. This tries a few tricks such as looking for single letter words (which must be a or i) and apostrophes (single letters after apostrophes are s,t; double letters are ll, re, ve)

Next an entry is attempted by calling SVD to find the vowels. As previously described, this returns several supposed vowels. These are checked against such common vowel characteristics as contact low frequency letters, wide variety of contacts, little contact among themselves and various positional data. Unlikely candidates are removed from the list of vowels. It should be noted that SVD occasionally produces a (mostly) correct partition wrongly labelled so that supposed vowels are really consonants and vice versa. This is accounted for by switching the labels if more than 6 vowels are found.

A routine which should be mentioned at this point is the fill-in and recording. This is called upon by several others. Whenever a guess of a cipher letter is made, this substitutes the guessed plaintext letter into the cipher message, records the substitution in the list of plaintext and the list of cipher letters, then calls word search. It also makes a note of which routine called it (made the decision of the letter guess). Each filling-in of a guess is made on a different level of a stack so that backtracking is possible. Any words which now have only one or two unknown letters are put onto the appropriate word stack for later viewing by the word search routine. Any just completed words are removed from the word stacks.

Another routine constantly called upon are the individual letter characteristics. They consist of a procedure for each English letter which contains basic information about its properties (mostly frequency and position). When a cipher letter is given it, a letter procedure returns an estimate of the probability that the cipher letter is that letter (by summing weighted comparisons of its characteristics). For example, for e - heavily weighted: very high frequency, very frequent final; moderately weighted: often second position, often doubled; lightly weighted: follows bhmw in two letter words. ([17] and [25] contain lengthy lists of letter characteristics.)

Continuing the mainstream of execution, SVD has returned several letters now "certified" as vowels so the vowel distinguishing routine is entered to determine which is which. To make its guesses, the vowel distinguisher goes through a series of standard characteristics such as the following (e.g. [25, pp. 11-13], [10, pp. 78-79]). E is the most frequent (and is a very common final letter). I has high frequency and often occurs as the antepenultimate letter. O and e are often doubled but rarely touch each other. (O is also frequent and often second or last.) A is frequent and often an initial letter. U and y are infrequent. There are various other frequency and positional criteria and the individual letter procedures are used as well.

After a vowel is identified, the fill-in routine is entered. Then word search is called to see if any words can now be guessed. If some can, fill-in and word search are repeated. When no words can be guessed the vowel distinguisher is returned to. (Note that even deducing one or two vowels will usually enable some short words to be guessed which in turn enables an entry and a solution.) The program typically identifies a few vowels then alternates between fill-in and word search. This process continues until all the vowels are determined as closely as possible (or a solution has been found). Then the frequency playoff routine is entered. If an impasse is reached (no deductions seem reasonable), the program backtracks. (Backtracking involves removing

several message development stack levels, leaving the decipherment in some previous state with fewer plaintext letters known. Often a good strategy is to choose among several possibilities than backtrack later if incorrect.)

The frequency playoff routine concentrates on trying to deduce what the most frequent cipher letters represent. This is done by using the individual letter procedures and relying on the fact that etaonirsh make up 70% of the letters in normal English. This, interacting with the above routines, executes until the cryptogram is "solved" or the program backs up to the start and fails.

Word Search

This is probably the most important development routine (for both the program and humans). The objective is to identify unknown cipher letters by guessing at words for which all but a few letters are known. The words thus examined are contained in two stacks, level 1 and 2, generated by the fill-in routine. Whenever a letter guess is made and fill-in substitutes it into the cryptogram, all of the words which now have only one unknown letter are placed on top of the level 1 stack; those with two unknown on the level 2 stack. (Words are deleted when completely guessed or moved from one stack to another.)

The word search routine first scans the level 1 stack. For each word in the stack, all possibilities for it are retrieved from a list of the 100 most common words in English [10, p. 226] (simple pattern matching against this list). If there is only one possibility and a brief comparison of the stack word's unknown cipher letter against the usual characteristics of the list's proposed letter is satisfied, the word is removed from the level 1 stack, fill-in is called, and another word search follows. If there are two or three possibilities, a note is made (of the possibilities for that cipher letter) and the search continues (unless a call to the individual letter procedures or to previous notes can resolve the conflict, e.g. some of the possible English

letters have already been guessed). If there are more than three possibilities, it is assumed that nothing can be concluded and the next word in the stack is examined. If no possible words have been found in the common word list, a search through the vocabulary (the 300 most common words from [3] excluding those in the common word list) is performed. The level 1 search continues through its stack trying each word. When a level 1 search is fruitless (no guessable words in the stack), a search through the level 2 stack is made in a similar manner (first trying common words, then vocabulary for each stack member).

Note that the most recent stack additions are examined before previous ones (i.e. new words are added at the top) and that words not in the vocabulary can still be deduced by guessing all of the letters. No knowledge of English such as plural, tense, or morphemic knowledge of words is included in the word search. Note there is a tradeoff between too few vocabulary words (providing no help in guessing) and too many (yielding too many possibilities).

The vocabulary lists are stored in the form of a digital tree [20, p. 489 This has the advantages of facilitating a fast unsuccessful search (necessary as there is often not any such word) and a fairly fast successful search (since the vocabulary is in frequency order). In addition, with encoding 5 bits per letter (a=00001 ... z=11010), each search takes at most 'five times the word length' comparisons, and this does not depend on the size of the vocabulary.

An example of the program's execution

A modest implementation in PL/I of the principles outlined above was run on an IBM 370/155 at Rice University, an IBM 360/195 at the IBM San Jose Research Laboratory, and an IBM 370/168 at the IBM T. J. Watson Research Laboratory. (The routines based on characteristics of English, particularly the individual letter procedures, were considered rather perfunctorily. The database and notes were somewhat incomplete. For facility

in stack manipulation and list processing, it would have been preferable to have used LISP or one of the new high-level LISP-like languages [2]. However, the version of LISP available did not have the necessary matrix manipulation and arithmetic capabilities.) A run typically took a few seconds.

Fig. 4. The sample cryptogram.

XVQIQ HIQ H MIQHX CHSO XVZSMA EY ZCCQSAQ ZSXQIQAX HPEKX LVZJV AJZQSJQ, HX NIQAQSX, BSELA TZXXTQ. ZX ZA XVQ UEP EY NVZTEAENVO XE BQQN ANQJKTHXZES HPEKX XVQAQ HTZDQ.

As an example of what the program can do, a sample cryptogram (Figure 4) will be solved. This is a fairly easy one of reasonable length (132 letters) with word divisions and no special attempt made to distort the normal characteristics of English. Throughout, plaintext will appear in lower case and ciphertext in capitals. The output reproduced is that of an actual program run, slightly abridged and annotated. The name of each routine, its results, and its "reasoning" appear along with comments in parentheses.

Read-in. (The cryptogram is read in and divided into words.)

Table generator. (Figures 5 and 6 show some of the frequency tables generated.)

Fig. 5. Letter frequencies in the cryptogram.

high		middle		low	
Q	20	I	5	B,L,M,O,Y	2
X	16	N	5	D,U	1
Z	11	T	5	F,G,R,W	0
A	10	J	4		
E	10	C	3		
H	9	K	3		
S	8	P	3		
V	8				

Fig. 6. More cryptogram letter frequencies.

initial		final	
H	5	Q	8
X	5	X	7
Z	4	A	3
A,B,E,N	2	O,Y	2
C,L,M,T,U	1	E,N,P,S,V	1

Special clues. A single letter, H, is found. Thus H = a or i. Since H is frequent, a frequent initial but never final, and not doubled (the a and i procedures were called), and since a is more often used as a single letter in cryptograms, guess

H = a (This indicated a guess and a call to fill-in.)

Word Search level 1. looking at aX. Too many possibilities.
Word Search level 2. looking at aIQ. Too many possibilities.

SVD. The vowels returned and checked are E O Q Z.
(Note there are only 4 and that a (=H) was not among them.) Also returned were consonant : ABDIJNPUX and neuter : CFGHKLMRSTVWY. (Here, in fact, x<0 and y>0 for vowels.)

Vowel distinguisher. The frequencies are Q 20, Z 11, E 10, O 2. Q is the most frequent and the most frequent final so

Q = e

Word Search level 1. looking at aIe. From common word list, I = r. Had age been on this list, a call to the g and r procedures would have revealed g's favorite position is last while r's favorites are second and next to last. I appears as letter 2 of 3, 2 of 5, 5 of 8, 2 of 7, 4 of 5. Four of these indicate r, none indicate g so

I = r

Word Search level 1. aX still unresolved.
Word Search level 2. looking at MreaX. Since great is on the common word list and X is very frequent (overall, initially, and finally) as is t,

M = g (g is never strongly checked.)

X = t (Important since t is the second most frequent English letter. Note that t could have been guessed from XVere implies there if great was not known.)

Fill-in. New word generated. at.
(At this point an entry has certainly been made since 5 letters, including 3 of the most frequent (e t a), have been guessed. Figure 7 gives the progress so far. Note the control still remains in the vowel distinguisher.)

Fig. 7. The cryptogram after an entry has been made.

```
t ere are a great  a   t   g        e  e   tere t a   t          e  e
XVQIQ HIQ H MIQHX CHSO XVZSMA EY ZCCQSAQ ZSXQIQAX HPEKX LVZJV AJZQSJQ,

at  re e t          tt e    t    t e                    t   ee
HX NIQAQSX, BSELA TZXXTQ.  ZX ZA XVQ UEP EY NVZTEAENVO XE BQQN

  e   at    a   t t e e a   e.
ANQJKTHXZES HPEKX XVQAQ HTZDQ.
```

Word Search level 1. looking at tE. From common word list (and since E is frequent vowel),

E = o

Word Search level 1. looking at oY (twice). From common word list, Y = f n r. r has already been guessed and there is no positive distinction now between f and n so a note of the two possibilities is made. looking at tVe. From common word list,

V = h

Fill-in. New words generated: there.

Word Search level 1. looking at theAe. From common word list (r alread guessed) and the fact that A is frequent (overall, initially, finally)

A = s

Word Search level 1. looking at Zs. From common word list, Z = i or u so record for later. looking at oY (twice) again. Still undecidable. looking at Zt. From common word list and since Z is a vowel, frequent letter, and frequent initial,

Z = i

Fill-in. New word generated: is.

Word Search level 1. looking at iSterest. Not in vocabulary. looking at thiSgs. From vocabulary,

S = n

Fill-in. New word generated: interest. (Note this was not in the vocabulary.)

Word Search level 1. looking at Nresent. Not in vocabulary. looking at oY. From common word list (since n and r have now been identified),

Y = f

Word Search level 1. looking at Nresent. Not in vocabulary.
Word Search level 2. looking at BnoLs, sJienJe, iCCense, CanO, aTiDe. All of these are not in the vocabulary or have too many possibilities. looking at TittTe. From vocabulary,

T = l

Fill-in. New word generated: little.

Word Search level 1. looking at aliDe, Nresent. Not in vocabulary.
Word Search level 2. looking at BnoLs, sJienJe, iCCense. Not in vocabulary. looking at CanO. The deducing of l has enabled a unique match in the vocabulary (previously land also matched). Hence CanO = many an

C = m

Fill-in. New word generated: immense.

O = y (second part of guessing many)

Word Search level 1. looking at aliDe, Nresent. None in vocabulary.
Word Search level 2. looking at NhilosoNhy, BnoLs, sJienJe. None in vocabulary. looking at LhiJh (Note this was added to word stack 2 some time ago but had not yet been reached, a consequence of adding new words to the top of the stack.) From common word list (there are so many constraints at this stage that letter characteristic checks are not made),

L = w

J = c

Fill-in. Words generated: which, science. (Finally have deduced scienc despite it not being in the vocabulary, i.e. deduced it letter by lett

Word search level 1. looking at Bnows, aliDe, Nresent. None in vocabulary.
Word search level 2. looking at sNecKlation, NhilosoNhy. None in vocabulary. looking at aPoKt (This was also added long ago but never reached.) From common word list,

P = b

K = u

Word search level 1. looking at sNeculation, Uob, Bnows, aliDe, Nresent None in vocabulary.

Word search level 2. looking at NhilosoNhy. looking at BeeN. There are many possibilities for this but only one matching unguessed letters in vocabulary,

B = k

Fill-in. New word generated: knows. (Finally got this. Although 'know' is in the vocabulary, the program can't get 'knows' from it.)

N = p (second part of guessing keep)

Fill-in. New words generated: present, philosophy, speculation, keep.

Fig. 8. The cryptogram after the program terminates its analysis.

there are a great many things of immense interest about which science
XVQIQ HIQ H MIQHX CHSO XVZSMA EY ZCCQSAQ ZSXQIQAX HPEKX LVZJV AJZQSJQ,

at present knows little it is the ob of philosophy to keep
HX NIQAQSX, BSELA TZXXTQ. ZX ZA XVQ UEP EY NVZTEAENVO XE BQQN

speculation about these ali e
ANQJKTHXZES HPEKX XVQAQ HTZDQ.

This is as far as the program got. See figure 8 for its final analysis (the plaintext message was paraphrased from [27]). The complete cryptogram has been deciphered except for two letters (U and D) which only occu once and then in words (job, alive) not in the vocabulary. (Note they also correspond to the low frequency letters j and v so that the frequency playoff cannot help.) This is true in general - the system deduces nearly all letters except for a few which appear only a few times and then in unknown words. Notice, though, how many of the words in this easy, but not "cooked-up", example were in the vocabulary, and how well the system did at deducing even those that were not. SVD did fairly well with the vowel identification, returning Q Z E D which turned out to be e i o y.

Summary and possible improvements

Some techniques for designing a program to solve cryptograms automaticall have been described. The program consists of an entry by identifying vowels (using a singular value decomposition method and a vowel distinguisher) and a development by guessing words (pattern matching against a vocabulary) plus referring to individual letter procedures. An implementation of these seemed fairly successful at solving simple examples.

There are a number of possible improvements which could be made. A larger vocabulary would help in guessing words and eliminating impasses caused by being unable to decipher odd letters appearing only in words not in the vocabulary lists. (Perhaps even a third word list to be used only when stuck? See [9] for a good discussion of how to decide when a cipher is solved.) Improving the information on characteristics of English is also important. This includes improving the probability estimation of the individual letter procedures, enhancing vowel distinguishing, taking contact data frequency into account, and perhaps adding such additional characteristics as pattern words or reversal data. Examining even elementary types of syntax or semantics (such as plurals, tenses, and word pieces) would help immensely. (For instance, matching 'knows' from 'know' or 'alive' from 'live' in the example worded here.) An easy way to do some of this is to allow partial word matching (e.g Tnc "matching" knows) but this could lead to sticky probability situations. Providing more informative notes and an evaluation function to determine which word to try to match next would be nice. A periodic examination of the notes to see if anything can be deduced from them yet might speed up the deciphering.

Conclusions

The current implementation appears to be capable of solving simple cryptograms and a completion and extension of the ideas outlined above could likely become quite good. It would be interesting to see how much semantic information is needed to write a system which would solve most

cryptograms in normal English. The author suspects that not too much would be necessary. The program here did quite well with very little (vowel identifying and word guessing, no very sophisticated programming) and probably just automating more of the well-known characteristics of English would suffice. Even the present system might be useful for some purposes (e.g. the "learning to problem solve" system envisioned by [26] which operates at several different levels (mentor, bookkeeper, partner, learner) while utilizing the solution of cryptograms as a basis for computer-assisted instruction).

People can generally learn to solve (easy) cryptograms with logical deduction and only little knowledge of frequency and positional characteristics (but of course a large vocabulary). Thus it should not be surprising that the program performs fairly well. (The non-human-like vowel identification could be replaced by other methods as mentioned in the section on possible approaches to the solution process. It could even be replaced by a very human-like set of hunt-and-peck procedures which make an entry by trying various tricks such as single letters, quotes, common words, frequency (letter, initial, final), and positional data (see [25], [17]). Another change would be doing the heavy computation like table generation and SVD only when normal solving techniques fail.) It may accordingly be of interest to note that yet another fairly substantial-appearing intellectual activity is "easier than first thought" and can be understood by programming a computer to perform it.

Acknowledgements

This originated as a independent project at Rice University sponsored by Prof. John Brelsford. Later versions were programmed at the IBM San Jose Research Laboratory and the IBM T. J. Watson Research Laboratory. I thank David Reed and Peter Blatman for comments, and especially Prof. Don Morrison for a copy of his paper on the SVD method. Finally, I would like to express a debt of gratitude to Prof. Joseph Schatz for many stimulating philosophical conversations.

REFERENCES

1. Barker, W.G. Cryptanalysis of the Simple Substitution Cipher with Word Divisions. (Laguna Hills, Calif.: Aegean Park Press, 1975).
2. Bobrow, D.G. and Raphael, B. New programming languages for artificial intelligence research. ACM Computing Surveys 6: 155-174 (1974)
3. Carroll, J., et.al. (eds) The American Heritage Word Frequency Book. (Boston: Houghton Mifflin, 1971), pp. 565-567.
4. Carroll, L. The hunting of the snark. in Alice in Wonderland and Other Favorites. (New York: Washington Square Press, 1951) (original 1876).
5. Edwards, D.J. OCAS - On-line Cryptanalysis Aid System. MIT Project MAC, TR-27 (May 1966).
6. Feistel, H. Cryptography and computer privacy. Scientific American 228: 15-23 (May 1973).
7. Feistel, H., Notz, W.A., and Smith, J.A. Some cryptographic techniques for machine-to-machine data communications. Proc. IEEE 63: 1545-1554 (1975).
8. Friedman, W.F. Elementary Military Cryptography. (Laguna Hills, Calif.: Aegean Park Press, 1976) (original 1935).
9. Friedman, W.F. and E.S. The Shakespearean Ciphers Examined. (Cambridge: Cambridge University Press, 1957).
10. Gaines, H.F. Cryptanalysis: A Study of Ciphers and Their Solutions. (New York: Dover Publications, 1956) (original 1939).
11. Gantmacher, F.R. The Theory of Matrices, Vol 2. (New York: Chelsea Pub. Co., 1960).
12. Girsdansky, M.B. Cryptology, the computer, and data privacy. Computers and Automation 21: 12-19 (April 1972).
13. Goddard, E. and T. Cryptodyct. (P.O. Box 441, Marion, Iowa 52302, 19
14. Golub, G. Singular value decomposition of a complex matrix. Communications ACM 12: 564-565 (1969), algorithm 358.
15. Golub, G. and Reinsch, C. Singular value decomposition and least squares solutions. Numerische Mathematik 14: 403-420 (1970).
16. Hammer, C. Signature simulation and certain cryptographic codes. Communications ACM 14: 3-14 (Jan 1971).
17. Harris, F.A. Solving Simple Substitution Ciphers. (Bethesda, Md.: American Cryptogram Assoc., 1959) (original 1943).

18. Hitt, P. Manual for the Solution of Military Ciphers. (Laguna Hills, Calif.: Aegean Park Press, 1976) (original 1920).

19. Kahn, D. The Codebreakers: The Story of Secret Writing. (New York: Macmillan Co., 1967).

20. Knuth, D.E. The Art of Computer Programming, Vol 3: Sorting and Searching. (Reading, Mass.: Addison-Wesley, 1973), section 6.3.

21. Mellen, G.E. Cryptology, computers, and common sense. Proc AFIPS Conf 42: 569-579 (1973).

22. Meyer, C.H. and Tuchman, W.L. Pseudorandom codes can be cracked. Electronic Design 23: 74-76 (Nov 1972).

23. Moler, C. and Morrison, D. Singluar value analysis of cryptograms. to appear.

24. Newell, A. and Simon, H. Human Problem Solving. (Englewood Cliffs, N.J.: Prentice Hall, 1972), chaps 5-7.

25. Ohaver, M.E. Cryptogram Solving. (Columbus, Ohio: Etcetera Press, 1973) (original 1933).

26. Peelle, H.A. and Riseman, E.M. Four faces of HAL: a framework for using artificial intelligence techniques in computer-assisted instruction. IEEE Trans. on Systems, Man, and Cybernetics SMC-5: 375-380 (May 1975).

27. Russell, B. Bertrand Russell Speaks His Mind. (New York: World Publishing Co, 1960), pp. 9-10.

28. Shannon, C. Communication theory of secrecy systems, Bell System Technical J. 28: 645-719 (1949).

29. Silver, R. Decryptor, pp. 57-60 in MIT Lincoln Laboratory Quarterly Progress Report, Division 5 (Information Processing), (Dec 1959).

30. Sinkov, A. Elementary Cryptanalysis: A Mathematical Approach. (New York: Random House, 1968).

31. Stewart, G.W. Introduction to Matrix Computation. (New York: Academic Press, 1973).

32. Tuckerman, B. A study of the Vigenere-Vernam single and multiple loop enciphering systems. IBM Research Report 2879 (Yorktown Heights N.Y., May 1970).

33. Tuckerman, B. Solution of a substitution-fractionation-transposition cipher. IBM Research Report RC 4531, (Yorktown Heights, N.Y., Sept 1973).

Appendix: The Moler-Morrison VFC Partitioning Theorem (SVD algorithm)

This proposes a criteria for partitioning the alphabet of a text (cipher or otherwise) into vowels and consonants (based on the signs of the second left and right singular vectors from a singular value decomposition). The criteria is shown to satisfy the vfc (vowel follows consonant) rule, i.e. the proportion of vowels following vowels is less than that of vowels following consonants. Many languages, including English, have predominantly vfc texts. (The theorem and proof are adapted from [23].

Define n-element column vectors v and c by
$v_i = 1$ if the i-th letter of the alphabet is a vowel, 0 otherwise.
$c_i = 1$ if the i-th letter of the alphabet is a consonant, 0 otherwise.
Then the vfc rule can be expressed as

$$\frac{v^T A v}{v^T A (v+c)} < \frac{c^T A v}{c^T A (v+c)}$$

where A is the digram matrix of the text. (This is the same rule given earlier in the paper. Note that A is non-negative.) Cross-multiplying and cancelling the common term yields

$$(v^T A v)(c^T A c) - (v^T A c)(c^T A v) < 0.$$

The singular value decomposition of A is $X^T AY = \mathrm{diag}(\sigma_1, \ldots, \sigma_n)$. So $A = \sigma_1 x_1 y_1^T + \sigma_2 x_2 y_2^T + \ldots + \sigma_r x_r y_r^T$ where r = rank A and x_i and y_i ar the i-th column vectors of X and Y. The following theorem can now be stated (assuming that the digram matrix A is actually (approximately) of rank 2).

Theorem (Moler-Morrison VFC Partitioning)

Let A be a non-negative matrix of rank 2 with singular value expansion $\sigma_1 x_1 y_1^T + \sigma_2 x_2 y_2^T$. Use the following to determine v and c:

$v_i = 1$ if $x_{i2} > 0$ and $y_{i2} < 0$, 0 otherwise.

$c_i = 1$ if $x_{i2} < 0$ and $y_{i2} > 0$, 0 otherwise.

Then the vfc rule

$D = (v^T Av)(c^T Ac) - (v^T Ac)(c^T Av) < 0$ is satisfied.

Remarks

x_{i2} is the 2nd element in the i-th column vector. The SVD algorithm is the direct analogue of the theorem, namely the i-th letter of the alphabet is a

vowel	if $x_{i2} > 0$ and $y_{i2} < 0$
consonant	if $x_{i2} < 0$ and $y_{i2} > 0$
neuter	if $\mathrm{sign}(x_{i2}) = \mathrm{sign}(_{i2})$

Proof

Let $z_i = \sigma_i x_i$ and note that sign (z_i) = sign (x_i) since $\sigma_i \geq 0$. Now substitute $A = z_1 y_1^T + z_2 y_2^T$ into the expression for D. This yields

$$
\begin{array}{llll}
 & (v^T z_1 y_1^T v)(c^T z_1 y_1^T c) & + & (v^T z_2 y_2^T v)(c^T z_1 y_1^T c) \\
+ & (v^T z_1 y_1^T v)(c^T z_2 y_2^T c) & + & (v^T z_2 y_2^T v)(c^T z_2 y_2^T c) \\
- & (c^T z_1 y_1^T c)(c^T z_1 y_1^T v) & - & (v^T z_2 y_2^T c)(c^T z_1 y_1^T v) \\
- & (v^T z_1 y_1^T c)(c^T z_2 y_2^T v) & - & (v^T z_2 y_2^T c)(c^T z_2 y_2^T v)
\end{array}
$$

Terms 1 and 5 cancel each other as do terms 4 and 8. (Just calculate the inner products recalling that x_i and y_i are column vectors of X and Y. e.g. $v^T z_1 y_1^T v = (\Sigma v_i z_{i1})(\Sigma y_{k1} v_k)$.)

Thus

$$
\begin{array}{llll}
D = & (v^T z_2 y_2^T v)(c^T z_1 y_1^T c) & + & (v^T z_1 y_1^T v)(c^T z_2 y_2^T c) \\
- & (v^T z_2 y_2^T c)(c^T z_1 y_1^T v) & - & (v^T z_1 y_1^T c)(c^T z_2 y_2^T v)
\end{array}
$$

Label these 8 elements $\underline{1} \ldots \underline{8}$. Which of them are negative? By the Perron-Frobenius Theorem, z_1 and $y_.$ have (all) non-negative components (see Lemma). c and v have non-negative components. Thus $\underline{2},\underline{3},\underline{6},\underline{7}, \geq 0$.

Now consider the valuations of v_i and c_i as given in the theorem statement. These imply that in $\underline{1},\underline{4},\underline{5}$, and $\underline{8}$, the only negative inner products are $y_2^T v$ (from $\underline{1}$ and $\underline{8}$) and $c^T z_2$ (from $\underline{4}$ and $\underline{8}$). Hence $\underline{1},\underline{4} < 0$ and $\underline{5},\underline{8} \geq 0$.

Accordingly all four terms in D are negative and D is negative as desired

Lemma

Show z and y_1 have (all) non-negative components.

Proof: The Perron-Frobenius theorem (see [11, p. 53, thm 2]) implies t if M is a non-negative, irreducible matrix, then the eigenvector corresponding to the maximal eigenvalue has positive components. The A here comes from the singular value decomposition $X^T AY = \mathrm{diag}(\sigma_1,\ldots,\sigma_n)$. So the x_i and y_i are eigenvectors of AA^T and $A^T A$ respectively. In particular, $AA^T x_1 = \lambda_1 x_1$ and $A^T A y_1 = \kappa_1 y_1$ where λ_i and κ_i are the appropriate eigenvalues. Now A non-negative and irreducible implies that AA^T and $A^T A$ are also. Hence since σ_1 is maximal we have $(\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0)$ and $\sigma_i = \sqrt{\lambda_i} = \sqrt{\kappa_i}$, λ_1 and κ_1 are maximal eigenvalues. Thus by the Perron-Frobenius theorem, x_1 and y_1 have positive components. Since $z_1 = \sigma_1 x_1$, both z_1 and y_1 have non-negative components.