Classical Cryptanalysis Tutorials

The best ones reviewed and reformatted by David Maskill, collected from around the internet and rated on a scale of 1-5 for how important it is that you read them (i.e. how much you learn from them) and another rating for length. It's better to read them in the order I've put them in and use my ratings to help decide whether to skip sections or not.

Before reading any of this, it is probably a good idea to quickly make sure you know how certain ciphers work. There are hundreds of different ones but here are the basics and the ones most likely to be used (the bold ones are those that you should know like the back of your hand):

- Atbash Cipher
- **ROT13 Cipher**
- **Caesar Cipher**
- **Affine Cipher**
- **Rail-fence Cipher**
- Baconian Cipher
- **Polybius Square Cipher**
- **Simple Substitution Cipher**
- Codes and Nomenclators Cipher
- **Columnar Transposition Cipher**
- **Autokey Cipher**
- Running Key Cipher
- **Vigenère, Beaufort, Variant and Gronsfeld Cipher**s
- Homophonic Substitution Cipher
- Four-Square Cipher
- Hill Cipher
- **Playfair Cipher**
- ADFGVX Cipher
- Bifid Cipher
- Straddle Checkerboard Cipher
- Trifid Cipher

Monogram, Bigram and Trigram frequency counts importance: 5/5. Length: 1/5

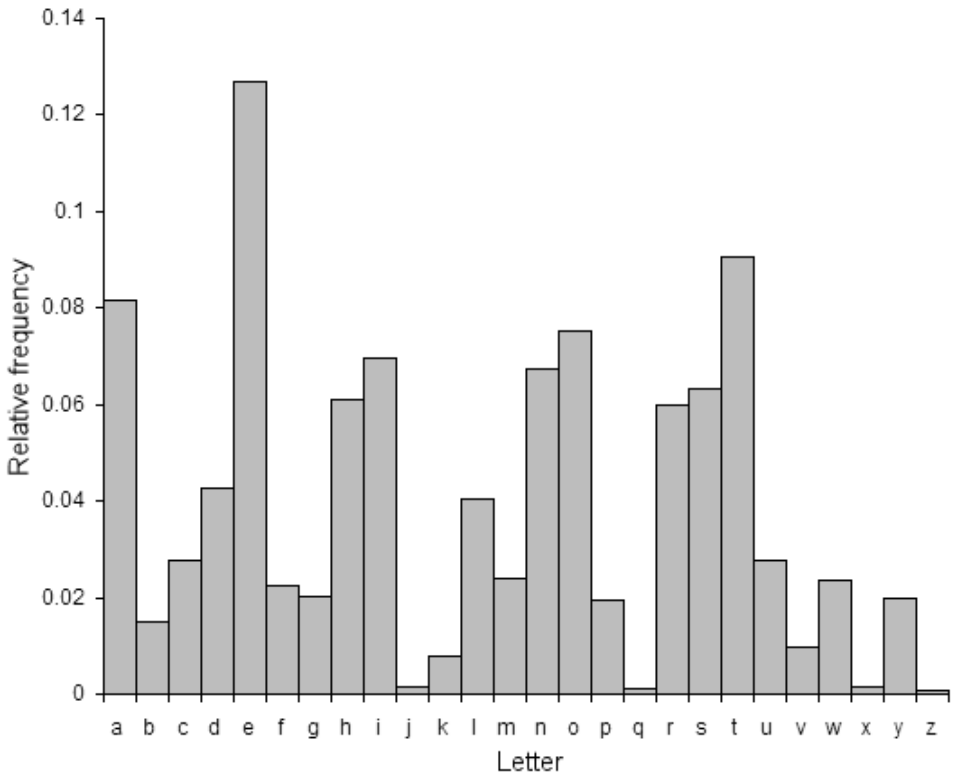
Introduction to Frequency Analysis

Frequency analysis is the practice of counting the number of occurrences of different ciphertext characters in the hope that the information can be used to break ciphers. Frequency analysis is not only for single characters, it is also possible to measure the frequency of bigrams (also called digraphs), which is how often pairs of characters occur in text. Trigram frequency counts measure the occurrence of 3 letter combinations.

When talking about bigram and trigram frequency counts, this page will concentrate on text characterisation as opposed to solving polygraphic ciphers e.g. playfair. The difference is that text characterisation depends on all possible 2 character combinations, since we wish to know about as many bigrams as we can (this means we allow the bigrams to overlap). When cracking playfair, we do not allow the bigrams to overlap.

Monogram Counts

Monogram frequency counts are most effective on substitution type ciphers such as the caesar cipher, substitution cipher, polybius square etc. It works because natural english text follows a very specific frequency distribution, which is not masked by substitution ciphers. The distribution looks like:



Chi-Squared statistic in Cryptanalysis importance: 4/5. Length: 2/5

The Chi-squared Statistic is a measure of how similar two categorical probability distributions are. If the two distributions are identical, the chi-squared statistic is 0, if the distributions are very different, some higher number will result. The formula for the chi-squared statistic is:

$$\chi^2(C,E) = \sum_{i=A}^{i=Z} \frac{(C_i - E_i)^2}{E_i}$$

where C_A is the count (not the probability) of letter A, and E_A is the expected count of letter A.
This page will describe the use of the chi-squared statistic for cryptanalysis. Ordinarily, statisticians use the chi-squared statistic for measuring the goodness of fit of data. Unlike statisticians, we make no assumptions about the distribution of our data, and draw no conclusions about the significance of the result. We simply use the method to suggest a possible decryption.

If we were to try solving a Caesar cipher by hand, a good first step would be to calculate the frequency distribution of the ciphertext characters. We could then compare them to the frequency distribution of english, and by shifting the two frequency distributions relative to one another we could find the shift that was used to encipher the plaintext. This would occur when the shifted english frequencies line up with the ciphertext frequencies i.e. common letters in english are common in the ciphertext and rare english letters are rare in the ciphertext.

The chi-squared statistic is a way for a computer to essentially perform this procedure. Let us say we have a message enciphered using the Caesar cipher:

aoljhlzhyjpwolypzvulvmaollhysplzaruvduhukzptwslzajpwoly

zpapzhafwlvmzbizapabapvujpwolypudopjolhjoslaalypuaolwsh

pualeapzzopmalkhjlyahpuubtilyvmwshjlzkvduaolhswohila

We also know the probabilities of characters occurring in normal english text. The chi-squared statistic uses counts, not probabilities. As a result we need to use the probabilities to calculate the expected count for each letter. If the letter E occurs with a probability of 0.127, we would expect it to occur 12.7 times in 100 characters. To calculate the expected count just multiply the probability by the length of the ciphertext. The cipher shown above is 162 characters, so we expect E to appear 162*0.127 = 20.57 times.

To solve the Caesar cipher, we decipher the ciphertext with each of the 25 possible keys and calculate the Chi-squared statistic for each key. This compares the letter counts in each decryption with what we would expect the counts to be if the text were english. To calculate the Chi-squared statistic for the ciphertext above, we see the letter A appears 18 times. If it were english, we would expect it to appear 162*0.082 = 13.284 times. Using this information we calculate the following

$$\frac{(18 - 13.284)^2}{13.284} = 1.674$$

We also need to perform this procedure for the other letters, then add the 26 results that we get. The result of this is around 1634.09. To find the correct key we have to do this for each key, the results of this can be seen below:

decryption key	plaintext	chi-squared
0	AOLJHLZHYJPWOLYPZVULVMAOL ...	1634.09
1	ZNKIGKYGXIOVNKXOYUTKULZNK ...	3441.13
2	YMJHFJXFWHNUMJWNXTSJTKYMJ ...	2973.71
3	XLIGEIWEVGMTLIVMWSRISJXLI ...	1551.67
4	WKHFDHVDUFLSKHULVRQHRIWKH ...	1199.40
5	VJGECGUCTEKRJGTKUQPGQHVJG ...	1466.62
6	UIFDBFTBSDJQIFSJTPOFPGUIF ...	1782.26
7	THECAESARCIPHERISONEOFTHE ...	**33.67**
8	SGDBZDRZQBHOGDQHRNMDNESGD ...	1747.07
9	RFCAYCQYPAGNFCPGQMLCMDRFC ...	1386.62
10	QEBZXBPXOZFMEBOFPLKBLCQEB ...	3423.96
11	PDAYWAOWNYELDANEOKJAKBPDA ...	809.38
12	OCZXVZNVMXDKCZMDNJIZJAOCZ ...	4646.96
13	NBYWUYMULWCJBYLCMIHYIZNBY ...	724.11
14	MAXVTXLTKVBIAXKBLHGXHYMAX ...	2159.43
15	LZWUSWKSJUAHZWJAKGFWGXLZW ...	1787.26
16	KYVTRVJRITZGYVIZJFEVFWKYV ...	3527.17
17	JXUSQUIQHSYFXUHYIEDUEVJXU ...	2967.66
18	IWTRPTHPGRXEWTGXHDCTDUIWT ...	1368.70
19	HVSQOSGOFQWDVSFWGCBSCTHVS ...	929.17
20	GURPNRFNEPVCUREVFBARBSGUR ...	461.19
21	FTQOMQEMDOUBTQDUEAZQARFTQ ...	4395.68
22	ESPNLPDLCNTASPCTDZYPZQESP ...	703.43
23	DROMKOCKBMSZROBSCYXOYPDRO ...	1226.79
24	CQNLJNBJALRYQNARBXWNXOCQN ...	1817.85
25	BPMKIMAIZKQXPMZQAWVMWNBPM ...	2939.16

We can now deduce that the key used to encipher the message was 7, since the chi-squared statistic is much lower when calculated on the text deciphered using that key.

Index of Coincidence Importance: 5/5. Length: 3/5

Summary

The index of coincidence is a measure of how similar a frequency distribution is to the uniform distribution. The I.C. of a piece of text does not change if the text is enciphered with a substitution cipher. It is defined as:

$$I.C. = \frac{\sum_{i=A}^{i=Z} f_i(f_i - 1)}{N(N-1)}$$

where f_i is the count of letter i (where i = A,B,...,Z) in the ciphertext, and N is the total number of letters in the ciphertext.

Introduction

Counting the frequency of letters is an important part of the cryptanalysis of most classical ciphers. If we compare the frequencies of text enciphered with a substitution cipher and another piece of text enciphered with the Vigenere cipher, we see that the frequency distribution is much flatter for the Vigenere text. The substitution cipher frequency distribution looks much 'spikier' or 'rougher' than the Vigenere distribution (see these images below for an example). The index of coincidence is a way of turning our intuitions about spikiness or roughness of the frequencies into a number. If the frequencies are very spiky, we get a higher number, if the frequencies are all roughly the same we get a lower number.
If we have a piece of english text roughly 1000 characters in length, we can be reasonably sure that the letter 'E' will appear around 127 times. As the text gets longer, we expect the proportion to remain roughly the same, in other words the probability of finding an 'E' is around 12.7%, or

$$p_E = 0.127$$

We also have the condition that all of the probabilities must add to 1, i.e.

$$\sum_{i=A}^{i=Z} p_i = 1$$

A Measure of Roughness

If all of the letters occurred with the same frequency, the probability for each letter would have to be 1/26, or around 0.0385. This would give a completely flat distribution. The amount that the probability of 'E', $p_E$differs from 1/26 is p_E - (1/26) = 0.127 - 0.0385 = 0.0885. If we wanted to calculate the amount that all the letter probabilities differed from (1/26), we could calculate the sum of the difference for each letter. Unfortunately some letters would have a positive difference, and some would have a negative difference. This would mean the total difference would cancel out, leaving us with a sum of 0 every time.

$$\sum_{i=A}^{i=Z} \left(p_i - \frac{1}{26} \right) = \sum_{i=A}^{i=Z} p_i - \sum_{i=A}^{i=Z} \frac{1}{26} = 1 - 26\frac{1}{26} = 0$$

To circumvent this problem, we could square the difference (p_i - (1/26)), ensuring the result is always positive. In this way we could calculate a measure of roughness, we will call it M.R.

$$\text{M.R.} = \left(p_A - \frac{1}{26}\right)^2 + \left(p_B - \frac{1}{26}\right)^2 + \ldots + \left(p_Z - \frac{1}{26}\right)^2$$

or more succinctly

$$\text{M.R.} = \sum_{i=A}^{i=Z} \left(p_i - \frac{1}{26}\right)^2$$

Expanding out, we get

$$\begin{aligned}\text{M.R.} &= \sum_{i=A}^{i=Z} p_i^2 - \sum_{i=A}^{i=Z} 2p_i \left(\frac{1}{26}\right) + \sum_{i=A}^{i=Z} \left(\frac{1}{26}\right)^2 \\ &= \sum_{i=A}^{i=Z} p_i^2 - \left(\frac{2}{26}\right) \sum_{i=A}^{i=Z} p_i + 26 \left(\frac{1}{26}\right)^2\end{aligned}$$

but we know the sum of p_i over all the letters is 1, so this further simplifies to

$$\text{M.R.} = \sum_{i=A}^{i=Z} p_i^2 - \frac{2}{26} + \frac{2}{26} = \sum_{i=A}^{i=Z} p_i^2 - \frac{1}{26}$$

If we calculate the sum of squares of each letter probability using the probabilities from e.g. wikipedia, we get a value of 0.066. If all our probabilities were (1/26), we get 0.038. As a result, our M.R. will give a number of 0.066-0.038 = 0.028 if the frequency distribution is similar to English, and a value of 0.038-0.038 = 0 if the frequency distribution is flat.

Approximating M.R.

What does our formula really mean? If p_A is the probability of a randomly selected letter being A, then p_A^2 is the probability that 2 letters selected at random will both be A. In the same manner, p_B^2 is the probability that 2 letters picked at random will be the letter B. If want to know the probability that two letters picked at random will be the same, but we don't care which letter it is, we calculate it like this

$$p_A^2 + p_B^2 + \ldots + p_Z^2 = \sum_{i=A}^{i=Z} p_i^2$$

Using this information, we can approximate the calculation of M.R. without actually calculating p_i for each letter. If there are 10 A's, there are 10 possible A's to pick. If we want to pick a second A, there are 9 left to choose from. This means there are 10*9 different ways to choose an A. In this count, each pair has been counted twice since the same pair can be obtained in 2 orders. Our final number of ways is (1/2)*10*9. If we have a cipher text with f_A occurrences of the letter A, the number of pairs of A's that can be formed is $f_A(f_A\text{-}1)/2$. In the same way, the number of ways to choose 2 B's is $f_B(f_B\text{-}1)/2$. If we want to know how many ways to choose 2 of the same letter, irrespective of the actual letter, we could calculate it as follows:

$$\sum_{i=A}^{i=Z} \frac{f_i(f_i-1)}{2}$$

If there are N letters in our ciphertext, there are N(N-1)/2 total ways of choosing 2 letters. The chance of 2 letters being the same is the total number of ways of choosing 2 of the same letter, divided by the total number of ways of choosing 2 letters. This is

$$\frac{\sum_{i=A}^{i=Z} f_i(f_i-1)}{N(N-1)}$$

Since this represents the chance that 2 letters are alike, it is called the Index of Coincidence, or I.C.

An Example

We will compare two ciphertexts, one which has been enciphered with the simple substitution cipher, and another that has been enciphered with the Vigenere cipher with a period of 6. The substitution cipher:

wmzfxtdhzfngfwxwnwxjevxdmzoxfkvxdmzowmkwmkfgzzexenfzpjotkebmneloz

lfjpbzkofxwvjefxfwfjpfngfwxwnwxeszyzobdhkxewzawvmkokvwzopjoklxppz

ozewvxdmzowzawvmkokvwzoxwlxppzofpojtvkzfkovxdmzoxewmkwwmzvxdmzokh

dmkgzwxfejwfxtdhbwmzkhdmkgzwfmxpwzlxwxfvjtdhzwzhbrntghzl

The Vigenere Example:

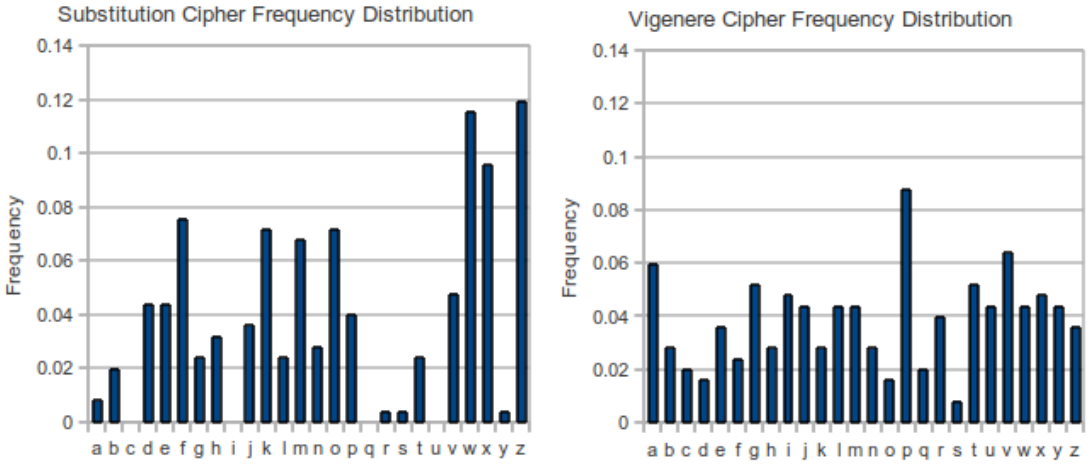
vptzmdrttzysubxaykkwcjmgjmgpwreqeoiivppalrujtlrzpchljftupucywvsyi

uuwufirtaxagfpaxzxjqnhbfjvqibxzpotciiaxahmevmmagyczpjxvtndyeuknul

vvpbrptygzilbkeppyetvmgpxuknulvjhzdtgrgapygzrptymevppaxygkxwlvtia

wlrdmipweqbhpqgngioirnxwhfvvawpjkglxamjewbwpvvmafnlojalh

If we look at the frequency distributions of both of these ciphertexts, it seems the Vigenere ciphertext is flatter than the substitution cipher.



If we calculate the I.C. of these two ciphertexts, we get:

Cipher	I.C.
Substitution	0.066
Vigenere	0.042

The I.C. is much closer to what we expect for English text for the substitution cipher. This means we can differentiate between ciphers easily using only the ciphertext.

Relation to Chi-squared statistic

The Chi-squared statistic compares two different probability distributions, whereas the I.C. compares one probability distribution to the uniform distribution. The measure of roughness we derived above (the M.R.) is related to the Index of Coincidence in the following way:

$$\mathrm{M.R.}(P) = \mathrm{I.C.}(P) - \frac{1}{26}$$

where P is the frequency distribution p_A,p_B... etc. The Chi-squared statistic is also related to M.R.,

$$\mathrm{M.R.}(P) = \frac{1}{26}\chi^2(P, U)$$

$$\chi^2(P, U) = \sum_{i=A}^{i=Z} \frac{(P_i - U_i)^2}{U_i}$$

where U is the uniform distribution p_A=p_B=...=1/26.

The Index of Coincidence - the Chi Test, the Kappa Test, and the mixed-alphabet Vigenere

importance: 3/5. Length: 3/5

William Friedman's discovery of the Incidence of Coincidence was arguably the greatest breakthrough in cryptanalysis since Al Kindi's development of frequency analysis in the 9th century.

The Index of Coincidence was first described by Friedman in a paper written by Friedman in 1920, and published as Riverbank Publication 22. "The Incidence Of Coincidence And Its Application In Cryptography". Friedman revised and improved his technique over the years, and was expanded on by his subordinates, after he began working for the War Department. Most significantly by Solomon Kullbeck, who created what became the phi and chi tests, derived from Friedman's Index of Coincidence, which became known as the kappa test.

A reprint of Friedman's original paper is available from Aegean Park Press as "The Riverbank Publications, Volume III". A later revision is also available as "The Index Of Coincidence And Its Applications In Cryptanalysis", Soloman Kullback's text describing the phi and chi tests is available as "Statistical Methods In Cryptanalysis", both from Aegean Park Press.

I'm going to try to explain how to calculate the Chi Test and the Kappa test, and how to use them in simple problems.

We'll start with a known fact. Plain language text has a distinctive frequency distribution. We've all seen it: "etaoin shrdlu", or whatever. Some letters appear frequently, some less frequently. Plotted in a graph, these form a recognizable shape. For a large sample of text, these are quite consistent.

```
        #
        #
        #                           #
        #                           #
#       #                   #       #
#       #       #         # #       #
#       #     # #         # #     # # #
#     # #     # #     #   # #     # # #
#   # # #     # #     #   # #     # # # #
#   # # # # # # # #     # # # # #   # # # #   #   #
# # # # # # # # #   # # # # # # #   # # # # # #   #
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
```

For a smaller sample of text, there can be variations, but the general shape is quite similar, even if the height of the peaks might be different:

```
                            #
                            #       #
                          # #       # #
#       #       #         # #       # # #
#       #       #         # #     # # # #
#       #       #         # #     # # # #       #
#     # #       #     #   # #     # # # #       #
# # # # #   # # #     # # # # #     # # # #   # # #
# # # # # # # # #   # # # # # # #   # # # # # # # #
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
```

Now suppose you were writing a computer program to automatically crack Caesar shift ciphers. Yes, I know that it's easy enough to work out all 26 possibilities, and then to pick out the right solution by eye. Or to write a computer program to generate all 26 possibilities, then to pick out the right solution by eye. But you're trying to write a program that would pick out the right solution automatically. What test would you use?

You could try to replicate what you're doing by eye - search the text to see if parts of it match words in a dictionary, see if bits match common English digrams, trigrams, tetragrams, etc. All of these are easy to do by eye, but they can be complicated to implement on computer. But what a computer can do, quickly and easily, is to determine how closely the frequency distribution of the text matches that of ordinary English text.

That's the Chi test. It gives you a measure of how similar the frequency distributions of two strings of text are.

To calculate it you need the frequency counts of the letters in the two strings, and the total number of letters in each string. It's calculated by multiplying the frequency count of one letter in the first string by the frequency count of the same letter in the second string, and then doing the same for all the other letters, and summing the result. This is divided by the product of the total number of letters in each string.

$$\chi = \frac{\sum_{i=1}^{k}(f_i.f_i^{'})}{n.n^{'}}$$

Where k is the number of distinct letters in the alphabet, f *is the number of times the i'th letter appears in the first string and f' is the number of times the i'th letter appears in the second string. And n and n' are the total number of characters in the first and second strings.*

For our automated Caesar breaker, we'd use frequency counts generated from our test decryption for one set of frequencies, and standardized frequency counts generated from large samples of English text for the other.

That's the chi test.

It gives us a way to determine if two strings have the same frequency distribution. It doesn't, though, tell us if they have similar frequency distributions. Consider a simple substiturion cipher. It doesn't have anything like the same distribution as ordinary text when it comes to the distances between peaks, but it has the same number of peaks, of the same height. They simply appear in different places, It would be nice to have a simple test by which we could determine if a text was a simple substitution of another.

And we do. That's the Index of Coincidence, or the kappa test. It's calculated by using the chi test to compare a text to itself, and then comparing the result to that of performing the kappa test on a sample of ordinary text, and to that of a random string of letters.

So where for the chi test, we did this: $\chi = \frac{\sum_{i=1}^{k}(f_i.f_i^{'})}{n.n^{'}}$

for the kappa test, we do

$$\kappa = \frac{\sum_{i=1}^{k}(f_i \, . \, f_{i-1})}{n(n-1)}$$

(There's some sort of statistician's black magic that makes them reduce counts by one when comparing distributions to themselves. I'm not going to try to explain why, I'm just going to point out that it's what they do, and you don't need to worry about it.)

What the kappa test measures is the probability that if you choose two letters from the text at random, they'll be the same letter. (OK, I am going to try to explain the -1. If you're selecting a letter from each of two distinct texts, you are selecting each randomly from a population of size n and a population of size n'. If you are selecting two letters from the same text, and you're not allowing the same letter to be chosen twice, the first letter is chosen from a population of size n, and the second from a population of size n-1.)

If you're working with an alphabet with 26 letters, the kappa of a string of random letters will be 1/26, or 0.0385, 3.85%. If you take the kappa of normal english text, your kappa will be around 0.0665.

The Index of Coincidence is done by scaling the kappa test by the kappa of random text. So if you're kappa is 0.0665, your IC is 0.0665/0.0385, or 1.73.

What happens when you calculate the IC of any of our attempts at cracking the Caesar shift? All of them, correct and incorrect, give you the same value. So will the result of applying any simple substitution cipher to the same text. The chi test can be of use in recognizing when a monoalphabetic cipher has been broken, the kappa and the IC are of no use at this.

Where the IC comes into its own is in polyalphabetic ciphers - where a number of different alphabets are used in encrypting a text.

Consider, now, the Vigenere cipher, where the text is encrypted by adding to each letter of the plaintext successive letters of the keyword, rotating back to the first letter of the plaintext after the last has been used.

```
attac katda wn
+ SECRE TSECR ET
= SXVRG DSXFR AG
```

The IC of vigenere ciphertext is very different than that of ordinary text - far closer to 1.0, or that of random text, than the 1.7 or thereabouts we'd expect from a plaintext or a monoalphabetic substitution, The more alphabets used, the closer the IC will be to 1.0.

So, how do we break a Vig? We find the period. How do we find the period? We look for slices of the ciphertext that have ICs that look like normal text. We start by working through possible key lengths. We assume the key length is two, and then calculate the IC of the even characters, and the IC of the odd characters, then take the average of these two ICs, and call that the result for key length two.

Then we assume the key length is three, split the text into three pieces, one containing the 1st, 4th, 7th, characters, one the 2nd, 5th, 8th, one the 3rd, 6th, 9th, and calculate ICs for each of them. We average these three ICs and call it the result for key length three.

Continue until you've calculated results for key lengths twice what you think is the maximum likely key length. It's tedious work by hand, absurdly easy with a computer.

Look at your results. Most of them will be close to 1.0. But your results for the actual key length will be close to 1.7, and so will be the results for multiples of the actual key length. If your actual key length is 6, you'll see peaks in your results at 6, 12, and 18.

So now you know your key length. What next? Well, if the key length was six, then your text was encrypted with six different alphabets, and since we're dealing with a standard Vig, each of those six is a simple Caesar shift. And we know how to crack Caesar shifts, we shift each 26 times, using the chi test to compare their frequency distributions to that of ordinary text. Again, a bit tedious to do by hand, but blindingly simple for a computer.

But, you say, what if it wasn't a standard Vig? What if it was a mixed alphabet Vig?

What, if they didn't use the standard Vig tableau:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
BCDEFGHIJKLMNOPQRSTUVWXYZA
CDEFGHIJKLMNOPQRSTUVWXYZAB
DEFGHIJKLMNOPQRSTUVWXYZABC
EFGHIJKLMNOPQRSTUVWXYZABCD
FGHIJKLMNOPQRSTUVWXYZABCDE
...
```

What if, instead, they used a mixed-alphabet tableau?

```
AUBCWDPOQKFZLIHETMVNSRYJGX
UBCWDPOQKFZLIHETMVNSRYJGXA
BCWDPOQKFZLIHETMVNSRYJGXAU
CWDPOQKFZLIHETMVNSRYJGXAUB
WDPOQKFZLIHETMVNSRYJGXAUBC
DPOQKFZLIHETMVNSRYJGXAUBCW
...
```

This looks as if it would be a great deal more complicated, but its not. Yes, each of the six alphabets was encrypted using a mixed alphabet, but they're the same alphabet shifted by six different amounts.

Again, we use the chi test. But now, instead shifting each piece of the text, and comparing its frequency distribution to that of standard english, we compare it to the other pieces. We shift piece one 26 times, and use the chi test to compare it to piece two. When the chi test peaks, we'll have aligned piece 1 and 2. Do the same with each piece of the text, until they're all aligned with each other.

What do we have, when we're done? We've eliminated the shifts between the separate alphabets. So what we have now is a monoalphabetic substitution. In other words, a simple substitution cipher. Which can easily be broken using the techniques we're all familiar with.

(This approach is described in Abraham Sinkov's "Elementary Cryptanalysis: A Mathematical Approach". He and Soloman Kullbeck were two of William Friedman's first three hires when he set up the Signals Intelligence Service for the War Department. The third being Frank Rowlett, who was later credited for being the key figure in the cracking of the Japanese PURPLE cipher.)

Contact Analysis importance: 5/5. Length: 4/5

In cryptanalysis, contact analysis is the study of the frequency with which certain symbols precede or follow other symbols. The method is used as an aid to breaking classical ciphers.

Contact analysis is based on the fact that, in any sample of any written language, certain symbols appear adjacent to other symbols with varying frequencies. Moreover, these frequencies are roughly the same for almost all samples of that language, even when the distribution of the symbols themselves differs significantly from normal. This is true regardless of whether the symbols being used are words or letters.

In some ciphers, these properties of the natural language plaintext are preserved in the ciphertext, and have the potential to be exploited in a ciphertext-only attack.

Although in a sense contact analysis can be considered a type of frequency analysis, most discussions of frequency analysis concern themselves with the simple probabilities of the symbols in the text:
$P(X_i = a)$ or $P((X_i = a) \cap (X_{i+1} = b))$.

Contact analysis is based on the conditional probability that certain letters will precede or succeed other letters:
$P(X_i = b \mid X_{i+1} = a)$, or $P(X_i = c \mid (X_{i-2} = a) \cap (X_{i-1} = b))$.

Where frequency analysis is based on first-order statistics, contact analysis is based on second or third-order statistics.

The process of contact analysis often begins with an attempt to categorize the symbols into classes that have different contact behavior. When the symbols represent letters, the first attempted categorization is usually between vowels and consonants, which have very different contact behaviors in all alphabetic languages.

Contact analysis for simple substitution ciphers

A common technique for applying the principles of contact analysis to simple substitution ciphers is the consonant line method. In this, the cryptanalyst first creates a chart indicating which ciphertext letters appear next to which other ciphertext letters. From this, a determination is made as to which letters are vowels and which are consonants.

English, as with other alphabetic languages, uses letters to spell out phonetically pronounceable words. This places limitations on the sequence of letters. Vowels and consonants tend to alternate, and when we see consecutive vowels or consonants, they are limited to those which can be pronounced. Vowels tend to occur adjacent to consonants, and consonants tend to occur next to vowels. Each will be found in contact with the letters of other class rather than in contact with its own.

Vowels tend to contact more distinct letters than do consonants. Vowels commonly contact the many different consonants, consonants most commonly contact the much fewer number of vowels. When vowels contact other vowels they show typical patterns: e rarely contacts o, ea is common, ae is rare. ei and ie are both common.

Using these relationships, many letters within the ciphertext can be classified as vowels (AEIOUY), high-frequency consonants (TNSHRDL), medium-frequency consonants (CMWFGPB), and low-frequency consonants (VKJXQZ), and certain letters within each group can be given provisional identifications.

An example

(For this example, uppercase letters are used to denote ciphertext, lowercase letters are used to denote plaintext or guesses at plaintext, and X~t is used to express a guess that ciphertext letter X represents the plaintext letter t.)

Suppose Eve has intercepted the cryptogram below, and suspects that it is an English text encrypted using a simple substitution cipher:

GYNVN CBFXH IXORD XIMFN DBHRJ HKBYD MIXTD XGOCR HKRHS MNDBF
GYNVK BDERO DBYYR UROOR IRIZR OQKBH HKRMO NYHDX IIBYJ NCDBF
FRJHK NCQRR EZGHQ KRYNH PRHCO NPKHJ XQYHX NHGYN VNCBP FXONT
NRJUN JRXPB IRMRX MFRJX YHJXC RONXG CQXOE XYGYN VCSCH RICHK
RSCRY JAXER CBOXG YJHKR QXOFJ XYGCR YRHXO QONHR BJURY HGORP
BIRCB YJORC RBODK MBMRO CRMXC HORBF MOXPO BIIRO CJXYH GCRMB
CDBFJ BHBIB HNXY

She will have prepared herself with charts describing the conditional probabilities for the plaintext language.

Her first task is to create a contact chart.

Contact Chart

A contact chart is a table with a column for every letter in the ciphertext, under which is written a pair of letters for each instance of that letter in the plaintext, the first being the letter prior to that instance of the letter, the second being the letter after. At the top of each column two numbers are written, the number of times that letter appears in the ciphertext (the letter frequency, which is the same as the number of letter pairs in that column), and the number of distinct letters that appear in that column, called the Variety of Contact or VOC.

```
 1 24 21 11  4 10 11 26 13 15 11  0 11 20 23  6  7 40  3  2  3  4  0 25 21  2
 2 12 13 12  5 10  7 16  8 11  9  0 10 15 15  8  7 20  4  3  3  3  0 17 10  4
 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
JX CF NB RX DR BX -Y XI HX RH HB    IF YV XR HR OK OD HM XD RR NN    FH GN IR
   DH OR NB RZ MN XO BR XM YN HR    DI VC GC NK CR HJ CC NN JN NK    IO BD EG
   KY ND YM OX BG FY JK MX RH VB    SN FD RD BF HK CH RC    JR NN    DI GN
   DF NQ TX XR BF ZH RK RR HX QB    RO MD RO XB XY KH           NC    IT BY
   KD HO NB    FR HY RS RZ RU HR    RR YV OR RB CX EO                 DG YR
   DY NB BE    PX XC BH XI NR HN    XF OY RQ XO RX YU                 DI NH
   KH XR OB    MR YY HK IB RX QR    KB JC MN    OO UO                 JQ BJ
   IY GQ HX    OJ XY YD BR HX PH    BR KC CN        OI                HN RN
   DF VS CB    BM YC JK RC YA HR    RX YH XN        II                FO QH
   CP SH OK    BJ HO GQ BR YH HR    FO OP RN        ZO                RP GN
   PI IH CB        HC NP BI FX DM    RB XH XE        KM                RM XH
   CO SR               RC IR BU          YV BX       FJ                JY XG
   RJ RB               KJ BB YO          VC XF       QR                JC GN
   PI GR               YX    CX          OT XQ       RE                NG RJ
   CY RB               NG    FB          TR QN       KY                QO GJ
   RO RR               YJ                UJ GR       PH                EY XG
   MM OR               CR                OX JR       NJ                AE RR
   RF XH               CK                YV BD       JX                OG RH
   OI OJ               JK                OH RC       IM                QO BJ
   MC GR               RX                HX HR       MX                JY XH
   DF BD               NR                   MX       FJ                HO X-
   JH                  YG                   PB       CO                MC
   HI                  CO                   RC       HI                OP
   IH                  YG                            KS                JY
                       BB                            CY                NY
                       BN                            EC
                                                     KQ
                                                     CY
                                                     YH
                                                     HB
                                                     UY
                                                     OP
                                                     IC
                                                     OC
                                                     CB
                                                     MO
                                                     CM
                                                     OB
                                                     IO
                                                     CM
```

Notice how in the column for H, there are 26 letter pairs, which indicates that H appears 26 times, but there are only 16 distinct letters in that column, giving H a VOC of 16.

While preparing this chart would have taken Eve a bit longer than generating a simple frequency count, it provides her with far more information about the structure of the text. It includes the single character frequency count as the top line of chart.

She can easily calculate the digram frequencies by counting the number of times each distinct letter appears in the left position in each column. HK appears six times.

She can calculate the trigram frequency by counting the number of times each distinct letter pair appears in each column. HKR appears three times.

In practice, she would not list all of the trigram and digram frequencies, but would simply identify the highest, by visual inspection of the chart, and perhaps highlighting those of interest. In this ciphertext, the most frequent letters are R, H, and X, the most frequent digrams are CR, HK, and RO, and the most frequent trigrams are GYN, HKR and YNV.

Just as a ciphertext can sometimes be broken only simple examination, or using only simple frequency distributions, sometimes the contact chart alone will provide sufficient insight into the text as to provide the crack necessary to break it. But for the sake of continuing the example, Eve failed to accomplish this.

Identifying vowels and consonants

Eve's next step would be to make a guess as to whether one or more letters were a consonant or a vowel. In the ciphertext, Eve might assume that R is a vowel. It has a very high contact number - 20 - indicating that it contacts every letter in the ciphertext save four.

But Eve knows that the most reliable assumption is usually that low-frequency, low-contact letters are consonants. The low-frequency consonants

make up around 20% of the a typical text. There are 314 characters in the ciphertext, 20% of which is 62. Using the chart, Eve identifies the lowest-contact letters that have a combined frequency of as close to 62 as possible, without going over.

```
Letter Freq VoC
P         6   8
I        13   8
Q         7   7
G        10   7
E         4   5
Z         2   4
S         3   4
V         4   3
U         3   3
T         2   3
A         1   2
Total: 55
```

From these, Eve would begin to build her vowel and consonant lines.

Vowel and Consonant Lines

A Consonant Line takes the form of a tee, with the letters thought to be consonants above the top line, and with the letters that precede or follow the letters at the top written below on each side of the vertical line. A Vowel Line is analogous.

Eve has identified a number of probable consonants. She would have created a consonant line by taking each probable consonant in turn, writing it at the top of the line, and then going down the column for that letter in the contact chart, taking each letter pair and writing the left letter on the left side of the consonant line and the right on the right.

```
               ATUVSZEGQIP
             ---------------------
              BBBB | BBBB
               III | II
          RRRRRRRR | RRRRRRRRRR
                 F | F
               CCC | CCCCCCC
        XXXXXXXXXX | XXXXXX
                 D | D
                 E |
                   | G
                YY | YYYYY
           HHHHHHH | H
               OOO | OOOO
                   | KKKK
                 M | MM
               JJJ |
             NNNNN | NNN
                 Z | ZZ
```

This shows which letters contact the low-frequency consonants. Those that do so the most are likely to be vowels. Any letter that does not contact the low-frequency consonants at all is likely to be a consonant itself, and can be added to the line. In this case, there are none.

This first consonant line gives Eve two probable vowels, R and X. R had been strongly suggested simply by it's high VOC

Eve decides to classify R and X as a vowels in this pass. So she prepares a vowel line

```
                   RX
             ---------------------
               BBB | BBBB
                   | FFF
               JJJ | JJ
        0000000000 | OOOOO
             CCCC | CCCCC
                R | RRRRRRRR
                D | DD
               XX | XXXXX
               EE | EE
                  | ZZ
             HHHHH | HHHHH
                   | K
                S | S
             IIIII | II
                U | U
             MMMMM | MM
                N | NNN
               QQ | QQ
              PPP | P
            YYYYYY | YY
               GGG | G
                 T |
```

his convinces Eve that O and H are consonants. C also shows up as frequently contacting vowels, but the consonant line shows it contacting consonants about as often, so she'll make no determination for it, yet.

Adding O and H produces a second consonant line.

```
               ATUVSZEGQIPOH
             ---------------------
        BBBBBBBBBB | BBBBBB
         HHHHHHHHH | HH
          NNNNNNNN | NNNNNNNNN
 RRRRRRRRRRRRRRRRR | RRRRRRRRRRRRRRRRR
               III | III
                 D | DDDD
  XXXXXXXXXXXXXXXX | XXXXXXXXXX
                 F | FF
           CCCCCCC | CCCCCCCCCC
                 K | KKKKKKKK
              OOOO | OOOOOO
                 E | E
               GGG | GGG
            YYYYYY | YYYYY
                 Q | QQQ
               MMM | MM
             JJJJJ | JJ
                 P | P
                 Z | ZZ
                   | S
```

Examining this second consonant line convinces Eve that B and N are the other two high-frequency vowels.

Vowel spacing

As Eve was identifying vowels, she was marking up the ciphertext, putting a bar over ever letter that represents a vowel. Long consonant clusters are rare in English. Five sequential consonants happen rarely (e.g., "running through"), more than five happen almost not at all. Similarly, long sequences of vowels are rare. She used these sequences, along with the number of contacts shown in the consonant line and the vowel line, in determining which letters were vowels and consonants.

After having identified the four high-frequency vowels, Eve sees that there are no instances of more than two vowels in sequence. Had there been any, it would have been an indication that she had decided incorrectly.

She finds only four five-long non-vowel sequences, EZGHQK. Of these, she has already identified E, Z, G, H, and Q as consonants. Either K is a vowel, or one of the others was mistakenly identified. (it's not uncommon for u and y to appear in the low 20%, and to be picked as consonants in the initial division.)

Vowel-vowel contact

Eve's next step is to try to determine which of the four high frequency vowels are which. She does this by looking at the frequencies of the digrams containing only these four letters.

```
  B  N  R  X
B 0  0  0  0
N 0  0  1  1
R 3  0  1  1
X 0  1  0  0
```

Eve knows that e is the vowel that is by far most likely to contact other vowels. She provisionally assigns R~e. The vowel that is most likely to precede e is i, which might be N, though the counts are so small as to make it uncertain.

She leaves the others, for now, and moves on to the trigrams to see if she can identify the high-frequency consonants.

High-Frequency Consonants

The high-frequency letters in the ciphertext are R, H, X, B, O, C, Y, N, J, and I.

Eve has identified R as e, and B, N, and X as high-frequency vowels.

Remaining are H, O, C, Y, J, and I, and the high-frequency consonants, t, n, s, r and d are likely among them.

```
  4 GYN
  4 HKR
  4 YNV
  3 DBF
  3 JHK
  3 JXY
  2 BIR
  2 BYJ
  2 CDB
  2 CRM
  2 CRY
  2 DXI
  2 FRJ
  2 GCR
  2 HJX
  2 NCB
  2 NDB
  2 NVN
  2 OCR
  2 PBI
```

```
2 QXO
2 RCB
2 RJH
2 ROC
2 VNC
2 XYG
2 XYH
2 YHG
```

Eve examines the high-frequency trigrams, to see if they can help her identify the remaining high-frequency consonants,. The one she immediately notices is HKR. She knows R~e, that H is a consonant, and that K is not a, e, i or o,. Of the trigrams consisting of two consonants followed by e, the is overwhelmingly the most common. She also sees that HH appears as a double letter, which doesn't contradict the assumption. So she provisionally assigns HKR~the.

Now she looks at the digrams;

```
7 CR
6 HK
6 RO
5 BF
5 DB
5 GY
5 JX
5 KR
5 ON
5 OR
5 XO
5 XY
5 YH
5 YN
4 BH
4 BI
4 BY
4 CB
4 IR
4 NC
4 NV
4 RH
4 RJ
4 RM
4 RY
4 YJ
```

She looks at BH. She has assumed that H is t, and is pretty sure that B is a vowel other than e. Of the four possibilities, at, it, ot, and ut, at is the most common. She decides to give B~a a try.

At this point, she has two high-frequency vowels left, i and o, which have to be either XN or NX. Purely arbitrarily, she tries NX~oi. Examining the partial plaintext, she sees that the message ends: BHNXY~atoi_, which doesn't seem right. Both oi and io are reasonably common, but as the ending of a word, ation is common, and atoi? is not.

So she changes her mind and tries NXY~ion. She now has the four high-frequency vowels BENX~aeio, and three consonants, KYH~hnt. The remaining high-frequency letters in the ciphertext are: O, C, J, and I. The unidentified high-frequency consonants are s, r and d.

She looks through the partial plaintext, looking to see where these might fit. She notices:

She looks through the partial plaintext, looking to see where these might fit. She notices:

```
_ a n n e _ e _ _ e _ e _ _ e _ _ h a t t h e
D B Y Y R U R O O R I R I Z R O Q K B H H K R
```

Most particularly, she notices that the unknown letters in this sequence include repeats of the high-frequency ciphertext letters, O and I. O is the higher-frequency of the two, so she mentally replaces it with the unidentified high-frequency consonants. The assignment O~r immediately leads to I~m, and the plaintext

```
_ a n n e _ e r r e m e m _ e r _ h a t t h e
D B Y Y R U R O O R I R I Z R O Q K B H H K R
```

Which is, of course:

```
c a n n e v e r r e m e m b e r w h a t t h e
D B Y Y R U R O O R I R I Z R O Q K B H H K R
```

At which point her plaintext is:

```
_ n i _ i _ a _ o t m o r e c o m _ _ i c a t e _ t h a n c _ m o _ c
G Y N V N C B F X H I X O R D X I M F N D B H R J H K B Y D M I X T D

o _ r _ e t h e t _ _ i c a _ _ n i _ h a c _ e r c a n n e v e r r e
X G O C R H K R H S M N D B F G Y N V K B D E R O D B Y Y R U R O O R

m e m b e r w h a t t h e _ r i n t c o m m a n _ i _ c a _ _ e _ t h
I R I Z R O Q K B H H K R M O N Y H D X I I B Y J N C D B F F R J H K
```

```
i _ w e e _ b _ t w h e n i t _ e t _ r i _ h t _ o w n t o i t _ n i
N C Q R R E Z G H Q K R Y N H P R H C O N P K H J X Q Y H X N H G Y N

_ i _ a _ _ o r i _ i e _ v i _ e o _ a m e _ e o _ _ e _ o n t _ o _
V N C B P F X O N T N R J U N J R X P B I R M R X M F R J X Y H J X C

e r i o _ _ w o r _ o n _ n i _ _ _ _ t e m _ t h e _ _ e n _ _ o _ e
R O N X G C Q X O E X Y G Y N V C S C H R I C H K R S C R Y J A X E R

_ a r o _ n _ t h e w o r _ _ o n _ _ e n e t o r w r i t e a _ v e n
C B O X G Y J H K R Q X O F J X Y G C R Y R H X O Q O N H R B J U R Y

t _ r e _ a m e _ a n _ r e _ e a r c h _ a _ e r _ e _ o _ t r e a _
H G O R P B I R C B Y J O R C R B O D K M B M R O C R M X C H O R B F

_ r o _ r a m m e r _ _ o n t _ _ e _ a _ c a _ _ a t a m a t i o n
M O X P O B I I R O C J X Y H G C R M B C D B F J B H B I B H N X Y
```

And from there, breaking the rest is trivial.

Alternative Chart

I'd noticed that the Wikipedia page on cryptanalysis was very weak on attacks on classical ciphers. It mentioned, IIRC, frequency analysis, Kasiski, and IC. Nothing on pattern words, anagramming, or contact tables.

I tried to write something, patterned on the existing page on frequency analysis, and it quickly became too long. That's what was on the scratch page you Googled. I took the first part, and posted it as contact analysis, figuring to put a second page up on cluster analysis later. That's when I hit Google's citation rules. Writing up something like this is a lot more fun than tracking down citations, trying to figure out which book I had learned these various ideas from. So I pretty much gave up on it.

With the recent surge in activity here, I decided to grab what I had intended for wikipedia, and to rewrite it for here.

When I said I was trying to figure out how to deal with Pats - simple substitution without word separation, this is what I meant. If a Pat is long enough for the statistics to be clear, they're pretty easy to deal with. If you have a good crib - as you do for the first half-dozen Pats in every issue of The Cryptogram - they're pretty easy to deal with. But short pats can be a problem. For them, these techniques *sometimes* work.

The basic approach, trying to distinguish vowels from consonants, is pretty much the essential first step. The consonant line is one method for trying to do so. I've seen others, and I often try the others. I've not found any that always work.

Another form of contact chart:

```
    |  R  H  X  B  O  C  Y|  N  J  I  D  G  K  M|  F  Q  P  E  V  S  U  T  Z  A
    | 40 26 25 24 23 21 21| 20 15 13 11 11 11 11| 10  7  6  4  4  3  3  2  2  1
----|--------------------|---------------------|-----------------------------
40-R|  1  4  2  3  6  3  4|    4  3  1       4|    1  1  1    1  1
26-H|  3  1  2  1  1  1   |  1  2  1  1  3  6  |    1  1       1
25-X|     1       5  2  5|  1     2     3    1|    1  2  1          1
24-B|     4       2  1  4|     1  4  1       1|  5    1
23-O|  5     2  1  1  3   |  5        2        |  1  2    1
21-C|  7  3     4  1      |     1     2        |     2          1
20-Y|  2  5             1|  5  4     1  2     |
----|--------------------|---------------------|-----------------------------
20-N|  1  3  2        4  1|     1     2        |        1     4        1
15-J|  1  3  5  1  1      |  1                 |                    2        1
13-I|  4     2  2     1   |        2          1|                          1
11-D|        3  5         |                1  1|           1
11-G|     1        2  3  5|                   |
11-K|  5  1     3         |  1                1|
11-M|  2     1  2  2      |  1     1          |  2
----|--------------------|---------------------|-----------------------------
10-F|  2     2           |  1  2        1    1|  1
 7-Q|  1     2     1     1|                2  |
 6-P|  1        2  1      |                1  |  1
 4-E|  2     1            |                   |                          1
 4-V|                 1   |  2             1  |
 3-S|                 2   |                   1|
 3-U|  2                  |  1                |
 2-T|                     |  1        1       |
 2-Z|  1                  |              1    |
 1-A|        1            |                   |
```

What you look for here are high-frequency letters that make frequent contact with low-frequency letters. These are usually vowels.

Another I've seen was to look at the 8 highest frequency letters and the 18 highest frequency digrams. In this case:

```
     7 CR           40 R
     6 HK           26 H
     6 RO           25 X
     5 BF           24 B
     5 DB           23 O
     5 GY           21 C
     5 JX           21 Y
     5 KR           20 N
     5 ON
     5 OR
     5 XO
     5 XY
     5 YH
     5 YN
     4 BH
     4 BI
     4 BY
     4 CB
```

What you're looking for is which of the high-frequency letters appears in the greatest number of distinct digrams. This is usually e. Of the remaining 7 high-frequency letters, the three that are found least often in contact with e are most often a, i, and o.

But again, these don't always work. What works best for me is to lay out all of the info, frequency counts, digrams, trigrams, repeated letters, ABA patterns, contact tables, consonant and vowel lines, etc., and to try to make guesses. Keeping my mind focused on what I'm looking for next is essential. I can read over something like "_anne_e__e_e__e__hatthe" dozens of times, without picking up that it means "canneverrememberwhat". It was only when I was looking at that with an eye to seeing where an r might fit, that I recognized the solution.

Ditto for keeping track of vowel separation. If you have your vowels right, your consonant clusters will be small. When you have some, but not all, of your vowels, you can be assured that the remaining vowels will be among the set of letters that appear in all of your consonant clusters. When you have the vowels right, you will have few long groups of consonants.

```
_ _ _  _ _   _ _  _   _  _ _   _
unixi salot morec ompli cated thanc
GYNVN CBFXH IXORD XIMFN DBHRJ HKBYD
```

The idea of using contact behavior to partition the ciphertext into distinct sets isn't limited to simple substitution. I was in a conversation in a SF forum, not long ago, over how strong his cipher system was. What he'd proposed was, in effect, a nomenclature. A small list of code words plus a cipher for spelling out words that weren't in the list. He was adamant that no one would be able to tell which of his code groups represented words and which would represent letters.

That is, of course, nonsense. The groups representing letters would be used to spell out words - that means they would appear in clusters. Ditto for the groups representing words. Examining the frequency with which each code group appeared in contact with each other group should quickly allow you to partition the code groups into the letters and the words. The letters can then be attacked as simple substitution. The words can then be guessed from context.

Types of Coincidence Indices: Kappa, Delta and Delta-bar importance: 4/5. Length: 3/5

The index of coincidence is useful both in the analysis of natural-language plaintext and in the analysis of ciphertext (cryptanalysis). Even when only ciphertext is available for testing and plaintext letter identities are disguised, coincidences in ciphertext can be caused by coincidences in the underlying plaintext. This technique is used to cryptanalyze the Vigenère cipher, for example. For a repeating-key polyalphabetic cipher arranged into a matrix, the coincidence rate within each column will usually be highest when the width of the matrix is a multiple of the key length, and this fact can be used to determine the key length, which is the first step in cracking the system.

Coincidence counting can help determine when two texts are written in the same language using the same alphabet. (This technique has been used to examine the purported Bible code). The *causal* coincidence count for such texts will be distinctly higher than the *accidental* coincidence count for texts in different languages, or texts using different alphabets, or gibberish texts.

To see why, imagine an "alphabet" of only the two letters A and B. Suppose that in our "language", the letter A is used 75% of the time, and the letter B is used 25% of the time. If two texts in this language are laid side by side, then the following pairs can be expected:

Pair	Probability
AA	56.25%
BB	6.25%
AB	18.75%
BA	18.75%

Overall, the probability of a "coincidence" is 62.5% (56.25% for AA + 6.25% for BB).

Now consider the case when *both* messages are encrypted using the simple monoalphabetic substitution cipher which replaces A with B and vice versa:

Pair	Probability
AA	6.25%
BB	56.25%
AB	18.75%
BA	18.75%

The overall probability of a coincidence in this situation is 62.5% (6.25% for AA + 56.25% for BB), exactly the same as for the unencrypted "plaintext" case. In effect, the new alphabet produced by the substitution is just a uniform renaming of the original character identities, which does not affect whether they match.

Now suppose that only *one* message (say, the second) is encrypted using the same substitution cipher (A,B)→(B,A). The following pairs can now be expected:

Pair	Probability
AA	18.75%
BB	18.75%
AB	56.25%
BA	6.25%

Now the probability of a coincidence is only 37.5% (18.75% for AA + 18.75% for BB). This is noticeably lower than the probability when same-language, same-alphabet texts were used. Evidently, coincidences are more likely when the most frequent letters in each text are the same.

The same principle applies to real languages like English, because certain letters, like E, occur much more frequently than other letters—a fact which is used in frequency analysis of substitution ciphers. Coincidences involving the letter E, for example, are relatively likely. So when any two English texts are compared, the coincidence count will be higher than when an English text and a foreign-language text are used.

It can easily be imagined that this effect can be subtle. For example, similar languages will have a higher coincidence count than dissimilar languages. Also, it isn't hard to generate random text with a frequency distribution similar to real text, artificially raising the coincidence count. Nevertheless, this technique can be used effectively to identify when two texts are likely to contain meaningful information in the same language using the same alphabet, to discover periods for repeating keys, and to uncover many other kinds of nonrandom phenomena within or among ciphertexts.

The same idea can be applied to a single text, where the sample is in effect compared with *itself*. Mathematically we can compute the index of coincidence **IC** for a given letter-frequency distribution as

$$\mathbf{IC} = \frac{\sum_{i=1}^{c} n_i(n_i - 1)}{N(N-1)/c},$$

where N is the length of the text and n_1 through n_c are the frequencies (as integers) of the c letters of the alphabet ($c = 26$ for monocase English). The sum of the n_i is necessarily N.

The products $n(n-1)$ count the number of combinations of n elements taken two at a time. (Actually this counts each pair twice; the extra factors of 2 occur in both numerator and denominator of the formula and thus cancel out.) Each of the n_i occurrences of the i-th letter matches each of the remaining $n_i - 1$ occurrences of the same letter. There are a total of $N(N-1)$ letter pairs in the entire text, and $1/c$ is the probability of a match for each pair, assuming a uniform random distribution of the characters (the "null model"; see below). Thus, this formula gives the ratio of the total number of coincidences observed to the total number of coincidences that one would expect from the null model.

The expected average value for the I.C. can be computed from the relative letter frequencies f_i of the source language:

$$\mathrm{IC}_{expected} = \frac{\sum_{i=1}^{c} {f_i}^2}{1/c}.$$

If all C letters of an alphabet were equally distributed, the expected index would be 1.0. The actual monographic I.C. for telegraphic English text is around 1.73, reflecting the unevenness of natural-language letter distributions. Expected values for various languages are:

Language	Index of Coincidence
English	1.73
French	2.02
German	2.05
Italian	1.94
Portuguese	1.94
Russian	1.76
Spanish	1.94

Sometimes similar values are reported without the normalizing denominator, for example $0.067 = 1.73/26$ for English; such values may be called κ_p ("kappa-plaintext") rather than "I.C.", with κ_r ("kappa-random") used to denote the denominator $1/c$ (which is the expected coincidence rate for a uniform distribution of the same alphabet, $0.0385 = 1/26$ for English).

Generalization

The above description is only an introduction to use of the index of coincidence, which is related to the general concept of correlation. Various forms of Index of Coincidence have been devised; the "delta" I.C. (given by the formula above) in effect measures the autocorrelation of a single distribution, whereas a "kappa" I.C. is used when matching two text strings. Although in some applications constant factors such as c and N can be ignored, in more general situations there is considerable value in truly *indexing* each I.C. against the value to be expected for the null hypothesis (usually: no match and a uniform random symbol distribution), so that in every situation the expected value for no correlation is 1.0. Thus, any form of I.C. can be expressed as the ratio of the number of coincidences actually observed to the number of coincidences expected (according to the null model), using the particular test setup.

From the foregoing, it is easy to see that the formula for **kappa I.C.** is

$$\mathrm{IC} = \frac{\sum_{j=1}^{N} [a_j = b_j]}{N/c},$$

where N is the common aligned length of the two texts *A* and *B*, and the bracketed term is defined as 1 if the j-th letter of text *A* matches the j-th letter of text *B*, otherwise 0.

A related concept, the "bulge" of a distribution, measures the discrepancy between the observed I.C. and the null value of 1.0. The number of cipher alphabets used in a polyalphabetic cipher may be estimated by dividing the expected bulge of the delta I.C. by the observed bulge, although in many cases (such as when a repeating key was used) better techniques are available.

Example

As a practical illustration of the use of I.C., suppose that we have intercepted the following ciphertext message:

QPWKA LVRXC QZIKG RBPFA EOMFL JMSDZ VDHXC XJYEB IMTRQ WNMEA
IZRVK CVKVL XNEIC FZPZC ZZHKM LVZVZ IZRRQ WDKEC HOSNY XXLSP
MYKVQ XJTDC IOMEE XDQVS RXLRL KZHOV

(The grouping into five characters is just a telegraphic convention and has nothing to do with actual word lengths.) Suspecting this to be an English plaintext encrypted using a Vigenère cipher with normal A–Z components and a short repeating keyword, we can consider the ciphertext "stacked" into some number of columns, for example seven:

QPWKALV
RXCQZIK
GRBPFAE
OMFLJMS
DZVDHXC
XJYEBIM
TRQWN…

If the key size happens to have been the same as the assumed number of columns, then all the letters within a single column will have been enciphered using the same key letter, in effect a simple Caesar cipher applied to a random selection of English plaintext characters. The corresponding set of ciphertext letters should have a roughness of frequency distribution similar to that of English, although the letter identities have been permuted (shifted by a constant amount corresponding to the key letter). Therefore if we compute the aggregate delta I.C. for all columns ("delta bar"), it should be around 1.73. On the other hand, if we have incorrectly guessed the key size (number of columns), the aggregate delta I.C. should be around 1.00. So we compute the delta I.C. for assumed key sizes from one to ten:

Size	Delta-bar I.C.
1	1.12
2	1.19
3	1.05

4	1.17
5	1.82
6	0.99
7	1.00
8	1.05
9	1.16
10	2.07

We see that the key size is most likely five. If the actual size is five, we would expect a width of ten to also report a high I.C., since each of its columns also corresponds to a simple Caesar encipherment, and we confirm this. So we should stack the ciphertext into five columns:

QPWKA
LVRXC
QZIKG
RBPFA
EOMFL
JMSDZ
VDH…

We can now try to determine the most likely key letter for each column considered separately, by performing trial Caesar decryption of the entire column for each of the 26 possibilities A–Z for the key letter, and choosing the key letter that produces the highest correlation between the decrypted column letter frequencies and the relative letter frequencies for normal English text. That correlation, which we don't need to worry about normalizing, can be readily computed as

$$\chi = \sum_{i=1}^{c} n_i f_i,$$

where n_i are the observed column letter frequencies and f_i are the relative letter frequencies for English. When we try this, the best-fit key letters are reported to be "EVERY," which we recognize as an actual word, and using that for Vigenère decryption produces the plaintext:

MUSTC HANGE MEETI NGLOC ATION FROMB RIDGE TOUND ERPAS
SSINC EENEM YAGEN TSARE BELIE VEDTO HAVEB EENAS SIGNE
DTOWA TCHBR IDGES TOPME ETING TIMEU NCHAN GEDXX

from which one obtains:

MUST CHANGE MEETING LOCATION FROM BRIDGE TO UNDERPASS
SINCE ENEMY AGENTS ARE BELIEVED TO HAVE BEEN ASSIGNED
TO WATCH BRIDGE STOP MEETING TIME UNCHANGED XX

after word divisions have been restored at the obvious positions. "XX" are evidently "null" characters used to pad out the final group for transmission.

This entire procedure could easily be packaged into an automated algorithm for breaking such ciphers. Due to normal statistical fluctuation, such an algorithm will occasionally make wrong choices, especially when analyzing short ciphertext messages.

Kasiski Examination importance: 4/5. Length: 2/5

The Kasiski examination allows a cryptanalyst to deduce the length of the keyword used in the polyalphabetic substitution cipher. Once the length of the keyword is discovered, the cryptanalyst lines up the ciphertext in *n* columns, where *n* is the length of the keyword. Then, each column can be treated as the ciphertext of a monoalphabetic substitution cipher. As such, each column can be attacked with frequency analysis.

The Kasiski examination involves looking for strings of characters that are repeated in the ciphertext. The strings should be three characters long or more for the examination to be successful. Then, the distances between consecutive occurrences of the strings are likely to be multiples of the length of the keyword. Thus finding more repeated strings narrows down the possible lengths of the keyword, since we can take the greatest common divisor of all the distances.

The reason this test works is that if a repeated string occurs in the plaintext, and the distance between corresponding characters is a multiple of the keyword length, the keyword letters will line up in the same way with both occurrences of the string. For example, consider the plaintext:

crypto is short for cryptography.

"crypto" is a repeated string, and the distance between the occurrences is 20 characters. We will line up the plaintext with first a six-character keyword "abcdef" (6 does not divide 20) and a five-character keyword "abcde" (5 divides 20).

abcdefabcdefabcdefab**cdefab**cdefabc
crypto is short for **crypto**graphy.

Notice that the first instance of "crypto" lines up with "abcdef" and the second instance lines up with "cdefab". The two instances will encrypt to different ciphertexts and the Kasiski examination will reveal nothing.

abcdeabcdeabcdeabcde**abcdea**bcdeabc
crypto is short for **crypto**graphy.

Note that both occurrences of "crypto" now line up with "abcdea". The two instances will encrypt to the same ciphertext and the Kasiski examination will be effective.

A string based attack

The difficulty of using the Kasiski examination lies in finding repeated strings. This is a very hard task to perform manually, but computers can make it much easier. However, care is still required, since some repeated strings may just be coincidence, so that some of the repeat distances are misleading. The cryptanalyst has to rule out the coincidences to find the correct length. Then, of course, the monoalphabetic ciphertexts that result must be cryptanalyzed.

1. A cryptanalyst looks for repeated groups of letters and counts the number of letters between the beginning of each repeated group. For instance if the ciphertext was **FGX**THJAQWN**FGX**Q, the distance between FGX's is 10. The analyst records the distances for all repeated groups in the text.
2. The analyst next factors each of these numbers. If any number is repeated in the majority of these factorings, it is likely to be the length of the keyword. This is because repeated groups are more likely to occur when the same letters are encrypted using the same key letters

than by mere coincidence; this is especially true for long matching strings. The key letters are repeated at multiples of the key length, so most of the distances found in step 1 are likely to be multiples of the key length. A common factor is usually evident.

3. Once the keyword length is known, the following observation of Babbage and Kasiski comes into play. If the keyword is *N* letters long, then every *N*th letter must have been enciphered using the same letter of the keytext. Grouping every*N*th letter together, the analyst has *N* "messages", each encrypted using a one-alphabet substitution, and each piece can then be attacked using frequency analysis.
4. Using the solved message, the analyst can quickly determine what the keyword was. Or, in the process of solving the pieces, the analyst might use guesses about the keyword to assist in breaking the message.
5. Once the interceptor knows the keyword, that knowledge can be used to read other messages that use the same key.

Superposition

Kasiski actually used "superimposition" to solve the Vigenère cipher. He started by finding the key length, as above. Then he took multiple copies of the message and laid them one-above-another, each one shifted left by the length of the key. Kasiski then observed that each *column* was made up of letters encrypted with a single alphabet. His method was equivalent to the one described above, but is perhaps easier to picture.

Modern attacks on polyalphabetic ciphers are essentially identical to that described above, with the one improvement of coincidence counting. Instead of looking for repeating groups, a modern analyst would take two copies of the message and lay one above another.

Modern analysts use computers, but this description illustrates the principle that the computer algorithms implement.

The generalized method

1. The analyst shifts the bottom message one letter to the left, then two letters to the left, etc., each time going through the entire message and counting the number of times the same letter appears in the top and bottom message.
2. The number of "coincidences" goes up sharply when the bottom message is shifted by a multiple of the key length, because then the adjacent letters are in the same language using the same alphabet.
3. Having found the key length, cryptanalysis proceeds as described above using frequency analysis.

Quadgram Statistics as a Fitness Measure importance: 3/5. Length: 2/5

When trying to break ciphers, it is often useful to try deciphering with many different keys, then look at the deciphered text. If the text looks very similar to English, we consider the key to be a good one.

What we need is a way of determining if a piece of text is very similar to English. This is achieved by counting 'quadgrams' (also known as 'tetragraphs'), or groups of 4 letters e.g. the quadgrams in the text ATTACK are: ATTA, TTAC, and TACK.
Note that single letter frequencies, bigrams and trigrams can also be used for this purpose.
My experience shows that quadgram frequencies work slightly better than trigrams, trigrams work slightly better than bigrams etc. but that going higher than 4 letters does not really add any benefit.

To use quadgrams to determine how similar text is to English, we first need to know which quadgrams occur in English. To do this we take a large piece of text, for example several books worth of text, and count each of the quadgrams that occur in them. We then divide these counts by the total number of quadgrams encountered to find the probability of each.

If we count the quadgrams in Tolstoy's "War and Peace", we have roughly 2,500,000 total quadgrams after all spaces and punctuation are removed. The probability of a specific quadgram is calculated by dividing the count of that quadgram by the total number of quadgrams in our training corpus. The log probability is simply the logarithm of this number. A few specific counts:

Quadgram	Count	Log Probability
AAAA	1	-6.40018764963
QKPC	0	-9.40018764963
YOUR	1132	-3.34634122278
TION	4694	-2.72864456437
ATTA	359	-3.84509320105

The table above shows that some quadgrams occur much more often than other quadgrams, this can be used to determine how similar to English a specific piece of text is. Note that QKPC appears zero times in our training corpus, yet the log probability is not -infinity, this is because we floor all probabilities.
If the text contains QPKC, it is probably not English, but if it contains TION, it is much more likely to be English.
To compute the probability of a piece of text being English, we must extract all the quadgrams, then multiply each of the quadgram probabilities.

For the text ATTACK, the quadgrams are ATTA, TTAC, and TACK. The total probability is

$$p(\mathrm{ATTACK}) = p(\mathrm{ATTA}) \times p(\mathrm{TTAC}) \times p(\mathrm{TACK})$$

where e.g.

$$p(\mathrm{ATTA}) = \frac{\mathrm{count}(\mathrm{ATTA})}{N}$$

In the equation above, count() is the number of times the particular quadgram occured, N is the total number of quadgrams in the training sample.

When multiplying many small probabilities, numerical underflow can occur in floating point numbers. For this reason the logarithm is taken of each probability.

The well known identity `log(a*b) = log(a)+log(b)` is used, so the final log probability is

$$\log\left(p(\text{ATTACK})\right) = \log\left(p(\text{ATTA})\right) + \log\left(p(\text{TTAC})\right) + \log\left(p(\text{TACK})\right)$$

This log probability is used as the 'fitness' of a piece of text, a higher number means it is more likely to be English, while a lower number means it is less likely to be English.

An Example

In this example we will use the words 'fitness' and 'log probability' interchangeably. This example assumes we have already gathered the log probabilities of all the quadgrams that occur in English text. We now find the likelihood that a new piece of text comes from the same distribution as English text does.

The following text:

```
ATTACK THE EAST WALL OF THE CASTLE AT DAWN
```

has a fitness of -129.24. This was achieved by extracting each quadgram, finding the log likelihood of this quadgram in English text (which we calculated earlier), then adding each of the numbers obtained in this way. If we use the Caesar cipher with a key of 5 to encipher this text to

```
FYYFHP YMJ JFXY BFQQ TK YMJ HFXYQJ FY IFBS
```

and re-calculate fitness we get -288.10. It is quite clear that the original text has a much higher fitness, which is exactly what we want from a fitness measure. The reason the fitness measure is higher for the first piece of text is that quadgrams like 'EAST' and 'OFTH' are quite common in English, so they contribute a higher score to fitness. The second piece of text contains quadgrams like 'FYYF' and 'BFQQ' which are very rare, these quadgrams contribute a much lower score to the fitness. Since the total fitness is the sum of all these contributions, English like text scores much higher that garbled text.

Unicity Distance importance: 1/5. Length: 1/5

Introduction

The Unicity Distance is a property of a certain cipher algorithm. It answers the question 'if we performed a brute force attack, how much ciphertext would we need to be sure our solution was the true solution?'. The answer depends on the redundancy of English. A short example should hopefully illuminate the problem:

Say for example we are given a message to decipher: FJKFPO, and we know it is enciphered with a substitution cipher. Can we decipher it? The answer is 'not really'. We can find many english words that fit the pattern, sure, but we can never know which was the actual original plaintext. For example, all of the following english fragments are legitimate decryptions for the ciphertext above, and we have no idea which is correct:

thatis

ofyour

season

onyour

thatwe

thetop

thetwo

oxford

thatin

thatof

and many more. The longer the ciphertext, the fewer possible decryptions there are. We wish to know, how long does a piece of ciphertext need to be, before it has only one possible decryption? This minimum length is given by the unicity distance.

Redundancy

The substitution cipher has a key that consists of 26 letters. The total number of keys is the number of ways we can jumble 26 letters, or 26! (factorial), this is very large number of possible keys. The amount of information this carries, measured in bits, is given by the logarithm to base 2:

$$\log_2(26!) = 88.28$$

or around 88 bits. The amount of information (per letter) carried by an alphabet of 26 letters is $\log_2(26) = 4.7$ bits. The actual amount of information carried by english has been reported to be around 1.5 bits per character. This means the redundancy of english is around 4.7 - 1.5 = 3.2 bits per character.

Unicity Distance

The unicity distance is the ratio of the number of bits required to express the key divided by the redundancy of english in bits per character. In the case of the substitution cipher above, this is 88.28/3.2 = 27.6. This means 28 characters are required, minimum, to be sure a particular decryption is unique. Different ciphers have different unicity distances, a higher unicity distance generally indicates a more secure cipher.

The unicity distance calculation is based on all keys being equally probably. If the key for a cipher is an english word, this limits greatly the number of possible keys. As a result the Unicity distance will be lower. The unicity distance calculated for several common ciphers is shown below:

Cipher	No. of keys	Unicity dist.	Details
Atbash cipher	1	0 characters	
Caesar cipher	25	2 characters	
Affine cipher	311	3 characters	
Simple Substitution cipher	26!	28 characters	
Vigenere cipher	26^N	N*1.47 characters	*N is the key length, e.g. N=10 means 15 characters are required.*
Autokey cipher	26^N	N*1.47 characters	*Same as Vigenere.*
Porta cipher	13^N	N*1.156 characters	*N is the key length, e.g. N=10 means 12 characters are required.*
Playfair cipher	25!	27 characters	
Foursquare cipher	(25!)^2	53 characters	

Note that just because you have 30 characters of ciphertext for a substitution cipher, this does not mean you will be able to break it. The Unicity distance is a theoretical minimum. In general you may need 4 times more characters (100 or more) to reliably break substitution ciphers. The same problem exists with the other ciphers.

One Time Pads

It was noted by the discoverer of the unicity distance that in a system with an infinite length random key, the unicity distance is infinite, and that for any alphabetic substitution cipher with a random key of length greater than or equal to the length of the message, plaintext cannot be derived from ciphertext alone. This type of cipher is called a one-time-pad, because of the use of pads of paper to implement it in WW2 and before.

The Vigenere cipher with a key chosen to be the same length as the plaintext is an example of a one-time pad (only if the key is chosen completely randomly!). For any ciphertext we get, every single decryption is just as valid as any other.

Proper Use of Cribs importance: 5/5. Length: 1/5

A "crib" is a sample of known plaintext that can help you decrypt a message.

When using a crib on a Vigenere, you look for letters that will repeat because of the period. For example, say your Vigenere had a period of 8. A good crib for that period would be:

"one ring to rule them all"

```
12345678  <-Vigenere period is 8
oneringt  <-this is our crib
orulethe
mall
```

Now notice the pattern. The "O"s in position 1 and 9 will both be encrypted by the same Vigenere key letter. That means they will BOTH encipher to the same letter. Also, the "L"s in positions 12 and 20 will both be enciphered by the same vigenere key letter and will be identical in the crypt text. Please note that it doesn't matter where this sequence starts within the Vigenere key, as long as the period is 8, the two "O"s will be encrypted with the same key letter, as will the two "L"s. It's the period, not the position within the key that matters.

So, we have a very distinct pattern to look for in our crypt text.

```
oneringtorulethemall
123456789*123456789*
A???????A??B???????B  (A could equal B here, that's not important)
```

If we find identical letters at this displacement anywhere in our crypt text, there is a very good chance we have located our crib.

Now playfair cribs are a little bit more complicated. Since a playfair divides everything into digram pairs, you have two possible configurations for each crib. Say our crib was: I LIKE WHITE KITTENS
It could be segmented as:

```
il ik ew hi te ki tx te ns   <-in playfair we seperate double letters with an x
or
-i li ke wh it ek it te ns
```

Now either way, we have VERY nice playfair patterns. In the first version we have repitition of the digram TE. BUT we also have the reverse pair IK, KI! Remember that in a playfair, any digraph and its reverse (eg AB and BA) will decrypt to the same letter pattern in the plaintext (eg RE and ER).

So in our crypt we would be looking for something like:

```
?? AB ?? ?? CD BA ?? CD ??
```

In the second version we have an IT repetition, and a KE, EK reversal. If thats the way it happened to line up we look for:

```
?? ?? AB ?? CD BA CD ?? ??
```

If "I LIKE WHITE KITTENS" is in our plaintext, we will find one of these two patterns in the crypt text.

Cryptanalysis of the Caesar/Affine Cipher importance: 5/5. Length: 1/5

The affine cipher is very slightly more complicated than the Caesar cipher, but does not offer much more security. The number of possible keys is 12*26-1 = 311. This is very easy for a computer to simply search all possible keys and pick the best. To find the best key, we must try deciphering the ciphertext with each key, then determine the 'fitness' of each piece of deciphered text. How do we determine how 'fit' a piece of text is? We calculate the statistics of the deciphered text, and compare these statistics to those calculated from standard english text. A fitness function will give us a number, the higher the number the more likely the particular key is the correct one. For this example we will be using quadgram statistics, but others are possible e.g. bigram or trigram statistics. These methods work by first determining the statistics of english text, then calculating the likelyhood that the ciphertext comes from the same distribution. An incorrectly deciphered (i.e. using the wrong key) message will probably contain sequences e.g. 'QKPC' which are very rare in normal english. In this way we can rank different decryption keys, the decryption key we want is the one that produces deciphered text with the fewest rare sequences.

The affine cipher has 2 key numbers, 'a' and 'b'. 'b' can range from 0 to 25, and 'a' can have any of the values 1,3,5,7,9,11,15,17,19,21,23,25. We iterate over each of these possible combinations, of which there are 311, determine the fitness of each combination, then chose the best. This is called a **'brute force'** method.

An Example - Our ciphertext is the following:

QUVNLAUVILZKVZZZVNHIVQUFSFZHWZQLQHQLJSNLAUVIFZLQ

LZYHKSVIFWKVQJFKKJMQUVFQQFNTZQUFQPJITFDFLSZQZHWZ

QLQHQLJSNLAUVIZLSFEELQLJSQJJQUVIFQQFNTZ

We start decrypting with all possible keys (only a few are shown in the table):

a	b	plaintext	fitness
(1,	1)	PTUMKZTUHKYJUYYYUMGH...	-1144.91
(3,	5)	VFOUCHFOBCYTOYYYOUSB...	-1112.00
(11,	18)	OMFJXWMFSXDEFDDDFJZS...	-1141.52
(17,	5)	THECIPHERISLESSSECUR...	**-593.58**
(21,	1)	XRWIYVRWJYQTWQQQWIEJ...	-1194.72
(25,	9)	TPOWYJPOBYKZOKKKOWCB...	-1117.56

As a result we find that the key corresponding to a=17, b=5 has the highest fitness, and indeed it is quite readable:

THECIPHERISLESSSECURETHANASUBSTITUTIONCIPHERASIT

ISVULNERABLETOALLOFTHEATTACKSTHATWORKAGAINSTSUBS

TITUTIONCIPHERSINADDITIONTOOTHERATTACKS

Cryptanalysis of the Autokey Cipher importance: 5/5. Length: 1/5

In Cryptanalysis of the Vigenere Cipher, it was possible to use the Index of Coincidence to identify the period of the cipher, unfortunately this method no longer works for the Autokey cipher.
This page deals with automated cracking of Autokey ciphers with no known crib. The only thing we know about the plaintext is that it is English. For cracking these ciphers by hand or with a crib, different techniques can be used.

For the approach described below, we need a way of determining how similar a piece of text is to English text. This is called rating the 'fitness' of the text. A piece of text very similar to English will get a high score (a high fitness), while a jumble of random characters will get a low score (a low fitness). For this we will use a fitness measure based on quadgram statistics. This method works by first determining the statistics of english text, then calculating the likelyhood that the ciphertext comes from the same distribution. An incorrectly deciphered (i.e. using the wrong key) message will probably contain sequences e.g. 'QKPC' which are very rare in normal English. In this way we can rank different decryption keys, the decryption key we want is the one that produces deciphered text with the fewest rare sequences.
To find the key length, we will use this technique starting with key length = 2, then try key length = 3 etc. until we get to 25 or so. This method is fast enough that we can search all key lengths in a fairly short time.

The Hill-climbing Algorithm

This example uses an assumed key length of 7. We start with an initial key, which could be chosen at random, or simply 7 'A's e.g. 'AAAAAAA'. We call this the 'parent' key. Try all possibilities of A-Z in the first key letter e.g. 'AAAAAAA', 'BAAAAAA', 'CAAAAAA',... These are called the 26 'child' keys. If any of the 26 child keys have a higher fitness than the parent, set the parent to the highest scoring child key. If 'FAAAAAA' turned out to be the best scoring child key, this becomes the parent. We now move to the second column and try all possibilities of 'FAAAAAA', 'FBAAAAA', 'FCAAAAA',... The parent is set to the best child once again. This procedure is repeated for all key letters. Once the 7th key letter is reached, start again at the first position and repeat the procedure. Once the algorithm goes through all key positions and the parent is not changed, we have found a local optimum and the algorithm stops.

To test the fitness of particular key, we try decrypting the message with that key, and calculate the fitness of the decrypted text using quadgram statistics. A modification that should be made is to only calculate fitness of the plaintext letters corresponding to the key letters we have searched. For example, if our current key is 'CIPHAAA', i.e. we have only searched 4 of the 7 key letters, (the original text is DEFENDTHEEASTWALLOFTHECASTLE)

current key: **CIPH**AAA**DEFE**RUL**HEEA**OCE**ALLO**JKZ

ciphertext: FMULRULKIJEFWPHPPOXMDENLGYEL
decrypted: **DEFE**RUL**HEEA**OCE**ALLO**JKZ**ECAS**PUM

we should only calculate fitness from the shaded parts of the decrypted text. This reduces the effect on the total fitness of the garbled text that is present due to unsearched components of the key.

Note that we do not search all possible keys - this would necessitate around 26^N trial keys to decipher, which for N > 5 starts to become a large number indeed. We search each position independent of the others; this cuts our number of trial keys to 26*N which is much more manageable. Note that sometimes the correct key will not be found, in which case you could try a different starting 'parent' and rerun the program. It may simply be that the ciphertext is too short, or contains too many rare quadgrams. This will mean garbled text may score higher than the original English. This is a limitation of any algorithm based on statistical properties of text, including single letter frequencies, bigrams, trigrams etc.

Cracking a Playfair importance: 3/5. Length: 5/5

First, a brief review of the playfair cipher:
write your keyword followed by the rest of the alphabet (leaving out j) into a 5x5 square. For example, with the keyword piano:

```
PIANO
BCDEF
GHKLM
QRSTU
VWXYZ
```

You can use other shapes as well, but to simplify matters we will assume a 5x5 square.

We encrypt by pairs, so break your plain text into pairs (digrams). No pair may be a double, so if there are any doubles, break them up with a null (usually x). Finally, pad out the last digram to two letters with a null if necessary. (also replace any "J"s with "I"s)

So our plain message: "little puppies" would become:

li tx tl ep up pi es <-note that we split up the tt with an x

We have 3 rules for encryption.
1: If the pair is in the same col, we replace each letter with the letter below it.
. TL encrypts to YT
2: If the pair is in the same row, we replace each letter with the letter to the right
. PI encrypts to IA
3: If the pair is diagonal, we replace each letter with the letter in the same row, but in the other letters col
. LI encrypts to HN

note that the square "wraps", so the letter to the right of M is G.

When decrypting, we reverse these rules, so if the encrypted pair are in the same col, use the letter above each, if they are in the same row, use the letter to the left, and if they are diagonal, use the letter in the same row but other letters col (note that the diag rule is symetrical, encryption and decryption are identical)

Now, for cracking the playfair some important things must be noticed. A normal frequency count will not help much, but a digram frequency count MIGHT. TH is the most common digram by far, but you have to have a pretty big hunk of text for that to help much.

Also, with a playfair, any digram pair that is reversed, will encrypt to the same but reversed crypt pair. For example: LI encrypted to HN, and IL encrypts to NH. So if you figure out any pair, you have also figured out it's reversal.

due to the nature of the playfair rules, no letter can encrypt to itself.

And one last important feature, because of the "wrapping" rule, a playfair square does not change the way it decrypts or encrypts when you shift it by rows or columns.

These two squares:

```
PIANO    TUQRS
BCDEF    YZVWX
GHKLM    NOPIA
QRSTU    EFBCD
VWXYZ    LMGHK
```

Are functionally identical, try it, in either one, PI encrypts to IA, PW encrypts to IV, and PB encrypts to BG.

Now, enough of the basics, it's time to attack the actual playfair challenge.

```
TM NX LR QG CR XE EW EG VK GS MH XM EV EK TV GV SU GZ KH IC NH NB TM SA VS KN

BH AN KT GI VO VA SF VA AR BV NI VE IV AV HX IQ NK IS EU LE BM HA LX VC BF ST

CN RX MI AS HV AS HB CI HY BM AR BU NX RU IS EU LE VA SF GZ KN HG GC RC IK BS

ES BP VA HU RE IR XE TY AB IU
```

The crib was "turkey eating title"

First we have to place the crib.

now there are two ways this crib might be broken up into pairs:

```
*t ur ke ye at in gt it le    <-no interesting patterns
or
```

```
tu rk ey ea ti ng ti tl ex    <-a very interesting pattern!
.           ^^    ^^
```

A reversal would have been interesting as well, ti and it for example. The first ordering wouldn't be much help to us, so we will assume that he has picked a crib that broke up in a way that was interesting, the second ordering. All we have to do now is find out where in our cryptogram this crib (broken up the interesting way) could possibly fit. Turns out there are two places that have the correct pattern:

```
AN KT GI VO VA SF VA AR BV
tu rk ey ea ti ng ti tl e
.           ^^    ^^
and
ST CN RX MI AS HV AS HB CI
tu rk ey ea ti ng ti tl e
.           ^^    ^^
```

If you try the first one (AN KT...) you rapidly run into problems with the decryption, so we will proceed working on the second location (ST CN...)

Now we have some known digram plain-encrypt pairs. It's time to start work on rebuilding our playfair square.

Some definitions:
"same row" or "same col" means in the same row or col, but exact relationship to the other letters is unknown.
"above" or "below" means the letters are touching.
"left of " or "right of" means the letters are touching,
data in parenthesis is conditonal
. N: (NG=HV row=left of H, diag=same row H same col V, col=over H)
Means for the plain=crypt pair NG=HV:
. if N and G are same row, then N is left of H
. if N and G are diagonal, then N is same row as H, same col as V
. if N and G are same col, then N is over H

plain=crypt
TU=ST
RK=CN
EY=RX
EA=MI
TI=AS
NG=HV
TI=AS
TL=HB
E*=CI

now, TU=ST implies that UTS are in the same row or column, and in exactly those positions relative to each other because you can't GET the same letter in a crypt digram and plain from the diagonal rules. If they are in the same row, we have problems later on, (go ahead, try it, you'll see!) so we go with same col.

u
t
s

TI=AS can not be in the same col because we've already proven that same col for T will encrypt T=S
TI=AS can not be in the same row since S is in the same col as T.
so they must be diags, so, we now know:
A: same row as T, same col as I
I: same row as S, same col as A

```
U
T   A
S   I
```

Note that we do NOT know the relationship between these two cols yet, they might be right next to each other, or several cols apart. I'm going to chart them with an empty col in between for now, but they may be right next to each other.

Lets consider EA=MI. Since I is in the same column as A, E and M must also be in that column, and M must be under E. they could fit in as either of these possibliities. (ignore the periods, they are just use to make everything line up correctly)

```
.       E
. E     M
U M   U
T A   T A
S I   S I
```

Remember that the playfair square is equivalent when shifted by cols or rows, so

```
. E         U
. M         T A
U     and   S I
T A           E
S I           M
```

are the SAME.

EY=RX is promising because we know that E and Y (and R and X) can't be in the same column, because EA=MI proved that E=M in the same column. Which means E and R must be in the same row. We don't know WHERE in that row R sits because we have nothing to locate Y yet, except that its not in the same column as E. could be in same row.
So we have either:

```
ER YX  OR  E R
.          X Y
```

which gives us:
. R: same row as E, (EY=RX row=right of E, diag=same col as Y)
. Y: same row as X, (EY=RX row=left of X, diag=same col as R)
. X: same row as Y. (EY=RX row=right of Y, diag=same col as E)
. E: same row as R. (EY=RX row=eft of R, diag=same col as X)

Remember that I am using parenthesis to specify conditional properties of letters.
this:
. R: Same row as E, (EY=RX row=right of E, diag=same col as Y)
means that I KNOW that R is in the same row as E. And if E and Y are in the same row, then R is right of E (ER), but if E and Y are diagonal, then R is in the same col as Y.

Next we look at RK=CN and for C we have E*=CI and we already know R is in the same row as E.

Now remember that from the playfair rules, same row and diag rules BOTH encrypt a letter to another letter in the same row. Only the same col rule can encrypt to a letter in a different row, and then only to the letter directly below the letter we are encrypting.

Now since R and E can BOTH encrypt to C, C has to be in the same row as both OR, either:
R is above C and E is in the same row as C
or E is above C and R in the same row as C.

Since I already know that R is in the same row as E, neither of the last two arrangements are possible, SO:
C: same row as E and R

Since R and C are in the same row, RK=CN can be same row, or diag, but not same col. So the possible relationships are:

```
RC KN  OR  R C
.          N K
```

so we now know:
. K: same row as N, (RK=CN row=left of N, diag=same col as C)
. N: Same row as K, (RK=CN row=right of K, diag=same col as R)
. C: same row as ER, (RK=CN row=right of R diag=same col as K)
. R: same row as C, (RK=CN row=left of C, diag=same col as N)

what about: TL=HB, it can't be same col since T did not encrypt to S. MIGHT be same row, might be diag. So the possible relationships are:

```
TH LB  OR  T H
.          B L
```

which tells us:
. H: Same row as TA (TL=HB row=right of T, diag=same col as L)
. L: Same row as B, (TL=HB row=left of B, diag=same col as H)
. B: Same row as L, (TL=HB row=right of L, diag=same col as T)
. T: same row as HA (TL=HB row=left of H, diag=same col as B)

Now that we know something about N and H, lets look at NG=HV

N must be same row as H or over H, G is either same row as V or over V. thats not much help, YET, but lets write it down anyway.

```
NH GV  OR  N H  OR  N
.          V G      H
.
.                   G
.                   V
```

. N: (NG=HV row=left of H, diag=same row H same col V, col=over H)
. G: (NG=HV row=left of V, diag=same row V same col H, col=over V)
. V: (NG=HV row=right of G, diag=same row G same col N, col=under G)

. H: (NG=HV row=right of N, diag=same row N same col G, col=under N)

So, lets look at what we have now:

```
. 12345        12345
1:           1:  E
2:  E        2:  M
3:U M    OR  3:U
4:T A        4:T A
5:S I        5:S I
```

. C: same row as ER, (RK=CN row=right of R diag=same col as K)
. E: same row as RC. (EY=RX row=eft of R, diag=same col as X)
. R: Same row as EC, (EY=RX row=right of E, diag=same col as Y)
. R: (RK=CN row=left of C, diag=same col as N)
. H: Same row as TA (TL=HB row=right of T, diag=same col as L)
. H: (NG=HV row=right of N, diag=same row N same col G, col=under N)
. T: same row as HA (TL=HB row=left of H, diag=same col as B)
. K: same row as N, (RK=CN row=left of N, diag=same col as C)
. N: Same row as K, (RK=CN row=right of K, diag=same col as R)
. N: (NG=HV row=left of H, diag=same row H same col V, col=over H)
. L: Same row as B, (TL=HB row=left of B, diag=same col as H)
. B: Same row as L, (TL=HB row=right of L, diag=same col as T)
. X: same row as Y. (EY=RX row=right of Y, diag=same col as E)
. Y: same row as X, (EY=RX row=left of X, diag=same col as R)
. V: (NG=HV row=right of G, diag=same row G same col N, col=under G)
. G: (NG=HV row=left of V, diag=same row V same col H, col=over V)

And we do not yet know the distance between the UTS and EMAI cols.

I really need to pin down the EM. so time to make some guesses and go the the real message. The sequance TM SA occurs in our message and should be helpful here:

if we use the arrangement:

```
. E
U M
T A
S I
```

then the sequance: TM SA decrypts to AUIT. There ARE a few words that end in AU, but they are rare and there are no words that begin with UI, and there is no occurance of AUI in the dictionary. so AUIT just seems very unlikely, so lets fix E and M's position:

```
. 12345
1:  E     cr
2:  M
3:U
4:T A       h
5:S I
```

Again, We still don't know the distance between these two collumns.
Note the cr and h, off to the right. They indicate that I know those letters are in those rows, but I don't know exactly WHERE in those rows. I put the H further over to clarify that I don't know if it is in the same col as c or r
now then, at the end of the crypt, we have AB IU, and we know from the above rules that IU decrypts to S?
Lets make some guesses at B and see where it takes us. We know:
. B: Same row as L, (TL=HB row=right of L, diag=same col as T)
Lets guess that T and L are in the same row. IF L is in the same row as T then B is right of L.

```
. 12345
1:  E     cr
2:  M
3:U
4:T A       h lb  <-l and b added on speculation that TL in same row
5:S I
```

with this speculative positioning, AB decypts to *L where the *=HT or B, so AB IU decrypts to HLS? or TLS? or BLS?

we need to find our possible word divisions, remembering that the ? on the end COULD be a null X.

The only two letter word starting with S is SO. But that leaves us with a word that ends in HL TL or BL, and somehow I just don't think AXOLOTL is likely to be in this message. So our word division would have to be further back, But there are no words LS?. AND there are no occurances of HLS, TLS or BLS in the dictionary at all. This means there is NO WAY we can make TL=same row work, we can now fix that TL=diag which gives us some good info:
. H: Same row as TA, same col as L
. H: (NG=HV row=right of N, diag=same row N same col G, col=under N)
. L: Same row as B, same col as H
. B: Same row as L, same col as T

good news, because since we now know that B is in the same col as T, we have it locked into two places on our square:

```
. 12345
1:b E    cr   <-B must be here
2:b M         <-or B must be here
3:U
4:T A      h
5:S I
```

has to be in one of those, so lets go back to AB IU. if we put b in the same row as M, we get AB decrypts to TM If we put B in the same row as E we get AB decrypts to TE, and that gives us:
AB IU decypts to TMS? or TES? (the ? may be a null X)

no word starts with MS except ms. No word ends in TM, and no TMS exist anywhere in the dictionary. So that combination is impossible. We must have TES? We know where B is! and this helps with L too.

```
. 12345
1:B E    lcr
2:  M
3:U
4:T A    h
5:S I
```

note we now know H is same col as L, so it has been placed in the same col. this is important because even without knowing it's exact position within the row, the diagonal rule can allow us to decrypt pairs if we know they are in the same col as other letters. For example, with the above square, we do not know the exact positions of H and L. But we know what row they are in, and we know they are in the same col. SO, that means we KNOW that the crypt pair BH is diagonal, and that it will decrypt to LT.

Lets finish lookin at TES? and see if we can figure out what that last letter is. There are no 3 letter words ES? so we have:
*TE SO
TEST
*TES? could be COURTESY, DISCOURTESY or lots and lots of words that end in TEST. (contest, fattest, greatest, etc) Discourtesy won't fit. Courtesy requires that UR=TY which is impossible. AND, if we assume TEST we have ST=IU which is impossible.

that puts us back to *TE SO. it could be ANTICIPATE SO or STATE SO, but I'm just not realy happy with ending a sentence with so. It seems unlikely. I think we can safely guess that that lest letter is a null X. and that WORKS. now we have TESX as the end of the message, good! makes SENSE! SX=IU gives us a perfect diagonal. And we know that Y is in the same row as X, so we can now say that EY is diag, which gives us:
Y: same row as X, same col as R
R: Same row as EC, same col as NY

```
. 12345
1:B E    lcr
2:  M
3:U X      y  <- y is same row UX same col R
4:T A    h
5:S I
.          n  <- n is down here because I know it's col but not its row
```

note that row 1 is full, we don't know the exact order, but there is no room for any more letters on that row. (yes, I am assuming a 5x5 grid). One of our questions was is RK=CN same row or diag? well, RK CAN'T be same row, there is no room for K on the same row as R, it has to be diag, which gives us:
. C: same row as ER, same col as K
. K: same row as N, same col as C
. N: Same row as K, same col as R
. R: Same row as EC, same col as N

Very nice, so, what I know now is:

```
. 12345
1:B E    lcr
2:  M
3:U X      y
4:T A    h
5:S I
.         kn
```

. C: same row as ERLB, same col as K
. R: Same row as ECBL, same col as NY
. L: Same row as BECR, same col as H
. H: Same row as TA, same col as L
. H: (NG=HV row=right of N, diag=same row N same col G, col=under N)
. K: same row as N, same col as C
. N: Same row as K, same col as R
. N: (NG=HV row=left of H, diag=same row H same col V, col=over H)
. Y: same row as X, same col as R
. V: (NG=HV row=right of G, diag=same row G same col N, col=under G)

. G: (NG=HV row=left of V, diag=same row V same col H, col=over V)

And we STILL do not yet know the distance between the BUTS and EMXAI cols.

ok, now back to the cryptogram where using the above we find:

```
IR XE TY AB IU  <-cipher
?E MI ?U TE SX  <-plain
```

obviously NU=TY to give us MINUTES
. N: same row as TAH, same col as Y
. Y: same row as UX, same col as N
. (note that this also gives us the row for K!)

and another fruitful sequance:

```
AS HB CI HY BM AR BU NX RU  <-cipher
TI TL E  N? E? NE S? AY BY  <-plain
```

has just GOT to be WEDNESDAY, lets try it:

```
HY BM AR BU NX  <-cipher
Nw Ed NE Sd AY  <-plain
```

that gives us the position of D (between B and U)

and what can we do with NW=HY? H is diag from Y so:

```
N H
Y W
```

giving us:
. H : same col as LW
. Y : same row as UXW
. W : same row as UXY, same col as LWH

I think I can see the keyword now, but lets clean up our last question first. we now know enough to deal with our NG=HV question. It can't be same col because N and H are same row. It can't be same row because there isn't room for G on the TAHKN row. So it is diag. And that makes our "what I know" look like:

```
. 12345
1:B E    lcr
2:D M
3:U X    w y
4:T A    hkn
5:S I
.        g v
```

. C: same row as ERLB, same col as K
. R: Same row as ECBL, same col as NY
. L: Same row as BECR, same col as H
. H: Same row as TANK, same col as LWG
. K: same row as NTAH, same col as C
. N: Same row as KTAH, same col as RYV
. Y: same row as UXW, same col as NR
. W: same row as UXY, same col as LGH
. V: same row G same col NYR
. G: same row V same col LWH

We still don't know the relationship between BDUTS and EMXAI cols, but lets fix that right now. Note that the letters TAHKN are all in the same row, and SI are in the row beneath them? and G and V are in the same row but we aren't certain which yet. Lets just take the obvious guess that keyword is THANKSGIV. That puts G and V in the row with S and I, and makes col 2 LWHG and col 4 RYNV and col 5 CK. with this guess, our square looks like:

```
. 12345
1:BLERC
2:D M
3:UWXY
4:THANK
5:SGIV
```

looking good, now lets shift it so that the thanks is on top. This isn't necessary, but it's prettier. (playfair squares are functionally equivalent when shifted by rows or cols)

```
. 12345
1:THANK
```

```
2:SGIV
3:BLERC
4:D M
5:UWXY
```

hmmm, obviously the key is more complicated than just thanksgiving. Probably thankgiving gobbler? Lets try it:

```
 12345
1:THANK
2:SGIVO
3:BLERC
4:DFMPQ
5:UWXYZ
```

And THAT, I do believe, is our playfair square. Testing it on the encrypted text gives us the entire message correctly.

Cryptanalysis of the Columnar Transposition Cipher importance: 2/5. Length: 2/5

This page deals with automated cracking of columnar transposition ciphers with no known crib. The only thing we know about the plaintext is that it is English. For cracking these ciphers by hand or with a crib, different techniques can be used. The book by Helen Fouche Gains "Cryptanalysis - a study of ciphers and their solution" and the book by Sinkov "Elementary Cryptanalysis" both describe at great length how to break columnar transposition ciphers by hand.

The columnar transposition is a surprisingly secure cipher when long keys are used (key words around length 20), but much weaker if shorter keywords are used. In addition, if we know the keyword length most of our work is done. If we have a columnar transposition cipher, and we don't know the keyword length, there are several things we can try.

For the approaches described below, we need a way of determining how similar a piece of text is to english text. This is called rating the 'fitness' of the text. A piece of text very similar to english will get a high score (a high fitness), while a jumble of random characters will get a low score (a low fitness). For this we will use a fitness measure based on quadgram statistics. This method works by first determining the statistics of English text, then calculating the likelyhood that the ciphertext comes from the same distribution. An incorrectly deciphered (i.e. using the wrong key) message will probably contain sequences e.g. 'QKPC' which are very rare in normal english. In this way we can rank different decryption keys, the decryption key we want is the one that produces deciphered text with the fewest rare sequences.

Enumerate all Short Keywords

The first step in attacking a columnar transposition cipher is to try all possible short keywords. If we check all keywords up to a length of 9 or so, we don't have to wait very long. For every keyword permutation we score the deciphered text, then chose the text with the highest score as our best candidate. The number of possible rearrangements of a length N key is N! (N factorial). This number grows very quickly as N gets larger. The number of possible keys for various length keywords is shown below:

Key Length	No. of permutations	Examples
2	2	AB, BA
3	6	ABC, BAC, CBA, ...
4	24	ABCD, ABDC, ACBD, ...
5	120	ABCDE, ABCED, ...
6	720	ABCDEF, ABDCFE, ...
7	5,040	ABCDEFG, ABDCGEF, ...
8	40,320	ABCDEFGH, ...
9	362,880	ABCDEFGHI, ...
10	3,628,800	ABCDEFGHIJ, ...
11	39,916,800	ABCDEFGHIJK, ...
12	479,001,600	ABCDEFGHIJKL, ...

Trying to test all possible combinations of a length 6 keyword is easy, 720 trial decryptions can be done very quickly. However, we have little hope of trying to enumerate all possible length 12 keywords, as it would take far too long. This is why we stop at around length 9 keywords.

If this stage fails to decrypt the ciphertext, there are two possibilities: the actual key is longer than 9, or the key is shorter than 9 but the plaintext contains many strange quadgrams, which leads us to discount it as a possibility due to its low score. This can be overcome by ensuring the pre-generated quadgram statistics come from text as similar as possible to the ciphers we are breaking.

Dictionary Attacks

If the first step failed, we now move on to the second. The columnar transposition cipher is almost always keyed with a word or short phrase, so we may not need to test all possible transposition keys, we may only need to test common words. This involves having a large list of dictionary words including place names, famous people, mythological names, historical names etc. From this we generate a text file of possible keys. We only need consider words longer than length 9, since we tested all the shorter words in step 1. We then try to decrypt the ciphertext with all possible dictionary words and record the keyword that resulted in plaintext with the highest quadgram fitness. Having 1,000,000 dictionary words would be a good comprehensive target.

This step can work if the keyword was one of the words that you included in the dictionary, but it fails to work if the keyword is something like 'THEBLOOMSINSEPTEMBER'. We can't hope to include all possible short phrases in our dictionary, as there are simply far too many of them. If this step fails, we must move on to step 3.

Hill Climbing

The hill climbing approach we use here is almost identical to that used to break substitution ciphers. We first assume the key length is 10, then chose a random starting keyword of this length. This is called the parent key. Child keys are generated by making random swaps in the parent keyword, and if any of the swaps lead to an increased fitness we replace the parent with the child that beat it. In addition to randomly swapping two elements, we also try rotating the key e.g. 'ABCDEFG' -> 'BCDEFGA', or cutting the key at a random point and swapping the ends e.g. 'ABCDEFG' -> 'EFGABCD'. This allows us to better search the keyspace.

This algorithm will very rarely get the correct answer on the first run, typically you would have to run it several hundred times, each time starting with a different random key, to be sure to get the correct decryption. If after many tries the correct key is not found, it is time to increment the key length to 11 and rerun everything. This must be performed for key lengths from 10,11,...,20. Keys of length 20 are very difficult to crack, and it gets much more difficult to crack keys as they become longer than 20.

This sort of cracking can take all night to complete, but if you really have to crack something you just have to take the time to do it. For long keys, length 20 and up, a simulated annealing algorithm would probably be a better choice and may reduce the number of iterations you need to perform to crack a cipher. But this introduces more variables: the starting temperature, the temperature step, and the number of iterations per step. These variables would also have to change for each different key length, which increases the complexity of use.

Solving a Vigeneres in under one second importance: 2/5. Length: 2/5

I was playing around with putting some optimizations into my Elijah3/Quagmire 3 solver when I realized that a Vigenere is the same cipher except that the "key" is null so the ciphertext alphabet is the standard alphabet. It occurred to me that my program should, therefore, solve any Vigenere instantly, so I put it to the test and discovered that it does, in fact, solve any Vigenere in under one second!

Even if the beta key length (the number of different alphabets used) is not known it can solve any Vigenere in a few seconds by trying to solve it with every possible beta key length starting with 1. No need to do index of coincidence or the Kasiski method, just brute force the key lengths since it takes under 1 second to verify that a given length is or is not correct.

C++ Source Code:

When given a Vigenere it solves it before it even has a chance to do the (optional) dictionary attack or hill climber, so the only function used in solving Vigenere's is GetScore() which finds the best alphabet alignment in under a second. One other text file is needed:

Code for logdigraph.txt (take out all line breaks before saving):

```
NRSSLQROSMQUSVMRKUULJQRQNRMQMJJTIHIPMERKHRHAQOMRKJAQATKPKTKJTRGQQKJTLKQNSQIKANGSRPQTQPRTNLQQQSPKQSTQNQAPGURTUTSRRSOOTTVSSOVVUPSSRRKRNONRQMPSLJPOMTOHRPSQLOBNFRNWNSOOSRKIPOPRNIRPRPKOANEUNOMWNLOUKJNOOTNIQPRPKOAOGRPTSSRRMKIPSSVTPLSUUMRNMGPNIJIOJGILIDHIGPICJJJPEJADADLLKRMKNQIIMLPOKEKPOKHMAMATPPSUPNOTKNTPNSPJORRQDOASGTQMLT

MKNSJIMPMSRGOQPQINAPETPSUTQUQSNPQPQTPLOTVPQQJQJQRRRPUQPQMPSTVSRJUSTURSMOKSLKJSLIPQIHRNISQESPQPGLALAAABAABAACAAFADDDAABDQADAAATPRRVPQQTMQQRRTQJQTTQPQFRHTQRPUQOSTMPQRQTRMPTVSNREPHTQQPUPNWUMMQQQUPKSSTRLRBRKQPQPQNQLQHKSQSMQESSSIKLKKJQFFFTEDFRDDEGEPEAHIGJEGAKATLLMSLJSSIIMMPRLFOOOKIMAMDNINMIFKNFEHIGKPEIJOJFJAJARPPOQPNPQL

LOPORPIORRMLQDLINGGDOFFGMBFJGBKEAEGHHEGAJL
```

Note: The output goes to a file called "results.txt" which shows the steps along the way plus the final solution.

David's Tips & Summary importance: 9/5. Length: just read it.

If the IoC types are confusing you:

The thing known as 'Kappa' can be calculated two ways:

$\kappa = \frac{\sum_{i=1}^{n}[a_i = b_i]}{n}$ This is only when you are comparing two pieces of text (the ciphertext and the ciphertext shifted). You count the number of times that a character in the original ciphertext is the same as the character taking up that position in the shifted ciphertext. The kappa is the number of these coincidences over the total number of characters in the original cipher text. You can either give it in this fraction form or put it as a percentage. This kind of Kappa is what you get from the 'Superposition' Kasiski examination in other words.

$\kappa = \frac{\sum_{i=A}^{Z} f_i(f_i - 1)}{n(n-1)}$ The idea is mathematically the same as the kappa used for comparing two texts, but this one is only used on one piece of text (the ciphertext). Obviously you can't compare a text with itself because the text and its copy are identical (that would give a kappa of 100%). Of course, the idea for both of these kappas is to see how often the text repeats itself – to find the frequency at which characters repeat and then see if this matches with what we would expect if it was English. This is the 'main' kappa used. For random text, the expected Kappa is 1/26 (0.0385). If you get a kappa from a ciphertext of about 0.067 then the ciphertext looks like it's English so it's a cipher in which the frequencies don't change (like a monoalph substitution or a transposition).

Now, the term 'Index of Coincidence' strictly only applies to $26 * \kappa$ because it is defined as your kappa scaled by the kappa for random text. In other words, if your kappa is 0.2, to get the actual IoC you should divide it by 0.0385 (dividing by 1/26 = multiplying by 26 obviously). However, the term 'IoC' is often thrown about without any thought and so to make it clearer, the full name of the Index of Coincidence was made 'Delta Index of Coincidence', with the others being labelled as 'Kappa Indices of Coincidence'. What people do now is refer to the kappa obtained through the Kasiski Examination Superposition as the 'kappa' you get from 'kappa testing' and don't call the other kind (the one-text only version) of kappa 'kappa' at all anymore. Instead they just call it the IoC. This means that you may see an IoC given in either of two forms: 0.068 or 1.768 (the scaled version). You are most likely to see a Kappa in just this form: 2.67%.

Now for general cryptanalysis tips:

- Brute forcing is the best method. Seriously. Most of the people who are considered 'gods of cryptography' agree. This is why it is better to program your way to the plaintext rather than do it with pencil and paper only. If you can, this is the first method you should use after identifying the cipher type. Only if this fails or the number of possibilities is too large should you begin proper cryptanalysis (otherwise you're wasting time).

- First thing you should always do is look at the ciphertext. See only A, D, F, G, V and X characters? Duh, it's an ADFGVX cipher. These are difficult to brute force because of the superencipherment. My method is to first identify the transp period by brute forcing as if it is a normal columnar transp cipher. The key thing is that you should not use normal fitness-assessment methods. You instead should judge which transp period results in the closest **Bigram Frequency** distribution to English. Once the most probable period is identified, then you can put together a digraph chart and see which digraph pops up most frequently as this is probably the letter E. Then you can start drawing up your Polybius and the rest is trivial. If the cipher type isn't obvious from looking at it, instantly calculate the IoC. Flat distribution? Try all the polyalphabetic brute-forcers and if nothing works, break out the pencil and paper. The IoC is close to 1.7? First

check for Affine (and caeser at the same time). This should take under a second. Now brute force for the keyword-type ciphers. This also takes a very short amount of time. If this doesn't work, you should use a hill-climbing algorithm (don't use genetic, they are slower) and that will do it, or you can decide that it is actually a kind of transp cipher and brute force for those instead. This however doesn't work a lot of the time so you often need to result to pencil and paper methods (transps are strangely some of the most secure classical ciphers – much more so than any polyalphabetic one). In this case, look for anagrams of what you think might be in there and figure out the key from there. If brute forcing fails, this anagram attack is guaranteed to work. Just remember that with route transps, you need to really think of everything.

- Obviously and an ADFGVX isn't the only one you can identify by just looking at it. One of the cipher challenges from a previous year was a baconian. Instead of designating one half of the alphabet to represent 'a' and the other half 'b', they went ahead and used the raw 'a' and 'b's so that it was blatantly obvious that it was a baconian at first sight. Also, if you see morse code, think pollux and similar things. If you see patterns of say, 1, 2 and 3 slashes only separated by some character, think trinary for example. You get the picture. If you see just numbers though, you want to hope it's not a homophonic cipher (which you would use hill-climbing for) but try the simple stuff first. It could just be a numerical caeser or a numerical vigenere (called a gronsfeld cipher). Always remember to try the simple stuff first. If you see a mix of different characters and punctuation marks, its most likely an ASCII rot (like the amazing rot-47 useful for encoding offensive messages on my facebook profile or in my e-mail signatures) or a homophonic (think zodiac killer). If you see repeated strings of the same characters, look at the patterns of spaces. It's all about pattern matching. When I say pencil and paper methods, statistics and pattern-matching is mostly what I mean.