18

Crosswords and Codebreaking

In this chapter we make a random walk through a few topics related to language modelling.

▶ 18.1 Crosswords

The rules of crossword-making may be thought of as defining a constrained channel. The fact that *many* valid crosswords can be made demonstrates that this constrained channel has a capacity greater than zero.

There are two archetypal crossword formats. In a 'type A' (or American) crossword, every row and column consists of a succession of words of length 2 or more separated by one or more spaces. In a 'type B' (or British) crossword, each row and column consists of a mixture of words and single characters, separated by one or more spaces, and every character lies in at least one word (horizontal or vertical). Whereas in a type A crossword every letter lies in a horizontal word *and* a vertical word, in a typical type B crossword only about half of the letters do so; the other half lie in one word only.

Type A crosswords are harder to *create* than type B because of the constraint that no single characters are permitted. Type B crosswords are generally harder to *solve* because there are fewer constraints per character.

Figure 18.1. Crosswords of types A (American) and B (British).

Why are crosswords possible?

If a language has no redundancy, then any letters written on a grid form a valid crossword. In a language with high redundancy, on the other hand, it is hard to make crosswords (except perhaps a small number of trivial ones). The possibility of making crosswords in a language thus demonstrates a *bound on the redundancy* of that language. Crosswords are not normally written in genuine English. They are written in 'word-English', the language consisting of strings of words from a dictionary, separated by spaces.

▷ Exercise 18.1.[2] Estimate the capacity of word-English, in bits per character. [Hint: think of word-English as defining a constrained channel (Chapter 17) and see exercise 6.18 (p.125).]

The fact that many crosswords can be made leads to a lower bound on the entropy of word-English.

For simplicity, we now model word-English by Wenglish, the language introduced in section 4.1 which consists of W words all of length L. The entropy of such a language, per character, including inter-word spaces, is:

$$H_W \equiv \frac{\log_2 W}{L+1}. \tag{18.1}$$

We'll find that the conclusions we come to depend on the value of H_W and are not terribly sensitive to the value of L. Consider a large crossword of size S squares in area. Let the number of words be $f_w S$ and let the number of letter-occupied squares be $f_1 S$. For typical crosswords of types A and B made of words of length L, the two fractions f_w and f_1 have roughly the values in table 18.2.

	A	B
f_w	$\frac{2}{L+1}$	$\frac{1}{L+1}$
f_1	$\frac{L}{L+1}$	$\frac{3}{4}\frac{L}{L+1}$

Table 18.2. Factors f_w and f_1 by which the number of words and number of letter-squares respectively are smaller than the total number of squares.

We now estimate how many crosswords there are of size S using our simple model of Wenglish. We assume that Wenglish is created at random by generating W strings from a monogram (i.e., memoryless) source with entropy H_0. If, for example, the source used all $A = 26$ characters with equal probability then $H_0 = \log_2 A = 4.7$ bits. If instead we use Chapter 2's distribution then the entropy is 4.2. The redundancy of Wenglish stems from two sources: it tends to use some letters more than others; and there are only W words in the dictionary.

Let's now count how many crosswords there are by imagining filling in the squares of a crossword at random using the same distribution that produced the Wenglish dictionary and evaluating the probability that this random scribbling produces valid words in all rows and columns. The total number of *typical* fillings-in of the $f_1 S$ squares in the crossword that can be made is

$$|T| = 2^{f_1 S H_0}. \tag{18.2}$$

The probability that one word of length L is validly filled-in is

$$\beta = \frac{W}{2^{L H_0}}, \tag{18.3}$$

and the probability that the whole crossword, made of $f_w S$ words, is validly filled-in by a single typical in-filling is approximately

$$\beta^{f_w S}. \tag{18.4}$$

This calculation underestimates the number of valid Wenglish crosswords by counting only crosswords filled with 'typical' strings. If the monogram distribution is non-uniform then the true count is dominated by 'atypical' fillings-in, in which crossword-friendly words appear more often.

So the log of the number of valid crosswords of size S is estimated to be

$$\log \beta^{f_w S} |T| = S\left[(f_1 - f_w L) H_0 + f_w \log W\right] \tag{18.5}$$

$$= S\left[(f_1 - f_w L) H_0 + f_w (L+1) H_W\right], \tag{18.6}$$

which is an increasing function of S only if

$$(f_1 - f_w L) H_0 + f_w (L+1) H_W > 0. \tag{18.7}$$

So arbitrarily many crosswords can be made only if there's enough words in the Wenglish dictionary that

$$H_W > \frac{(f_w L - f_1)}{f_w (L+1)} H_0. \tag{18.8}$$

Plugging in the values of f_1 and f_w from table 18.2, we find the following.

Crossword type	A	B
Condition for crosswords	$H_W > \frac{1}{2}\frac{L}{L+1} H_0$	$H_W > \frac{1}{4}\frac{L}{L+1} H_0$

If we set $H_0 = 4.2$ bits and assume there are $W = 4000$ words in a normal English-speaker's dictionary, all with length $L = 5$, then we find that the condition for crosswords of type B is satisfied, but the condition for crosswords of type A is *only just* satisfied. This fits with my experience that crosswords of type A usually contain more obscure words.

Further reading

These observations about crosswords were first made by Shannon (1948); I learned about them from Wolf and Siegel (1998). The topic is closely related to the capacity of two-dimensional constrained channels. An example of a two-dimensional constrained channel is a two-dimensional bar-code, as seen on parcels.

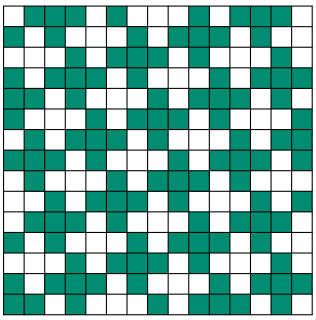

Figure 18.3. A binary pattern in which every pixel is adjacent to four black and four white pixels.

Exercise 18.2.[3] A two-dimensional channel is defined by the constraint that, of the eight neighbours of every interior pixel in an $N \times N$ rectangular grid, four must be black and four white. (The counts of black and white pixels around boundary pixels are not constrained.) A binary pattern satisfying this constraint is shown in figure 18.3. What is the capacity of this channel, in bits per pixel, for large N?

▶ 18.2 Simple language models

The Zipf–Mandelbrot distribution

The crudest model for a language is the monogram model, which asserts that each successive word is drawn independently from a distribution over words. What is the nature of this distribution over words?

Zipf's law (Zipf, 1949) asserts that the probability of the rth most probable word in a language is approximately

$$P(r) = \frac{\kappa}{r^{\alpha}}, \tag{18.9}$$

where the exponent α has a value close to 1, and κ is a constant. According to Zipf, a log–log plot of frequency versus word-rank should show a straight line with slope $-\alpha$.

Mandelbrot's (1982) modification of Zipf's law introduces a third parameter v, asserting that the probabilities are given by

$$P(r) = \frac{\kappa}{(r+v)^{\alpha}}. \tag{18.10}$$

For some documents, such as Jane Austen's *Emma*, the Zipf–Mandelbrot distribution fits well – figure 18.4.

Other documents give distributions that are not so well fitted by a Zipf–Mandelbrot distribution. Figure 18.5 shows a plot of frequency versus rank for the LaTeX source of this book. Qualitatively, the graph is similar to a straight line, but a curve is noticeable. To be fair, this source file is not written in pure English – it is a mix of English, maths symbols such as 'x', and LaTeX commands.

Figure 18.4. Fit of the Zipf–Mandelbrot distribution (18.10) (curve) to the empirical frequencies of words in Jane Austen's *Emma* (dots). The fitted parameters are $\kappa = 0.56$; $v = 8.0$; $\alpha = 1.26$.

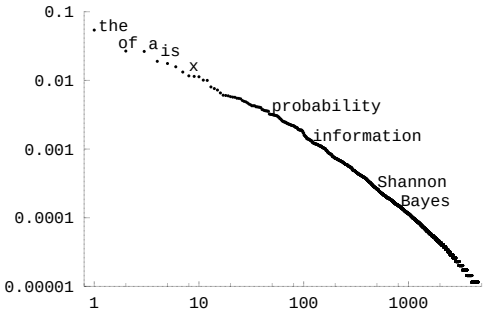

Figure 18.5. Log–log plot of frequency versus rank for the words in the LaTeX file of this book.

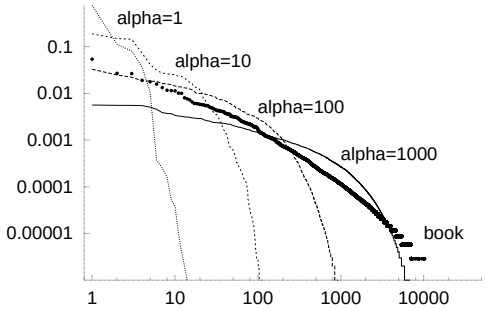

Figure 18.6. Zipf plots for four 'languages' randomly generated from Dirichlet processes with parameter α ranging from 1 to 1000. Also shown is the Zipf plot for this book.

The Dirichlet process

Assuming we are interested in monogram models for languages, what model should we use? One difficulty in modelling a language is the unboundedness of vocabulary. The greater the sample of language, the greater the number of words encountered. A generative model for a language should emulate this property. If asked 'what is the next word in a newly-discovered work of Shakespeare?' our probability distribution over words must surely include some non-zero probability for *words that Shakespeare never used before*. Our generative monogram model for language should also satisfy a consistency rule called *exchangeability*. If we imagine generating a new language from our generative model, producing an ever-growing corpus of text, all statistical properties of the text should be homogeneous: the probability of finding a particular word at a given location in the stream of text should be the same everywhere in the stream.

The Dirichlet process model is a model for a stream of symbols (which we think of as 'words') that satisfies the exchangeability rule and that allows the vocabulary of symbols to grow without limit. The model has one parameter α. As the stream of symbols is produced, we identify each new symbol by a unique integer w. When we have seen a stream of length F symbols, we define the probability of the next symbol in terms of the counts $\{F_w\}$ of the symbols seen so far thus: the probability that the next symbol is a new symbol, never seen before, is

$$\frac{\alpha}{F+\alpha}. \tag{18.11}$$

The probability that the next symbol is symbol w is

$$\frac{F_w}{F+\alpha}. \tag{18.12}$$

Figure 18.6 shows Zipf plots (i.e., plots of symbol frequency versus rank) for million-symbol 'documents' generated by Dirichlet process priors with values of α ranging from 1 to 1000.

It is evident that a Dirichlet process is not an adequate model for observed distributions that roughly obey Zipf's law.

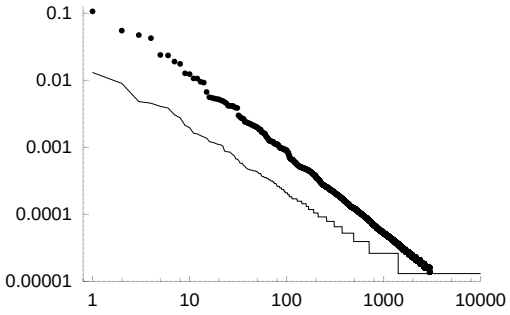

Figure 18.7. Zipf plots for the words of two 'languages' generated by creating successive characters from a Dirichlet process with $\alpha = 2$, and declaring one character to be the space character. The two curves result from two different choices of the space character.

With a small tweak, however, Dirichlet processes can produce rather nice Zipf plots. Imagine generating a language composed of elementary symbols using a Dirichlet process with a rather small value of the parameter α, so that the number of reasonably frequent symbols is about 27. If we then declare one of those symbols (now called 'characters' rather than words) to be a space character, then we can identify the strings between the space characters as 'words'. If we generate a language in this way then the frequencies of words often come out as very nice Zipf plots, as shown in figure 18.7. Which character is selected as the space character determines the slope of the Zipf plot – a less probable space character gives rise to a richer language with a shallower slope.

▶ 18.3 Units of information content

The information content of an outcome, x, whose probability is $P(x)$, is defined to be

$$h(x) = \log \frac{1}{P(x)}. \tag{18.13}$$

The entropy of an ensemble is an average information content,

$$H(X) = \sum_x P(x) \log \frac{1}{P(x)}. \tag{18.14}$$

When we compare hypotheses with each other in the light of data, it is often convenient to compare the log of the probability of the data under the alternative hypotheses,

$$\text{'log evidence for } \mathcal{H}_i\text{'} = \log P(D \,|\, \mathcal{H}_i), \tag{18.15}$$

or, in the case where just two hypotheses are being compared, we evaluate the 'log odds',

$$\log \frac{P(D \,|\, \mathcal{H}_1)}{P(D \,|\, \mathcal{H}_2)}, \tag{18.16}$$

which has also been called the 'weight of evidence in favour of $\mathcal{H}_1$'. The log evidence for a hypothesis, $\log P(D \,|\, \mathcal{H}_i)$ is the negative of the information content of the data D: if the data have large information content, given a hypothesis, then they are surprising to that hypothesis; if some other hypothesis is not so surprised by the data, then that hypothesis becomes more probable. 'Information content', 'surprise value', and log likelihood or log evidence are the same thing.

All these quantities are logarithms of probabilities, or weighted sums of logarithms of probabilities, so they can all be measured in the same units. The units depend on the choice of the base of the logarithm.

The names that have been given to these units are shown in table 18.8.

Unit	Expression that has those units
bit	$\log_2 p$
nat	$\log_e p$
ban	$\log_{10} p$
deciban (db)	$10 \log_{10} p$

Table 18.8. Units of measurement of information content.

The *bit* is the unit that we use most in this book. Because the word 'bit' has other meanings, a backup name for this unit is the *shannon*. A *byte* is 8 bits. A megabyte is $2^{20} \simeq 10^6$ bytes. If one works in natural logarithms, information contents and weights of evidence are measured in *nats*. The most interesting units are the *ban* and the *deciban*.

The history of the ban

Let me tell you why a factor of ten in probability is called a ban. When Alan Turing and the other codebreakers at Bletchley Park were breaking each new day's Enigma code, their task was a huge inference problem: to infer, given the day's cyphertext, which three wheels were in the Enigma machines that day; what their starting positions were; what further letter substitutions were in use on the steckerboard; and, not least, what the original German messages were. These inferences were conducted using Bayesian methods (of course!), and the chosen units were decibans or half-decibans, the deciban being judged the smallest weight of evidence discernible to a human. The evidence in favour of particular hypotheses was tallied using sheets of paper that were specially printed in Banbury, a town about 30 miles from Bletchley. The inference task was known as Banburismus, and the units in which Banburismus was played were called bans, after that town.

▶ 18.4 A taste of Banburismus

The details of the code-breaking methods of Bletchley Park were kept secret for a long time, but some aspects of Banburismus can be pieced together. I hope the following description of a small part of Banburismus is not too inaccurate.[1]

How much information was needed? The number of possible settings of the Enigma machine was about 8×10^{12}. To deduce the state of the machine, 'it was therefore necessary to find about 129 decibans from somewhere', as Good puts it. Banburismus was aimed not at deducing the entire state of the machine, but only at figuring out which wheels were in use; the logic-based bombes, fed with guesses of the plaintext (cribs), were then used to crack what the settings of the wheels were.

The Enigma machine, once its wheels and plugs were put in place, implemented a continually-changing permutation cypher that wandered deterministically through a state space of 26^3 permutations. Because an enormous number of messages were sent each day, there was a good chance that whatever state one machine was in when sending one character of a message, there would be another machine *in the same state* while sending a particular character in another message. Because the evolution of the machine's state was deterministic, the two machines would remain in the same state as each other

[1] I've been most helped by descriptions given by Tony Sale (`http://www.codesandciphers.org.uk/lectures/`) and by Jack Good (1979), who worked with Turing at Bletchley.

for the rest of the transmission. The resulting correlations between the outputs of such pairs of machines provided a dribble of information-content from which Turing and his co-workers extracted their daily 129 decibans.

How to detect that two messages came from machines with a common state sequence

The hypotheses are the null hypothesis, $\mathcal{H}_0$, which states that the machines are in *different* states, and that the two plain messages are unrelated; and the 'match' hypothesis, $\mathcal{H}_1$, which says that the machines are in the *same* state, and that the two plain messages are unrelated. No attempt is being made here to infer what the state of either machine is. The data provided are the two cyphertexts $\mathbf{x}$ and $\mathbf{y}$; let's assume they both have length T and that the alphabet size is A (26 in Enigma). What is the probability of the data, given the two hypotheses?

First, the null hypothesis. This hypothesis asserts that the two cyphertexts are given by

$$\mathbf{x} = x_1x_2x_3\ldots = c_1(u_1)c_2(u_2)c_3(u_3)\ldots \tag{18.17}$$

and

$$\mathbf{y} = y_1y_2y_3\ldots = c'_1(v_1)c'_2(v_2)c'_3(v_3)\ldots, \tag{18.18}$$

where the codes c_t and c'_t are two unrelated time-varying permutations of the alphabet, and $u_1u_2u_3\ldots$ and $v_1v_2v_3\ldots$ are the plaintext messages. An exact computation of the probability of the data $(\mathbf{x}, \mathbf{y})$ would depend on a language model of the plain text, and a model of the Enigma machine's guts, but if we assume that each Enigma machine is an *ideal* random time-varying permutation, then the probability distribution of the two cyphertexts is uniform. All cyphertexts are equally likely.

$$P(\mathbf{x}, \mathbf{y} \,|\, \mathcal{H}_0) = \left(\frac{1}{A}\right)^{2T} \quad \text{for all } \mathbf{x}, \mathbf{y} \text{ of length } T. \tag{18.19}$$

What about $\mathcal{H}_1$? This hypothesis asserts that a *single* time-varying permutation c_t underlies both

$$\mathbf{x} = x_1x_2x_3\ldots = c_1(u_1)c_2(u_2)c_3(u_3)\ldots \tag{18.20}$$

and

$$\mathbf{y} = y_1y_2y_3\ldots = c_1(v_1)c_2(v_2)c_3(v_3)\ldots\ . \tag{18.21}$$

What is the probability of the data $(\mathbf{x}, \mathbf{y})$? We have to make some assumptions about the plaintext language. If it were the case that the plaintext language was completely random, then the probability of $u_1u_2u_3\ldots$ and $v_1v_2v_3\ldots$ would be uniform, and so would that of $\mathbf{x}$ and $\mathbf{y}$, so the probability $P(\mathbf{x}, \mathbf{y} \,|\, \mathcal{H}_1)$ would be equal to $P(\mathbf{x}, \mathbf{y} \,|\, \mathcal{H}_0)$, and the two hypotheses $\mathcal{H}_0$ and $\mathcal{H}_1$ would be indistinguishable.

We make progress by assuming that the plaintext is not completely random. Both plaintexts are written in a language, and that language has redundancies. Assume for example that particular plaintext letters are used more often than others. So, even though the two plaintext messages are unrelated, they are slightly more likely to use the same letters as each other; if $\mathcal{H}_1$ is true, two synchronized letters from the two cyphertexts are slightly more likely to be identical. Similarly, if a language uses particular bigrams and trigrams frequently, then the two plaintext messages will occasionally contain the same bigrams and trigrams at the same time as each other, giving rise, if $\mathcal{H}_1$ is true,

```
      u  LITTLE-JACK-HORNER-SAT-IN-THE-CORNER-EATING-A-CHRISTMAS-PIE--HE-PUT-IN-H
      v  RIDE-A-COCK-HORSE-TO-BANBURY-CROSS-TO-SEE-A-FINE-LADY-UPON-A-WHITE-HORSE
matches:  .*....*..******.*.............*..........*...............*..........
```

Table 18.9. Two aligned pieces of English plaintext, **u** and **v**, with matches marked by ∗. Notice that there are twelve matches, including a run of six, whereas the expected number of matches in two completely random strings of length $T = 74$ would be about 3. The two corresponding cyphertexts from two machines in identical states would also have twelve matches.

to a little burst of 2 or 3 identical letters. Table 18.9 shows such a coincidence in two plaintext messages that are unrelated, except that they are both written in English.

The codebreakers hunted among pairs of messages for pairs that were suspiciously similar to each other, counting up the numbers of matching monograms, bigrams, trigrams, etc. This method was first used by the Polish codebreaker Rejewski.

Let's look at the simple case of a monogram language model and estimate how long a message is needed to be able to decide whether two machines are in the same state. I'll assume the source language is monogram-English, the language in which successive letters are drawn i.i.d. from the probability distribution $\{p_i\}$ of figure 2.1. The probability of $\mathbf{x}$ and $\mathbf{y}$ is nonuniform: consider two single characters, $x_t = c_t(u_t)$ and $y_t = c_t(v_t)$; the probability that they are identical is

$$\sum_{u_t, v_t} P(u_t) P(v_t) \, \mathbb{1}[u_t = v_t] \;=\; \sum_i p_i^2 \;\equiv\; m. \tag{18.22}$$

We give this quantity the name m, for 'match probability'; for both English and German, m is about 2/26 rather than 1/26 (the value that would hold for a completely random language). Assuming that c_t is an ideal random permutation, the probability of x_t and y_t is, by symmetry,

$$P(x_t, y_t \,|\, \mathcal{H}_1) \;=\; \begin{cases} \frac{m}{A} & \text{if } x_t = y_t \\ \frac{(1-m)}{A(A-1)} & \text{for } x_t \neq y_t. \end{cases} \tag{18.23}$$

Given a pair of cyphertexts $\mathbf{x}$ and $\mathbf{y}$ of length T that match in M places and do not match in N places, the log evidence in favour of $\mathcal{H}_1$ is then

$$\log \frac{P(\mathbf{x}, \mathbf{y} \,|\, \mathcal{H}_1)}{P(\mathbf{x}, \mathbf{y} \,|\, \mathcal{H}_0)} \;=\; M \log \frac{m/A}{1/A^2} + N \log \frac{\frac{(1-m)}{A(A-1)}}{1/A^2} \tag{18.24}$$

$$\;=\; M \log mA + N \log \frac{(1-m)A}{A-1}. \tag{18.25}$$

Every match contributes $\log mA$ in favour of $\mathcal{H}_1$; every non-match contributes $\log \frac{A-1}{(1-m)A}$ in favour of $\mathcal{H}_0$.

Match probability for monogram-English	m	0.076
Coincidental match probability	$1/A$	0.037
log-evidence for $\mathcal{H}_1$ per match	$10 \log_{10} mA$	3.1 db
log-evidence for $\mathcal{H}_1$ per non-match	$10 \log_{10} \frac{(1-m)A}{(A-1)}$	−0.18 db

If there were $M = 4$ matches and $N = 47$ non-matches in a pair of length $T = 51$, for example, the weight of evidence in favour of $\mathcal{H}_1$ would be +4 decibans, or a likelihood ratio of 2.5 to 1 in favour.

The *expected* weight of evidence from a line of text of length $T = 20$ characters is the expectation of (18.25), which depends on whether $\mathcal{H}_1$ or $\mathcal{H}_0$ is true. If $\mathcal{H}_1$ is true then matches are expected to turn up at rate m, and the expected weight of evidence is 1.4 decibans per 20 characters. If $\mathcal{H}_0$ is true

then spurious matches are expected to turn up at rate $1/A$, and the expected weight of evidence is -1.1 decibans per 20 characters. Typically, roughly 400 characters need to be inspected in order to have a weight of evidence greater than a hundred to one (20 decibans) in favour of one hypothesis or the other.

So, two English plaintexts have more matches than two random strings. Furthermore, because consecutive characters in English are not independent, the bigram and trigram statistics of English are nonuniform and the matches tend to occur in bursts of consecutive matches. [The same observations also apply to German.] Using better language models, the evidence contributed by runs of matches was more accurately computed. Such a scoring system was worked out by Turing and refined by Good. Positive results were passed on to automated and human-powered codebreakers. According to Good, the longest false-positive that arose in this work was a string of 8 consecutive matches between two machines that were actually in unrelated states.

Further reading

For further reading about Turing and Bletchley Park, see Hodges (1983) and Good (1979). For an in-depth read about cryptography, Schneier's (1996) book is highly recommended. It is readable, clear, and entertaining.

▶ 18.5 Exercises

▷ Exercise 18.3.[2] Another weakness in the design of the Enigma machine, which was intended to emulate a perfectly random time-varying permutation, is that it never mapped a letter to itself. When you press `Q`, what comes out is always a different letter from `Q`. How much information per character is leaked by this design flaw? How long a crib would be needed to be confident that the crib is correctly aligned with the cyphertext? And how long a crib would be needed to be able confidently to identify the correct key?

[A *crib* is a guess for what the plaintext was. Imagine that the Brits know that a very important German is travelling from Berlin to Aachen, and they intercept Enigma-encoded messages sent to Aachen. It is a good bet that one or more of the original plaintext messages contains the string `OBERSTURMBANNFUEHRERXGRAFXHEINRICHXVONXWEIZSAECKER`, the name of the important chap. A crib could be used in a brute-force approach to find the correct Enigma key (feed the received messages through all possible Engima machines and see if any of the putative decoded texts match the above plaintext). This question centres on the idea that the crib can also be used in a much less expensive manner: slide the plaintext crib along all the encoded messages until a perfect *mismatch* of the crib and the encoded message is found; if correct, this alignment then tells you a lot about the key.]