

Parallel Text Compression

in the beginning
god created the
heavens and the
earth
now the earth was
formless and
empty darkness
was over the
surface of the
deep
spirit of god
hovered over
the waters
and god said
let there be light
and god saw
that light was
good and he
divided the
darkness from
the light
and god called
the light day
and the darkness
night
and there was
evening and
morning
the first day
was finished
and the second
day
god said let
waters be
gathered
together
in one
place
and let
dry land
appear
and god
called the
dry land
earth
and the
gathered
waters
he called
seas
and god
called the
earth
earth
and the
seas
seas
and there
was evening
and morning
the second
day was
finished
and the third
day
god said
let the earth
bring forth
vegetation
and let trees
bring forth
fruit
and let the
earth bring
forth living
creatures
and let man
be ruler
over the
fish of the
sea and
the birds
of the air
and the
beasts of
the earth
and man
be ruler
over them
and god
blessed the
earth and
said to the
earth bring
forth
vegetation
and let trees
bring forth
fruit
and let the
earth bring
forth living
creatures
and let man
be ruler
over the
fish of the
sea and
the birds
of the air
and the
beasts of
the earth
and man
be ruler
over them
and god
blessed the
earth and
said to the
earth bring
forth
vegetation
and let trees
bring forth
fruit
and let the
earth bring
forth living
creatures
and let man
be ruler
over the
fish of the
sea and
the birds
of the air
and the
beasts of
the earth
and man
be ruler
over them

Craig G. Nevill

Honours Project 1989

Supervisor Tim Bell

Department of Computer Science
University of Canterbury

A Sad Story...

Imagine yourself in ten years time. Ethernet now does run through the ether, and communication costs are minimal. All computing is now done distributively, and if the loads are too high on the local network, it is worthwhile to log into a computer in Hong Kong.

A distributed database has been set up with all literature, encyclopaedias, official statistics, software and news available. Technology has continued its rampant progress, but as usual, human society with its understandable reactionary tendencies has remained much less 'globalised'. In particular, no language group is willing to give up the special cultural identity represented by their language or dialect. This means that this *huge* database has to be maintained in a host of different languages. Machine translation, always recognised as a difficult problem has not yet advanced to such a point as to fool readers into thinking that they are reading a document written in their own native language, so a complete version of the database has to be stored for each language.

The database is as large as it is allowed to be, and Parkinson's law still holds — there is still not enough space for everyone! Language groups small in number are not adequately represented in this global system, and the nations that most need the information are unable to support the maintenance costs of a version in their native tongue. Meanwhile, special interest groups are clamouring for representation of their literature in the database, and the international system administrators are the most loved and hated people in the world.

Sigh.

"What we need," cry the peoples of the word, "is some way of storing all these different versions, semantically identical, in an even more concise form than they are. Surely it must be possible!..."

Contents

| | | |
|----------|--------------------------------------------------------------|-----------|
| 1 | Introduction..... | 3 |
| 1.1 | Parallel Texts | 3 |
| 1.2 | Compression | 3 |
| 1.3 | Parallel Texts and Compression | 4 |
| 1.4 | Overview of the Report | 6 |
| 2 | The Competition..... | 7 |
| 2.1 | Aims | 7 |
| 2.2 | Method | 8 |
| 2.2.1 | NextText — an Automated Betting Program | 8 |
| 2.2.2 | Operation and Mathematical Relationships | 11 |
| 2.2.3 | User Interface | 12 |
| 2.3 | Organising People | 13 |
| 2.4 | Texts | 13 |
| 2.5 | Results | 15 |
| 2.6 | Conclusion | 18 |
| 3 | Extraction of Synonyms from Parallel Texts..... | 19 |
| 3.1 | The Expectations | 19 |
| 3.2 | The Problems | 19 |
| 3.3 | Significance Criteria | 20 |
| 3.4 | Results | 20 |
| 4 | Compression as Encryption..... | 22 |
| 4.1 | Finding a Thesaurus | 22 |
| 4.2 | Finding Texts | 23 |
| 5 | Implementation of Parallel Compression..... | 25 |
| 5.1 | PPM Model | 25 |
| 5.2 | Parallel Model | 26 |
| 5.3 | Combining the Models | 27 |
| 5.4 | Tuning | 29 |
| 6 | Conclusion | 38 |
| 7 | References..... | 40 |
| | Appendix A: Quantisation of Betting Levels..... | 41 |
| | Appendix B: Implementation of Synonym Extraction..... | 42 |
| | Appendix C: Synonyms Deduced from Parallel Texts..... | 44 |
| | Appendix D: Random Parallel Texts | 48 |

1 Introduction

1.1 Parallel Texts

Parallel texts come in two forms: either the two texts are in the same language, or in two different languages. The most common origin of a pair of parallel texts is some translation process. A text may be available in different languages, for example an English translation of a French novel, or the translation of a technical paper from Russian. Also, several translations of significant texts may have been made into the same language, for example Classical texts or the Bible translated into English.

The two cases are quite different in the sense that while the same-language pairs (or parallel translations) are few (they occur mostly in translation from ancient documents such as the Bible or Classical writings), they are significant and widely used. Two language pairs appear wherever translations are held together with the source document, or where documents are maintained in several different languages. This may occur especially in multi-lingual environments such as multi-national organisations and governments of multi-lingual countries (for example, Canada).

While the two cases are different, a two-language pair of texts can be transformed into a same-language pair by the automatic translation of one text into the language of the other. For the purposes of compression (see next section) this translation does not have to be stylistically perfect, so existing techniques of machine translation could be used to convert a two-language pair into a same-language pair. This reduces the problem addressed by this report to the treatment of same-language texts.

An informal definition of parallel texts with relevance to compression is “a pair of texts which say approximately the same thing using different words”, or more formally, “a pair of natural-language texts with equivalent semantic content”

1.2 Compression

In the 1940s, Shannon heralded the birth of a new discipline, *Information Theory*, with a paper showing how the *predictability* or *redundancy* of a text was related to its information content, or *entropy*. His thesis was essentially this: if it is possible to predict with a high degree of accuracy what will follow a given portion of text, the amount of information contained in the text predicted is small. The *entropy*, or *lack of predictability* of the text is where the actual information lies. The aim of compression is to store *only the information contained in the text*. The redundant

part of the text is not stored, as it is predictable, and can therefore be re-created when the information is 'decompressed'.

Many different models of a text can be constructed, ranging from those which look only at the frequency of individual letters, through those that look at groups of letters, up to the most sophisticated models which exist only in the human mind. The latter are based on an *understanding* of the text, on the experience of different texts of the same genre, and on many other cognitive processes not yet properly understood. The better the model of the text, the more redundancy is apparent, and the more concisely the information is stored. For example, rarely does the human brain store a piece of text verbatim; when it is recalled, the information is usually re-created in a slightly different way, due to the redundancy of natural language and human communication, but it retains its essential meaning.

This is obviously a very vague description of the mechanics of actual compression, but it is sufficient for the moment to note that the better the model of a piece of text, the better the predictions made by the model, the less redundancy is stored, the smaller the compressed text is, and the better the compression: prediction is tantamount to compression.

In his work, Shannon quantified the information content of a symbol with respect to a certain model. The entropy (H), or information content of a symbol s if it is predicted with probability p , is

$$H(s) = -\log_2 p(s) \text{ bits}$$

A bit is the basic measure of information, and is equivalent to one yes or no, true or false answer.

This means that if you are 90% sure that the letter after "Mary had a little l" is an 'a', and that letter occurs, the information content is $H('a') = -\log_2 0.9 \approx 0.05$ bits. If however the phrase is "Mary had a little lemma", and an 'e' was assigned a probability of 2%, the information content of the 'e' would be $H('e') = -\log_2 0.02 \approx 1.7$ bits.

One final note about models. Most good models of text *adapt* as they find out more about the text. Human models tend to classify the text quickly, and use experience with that genre as a starting point for the model (all this is of course subconscious). After that point, both human and automated models adapt to the specific vocabulary and style of a text, getting better as they learn more. In the case of a computer model being used for compression, the compression usually improves as processing proceeds.

1.3 Parallel Texts and Compression

How can parallel texts help compression? In the light of the previous discussion, this question can be restated as:

“How can parallel texts help *prediction*?”

To answer this, consider an example. The following text finishes in the middle of a word. Without looking at the text below it, try to predict what will come next:

Yet the age was not so barren in noble qualities, as not also to exhibit examples of virtue. Mothers accompanied the flight of their sons; wives followed their husbands into exile; there were brave kinsmen and faithful sons in law; there were slaves whose fidelity defied even t

Tacitus: The Histories 1.3

Translation: Wellesley

Hopefully, given enough guesses, it would be possible to complete word correctly. Armed with a parallel text, however, the problem becomes greatly easier:

However, the period was not so barren of merit that it failed to teach some good lessons as well. Mothers accompanied their children in flight, wives followed their husbands into exile. There were resolute kinsmen, sons-in-law who showed steadfast fidelity, and slaves whose loyalty scorned the rack. Distinguished men driven to suicide... (etc)

Tacitus: The Histories 1.3

Translation: Hadas

The problem is still not trivial, but the choices are narrower.

The phrase could be ‘the rack’ — but it isn’t. The rack is a form of *torture*, and this is in fact what the word is.

An even more marked example is where a proper noun is involved — proper nouns are usually invariant between parallel texts, for example

Among the provinces, Spain was under the government of Cluvius Rufus, an eloquent man, who had all the accomplishments of civil life, but who was without experience in war.

Of the provinces, Spain was governed by C__ R__, a fine orator, who was tried in the arts of peace, but untried in wars

No prizes for guessing what the missing letters are!

Several useful properties of parallel texts become apparent when an attempt at prediction is made.

There are four broad levels at which texts correspond:

- (i) Often, the words used in the two texts are the same. This can arise is where there are not many appropriate synonyms for a word, or when one text is based on the other (this is often the case when the text has been modified in the light of new information, or the text is rewritten to

change the style). In this case, it is very easy to predict the target text.

- (ii) As the two texts contain the same information, synonyms of words often appear in the target text. In this case, a thesaurus can be used to generate candidate words for the target text.
- (iii) Usually not all of the words correspond exactly to themselves or to synonyms in the target text. However, there is often some phrase which corresponds between the texts. Thus a phrase can be related to a *synonym phrase* in the target text. A model using these correspondences would either need a very large thesaurus relating phrases to phrases, or be able to *understand* the two texts.
- (iv) The very worst case is where no words or phrases correspond in the texts, but the whole texts are known to have the same semantic content. Practically, this very worst case does not arise, but it is not uncommon for the smallest corresponding chunks to be a sentence, paragraph or some other chunk. Very powerful artificial intelligence techniques would be required for a model which will take advantage of these, so that the actual *meaning* of the text will be reduced to some internal meaning representation and re-created in the style of the target text.

Parallel texts can help prediction, and can therefore enable compression to a degree not obtainable otherwise.

1.4 Overview of the Report

This report describes an investigation of the feasibility of parallel text compression, and experiments with practical algorithms to take advantage of the properties of parallel texts.

To investigate the feasibility of parallel text compression, a series of experiments was run based on existing techniques for estimating the entropy of text. These experiments required subjects to predict characters in a text given the preceding text, as well as a parallel text. Their performance was measured against a control group without a parallel text.

The implementation of a practical compressor was the culmination of several threads of research, involving a detailed study of the nature of parallel texts, cryptographic attacks on several files which had been compressed with unknown schemes, the development of a prediction algorithm, and a system for combining two different models

for optimum result



2 The Competition

2.1 Aims

To determine the degree of success that could be expected from parallel text compression, an experiment was conducted in which subjects were asked to predict a text letter by letter, betting money on the possible letters.

This was an extension of work by Shannon [Shannon], Cover and King [Cover], and Zunde and Kelman [Zunde]. Shannon originally tried to place bounds on the entropy of English by asking subjects to guess the next letter of a text until they were correct. The number of guesses needed was used as a measure of the entropy of the text. Cover and King performed similar experiments, requiring subjects not only to guess a letter, but also to say how sure they were that they were right — introducing the concept of betting money on letters. Zunde and Kelman let subjects choose sets of letters to bet on, narrowing down their options until they found the one correct letter. They also automated the process using character terminals.

The subject's aim is to maximise their winnings. It can be shown (see *Operation and Mathematical Relationships*) that this minimises the entropy reflected by their bets. To maximise winnings, they must place their money according to their best judgement. In other words, they must be risk neutral — a risk taker will lose more by ending up with less to bet on other letters when their initial guesses are wrong, while a risk averse person will not obtain the optimal return for their bet each time. The percentage of the money that the subject bets on a letter should therefore reflect the likelihood of that letter according to their internal 'model' of the text.

In this way, the entropy of the text being predicted with respect to the subject's internal model can be determined. The lower bound on this entropy over several subjects indicates the actual information content of that piece of text with respect to the best possible model.

The aim of the experiments in the context of parallel text compression was to ascertain the amount of help a parallel text is in the prediction of a given text.

To see how this could work, imagine a pair of identical twins who have the peculiar property of thinking about a problem in *exactly* the same way. Given identical texts, they will assign exactly the same probabilities to the possible letters.

A technique known as arithmetic coding will take a stream of probabilities and will convey this information in exactly the number of bits dictated by the entropy. — i.e. transmit *exactly* the information reflected by the percentages, and

no more. If the predictions of one of the twins are used as the basis for a compressor, the text can be encoded using the exact number of bits represented by the probabilities assigned to the letters. The other twin's predictions could then be used as the basis of a decompressor, enabling the compressed text to be decompressed. This situation has its analog in automated compression, where the twins are just two identical programs running on two identical machines. Machines, however, are not as clever as humans!

One set of subjects were asked to predict a portion of text, given the preceding text for about ten pages. The other group were also given a parallel text up to and past the point at which the corresponding unknown text started.

For example, subjects were given the text up to and including "*What do you want? Tell me and you* ", where the text following was "*shall have it.*", and the parallel text was "*What is thy petition, queen Esther? and it shall be granted thee.*".

The predictions of the best person with the parallel text and those of the best person without were:

| Letter | Prediction with parallel text | Entropy (bits) | Prediction without parallel text | Entropy (bits) |
|---------------|-------------------------------|----------------|----------------------------------|----------------|
| s | 85 | 0.23 | 40 | 1.3 |
| h | 90 | 0.15 | 80 | 0.23 |
| a | 99 | 0.014 | 95 | 0.074 |
| l | 99 | 0.014 | 95 | 0.074 |
| l | 99 | 0.014 | 98 | 0.029 |
| Total Entropy | | 0.422 | | 1.707 |

Over several experiments, the difference between the lowest entropy without a parallel text and the lowest entropy *with* a parallel text should give a quantitative indication of the helpfulness of a parallel text in terms of a change of entropy.

2.2 Method

2.2.1 NextText — an Automated Betting Program

To automate the betting procedure, and to speed the learning of the subjects, a Macintosh Hypercard Stack, *NextText*, was developed. The initial development, based heavily on Zunde and Kelman's implementation on dumb terminals, was done by Greg Ewing. The original format of the screen is shown in figure 2.1.

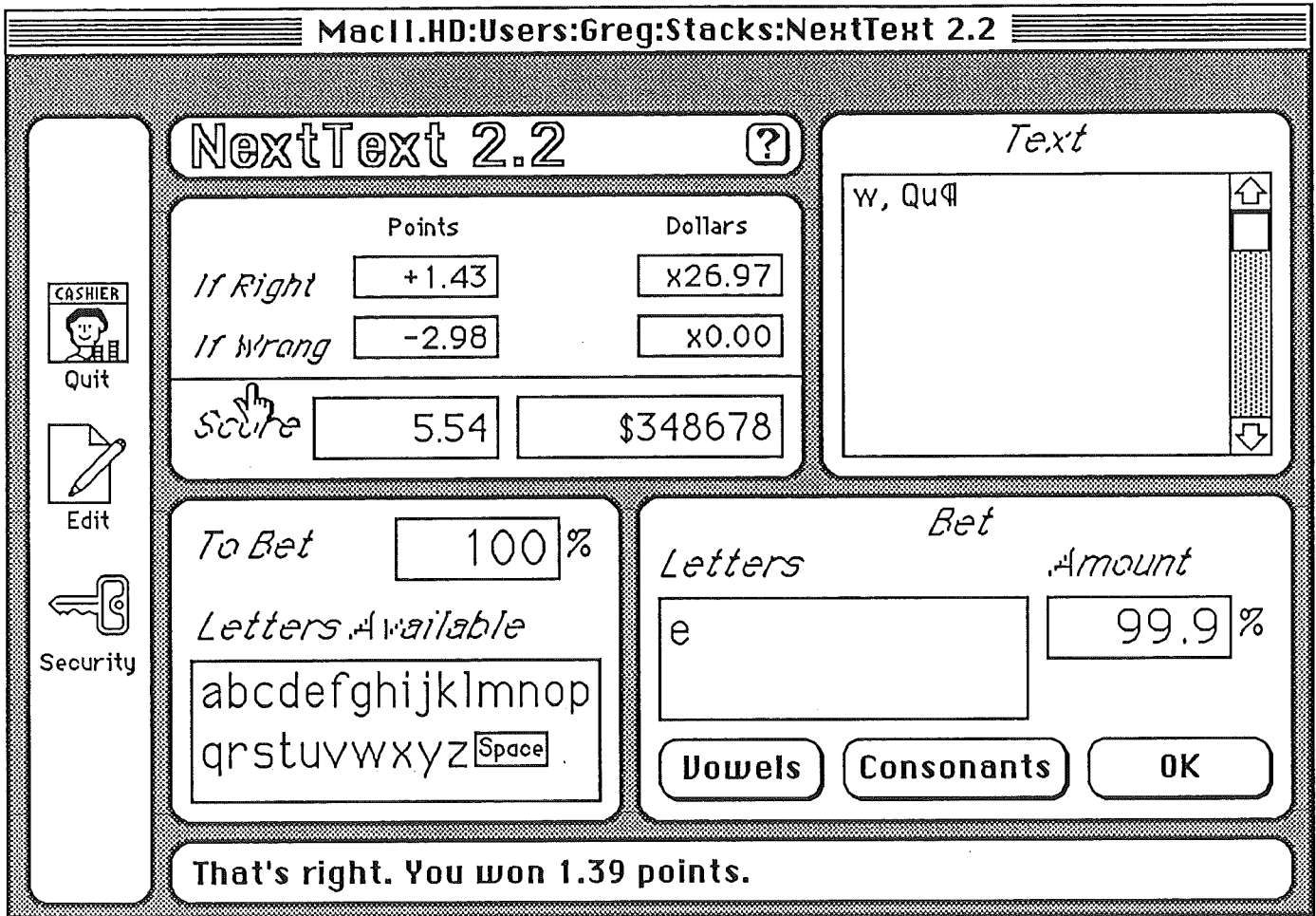


Figure 2.1 The original NextText format

The program was used for the first set of experiments. It worked fairly well, but subjects without computer experience had trouble with tasks such as typing in percentages. The program was also unpredictable, due mainly to the limitations of the Hypercard environment. Also, subjects would often make rash bets, and when the correct letter was not always obvious, percentages became rather small, in some cases as low as 0.01%. Subjects found it hard to work with such small percentages, and performed badly because of this.

The new version of *NextText* incorporated a redesign of the user interface, where the keyboard is no longer used for entering letters (fig. 2.2). Since the prediction is over a limited alphabet of lower case letters and a space, the keyboard could be represented by a matrix of 'buttons' on the screen. To make a letter choice, the subject must use the mouse to click on a letter on the screen and choose a level from a slider.

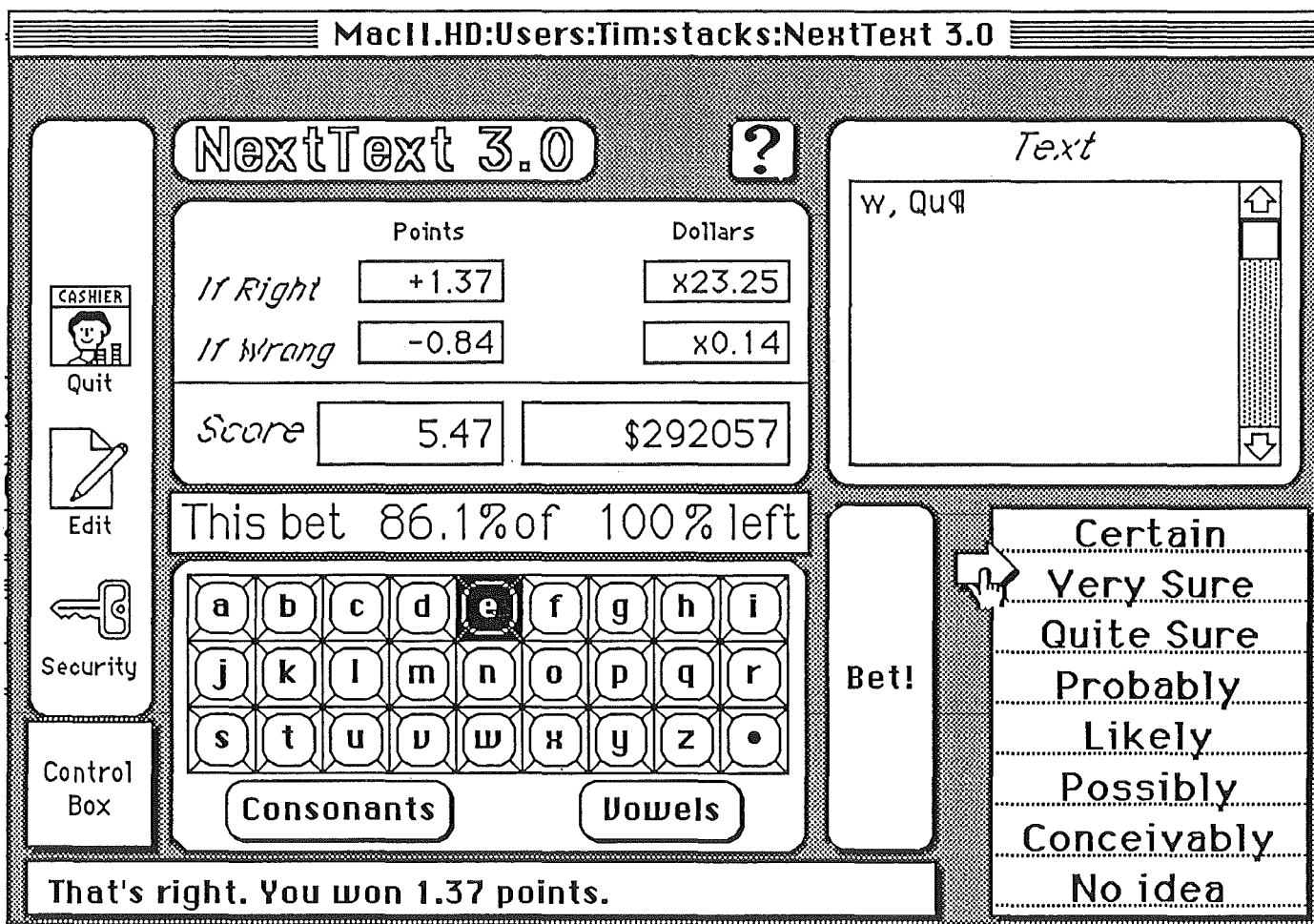


Figure 2.2 The new NextText format

The slider overcame problems with the selection of percentages. It was labelled from “No Idea” through “Conceivably”, “Possibly”, “Likely”, “Probably”, “Quite Sure” “Very Sure” to “Certain”. The reason why eight graduations were chosen was firstly to conform to the 7 ± 2 rule for cognitive processes [Miller], and to be an even number, so that there is no ‘middle’ graduation might encourage people to be lazy in their decision-making and take the middle mark. Nevertheless, the slider can be placed at any position on the scale, at any point between these graduations. The position of the slider relates to the amount of money bet, and therefore the amount to be potentially won or lost.

Before a slider was implemented, a version was used which had eight ‘buttons’ with the same labels, thus quantising the choice of percentage. An analysis of the projected effects of this can be found in Appendix A. A linear progression between the buttons did not introduce an excess of more than 2% in the final entropy, but in some cases reduced entropy up to 23% by enforcing sensible bets.

The slider can only select a percentage from the breakeven point to 95%. The breakeven point is where the probability assigned to each character is the same — so the breakeven percentage for m characters out of n left where there is $p\%$ of the original amount of money left is:

$$\text{breakeven} = \frac{m}{n^p}$$

The rationale for imposing a lower limit was that betting lower than the breakeven point implied that this selection was less likely than its complement. In this case, the subject should really select the complement and bet a higher than breakeven percentage on it.

The upper limit of 95% prevents rash bets being made. In the first set of experiments, subjects would often bet 99% or 99.99% of their money if they were very sure. As explained below, the gains of such high percentages are very small, while the potential losses are great.

2.2.2 Operation and Mathematical Relationships

Several letters can be selected at one time. The process of finding the correct letter is therefore reduced to a series of two-way choices. The slider position represents the probability that the correct letter is *one* of those selected.

If the subject selects more than one letter and is correct, the letters not selected disappear from the keyboard, and the subject has to make a further selection from those left. The amount of money left is equal to whatever was bet on the selection.

If the correct letter is not in the subject's selection, the letters in the selection are removed from the keyboard, and the subject must make a further selection. The amount of money left is the previous total less the amount bet on the selected letters.

This process continues until either the correct letter is selected on its own, or the correct letter is the only one remaining. At this point, the final percentage bet on the correct letter is used to calculate the change in winnings.

The winnings multiplier (m) is related to the proportion bet (p) by

$$m = \frac{27}{n^p}$$

where there are 27 characters available (a..z + space) and n letters are selected.

Because the winnings rapidly became large, e.g. $\$10^{20}$, a points value was also shown, where

$$\begin{aligned} \text{points} &= \log_{10} m \\ &= \log_{10} 27p \end{aligned}$$

This information is updated as the slider is moved up and down before the subject commits themselves to a particular choice. This was an important addition — it gives the subject a feel for the relationship between the slider markings, the percentage, and the winnings.

The entropy, H , of a certain choice is related to p thus:

$$H(\text{choice}) = -\log_2 p$$

also,

$$\begin{aligned} \Delta \text{points} &= \log_{10} 27p \\ &= \log_{10} 27 + \log_{10} p \\ &= \log_{10} 27 + \frac{\log_2 p}{\log_2 10} \end{aligned}$$

$$\begin{aligned} \Rightarrow H(\text{choice}) &= -\log_2 10 \cdot (\Delta \text{points} - \log_{10} 27) \\ &\approx 3.32(1.43 - \Delta \text{points}) \end{aligned}$$

So the points and the entropy are linearly related — as the points increase, the entropy decreases, so maximising the points (and thus the winnings) means minimising the entropy.

As noted before, betting higher than about 95% was undesirable, as the gain in winnings was very small, but if the selection was wrong, a much smaller amount of money was left to bet on.

For example, if a single letter is selected as the first bet, and a 95% probability is chosen, the entropy if right is $-\log_2 0.95 = 0.074$ bits, and if wrong (on average) is $-\log_2(0.05 / 26) = 9.0$ bits.

If the probability chosen is 99.9%, the entropy if right is $-\log_2 0.999 = 0.015$ bits, and if wrong (on average) are $-\log_2(0.01 / 26) = 11.3$ bits — so there is a very small change in the points if right (0.059 bits less), but a large change if wrong (2.3 bits more).

2.2.3 User Interface

Because the betting system doesn't use the keyboard, people inexperienced in using a computer only have to get to grips with a mouse.

Betting can be thought of in several ways, letting the subject choose the most meaningful for them. It can be thought of as betting money on a letter, giving a probability, or selecting a word which best describes the subjects confidence.

An online help system is available which describes the function of each part of the screen.

2.3 Organising People

To find people who were willing to take part in the competition, an advertisement was placed in the *University Chronicle*. It was hoped that that the competition would attract 'educated intellectuals' eager to prove their quickness of mind. Barring people who were never in their office and those who caught the 'flu the day before they were to take part, 10 people were found to participate. The competition was organised as a tournament so that the best people took part several times. They were organised in two groups, with a text each. Three out of each group were asked to predict the text given a parallel text, the other three predicted the text without a parallel text, as a control.

The winner from each group went through to the semi-finals, where those that had a parallel text in the first instance had none, and vice versa. Finally, the winner from each of the two groups went through to the finals, which were with a parallel text. A control wasn't possible in the finals since this would mean an unfair competition between the last two people.

One thing that I learnt when conducting the competition was that the process could not be too well explained. At least half an hour should be spent explaining the operation of the stack, and especially how the scoring worked. Practice runs on another piece of text were essential.

I found organisation very time consuming, with letters, telephone calls, finding texts, explaining procedures, checking on subjects, analysing results, improving methods of running the tournament, notifying people of results... Ideally, the experiment should be run with lots of people, preferably on larger texts, but the process is very time consuming and tiring for the subjects, and after one bout of three or four hours, most have had enough. In fact, after finding three groups of three people for the heats, it was not possible to find more than one person for the last heat. The semi-finals coincided with a very busy time at the end of the year, and the two fourth year Computer Science students who had reached the semi-finals were not willing to spend the hours needed to complete the competition, so this will be completed later.

2.4 Texts

There were three parallel texts involved: two for the heats and one for the semi-finals.

Text for heat one:

'And so the King and Haman went to eat with Esther for a second time. Over the wine the king asked her again, "no w, Queen Esther, what do you want? Tell me and you shall have it. I'll even give you half the empire."

Queen Esther answered, "If it please your majesty to grant my humble request, my wish is that I may live and that my people may live..."

Esther 7:2-3, Today's English Version Bible

Parallel text for heat one:

"...And the king said again unto Esther on the second day at the banquet of wine, What is thy petition, queen Esther? and it shall be granted thee: and what is thy request? it shall be performed, even to the half of the kingdom.

Then Esther the queen answered and said, If I have found favour in thy sight, O king, and if it please the king, let my life be given me at my petition and my people at my request:"

Esther, King James Version Bible

Text for heat two:

"King Xerxes asked Queen Esther, "Who is he? Where is the man who has dared to do such a thing?" Esthe

r said,"The adversary and enemy is this vile Haman."

Then Haman was terrified before the king and queen. The king got up in a rage, left his wine and went out into the palace garden. But Haman, realizing that the king had already decided his fate..."

Esther 7:6-7, New International Version Bible

Parallel text for heat two:

...Then King Xerxes asked Queen Esther, "Who dares to do such a thing? Where is this man?"

Esther answered, "Our enemy, our persecutor, is this evil man Haman!"

Haman stared at the king and queen in terror. The king got up in a fury, left the room, and went outside to the palace gardens. Haman could see that the king was determined to punish him for this, so he stayed behind to beg Queen Esther for his life...

Esther 7:5-7, Today's English Version Bible

Text for semi-final:

He was son of that Vitellius who was censor and three times consul; this was thought sufficient recommendation. In the army of Britain there was no angry feeling; indeed no tro

ops behaved more blamelessly throughout all the troubles of these civil wars, either because they were far away and separated by the ocean from the rest of the empire, or because continual warfare had taught them to concentrate their hatred on the enemy.

Tacitus: The Histories 1.9 (Translation Hadas)

Parallel text for semi-final:

Aulus Vitellius, son of the Vitellius who had held the censorship and three consulships. These, it seemed, were qualifications enough.

In the army of Britain there were no hard feelings. Indeed, throughout the period of civil war, no other legions acted with greater propriety. The reason may lie in the fact that they were far away, beyond the barrier of the North Sea; or perhaps they had learnt from continual campaigning to reserve their hatred for the enemy.

Tacitus: The Histories 1.9 (Translation Wellesley)

2.5 Results

Results were gleaned from *log files* which were kept by the betting program. The log files contain information about what letters were selected, what probability was given, and the time of each bet. From this, the entropy of each bet, the total entropy and the average entropy can be determined.

The final entropies for each of the subjects for the three texts are given in table 2.1.

Table 2.1

| Text One | | Text Two | | Text Three | |
|----------|-------------|----------|-------------|------------|-------------|
| Parallel | No Parallel | Parallel | No Parallel | Parallel | No Parallel |
| 0.47 | 0.92 | 1.0 | 1.5 | 1.2 | 1.5 |
| 0.71 | 1.3 | 1.1 | | | |
| 0.90 | 3.4 | 1.7 | | | |

(all quantities are in bits per character)

A few of the results were unexpected — the 3.4 bits per character was due to one subject who had trouble with the concept of the prediction, and with the operation of the program, and eventually gave up. The 1.7 bits per character was due to a similar situation. Age seemed to play a large part in how well subjects picked up the concepts involved and therefore the subjects' final entropy.

Overall, however, the average entropy of subjects supplied with a parallel text was significantly lower than that those without, about twice as well for the first text, and about one third better for the second text (although more samples would be needed to verify this second result). The third text seems even harder, with only one fifth improvement.

Apart from the actual entropy statistics, there were many interesting properties to note from graphs taken from the log files of the subjects.

For example, the graph of the entropy of each letter as they are predicted by a subject is shown in figure 2.3.

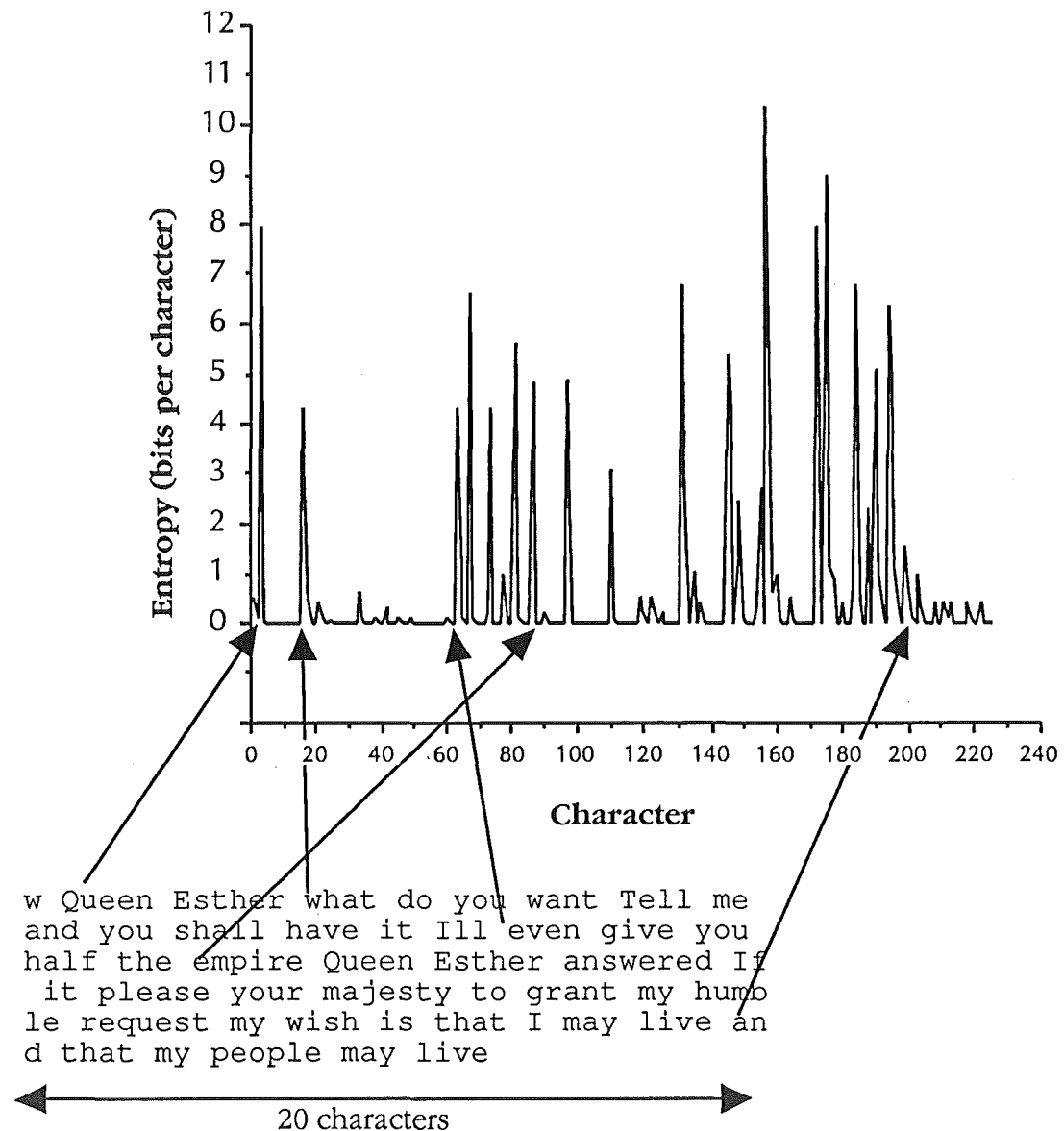
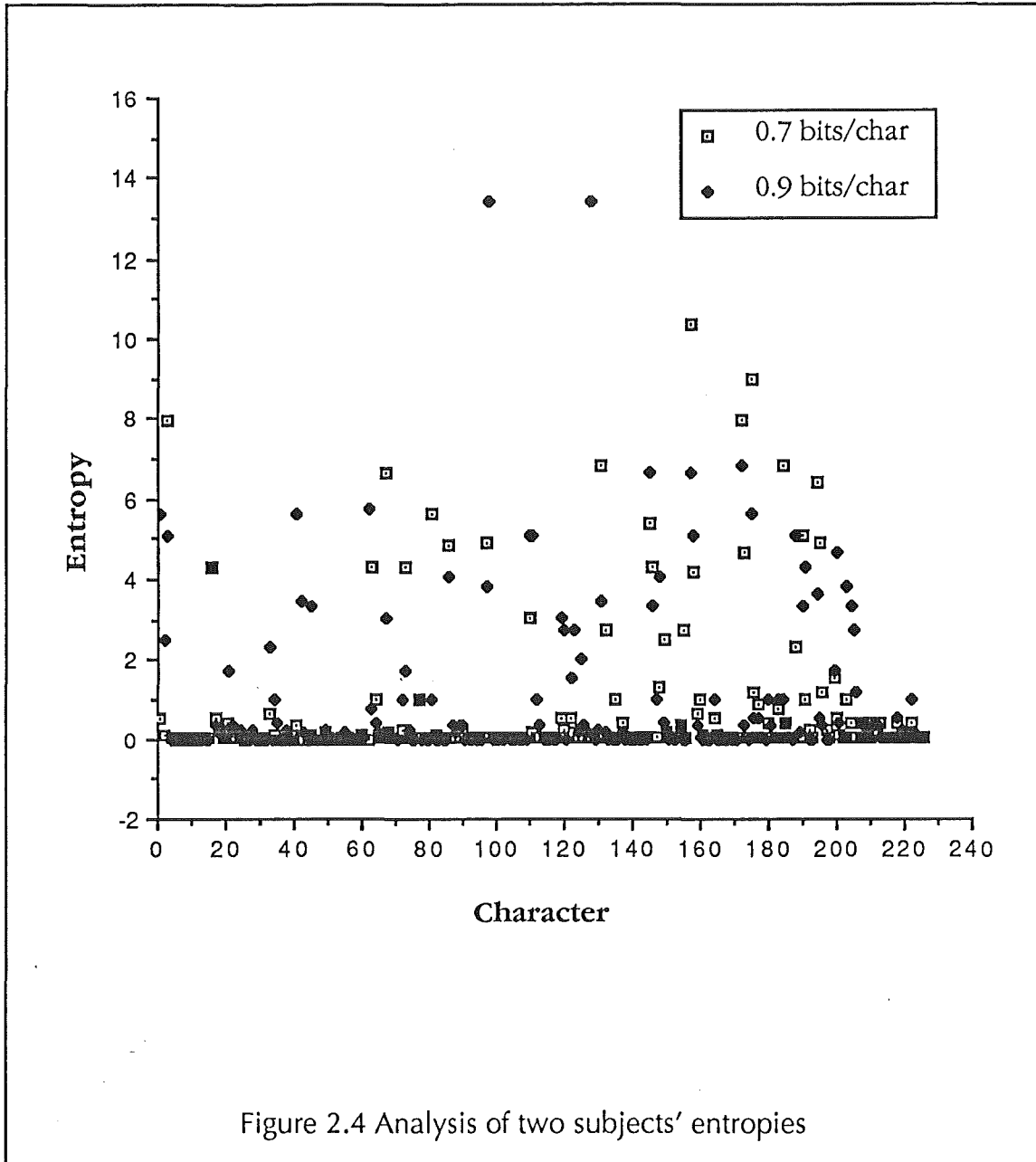


Figure 2.3 Entropy of the characters in a text

Note the extreme nature of the graph; most of the points are very close to zero, but there many peaks to four bits and over. These peaks usually correspond to the beginnings of words, and the flatter periods after the initial peak is the rest of the word, which is usually much easier to predict and therefore has a much lower entropy.



What makes the difference between a good better and a bad one? In figure 2.4, the entropy for each character for two different subjects are graphed together. One produced a final entropy of 0.9 bits/char, the other 0.7 bits/char. The difference does not seem to lie so much in the (many) values near zero, but instead in the high values. The better subject (entropy of 0.7) has managed to 'cut their losses' probably by not making rash bets, and always having some reasonable probability to fall back on.

The importance of the unexpected characters producing large entropies is shown in figure 2.5.

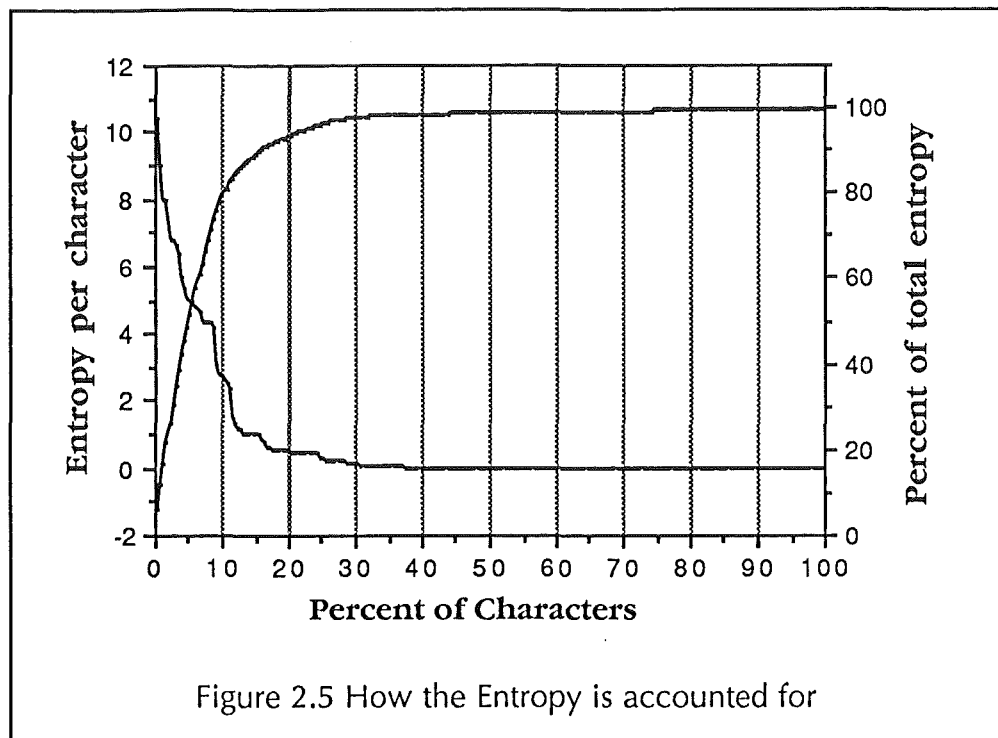


Figure 2.5 How the Entropy is accounted for


The line falling to the right in figure 2.5 is the entropy used by each character, sorted into descending order of entropy, from least expected symbol to most predictable. The line *rising* to the right is the cumulative sum of these entropies. This is the amount of the final entropy accounted for by symbols whose entropy are over a given value. For example, 80% of the entropy is accounted for by 10% of the symbols — those with an individual entropy of three or more.

This reinforces once more the value of minimising the losses incurred by these bad predictions.

2.6 Conclusion

The results of the competition have shown that parallel texts *are* helpful, but their usefulness varies. Some parallel texts can reduce the entropy of a text by a factor of two, others take a third or a fifth off.

When compressing, special note needs to be taken of catering for unexpected characters — damage control and not dare-devil tactics are required

The information gleaned from the competition forms an excellent basis for the implementation of a practical parallel compression technique 

3 Extraction of Synonyms from Parallel Texts

3.1 The Expectations

In the introduction, it was noted that the first requirement for a practical parallel compression technique was a thesaurus. This presented a problem: no thesaurus was readily available in machine-readable form. However, a possible solution became apparent after a little thought: if a pair of parallel texts were related partly by the fact that words corresponded with synonyms, it should be possible, over a large enough piece of text, to deduce those synonyms and thereby build up a static thesaurus for that pair of texts. Some correspondences may be peculiar to that combination of texts, but many should be general enough to be useful as a basis for a general-purpose static thesaurus.

Two texts were available: the Bible in *Today's English Version* (obtained from the *English Bible Society*) and the New Testament in the *King James Version* (obtained from a public domain HyperCard Stack).

The algorithm for the extraction of synonyms was fairly simple: Firstly, read a verse from each version. For each word in the source version, make a note of its occurrence with every word in the target version. Do this for each verse in the texts. At the end, apply some selection criteria for extracting valid synonyms.

3.2 The Problems

One problem became immediately apparent: the memory requirements of this algorithm are huge. Given the whole of the New Testament, with 8000 distinct words and 7000 verses, the number of correspondences that have to be recorded is huge.

One way to reduce the number of candidate synonym pairs is to tag words with their parts of speech. As synonyms must have the same part of speech, correspondences between two words can be immediately ruled out if they have different parts of speech.

To this end, 2246 words were manually tagged with their parts of speech. Many had to be tagged with several tags, for example most ending in 'ed', such as 'borrowed' were recorded as both an adjective ('a *borrowed* book') and the past participle of a verb ('she *borrowed* a book'). Some words, like *still* (noun, verb, adjective, and adverb) and *down* (noun, verb, adjective, adverb, preposition) were

tagged with even more tags. This whole process was very time-consuming, but made the synonym finding feasible.

Static tagging is only a small part of determining the grammatical category *h* of a word in context; there have been several papers published on dynamic disambiguation, where context and syntax are taken into account to decide on the part of speech of a word in a specific context. This would have made the process more fruitful, as the more combinations that can be ruled out on the basis of grammatical category, the less have to be weeded out statistically later on.

For a description of the implementation of the program and the special memory problems associated with the thousands of connections, see Appendix B.

3.3 Significance Criteria

The selection of sensible, effective selection criteria was difficult, but finally, the following criterion was chosen. The number of times that two words occur together in a verse must be more than the number of times that they would be expected to occur together by chance.

The number of times the synonym is expected in a verse in the target text is $\frac{\textit{synonym frequency in target}}{\textit{number of target verses}}$. The number of times the two words would be expected to be heard together is $\frac{\textit{synonym frequency in target} * \textit{word frequency}}{\textit{number of target verses}}$.

Here, *word frequency* (the number of times this word has appeared in the source text) acts as an approximation to the number of verses that this word has appeared in. If the number of times this combination has occurred is greater than the value of the above expression by a certain factor, then that combination is selected.

Selecting an appropriate threshold factor was interesting — the value that didn't print too many 'silly' combinations and didn't exclude too many useful ones was about 500. This means that a pair must occur with a frequency 500 times higher than would have been expected to be significant!

3.4 Results

The result of this process were disappointing. The reasons for this are several:

- (i) it is not possible to distinguish between synonyms and words that simply occur often together (e.g. antonyms!)
- (ii) Grammatical category disambiguation would produce much more accurate results
- (iii) Texts that are closer in style of expression should be used (the KJV and the TEV are a long way apart on the stylistic spectrum)

- (iv) Larger texts should be used. Correspondences can only be deduced when words occur *many times*. The ideal texts for a more successful attempt would be many times longer than those that were available
- (v) The most common words are often common because they do not have many synonyms, but they are the only words that occur often enough to be statistically significant, and worthwhile tagging. The more interesting words that make up the bulk of the unique words (80%) make up only 20% of the actual text; they are too numerous to be tagged, and do not occur often enough.

Despite this, some interesting results were obtained; full lists can be found in Appendix C. A selection of interesting ones are:

| TEV | KJV |
|-----------|------------|
| despiseth | rejects |
| covetous | greedy |
| pit | abyss |
| purified | salted |
| grumbling | murmured |
| bag | purse |
| Noah | Noe |
| closed | straitened |
| cold | hot |
| Priscilla | Aquila |

Note: strait is an archaic word for “narrow, limited, confined or confining”!



4 Compression as Encryption

4.1 Finding a Thesaurus

A good thesaurus is essential to a realistic attempt at parallel compression. The only thesaurus in computer-readable form that was readily available was the “Word Finder” desk accessory for the Apple Macintosh. Word Finder consists of two files — the program, and a 305 kilobyte ‘Large Thesaurus’ file. A scan of the latter for English words revealed nothing, as it was assumed that the thesaurus was compressed in some way.

Several properties of the compression method could be assumed based on the type of access needed. Firstly, as random access into the file would be required, the compression scheme could not rely on a context model built over a large part of the file — it would have to be static. The main candidate was a form of Huffman coding. Random access also implied some sort of alignment of each word or list of words, as this would simplify indexing.

Studying the file, it became apparent that it was divided into blocks of 512 bytes, padded with 0xff bytes if necessary. Apart from this, there was very little to offer a foothold for the attack. What was really required was some idea of where a given piece of information was located, and from there, a known or even chosen plaintext attack could be mounted. In other words, there had to be some way of ‘seeing’ the information that the Word Finder was getting from the disk.

To this end, the Large Thesaurus file was transferred to the Sun network using the Apple-Unix file server (Aufs). A weekend-long examination and modification of the server code yielded a version that would display information on a Sun about which portion of the file was being read by Word Finder, which was running on the Macintosh. It was immediately obvious that the file was read in blocks of 512 bytes — which meant that no accurate knowledge of the correspondence between a word displayed on the Macintosh and a sequence of bytes in the main file could be gleaned.

However, after a large amount of fruitless exploration, three properties of the accesses were discovered. Firstly, it was found that to find synonyms for a word, the first block was read, followed by one block (or a series of consecutive blocks) from the first 304 blocks, then one or more blocks from the last half of the file. This was perhaps indicative of hashing into the first half with reference to the first block, yielding indexes into the second half.

Secondly, the position of the block read in the first half of the file was roughly proportional to the position of the word in lexical order.

Thirdly, it was found that finding synonyms for words starting with 'a' produced accesses to the early part of the second half of the file, however, as the word chosen moved along the alphabet, the accesses became less predictable.

Due to this last property, it was possible to compile a list of all the words contained in the first block of the second half of the file by asking for words in alphabetical order until an access was made to the second block in the second half. The words and the corresponding block from the file (in hexadecimal) were painstakingly compared, and there seemed to be a relationship between them, but the exact connection was rather elusive. Finally, as a last resort, a binary dump of the block in question was made, and after an hour of study, all became clear.

The basis of the compression technique was exceedingly simple — each character was represented by a five bit number; 1 for 'a', 2 for 'b', ..., 26 for 'z', 27 for '-', 28 for `` and 299 for a space. Each word ended with five zero bits, and each list of words ended with five one bits. Each list of words corresponding to one entry in the thesaurus was preceded by four bits representing the part of speech of that particular list. These four bits at the start of each list were byte-aligned.

The first half of the file consisted of a word in the five-bit code described above, followed by a series of 21-bit numbers pointing to the synonym lists in the second half of the file. The first ten bits of the pointer was the offset in blocks from the start of the second half, the last ten bits were the byte-offset from the start of the block, and the bit in the middle was set if this pointer was the last list for the word.

Having spent a week following red herrings and making the occasional discovery, it was time to write some more code to decode the thesaurus. Soon it appeared in text form, all ready to be used in the rest of the project. The obvious thing to do now was to write a UNIX™ program that would find a synonym given a word. Using the experience in hashing gained with the synonym-generator, a program was written that would find a list of synonyms and write them to a file, taking about 2 ms per word.

4.2 Finding Texts

One large problem still stood in the way of the implementation of a parallel compressor. There was very little text on which it could be tested. To get copies of parallel texts, a Macintosh-based Bible-searching program was ordered. When it arrived, it was obviously compressed, but by using the same techniques as before, the text was eventually

extracted. The compression scheme used was this: a word is stored as a two byte index into an array of words. If the index is negative, the word at the corresponding positive index is capitalised. If the index starts with a certain range of codes, this is treated as an index into an array of punctuation. Indexes pointing to the start of books, chapters and verses are also present. Within a few days, the Bible in three versions was available for the parallel compressor to work on.



5 Implementation of Parallel Compression

Throughout the design of this project, constant recourse has been made to the human model — how would a human predict text given a parallel text? The human model in this case really consists of two parts — knowledge about English text quite apart from the parallel text, and the special clues provided by the parallel text. When nothing sensible seems to be suggested by the parallel text, a human will rely simply on the style and meaning of the target text, as well as on their general grasp of English expression. This subconscious model-switching ensures that prediction is, on average, no worse than what it would be without a parallel text. Sometimes a parallel text will be a hindrance, if it is not close enough to the target to be useful, but mostly it will be useful to some degree.

The goal of this project is to compress better than the best compression possible without a parallel text. The marriage of a model specific to parallel texts and a general-purpose model would take the following form: Both models must monitor how well they they think that they are doing. A parallel model will sometimes be totally stumped, if the letters appearing in the target text do not seem to correspond to any word, synonym, synonym phrase or chunk of the source. In this case, the general model will take over. Sometimes the opposite may be true, and the parallel model will take over then the general model runs into difficulty. Controlling the whole operation will be some blending program which will take the two sets of predictions for letters, blend them together according to the confidence of each of the models, and perform the compression.

This concept applies more generally than to parallel text compression. In the future, for compression to improve, Artificial Intelligence techniques will need to be used in conjunction with the present models to take advantage of both the semantic and the syntactic components of texts. This follows closely the way a human performs prediction, and involves the blending of at least two different models. The design decisions made here for parallel compression may well be precursors of those made in the near future for the next generation of text compression techniques.

5.1 PPM Model

The Partial Pattern Matching (PPM) compression technique is one of the best of the world. Last year, it won a \$30,000 prize in a compression competition [Moffat].

PPM works by keeping a note of which characters occur in which contexts for a number of different context lengths. When a character is to be predicted, PPM looks at the preceding three characters. If the character to be predicted has been seen in this context before, the probability assigned to that character is the number of times that it has occurred in that context over the total number of times that context has been seen. If it does not occur in the three-character context, PPM looks at the preceding *two* characters and does the same thing. PPM continues until it finds a context in which it has seen the character to be encoded, otherwise it encodes the character with the probability it would have if every letter was given an equal probability.

This means that PPM can adapt to find the best context length to predict a given character. However, this freedom of choice does have its costs. Because the decoder has to generate exactly the same probabilities for characters, it needs to know if the encoder changes to a shorter context. To this end, part of the codespace (the range from 0 to 1 which all the probabilities must fit into) is reserved as an *escape*. Before encoding a symbol, PPM can encode a number of *escapes*, notifying the decoder that it has gone to a smaller context.

For example, if the character 'x' is to be encoded, and it has been seen before in the same three-character context, it can be encoded using the probability mentioned above. If 'x' has only been seen in the same *one*-character context, PPM will encode <esc><esc>x, and the decoder will know to change to a one-letter context. An escape is encoded as if it were just another character, but it has special significance to the encoder and decoder. An escape has its own probability as well; the version of PPM, PPMC calculates the probability that it assigns to the escape in a context by

$$\frac{\text{number of } \textit{different} \text{ characters seen in this context}}{\text{total number of characters seen in this context}}$$

5.2 Parallel Model

In the introduction, four levels of relationships between parallel text pairs were outlined. These corresponding to four levels of increasing sophistication required for the model to take advantage of these relationships. The model actually implemented only covers the first two cases — the trivial case where words correspond to words, and the slightly harder case where words correspond to synonyms.

Using the thesaurus, the source (parallel) text is pre-processed to produce a (much larger) temporary file containing the text, and for every word with known synonyms, a list of its synonyms.

The parallel model reads a chunk of the source (where a chunk is the smallest portion of the text for which an exactly equivalent portion can be found in the target text) and sets about predicting the target text.

The original implementation worked in the following way: each word in the parallel text is compared to the context from the target text. If there is a match, the next letter in the source word is noted as a candidate for the next letter in the target text. For example, given the texts used in the introduction:

Yet the age was not so barren in noble qualities, as not also to exhibit examples of virtue. Mothers accompanied the flight of their sons; wives followed their husbands into exile; there were brave kinsmen and faithful sons in law; there were slaves whose fidelity defied even t

And the parallel text:

However, the period was not so barren of merit that it failed to teach some good lessons as well. Mothers accompanied their children in flight, wives followed their husbands into exile. There were resolute kinsmen, sons-in-law who showed steadfast fidelity, and slaves whose loyalty scorned the rack. Distinguished men driven to suicide... (etc)

To predict the letter following the 't', the simplest technique is to look at all the words in the parallel text beginning with 't': the, that, teach, their and there. Candidates for the next letter would be 'h' (x4) and 'e'.

If the source word is completely matched by the incoming letters, a space is noted as a candidate for the next letter. Noting a candidate simply involves adding a value to an array of weights for letters.

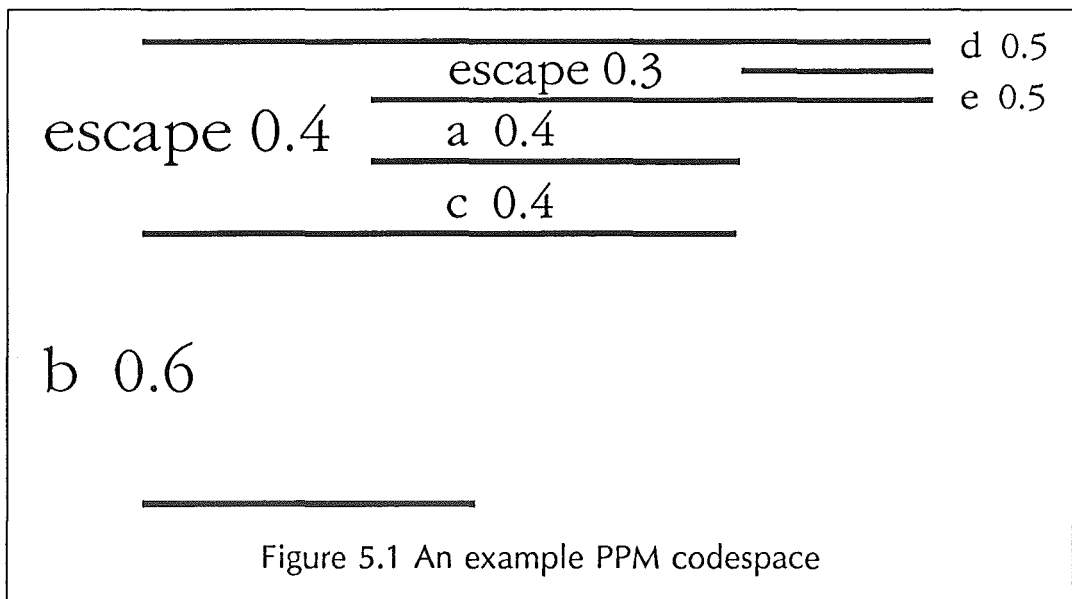
Next, the synonyms for the the first word are searched for ones which start with the letters given. If one is found, the procedure is the same as for a word, except that the weighting is less. For example, one of the synonyms for 'rack' could be 'torture', so 'o' would be noted as a candidate for the next letter.

5.3 Combining the Models

The two requirements for blending the two models is that they both produce a distribution of probabilities for each symbol in the alphabet, and give some indication of how accurate they think their predictions are. The parallel model easily produces the required distribution using the algorithm

described above, and the calculation of the confidence estimate is described in the discussion of tuning below. PPMC is a somewhat more difficult case, as the distribution it creates has some of the codespace put aside for the escape, and the codespace for some characters lie within the escape (see figure 5.1)

However, all the effective probabilities for all the symbols in the alphabet *still add up to 1*. This means that all the probabilities of all the symbols can be moved from their position within the escape codespace to their place in alphabetical order (figure 5.1)



The code for the PPMC model was that written by Alistair Moffat. The algorithm to produce the distribution is this: a real variable *escape* is initially set to 1. Each of the symbols which have occurred in the longest context are given their normal probabilities as calculated above, multiplied by the escape (which is 1 at this point). Next, *escape* is multiplied by the escape probability in that context. The symbols that have occurred in the next context level are assigned probabilities as before, but scaled by the new value of *escape*. This continues until all letters are accounted for. In the above example,

| | |
|------------------------|-------------|
| p(a) | = 0.6 |
| p(c) = 0.4 x 0.4 | = 0.16 |
| p(a) = 0.4 x 0.3 | = 0.12 |
| p(e) = 0.4 x 0.3 x 0.5 | = 0.06 |
| p(d) = 0.4 x 0.3 x 0.5 | = 0.06 |
| <hr/> Total | <hr/> = 1.0 |

These probabilities are then put into the required array.

The final product is an array of the effective probabilities that the PPMC model would have produced for each symbol. Using these probabilities will achieve exactly the same compression as the original compressor.

Two notes about the recording the occurrences of a symbol. In the actual implementation, if a 'b' is seen in the context 'bab', 'b's count in that context will be incremented, but its count in the contexts 'ab', 'b', and so on will not be changed. This is called *update exclusion*, and speeds the algorithm, as well as usually increasing compression.

Also, the original model updated the model as it encoded a symbol, but this is unsuitable in this context, because a probability for each of the symbols have to be generated under the same conditions to produce the distribution. A new procedure, *build_ppm_distribution* searches through the data structure (a trie) to gather the probabilities.

The first requirements for this blended models compressor have now been satisfied: a parallel model producing predictions for the next letter in a text, and a PPMC model producing similar predictions.

The main program was relatively easy to write once the two models had been developed. The main problem was integrating the reading of characters so that the models were kept in step.

Originally, blending was fairly simple; no adaptive blending was implemented; the function determining the weighting given to each model was constant throughout the compression. The application was compiled, the first one hundred verses from Genesis in the *Revised Standard Version* were passed through the modified thesaurus program, and stored in the file *source*, and the same portion of the *King James Version* stored in a file *target*. The program was run, and produced a file 2750 bytes long. It even decompressed! The target text was originally 12672 bytes. By setting the parallel model confidence to 0, the parallel model is ignored, and the whole model is equivalent to the PPMC. The size of the compressed file without the parallel model was 3443 bytes, so the parallel model yielded another 20% compression.

5.4 Tuning

From the results of the competition, it seemed that it could be possible on some texts to achieve up to twice the compression with a parallel text as without. A realistic goal, I thought was 30-40% better with tuning.

The parallel model has many parameters, and finding optimal values for these seemed the best place to start for the tuning effort.

Firstly, when looking at a pair of parallel texts, it is obvious that the more letters a model has of a word, the more sure it is about the predictions that it makes. Exactly how this 'confidence function' varies with the length of the string had to be determined.

One very useful property of the model is that the optimal weights for different lengths are independent of one another. For example the weight given to the parallel model if three letters of a word have been seen is completely irrelevant if only two letters have been seen! Therefore, to find the optimal value for the weighting for a given string length, all that is necessary is to set all other weights to 0, and vary the weight in question. The parallel model weighting is given as a percentage, and the PPMC model receives the rest of the weighting out of 100%. For example, a 30% weighting would mean that PPMC would get 70%; about a 1:2 weighting in favour of PPMC. The model was tested with 11 different weightings (0 to 100% in 10% increments), for lengths from 1 to 10. The graph of the number of bytes saved by each of the weightings is in figure 5.2. The optimal values of each of the weightings is where each line is at its maximum.

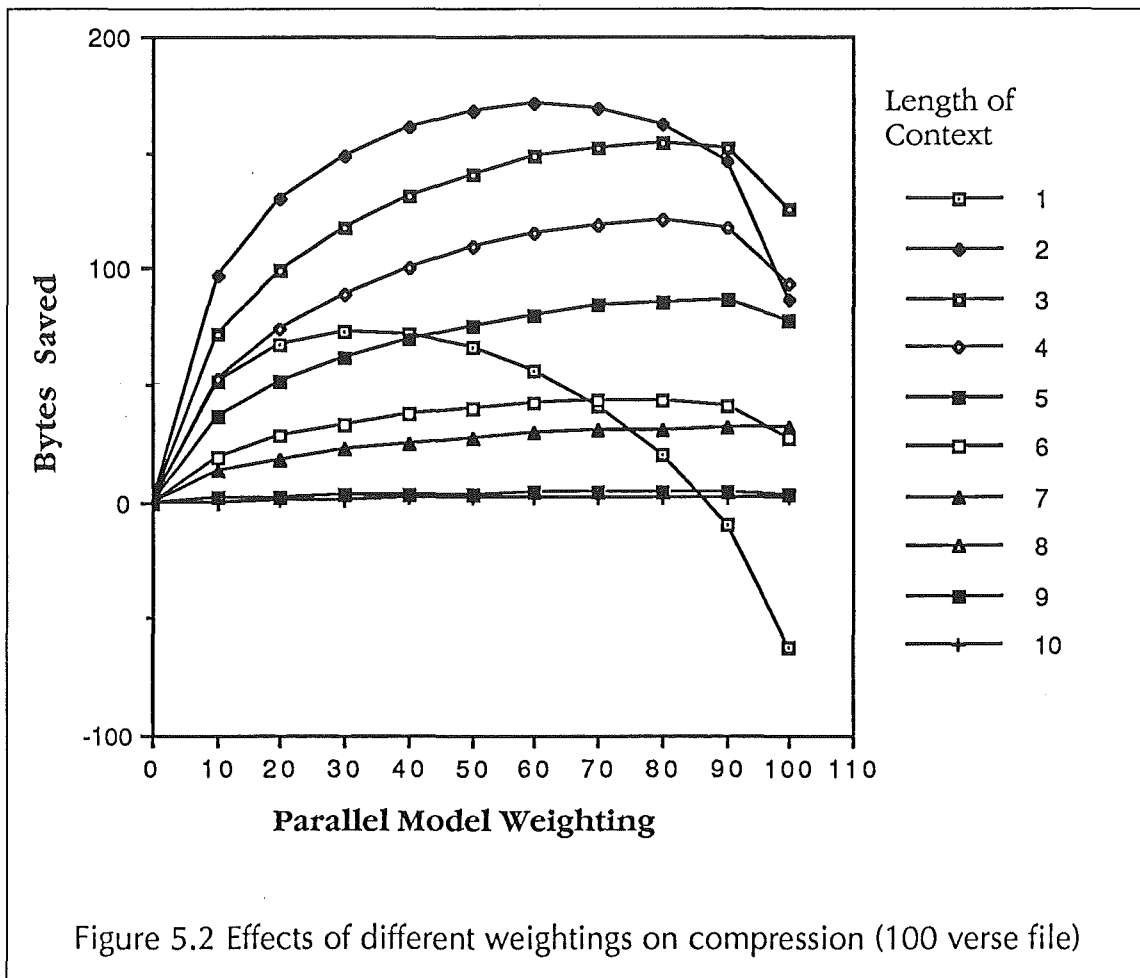


Figure 5.2 Effects of different weightings on compression (100 verse file)

Because the weights are independent, the number of bytes saved for the optimal value of *each* weight can be summed to produce the total number of bytes saved by the parallel model. For example, if by tuning the weight for context lengths of one gains 500 bytes over ignoring the parallel model, tuning for lengths of two gains 300, 200 for lengths of three and so on, the total number of bytes gained by tuning all of them is $500 + 300 + 200 + \dots$. Note that the amount of compression gained by the tuning of each weight from two onwards decreases. This is probably simply due to the rarity of the longer words. Probably the most important point about the shape of these graphs is the insensitivity of the compression ratio to the weights around the optimal weight. This means that the selection of weights is not critical, and could possibly be static. The same optimisation was tried on a larger (more realistic) 1000 verse file (130 kilobytes) (see figure 5.3), producing reassuringly similar results (see table 5.1).

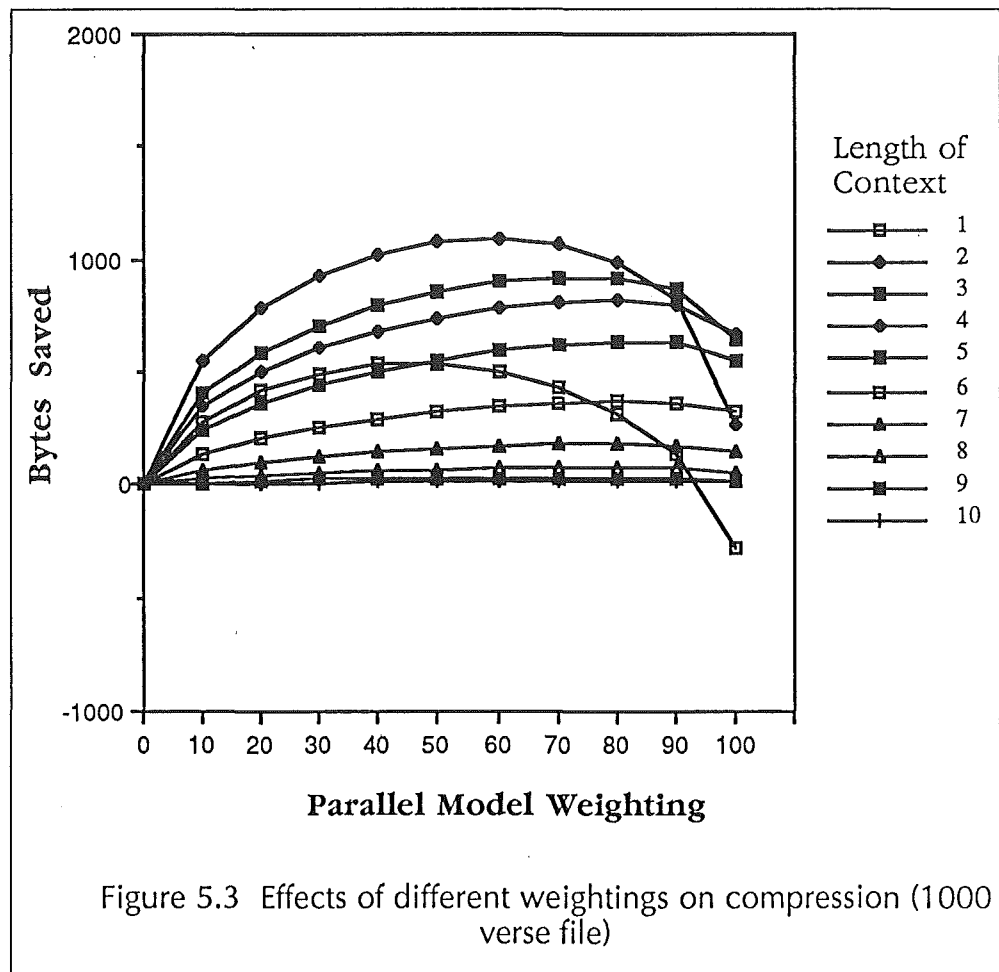


Table 5.1 Optimal weights for small and large files

| Length of context | Optimal parallel model weighting for small file | Optimal parallel model weighting for large file |
|-------------------|-------------------------------------------------|-------------------------------------------------|
| 1 | 30 | 40 |
| 2 | 60 | 60 |
| 3 | 80 | 80 |
| 4 | 80 | 80 |
| 5 | 90 | 80 |
| 6 | 90 | 80 |
| 7 | 90 | 80 |
| 8 | 90 | 90 |
| 9 | 90 | 90 |
| 10 | 90 | 90 |

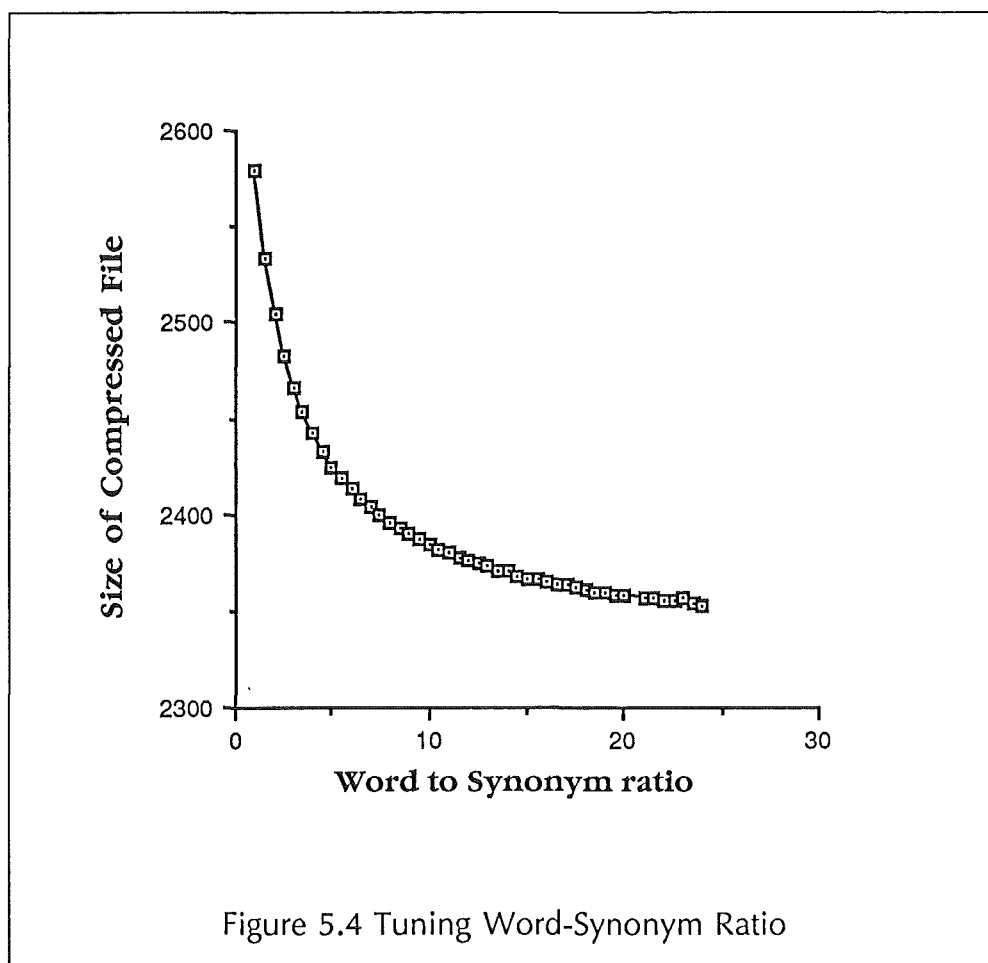
Using these optimal weights reduced the size of the file by another 2-3%.

The next parameter needing optimisation was the weighting of unexpected symbols. The dilemma was essentially this: if a completely unexpected letter occurs, it must still have been assigned some codespace, otherwise it cannot be encoded. However, if all of the letters are weighted, the codespace for *expected* letters becomes smaller. The PPMC model caters for unexpected symbols by switching to a shorter context; should the PPMC model do the same? Looking at the issue from a practical point of view, it was found when analysing the results of the competition that it was desirable to limit the maximum bet to 95%, in order to leave some room for error. The extra compression from a 99% bet over a 95% bet was little, but the possible losses were higher.

The original version assigned a count of one to everything, then added 40 for a matching word, and 20 for a matching synonym. Changing the model to assign zero to all counts before building the distribution produced a 2300 byte file — a 5% improvement! It seems that letting PPMC cater for the unexpected symbols is desirable.

Next, the relative weightings of words and synonyms. If a word in the parallel text matches the given context, this is more useful than a matching *synonym* of a word in the parallel text. But *how much* more useful? Setting the weight for a synonym to 20, the file was compressed with word weightings of 20 through 400 in increments of 20 (matching words are assumed to be *at least* as useful as a matching synonyms). Looking at the graph in figure 5.4, the graph is still heading downhill when the weighting for a synonym is

less than 1/20. This indicated that synonyms were practically useless.



After examination of the synonyms produced for the words in a verse, the problem became clear. The most lucrative time for the parallel model was in the first few characters (see figure 5.2) — the most compression can be gained from making good predictions based on two or three letters. The number of synonyms produced by a verse were huge. This means that for a given one- or two-letter string, many matching but totally useless synonyms can be found. This information is more of a hindrance than a help, so compression is better without it. Another reason for the lack of usefulness for synonyms was that the RSV-KJV pair are very close, in that the RSV was a revision of the KJV, and many original wordings have been retained. For this reason, words were much more useful than synonyms for this particular text.

Removing the use of synonyms entirely reduced the compressed file size another 2%, but another addition to the algorithm to use synonyms more wisely reduced this still further. The new, simple addition to the algorithm involved

using synonyms only when *no matching words* could be found, provided the context length was greater than two. This meant that the previous confusion caused by the large number of words matching small contexts was removed. The condition for use of synonyms was changed from

```
(words_matched == 0 && length > threshold)
```

to

```
(words_matched == 0 || length > threshold)
```

which meant that synonyms would be used if there were no more matching words *or* the context length is long enough to reduce the number of matching synonyms. In the first case, there is no time when *both* words and synonyms are used, so the weightings are independent. The only parameter to optimise is the threshold at which synonyms are used. After tuning, a threshold of two was found to be best.

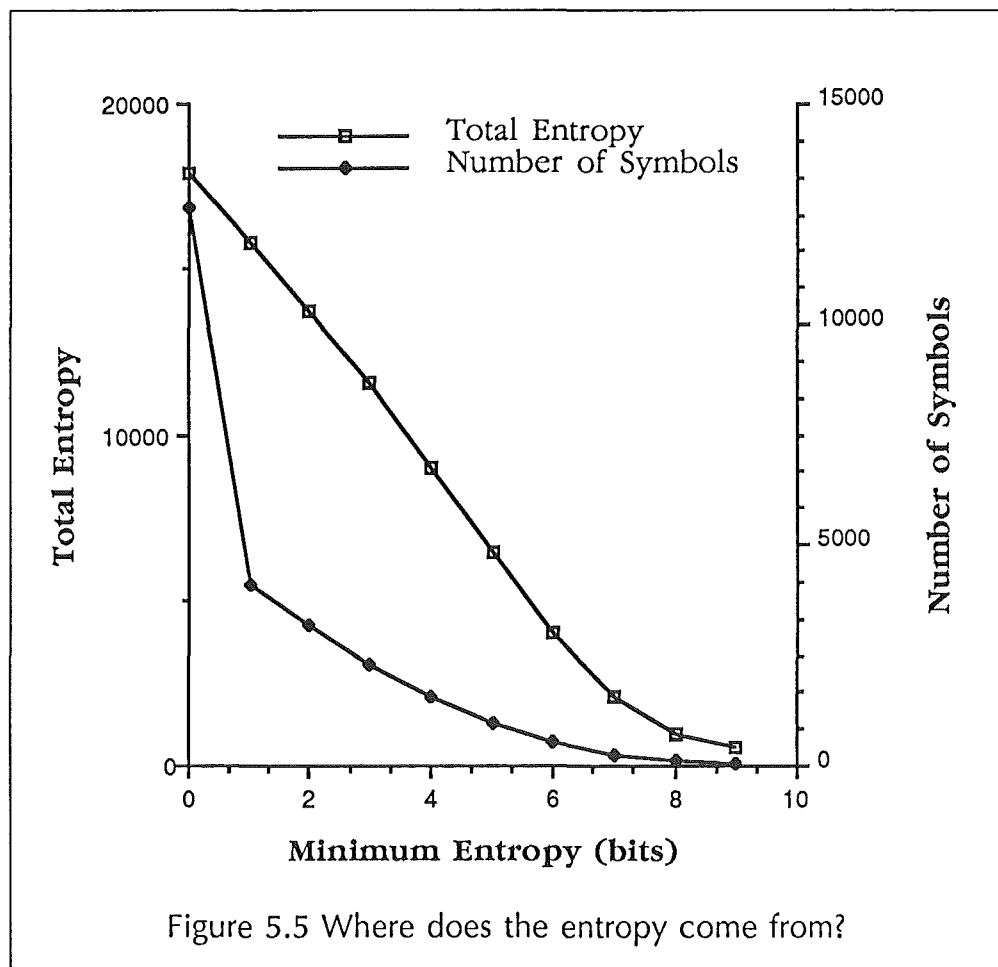
In the second case, the word to synonym ratio has to be re-optimised. It became evident after all this testing that the amount of extra compression involved in these choices was negligible, so the most straightforward option, where weightings are independent, was chosen.

Next, some attention needs to be paid to the position of a candidate word or synonym in a verse. If exactly one word or exactly one synonym is matched completely, the model should assume that the position of that word is the current position in the text. If such a position is found, each candidate for the next word should be weighted according to its distance from the assumed position. If the next word, when completely read, is not matched exactly once, the assumed position is incremented, but the confidence that this is the correct position is reduced, to a point where after several unmatched words, it is no longer considered. In this way, if phrases are interchanged, the model can easily follow the appropriate path.

However, the improvement in compression when this was added was negligible — it is interesting to note that the very simplest of techniques yield large returns, while the more sophisticated ones yield only small improvements.

Using the techniques described so far, the small test file was around 35% smaller than the file compressed using just PPMC. To increase the compression further, it was necessary to find what made up the bulk of the file — was it the unexpected characters which take many bits to encode, or was it the more predictable characters which if improved slightly would bring the average down significantly. Entropy for symbols which were encoded in more than nine bits were summed, then the entropy for those encoded in more than eight bits, and so on. The results are summarised in figure 5.5. About half of the entropy is made up by those symbols encoded in more than 4 bits. The number of

symbols this represents about one seventh of the symbols in the target file. A large part of the size of the compressed file was coming from the larger entropy symbols.



Looking more closely at the file, it was evident that a lot of the bits were being generated on the first character of a word. This is consistent with the results from the competition — most of the losses in compression are made at the start of the word, as the later characters in a word are usually predicted highly accurately. To predict the first character of a word, the PPMC model has to look at the contexts where the last letter was a space, so has no more idea about the next word than the parallel model. The parallel model should have some idea of the possible first letters of a word; if the position is correct, it should give a very good indication. Using the parallel model to generate a distribution for the first character of a word, and after tuning had been performed, it was found that its predictions were worth a weighting of about 60%, and reduced the size of the file to 2089, about 40% better than PPMC.

Up to this point, the model did not *learn*; the model was a static one, which did not adapt to the text. What would be

desirable is if a word corresponds to *itself* often, to record this, but if it often corresponds to some synonym, then that connection should also be recorded. As the model goes along, then, it may learn for example that the word 'food' in one text corresponds to the word 'nourishment' in the other. In this way, it should improve its predictions as it gets further through the file.

This was implemented using two hash tables of words; one containing counts for the number of times a word corresponds to itself, and the other containing references to common synonyms. When a full target word has been processed, any connections found are noted (in fact, at the same time as the position is found). As words are searched, common synonyms have the same priority (i.e. other synonyms will be searched only if all the words and the common synonyms have been exhausted). This produced no real increase in compression, but would be useful on more demanding files.

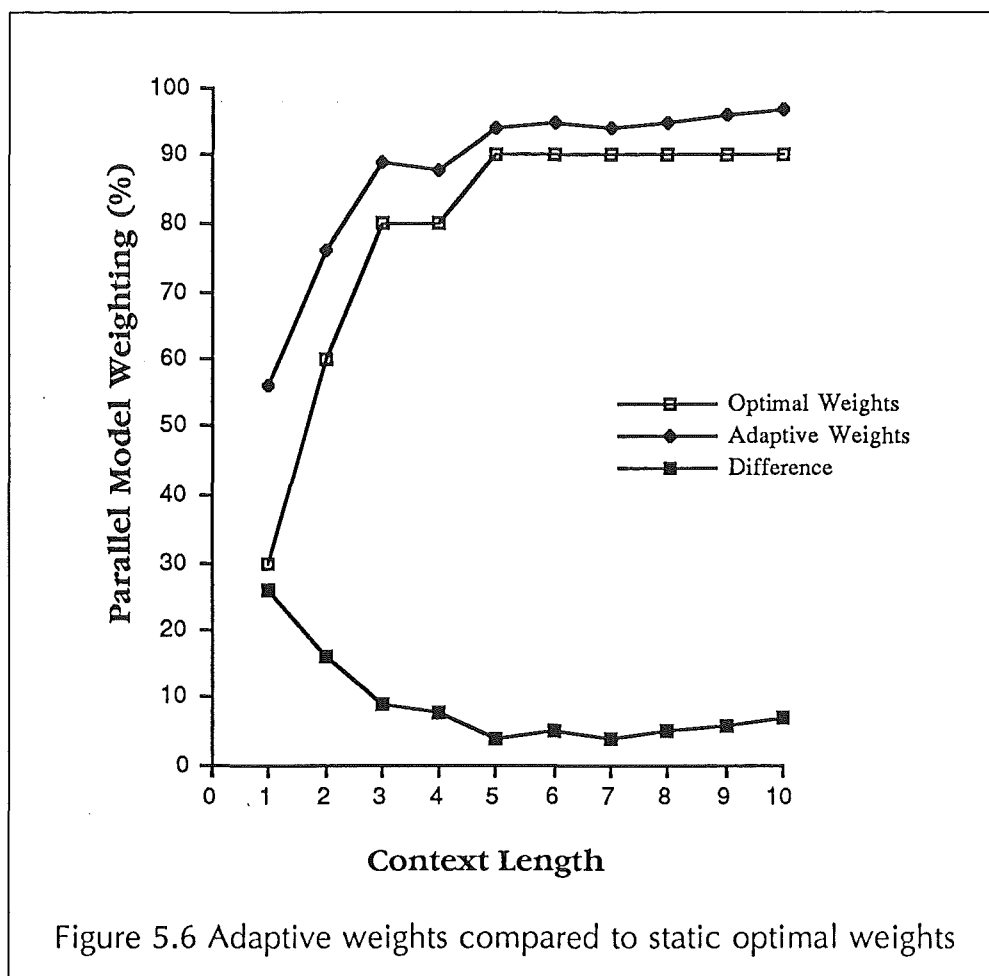
This discussion has described the increase in compression on one file, the first one hundred verses of Genesis, with the RSV as the source and the KJV as the target. This file is probably near a best case because of stylistic proximity as described above, and also by virtue of its length; PPMC takes a while to start compressing well, so the comparison is somewhat unfair. However, the principles gleaned from this file are useful, as at each point, a larger 116679 kilobyte file was also compressed to make sure that the approach used and the parameters chosen were valid over a wider range of text. The final figure for the improvement over PPMC on the longer text was 31%.

The tuning done on the weights for the parallel model should ideally be done dynamically. While the compression has been shown to be relatively insensitive to the value of the weighting near the optimum, some texts may be of quite a different nature. To achieve this, the program keeps a tally of the number of bits that each model would have used to encode each symbol if left on its own. A separate tally is kept for each context length. Weights are calculated by:

$$\text{parallel_weight} = \frac{\text{ppm_entropy}}{\text{ppm_entropy} + \text{parallel_entropy}}$$

$$\text{ppm_weight} = 1 - \text{parallel_weight}.$$

The model with the least entropy so far will therefore get the biggest weight. This, however, does not produce optimal weights. On the small file, the weights were generally higher than they should have been (see figure 5.6). The graph shows the optimal weights, the weights generated by the entropy ratios, and the difference between them. The size of the compressed file using this technique was 2369, about 300 bytes more than the best with optimal weights, so some better adaptive confidence function needs to be developed.



One other technique was tried where the model with the lowest entropy for all the characters was weighted more, where the entropy of a set of probabilities for symbols is

$$H(\text{distribution}) = -\sum_s p(s) \cdot \log_2 p(s).$$

This performed badly, probably because the parallel model is usually very bold about choosing one symbol and assigning other symbols 0 probability — an effectively infinite entropy. A special log function where $\log 0 = 0$ had to be constructed, but this is unrealistic, producing an invalid measure. The size of the compressed file under this scheme was about 2600 bytes.

This was as far as the tuning of the model went. Due to the time taken to acquire appropriate texts, and the fact that the desperation level had to rise considerably before the attack on the thesaurus was attempted, there was not enough time to implement other improvement possibilities. Also, the function of results versus effort becomes very flat after a certain amount of effort has been put in; it was already evident that the more sophisticated additions to the model were yielding minimal results, and that the very simplest techniques were the most effective.

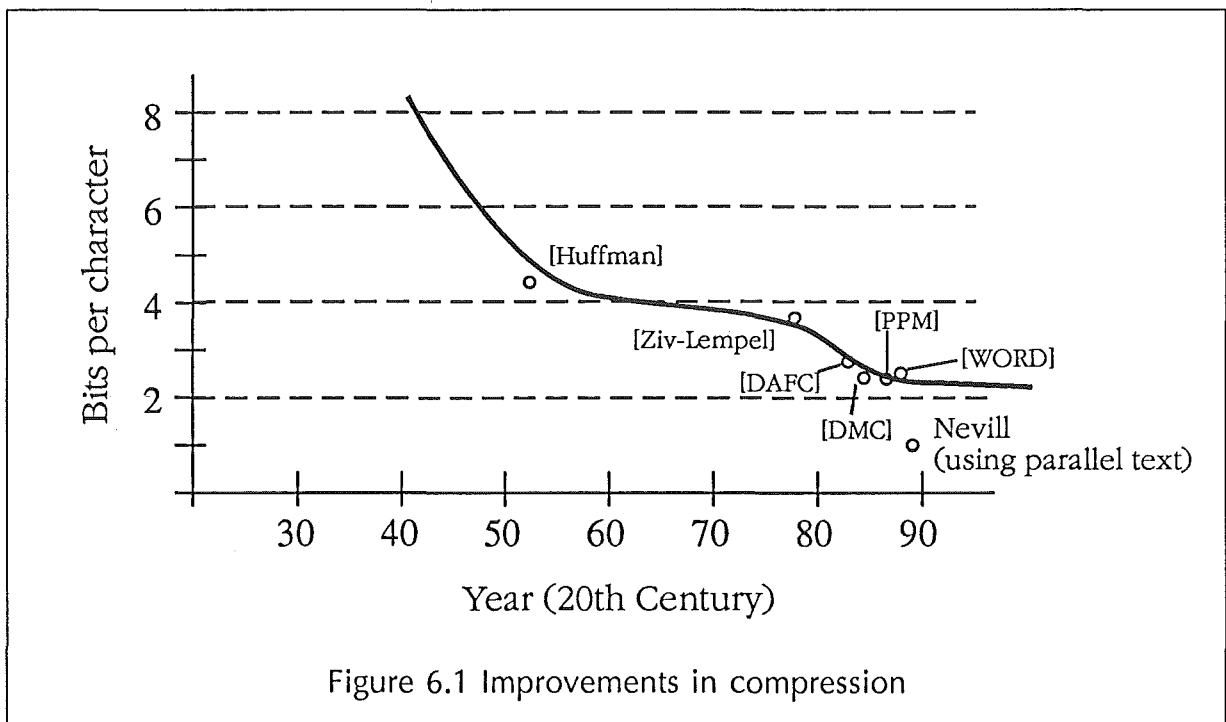


6 Conclusion

Parallel compression has been shown to be feasible, and the results from techniques implemented so far are very encouraging. The scenario described at the beginning of this report is not without its possibilities of fulfilment, and as the world becomes more and more globalised, linguistically orientated algorithms like these will become more and more important.


It seems that parallel texts can make an improvement of up to 50% on the entropy of the predictions of humans, and up to 40% on automated prediction. There is obviously a fair amount of information in a text quite apart from the raw semantic information — as this is more or less given to the subjects it should not be encoded, but it only seems to account for 50% of the information. The rest is style, vocabulary, and subtle changes in connotation.

A general rule for compression is that achieving a small amount of extra compression is harder and harder as the compression gets better.



Coding a 28-symbol alphabet in eight bits requires virtually no work. Packing the it into five bits per character is somewhat more difficult. With Huffman codes, three to four bits per character is achievable. The most sophisticated general compression scheme in the world manages about two bits per character, the relatively sophisticated techniques using a parallel text described here

can almost cut this in half again, to 1.2 bits per character, and to go lower requires a *much* greater amount of work. Figure 6.1 shows that breaking the 2 bits per character barrier is a considerable achievement.

Apart from developing and tuning a highly effective compression method for parallel texts, this project has made additions and improvements to the gambling method of estimating entropy, developed a practical system for mounting a cryptographic attack on a compressed file, experimented with the deduction of synonyms from a pair of parallel texts and constructed a framework for blending two prediction models, which will be useful in the future for sophisticated Artificial Intelligence based compressors. 

7 References

- [Moffat] Moffat, A.(1988) "A data structure for arithmetic encoding on large alphabets" *Proc 11th Australian Computer Science Conference*, 309-317, Brisbane, Australia, February.
 "A note on the PPM data compression algorithm" Research Report 88/7, Department of Computer Science, University of Melbourne, Parkville, Victoria 3052, Australia.
- [Miller] Miller G.A. (1956) "The magical number seven, plus or minus two: some limits on our capacity for processing information" *Psychological Review*, 63 (2) 81-97, March
- [Shannon] Shannon, "Prediction and entropy of printed English" *Bell System Technical J*, 50-64, January.
- [Cover] Cover, T.M. and King, R.C. (1978) "A convergent gambling estimate of the entropy of English" *IEEE Trans Information Theory*, IT-24 (4) 413-421, July.
- [Zunde] Zunde, P. and Kelman, D. 1985 "An Information-theoretical metric for testing program comprehension" Georgia Institute of Technology, August.
- [Huffman] Huffman, D.A. (1952) "A method for the construction of minimum-redundancy codes" *Proc Institute of Electrical and Radio Engineers*, 40 (9) 1098-1101, September.
- [WORD] Moffat, A. (1987) "Word based text compression" Research Report, Department of Computer Science, University of Melbourne, Parkville, Victoria 3052, Australia.
- [Ziv-Lempel] Ziv, J. and Lempel, A. (1978) "Compression of individual sequences via variable-rate coding" *IEEE Trans Information Theory*, IT-24 (5) 530-536, September.
- [PPM] Cleary, J.G. and Witten, I.H. (1984) "Data compression using adaptive coding and partial string matching" *IEEE Trans Communications*, COM-32 (4) 396-402, April.
- [DMC] Cormack, G.V. and Horspool, R.N. (1987) "Data compression using dynamic Markov modelling" *Computer J*, 30 (6) 541-550, December.
- [DAFC] Langdon, G.G. and Rissanen, J. (1983) "A double-adaptive file compression algorithm" *IEEE Trans Communications*, COM-31 (11) 1253-1255, November.




Appendix A: Quantisation of Betting Levels

When the betting program was being rethought, the possibility of having eight buttons to select the probability bet with was investigated. The question was how much of an error would be introduced by this quantisation of the bets. The buttons would cover the same range as the slider — from breakeven to 95%. A program was written to take the log files from the subjects who had taken part in the competition, and modify their predictions to the closest graduation under the new system. Results are shown in table A.1.

Table A.1 Effects of quantising probabilities

| Person | Original Entropy | Entropy with modified betting | % difference |
|--------|------------------|-------------------------------|--------------|
| 1 | 0.904 | 0.844 | -6.60 |
| 2 | 3.42 | 2.62 | -23.00 |
| 3 | 0.921 | 0.904 | -1.80 |
| 4 | 1.746 | 1.537 | -12.00 |
| 5 | 1.093 | 1.106 | +1.10 |
| 6 | 1.300 | 1.267 | -2.50 |
| 7 | 0.708 | 0.723 | +2.10 |
| 8 | 1.008 | 1.006 | -0.20 |
| 9 | 1.477 | 1.475 | -0.14 |
| 10 | 0.477 | 0.453 | -5.00 |

It is interesting to note that most people's entropy decreased — probably because they were constrained to sensible choices.

The program can be easily modified to simulate any function for the values of the buttons, so has the potential to be quite a useful tool for experiment design. 

Appendix B: Implementation of Synonym Extraction


The code was originally written fairly simply, as the data structures were complicated enough to get right on their own, but it was soon apparent that the linear search that the program used to find a word was grossly inefficient, and that hashing was probably a better way of implementing the search. The hash function is fairly simple, hashing into an array of 59 999 pointers, with collisions handled by chaining. Each word has a linked list of potential synonyms attached to it. Each word node contains information on its parts of speech, the number of times it occurred in total, the number of times it occurred in the same chunk in both versions and the number of times it occurred in the target version. Each synonym node held the number of times this synonym had occurred with the word.

Despite the best attempts to keep the memory usage down, it grew quickly to about six or seven megabytes. As long as there is enough actual memory for the program, the algorithm is very fast. However, as soon as parts of the program data space starts being paged out, the random access to data meant that performance decreases markedly. In fact, the first fifth of the New Testament could be processed in a few minutes, while the last four fifths would take all night, with the paging disk in constant use.]

One problem with the running time even on small files was that the selection of the validity criteria had to be done more or less by trial and error, and the constant re-compiling and re-running to ascertain whether the criteria were good was slow. It was decided to make the main program to output *everything* held in the data structures, and to write a program to post-process this information. This post-processing was very fast and allowed fine-tuning of the criteria.

This development led the way to the final solution to the memory problem. As there were now procedures for writing the data structure to disk and re-parsing it into the same data structure for post-processing, the main program could be modified to operate in the following way: While the program was running, it would monitor its resource usage, in particular the number of page swaps that had been performed on its behalf. It was noted that after around one hundred page swaps had been performed, the performance went downhill *extremely* fast. When one hundred page swaps had been performed, the program would print out all of its results where a synonym had occurred more than once. It would then read in the rest of the input and write it to a temporary file, start up another instantiation of itself

and exit. The new version of itself would start up, notice the presence of the results, and the temporary input file, and take its input from these.

The net result was that each time the performance was downgraded, the program would throw out all of those correspondences that had only occurred once, and reclaim a large amount of memory. This, of course, could result in the loss of information. For example, if the the program restarted five times (typical for the New Testament), correspondences that occurred up to five times could be lost. However two words that correspond only five times are not particularly significant. 

Appendix C: Synonyms Deduced from Parallel Texts

| King James Version | Today's English Version |
|--------------------|-------------------------|
| box | jar |
| box | alabaster |
| pit | abyss |
| pit | sunlight |
| pit | furnace |
| woe | horrible |
| scarlet | pearls |
| bade | host |
| band | regiment |
| bind | belt |
| colt | untying |
| corn | cornfields |
| crow | crows |
| crow | cock |
| crow | tonight |
| hail | plague |
| juda | judah |
| loss | reckon |
| lump | dough |
| lump | batch |
| reed | rod |
| reed | measuring |
| salt | friendship |
| salt | salty |
| salt | saltiness |
| rent | tore |
| rent | patch |
| pull | speck |
| sion | zion |
| vial | bowl |
| west | north |
| west | harbour |
| west | phoenix |
| wood | bronze |
| despiseth | rejects |
| despiseth | listens |

| Today's English Version | King James Version |
|-------------------------|--------------------|
| perfume | box |
| bag | purse |
| bag | scrip |
| buy | eyesalve |
| mob | legion |
| mud | clay |
| rid | malice |
| sow | sowest |
| grumbling | murmured |
| affection | kindness |
| mustard | grain |
| lightning | lightnings |
| lightning | thunderings |
| purified | salted |
| sandals | unloose |
| sandals | latchet |
| benjamin | tribe |
| benjamin | sealed |
| removed | vail |
| abel | cain |
| adam | adams |
| baby | babe |
| cain | slew |
| calf | fatted |
| bent | stooped |
| coat | agreeth |
| bowl | vial |
| cold | hot |
| fort | castle |
| noah | ark |
| noah | noe |
| pigs | swine |
| ripe | harvest |
| riot | murder |
| ship | boards |
| ship | swim |

P A R A L L E L T E X T C O M P R E S S I O N

| | |
|--------------|-----------|
| tetrarch | abilene |
| covetous | greedy |
| abound | increased |
| robbers | dangers |
| bottles | wineskins |
| bottles | fresh |
| bottles | ruined |
| bottles | skins |
| bottles | burst |
| candle | lampstand |
| candle | lights |
| singing | hymns |
| beware | insist |
| sinneth | continues |
| castle | fort |
| choked | thorn |
| eateth | refuses |
| putteth | skins |
| burdens | loads |
| brotherly | affection |
| murmured | grumbling |
| double | grief |
| double | queen |
| double | luxury |
| hateth | hates |
| flower | flowers |
| covetousness | greed |
| leaven | dough |
| leaven | batch |
| steward | manager |
| makest | invite |
| drunken | drunk |
| mayest | nakedness |
| mayest | advise |
| committeth | commits |
| solomons | porch |
| tribute | taxes |
| ungodly | godless |
| stripes | whipping |
| pearls | scarlet |
| elisabeth | elizabeth |
| perils | dangers |
| perils | seas |
| perils | wilds |
| perils | cities |

| | |
|-------------|--------------|
| ship | stuck |
| ship | forepart |
| ship | aground |
| ship | seas |
| ship | mainsail |
| ship | hoised |
| ship | rudder |
| ship | creek |
| sing | singing |
| skin | lepers |
| roof | housetop |
| sows | reapeth |
| west | north |
| west | haven |
| priscilla | aquila |
| pontius | tetrarch |
| teachings | doctrines |
| upstairs | upper |
| upstairs | chamber |
| citizen | roman |
| travelling | journeyed |
| babies | suck |
| robbers | perils |
| collector | publican |
| disease | leprosy |
| discouraged | faint |
| animal | strangled |
| bitter | wormwood |
| earned | gained |
| aquila | priscilla |
| closed | straitened |
| bushes | thorns |
| bushes | choked |
| dreaded | lepers |
| driving | followeth |
| thunders | uttered |
| grapes | winepress |
| greedy | extortioners |
| springs | fountains |
| cursing | railing |
| weapons | holds |
| weapons | pulling |
| weapons | warfare |
| solomons | porch |
| locked | shut |

P A R A L L E L T E X T C O M P R E S S I O N

| | |
|------------|------------|
| perils | danger |
| perils | robbers |
| perils | floods |
| perils | travels |
| drinketh | drinks |
| whatsoever | lovely |
| bottomless | abyss |
| sealed | benjamin |
| fragments | baskets |
| tanner | leather |
| begat | famous |
| eaten | swim |
| eaten | aground |
| thrice | crows |
| thrice | tonight |
| burst | wineskins |
| burst | skins |
| haste | hurried |
| haste | zacchaeus |
| durst | dared |
| goats | bulls |
| grain | mustard |
| soweth | reaps |
| meats | foods |
| meats | stomach |
| jonas | jonah |
| jonas | nineveh |
| hymns | psalms |
| iniquity | entirely |
| penny | wages |
| olive | cultivated |
| north | west |
| swords | clubs |
| porch | solomons |
| winter | harbour |
| winter | phoenix |
| shore | swim |
| shore | aground |
| sleep | sleepier |
| purse | bag |
| smite | strike |
| smoke | darkened |
| songs | psalms |
| songs | hymns |
| tribe | benjamin |

| | |
|------------|-------------|
| palace | hall |
| exactly | double |
| godless | ungodly |
| plague | hail |
| emperors | caesars |
| tables | overthrew |
| psalms | songs |
| psalms | hymns |
| sickle | ripe |
| greeted | saluted |
| severe | severity |
| abyss | pit |
| abyss | bottomless |
| simons | wifes |
| cheap | vinegar |
| purify | cleanse |
| bowls | vials |
| bulls | goats |
| laodicea | laodiceans |
| burst | bottles |
| enoch | translated |
| curse | hangeth |
| spring | fountain |
| immortal | immortality |
| hymns | singing |
| hymns | songs |
| hymns | psalms |
| judah | sealed |
| judah | tribe |
| mixed | mingled |
| prostitute | whore |
| prostitute | harlot |
| loses | salt |
| loses | lose |
| olive | wert |
| olive | fatness |
| swords | staves |
| seems | seemeth |
| porch | solomons |
| sends | saluteth |
| prize | fatted |
| owner | goodman |
| tenth | tithes |
| purse | scrip |
| queen | double |

P A R A L L E L T E X T C O M P R E S S I O N

| | |
|-------------|-------------|
| tribe | zebulun |
| wheat | swim |
| wheat | aground |
| swine | pigs |
| swine | herd |
| upper | upstairs |
| yield | purposes |
| melchisedec | melchizedek |
| kneeled | knelt |
| teachest | steal |
| kindness | affection |
| multiplied | larger |
| countrymen | dangers |
| bringeth | treasure |
| beckoned | motioned |
| immortality | immortal |
| infirmities | weaknesses |
| remission | demonstrate |
| | |
| | |
| | |
| | |

| | |
|------------|-------------|
| thorn | thorns |
| thorn | choked |
| spend | haven |
| wages | penny |
| ruler | abilene |
| ruler | lysanius |
| ruler | trachonitis |
| ruler | ituraea |
| storm | tempest |
| untie | unloose |
| untie | latchet |
| yeast | leaven |
| approval | experience |
| manager | steward |
| marries | committeth |
| welcomes | receiveth |
| listens | despiseth |
| outcasts | publicans |
| ability | kinds |
| ability | discerning |
| motioned | beckoned |
| collectors | publicans |



Appendix D: Random Parallel Texts

It is possible to use a Thesaurus to *generate* parallel texts, by replacing words with random synonyms. What follows is the first few verses of Mark, from the Bible.

this is the interest programs apropos jesus christ the boy of favorite

(This is the good news about Jesus Christ the Son of God)

it began during the predictor isaiah had written into evacuate the style for you

(It began as the prophet Isaiah had written to clear the way for you)

someone is shouting influence the wild obtain the approach apt for the potentate force each aboveboard road for him opposite itinerate

(etc...)

so potty appeared authority the barren baptizing additionally pass detour missing from your sins likewise endure baptized he told the help likewise idol adjudicate reprieve your sins

multitude folks from the entity of judaea further the municipality of jerusalem went extinguish into determine toilet they confessed their sins likewise he baptized them current the tributary jordan

lavatory wore clothing made of camels hair and some leather hit spiral his waist besides his entree was locusts moreover rambling honey

he announced into the breed the entity who impart arrive succeeding me is glut largest than i am i am not respectable amply smooth clear turn destroy too unclasp his sandals

i dedicate you colony only he wish nickname you furthermore the celestial elan

not prolonged afterwards jesus came from nazareth fashionable the sphere of galilee likewise was baptized on head current the jordan

whilst urgently while jesus came increase put out of the field he saw utopia preparatory moreover the individuality approach overthrow nearby him enjoy one dove

as well as several say came from ecstasy you are my individual exorbitant junior i am cheerful amid you

nearby erstwhile the heart made him stab to the pirouette

where he stayed forty days organism tempted adjacent demon
acknowledge animals were there in addition nevertheless
angels came with helped him

in pursuit we had been state control cooler jesus went waste
galilee as well preached the account data from divinity

the advantage long time has occur he said as well as the domain
of idol is aside reel gone from your sins with comprehend
the sake soaps

while jesus walked along the buoy of pond galilee he saw two
fishermen simon as well as his brother andrew
communicable angle within each gain 