Computer Correction of Real-word Spelling Errors in Dyslexic Text

Jennifer Pedler

Birkbeck, London University

2007

Thesis submitted in fulfilment of requirements for degree of PhD

Abstract

This thesis describes the development, implementation and testing of a program to detect and correct real-word spelling errors made by dyslexics. These errors – which occur when a user mistakenly types a correctly spelled word when another was intended – cannot be detected without the use of context and so are largely ignored by spellcheckers which rely on isolated-word dictionary look-up.

The method used here to detect and correct real-word errors is to identify sets (often pairs) of words that are likely to be confused, such as *loose* and *lose*, and then, when encountering one of the words (say *loose*) in the text being checked, to determine whether the other one (*lose*) would be more appropriate in the context.

The first stage of the program uses an extended part-of-speech tagger to consider words that differ in their parts-of-speech. This required, among other things, the substantial enhancement of an on-line dictionary. Decisions for words that have the same parts-of-speech are left for the second stage which uses semantic associations derived from WordNet. Much previous research using the confusion set approach has been limited to a small number of 'commonly confused' words and has used artificial test data. In contrast, the work presented in this thesis uses several thousand sets of confusable words and a large corpus of actual errors produced by dyslexics.

Contents

List of Tables

List of Figures

Declaration

This thesis it the result of my own work, except where explicitly acknowledged in the text.

Signed ....

Jennifer Pedler, May 2007

Acknowledgements

My greatest thanks go to my supervisor, Dr Roger Mitton, for his patience, meticulous eye for detail and keen interest in my work – and also for not giving up on me, even in the darkest hours. Without his encouragement I would never have started, let alone finished. I would also like to thank my second supervisor, Dr Trevor Fenner, for his insightful comments on the algorithmic and mathematical aspects of this work.

Other colleagues have also encouraged and supported me along the way, in particular Dr Jason Crampton who reviewed Chapters 8 and 9 and Janet Billinge who thoroughly proofread the early chapters.

Thanks also to all contributors to the error corpus and to George Mitton for .keying in additional dictionary entries and marking-up the errors in the corpus used for the final evaluation of my program.

All my family and friends will no doubt be relieved that I have finished at last. I'll be able to see more of you all now and we'll have to find something else to talk about! Particular thanks to my daughter Jade who kindled my initial interest in computer spellchecking and to .Jasmine for all her advice, encouragement and great meals.

Introduction

Most word processors have a built-in spellchecker that highlights misspelled words in some way and offers the facility to correct these misspellings by selecting an alternative from a list. To detect these misspellings in the first place, most spellcheckers take each word in a text in isolation and check it against the words stored in a dictionary. If the word is found in this dictionary it is accepted as correct without regard to its context. Although this approach is adequate for detecting the majority of typos, there are many errors that cannot be detected in this way. These are referred to as *real-word errors* – correctly spelled English words that are not the word the user intended. Take, for example, the following extract from a rhyme that has appeared in many variations – almost every word is misspelled but since they are real-word errors, none is highlighted by the spellchecker.

> I halve a spelling chequer
>
> It cam with my pea see
>
> Eye now I've gut the spilling rite
>
> Its plane fore al too sea
>
> ...
>
> Its latter prefect awl the weigh
>
> My chequer tolled mi sew.

Dyslexia is estimated to affect about 10% of the population in some form, about 4% severely, according to research reported on by the British Dyslexia Association (BDA). It is not just a problem with spelling, nor is it simply a difficulty with learning to read. The BDA definition is:

> "Dyslexia is a combination of abilities and difficulties which affect the learning process in one or more of reading, spelling and writing. Accompanying weaknesses may be identified in areas of speed of processing, short term memory,

organisation, sequencing, spoken language and motor skills. There may be difficulties with auditory and / or visual perception. It is particularly related to mastering and using written language, which may include alphabetic, numeric and musical notation."

(Tresman, 2006)

Dyslexia cannot be 'cured', nor is it something that children 'grow out of'. It is a lifelong disability but there are strategies and aids that can help to mitigate the difficulties. Among these aids is the computer spellchecker.

Until my dyslexic daughter began to use a word processor, her written work was often illegible as well as being poorly spelled. Word processing improved the legibility but spelling was still a problem; even after using the spellchecker a large amount of manual intervention was required to get it right. Although non-word errors were detected, they were often not corrected; in some cases the spellchecker simply made no suggestions whereas at other times it produced a long list of words, none of which were the word intended. This latter case often resulted in the introduction of real-word errors – for example, she was writing about a room that was *uninhabitable* (which, not surprisingly, she misspelled); the computer suggested, and she accepted, *inedible*. In addition to these real-word errors that were introduced through the correction process, there were others that had occurred when her misspelling of a word resulted in another dictionary word such as (with the intended word in parentheses) *fowling* (*following*), *mad* (*made*) and *quit* (*quite*). Errors such as these were simply ignored by the spellchecker. It was this experience that led me to develop my interest in computer spellcheckers, in particular how they could be improved to meet the needs of dyslexic users. Earlier research (Pedler, 1996, Pedler 2001a) showed that a large proportion of the errors that went uncorrected by commercial spellcheckers were real-word spelling errors. This is the problem that I have addressed in this research.

Since the 1980's, when my daughter was assessed, moves toward widening participation in education and a greater emphasis on accessibility and inclusivity have led to a wider acceptance of dyslexia as a disability – despite a recent report that at least one LEA still refuses to use the term *dyslexia* (Hill, 2005). Improving facilities for disabled users often has the effect of making an improvement for all users (as demonstrated in work to make library signage more appropriate to the needs of dyslexics (Nancholas, 2005), for example). Similarly, improvements in the handling of real-word errors are likely to benefit word-processor users in general.

The focus of the research presented in this thesis is the engineering of a spelling checker/corrector to deal with real-word spelling errors in text written by dyslexics. Therefore it does not cover the educational literature in this field. The thesis addresses the problem in the context of previous research into the detection and correction of real-word spelling errors in text corpora and does not examine more general error detection/correction literature in computer science and its application to such tasks as optical character recognition or machine translation.

Chapter 1: Computer Spellcheckers and Dyslexic Misspellings

Real-word spelling errors are errors that occur when a user mistakenly types a correctly spelled word when another was intended. Errors of this type generally go unnoticed by most spellcheckers as they deal with words in isolation, accepting them as correct if they are found in the dictionary and flagging them as errors if they are not. This approach would be sufficient to detect the non-word error *peice* in “I’d like a *peice* of cake” but not the real-word error *peace* in “I’d like a *peace* of cake.” To detect the latter, the spellchecker needs to make use of the surrounding context such as, in this case, to recognise that *piece* is more likely to occur than *peace* in the context of *cake*. Such real-word errors are a nuisance to everyone who uses a spellchecker but they present a particular problem for dyslexics who make more and worse errors than the average user (Mitton, 1987) and also have poorer proof-reading skills.

Kukich (1992), in a comprehensive survey of spellchecking techniques, claimed that "Developing context-based correction techniques has become the foremost challenge for ... error correction in text". The fact that her paper remains the definitive survey is perhaps indicative of the small amount of progress that has been made in the last decade or so. The research described in this thesis takes up this challenge. It considers both syntactic and semantic approaches to the problem and assesses their performance when applied to real-word errors produced by dyslexics. This chapter sets the problem in context. It looks at the development of spellchecking technology from its origins in the 1960s and then examines the shortcomings of currently available commercial spellcheckers when run over samples of dyslexic text.

1.1 Spellchecking techniques

Word-processors are now widely used by the general population whose spelling abilities range over a continuum from excellent to poor. For those at the lower end of this spectrum, many of whom may be dyslexic, the spellchecker has become an essential tool. However, although user requirements have evolved, the technology has done little to keep pace; today's spellcheckers still rely heavily on findings from research dating back to the 1960s.

Before examining the ways in which these early techniques are integrated into current spellcheckers, it is useful to consider the spellchecking process: The term 'computer spellchecker' is now widely taken to encompass both error detection and error correction although, strictly speaking, a *checker* is a program that detects errors which are then passed to a *corrector* that produces a list of suggestions. For some applications it may be sufficient for the program to flag potential errors which can then be manually corrected by the user, but today we generally expect a spellchecker not only to tell us that we have made a mistake but also to suggest how we can correct it. Spellcheckers that (attempt to) perform both tasks are incorporated into all widely used wordprocessors and today often built in to many other text entry programs such as email software.

For non-word errors the detection phase initially appears to be a simple matter of looking up each word in a word-list or dictionary and flagging any that are not found as potential misspellings. But this is not as straightforward as it seems at first. Its efficiency depends largely on the dictionary being used; if this includes too few words many correctly spelled but less common words will be questioned; if it contains too many words the real-word error problem will be exacerbated as misspellings of common words which happen to result in a rare word included in the computer's dictionary will go undetected. Spellchecker dictionaries are discussed further, briefly in section 1.2.4 below and in more detail in Chapter 4. Since most spellcheckers use isolated word look-up to detect errors, real-word errors are largely ignored as shown in the next section.

After a potential error is spotted the spellchecker's next task is to produce a list of proposed corrections. How should it go about this? Intuitively we would expect the error to be similar in some way to the intended word and this is confirmed by early studies of, mostly, typing errors, which found that over 80% of misspellings contain just one error – a wrong, missing or extra letter or a single transposition (Damerau, 1964; Pollock & Zamora, 1984). The method for correcting this type of error (described in detail by Peterson, 1980) is straightforward to implement and has underpinned much subsequent development.

Another useful early finding was that the first letter of a misspelling is correct in the majority of cases. (Yannakoudakis & Fawthrop, 1983; Pollock & Zamora, 1984.) Making this assumption greatly reduces the number of words that have to be considered as possible candidates for correction.

These methods are the basic techniques used by early spellcheckers which were designed for specific applications, such as the correction of errors in scientific databases. They were also adequate when word-processing moved into the office and spellcheckers began to be used to correct less restricted text. It is more accurate to describe their purpose in this context as the correction of mistypings rather than misspellings as the majority of typos produced by a reasonably accurate speller are likely to be one of the simple error types introduced above. Dyslexic misspellings are not so easy to categorise. Although they do generally bear at least some resemblance to the intended word, the exact relationship may sometimes be rather obscure as can be seen from some of the examples in the next section.

1.2 Spellcheckers and dyslexic spelling errors

To assess the performance of currently available commercial spellcheckers on 'dyslexic' text, I ran four spellcheckers[1] over three samples of text chosen as representative of the types of spelling error made by dyslexics (Pedler, 2001a). This section summarises the findings presented in that paper.

1.2.1 Samples of dyslexic text

The samples used for this initial research comprised a total of 3134 words including 636 spelling errors. Some of these errors occurred more than once but as the spellcheckers dealt with them in the same way on each occasion they were only counted once. Removing these duplicates left a base

[1] Three wordprocessors - MS Word, Corel Word Perfect, Lotus WordPro - and a dedicated spellchecker aimed at dyslexics - TextHelp Read & Write.

total of 577 distinct errors. This text formed the initial contribution to the dyslexic error corpus which is described in detail in Chapter 3.

Here is a short passage from each of the samples:

> I think I did well *becoser* I got *of* to a good *stare* and I have almost *finsder* my booklet and I have done a *fuwe peturs* on the computer and now I am doing a *couver*. My book is on *submaren*. I have had to do a *pituwer* of how a *submaren* works. I *havent* done *alot* of good work but I think I have done well.
>
> Sample 1.
>
> The cat working in the mill *spys* a *moues* feeding on corn. The cat *scillfully creps* up behind a sack and all of a *suden* his *musirls* are *tens* he *spring* and a little *squick* from the mouse I *herd* as the *cats clors sunck* deep into the mouse the cat *quilly ete* it and then *cerled* up on a sack to *slip*.
>
> Sample 2.
>
> There *seams* to be some *confusetion*. *Althrow* he *rembers* the *situartion*, he is not clear on *detailes*. With regard to *deleteing* parts, could you *advice* me of the *excat* nature of the *promblem* and I will *investgate* it *imeaditly*.
>
> Sample3.

Sample 1 is a collection of word-processed homework (saved before it was spellchecked) produced by my dyslexic daughter when she was in her third year of secondary school. Roger Mitton supplied me with the other two samples which he used for a comparative test of spellcheckers described in Mitton (1996). Sample 2 is made up of short passages of creative writing produced by secondary school children of low academic ability in the 1960s. These were originally taken from *English for the Rejected* (Holbrook, 1964). Dyslexia was not widely recognised at this time but it is likely that many of these poorer spellers would be classed as dyslexic given today's assessment procedures. Sample 3 consists of a number of misspellings from office documents (all produced by the same person).

1.2.2 Types of error

A simple error (Damerau, 1964) differs from the intended word by only a single letter. The four possibilities can be seen in the passages above: substitution – *cerled (curled)*, insertion - *couver (cover)*, omission - *investgate (investigate)* and transposition - *excat (exact).* Damerau found 80% of misspellings in the samples used for his research fell into one of these four simple error categories.

The errors in my initial samples present a different picture. Only 53% are simple errors. More than a third (39%) differ in more than one letter; I refer to these as multi-errors. Some errors, such as *submaren (submarine)*, closely resemble the intended word while others, such as *pituwer (picture)*, are not so obvious. The remaining 8% are word boundary infractions (run-ons and split words), which are special cases of omission and insertion errors. A run-on is the result of omitting a space, such as *alot (a lot).* A split word occurs when a space is inserted in the middle of a word, such as *sub marine.* These cause problems for a spellchecker because it treats spaces as word delimiters so a run-on will be treated as one word while a split word will be treated as two.

Misspellings that result in another dictionary word, real-word errors, are an additional complication. These may often be simple errors, such as *stare (start)* and *seams (seems)* from the examples above, but because the spellchecker will find them in its dictionary they will not be flagged as errors. Some real-word multi-errors in the samples, such as *no (know)* and *witch (which),* are homophones and could be the result of confusion over the correct spelling; others, like *fowling (following)* or *petal (petrol),* seem more like an unlucky coincidence. Some split words in the samples (like *sub marine*) resulted in two valid words but all the run-ons produced non-words.

The first letter was correct for 95% of the errors in my samples. This confirms the findings of Yannakoudakis & Fawthrop (1983) and Pollock & Zamora (1984) mentioned above. Several of the misspellings where the first letter was incorrect show confusion over vowel sounds e.g. *ete (ate)* and silent letters, e.g. *no (know)*.

The proportion of errors falling into each type in these samples is shown in Table 1.1

Total words	3134	
Total errors	636	
Distinct errors	577	
Simple errors	307	53%
Multi errors	223	39%
Word boundary errors	47	8%
	577	100%
Real-word errors	100	17%
Non-word errors	477	83%
	577	100%
First letter errors	30	5%

Table 1.1: Proportions of types of error in the initial samples

1.2.3 Spellchecker Performance

The error detection stage is the first point at which a spellchecker can fail. Spellcheckers, as discussed above, generally detect errors by looking the words up in a dictionary; words that are found in the dictionary are assumed to be correct and not flagged as errors with the result that misspellings that turn into another word are ignored. Words that are not found in the dictionary are assumed to be errors and passed to the checking process. The spellchecker can fail at this next point by not including the intended word in its list of suggestions. This means that there are three possible outcomes when a spellchecker is run over text containing errors. I have classed these as:

- Corrected – the error was flagged and the intended word was in the spellchecker's list.
- Detected – the error was flagged but the intended word was not in the list of suggestions.

- Not detected – the error was not found by the spellchecker.

Table 1.2 summarises the performance of the spellcheckers, when run over the initial samples. Each column of the table shows the proportion of errors falling into each of the categories above (as a range from the worst to the best spellchecker).

Column a shows that overall only about half of the errors were corrected and just over a quarter were detected (meaning that the user was at least alerted to the potential error) but the remainder (around 20%) went undetected. This correction rate falls approximately mid-way between the range of correction rates (30 - 73%) for spellcheckers (many specially designed for dyslexics) tested in work carried out at the University of Edinburgh (Nisbet et al. (1999)).

	a	**b**	**c**	**d**	**e**	**f**
	All errors	**Simple errors**	**Multi-errors**	**Run-ons**	**Split words**	**First letter errors**
Corrected	49 - 55%	73 - 79%	27 - 37%	0 - 58%	0%	27 - 40%
Detected	24 - 28%	2 - 4%	46 - 57%	42 - 100%	39 - 50%	27 - 37%
Not detected	19 - 20%	19 - 23%	17 - 19%	0%	50 - 61%	33 - 37%
Total errors (100%)	577	307	223	19	28	30

Table 1.2: Spellchecker performance on initial samples

Simple errors

In contrast to Damerau's finding (mentioned above) that 80% of the errors in his corpus were simple errors – a single insertion, omission, substitution or transposition - only slightly over half of the errors in my samples were of this type. Unsurprisingly, since most spellcheckers rely heavily on Damerau's findings, all the spellcheckers I tested performed best for errors of this type with a correction rate between 73% and 79% (column b). In general, with simple errors, if an error was detected, it was corrected; the failures were almost all in the 'not detected' category. As might be expected, the majority of these were real-word errors such as: *of (off), stare (start), tens (tense)* and *seams (seems.)* - all words that we would expect to find in the dictionary.

Multi-errors

Almost 40% of the errors in my samples contained more than one wrong letter. All the spellcheckers performed poorly for these multi-errors, correcting only around a quarter to a third of them. The main difficulty was not in detecting the errors in the first place but in correcting them once they had been detected. Around half were detected but not corrected but although there were differences between the spellcheckers, a large proportion of these multi-errors were not corrected by any of them. Some, such as *vrnegest (varnished)*, might reasonably be considered uncorrectable. Others look correctable - such as *futcher* (*future*) - but none of the spellcheckers managed to correct them. In some cases, such as *poticular (particular)* and *unforchanely (unfortunately)*, they made no suggestions at all. Although it seems surprising that they did not attempt corrections for errors such as these, this is perhaps preferable to producing a long list that does not contain the correct word.

Often the errors were phonetically similar to the intended word and a homophone of the correct word was suggested. For example, two of the spellcheckers managed to suggest *mussels* as a correction for *musirls* but failed to come up with the intended word – *muscles.* All the spellcheckers suggested *problem* as a correction for *probley (probably.)* This is a similar error to *rembers (remembers.)* Both errors show a common dyslexic tendency to omit repeated syllables in the middle of a word.

Word boundary errors

The small proportion of word boundary infractions (run-ons and split words) caused a lot of difficulty for the spellcheckers.

Word did not manage to correct any of the run-ons although the other spellcheckers managed 42 - 58% of them. In just over half of the run-ons the only error was an omitted space (e.g. *afew, allthat).* TextHelp (the spellchecker designed specifically for dyslexics) corrected all of these but at the expense of making some very odd suggestions for some of the other errors (such as suggesting *cu ms* as a correction for *cums* (*comes*)). Word stores a large number of run-ons in its AutoCorrect feature, which is separate from the spellchecker and intended merely to correct common typos. If you type *alot*, and AutoCorrect is switched on, it will automatically and silently be replaced by *a lot*. This might explain the spellchecker's failure to correct any of the run-ons in the samples; the designers perhaps decided that AutoCorrect would already have dealt with these and so the spellchecker need not attempt any of them.

None of the split words was corrected by any of the spellcheckers. Those where both halves resulted in a dictionary word were not flagged as errors, as might be expected. In some cases the only error was the inserted space (e.g. *sub marine, some one.)* In others there were additional changes, e.g. *throw out (throughout), where ever (wherever).* Word corrected the latter but only in a roundabout way; its grammar checker flagged it as a compound word and suggested correcting it to *whereever*. Accepting this replacement resulted in the word being flagged by the spellchecker which then gave *wherever* as its first suggestion.

The rest of the split words resulted in a non-word for one or both parts, e.g. *in adequacis (inadequacies*), *rey mebber (remember*.) The non-word parts were flagged (and in some cases 'corrected' e.g. *re* was suggested for *rey* and *member* for *mebber*) but, as a single-word correction was not produced, I classed these as not corrected.

First letter errors

About a quarter of the misspellings with an incorrect first letter were capitalisation errors; some were otherwise correctly spelled, e.g. *edith (Edith*), others had additional errors, e.g. *engleish (English*.) Most of these were corrected by the spellcheckers. One, *icoud (I could)* was also a run-on and not corrected by any.

A similar number were real-word errors and were not flagged by the spellcheckers, e.g. *are (our), bark (park), every (very.)*. Word's grammar checker, which does make some attempt to correct real-word errors (as discussed further below), successfully corrected *new (knew*).

Real-word errors

Undetected errors made up 19%-20% of all the errors (Table 1.2) and most of these were real-word errors. Even though real-word errors contribute only a minority of errors, they are a problem which needs addressing if spellcheckers are to achieve a reasonable correction rate. In some cases the proportion could be even higher than in my samples. Mitton (1996) estimates that real-word errors account for about a quarter to a third of all spelling errors, a high proportion of them occurring on function words (words like *the* and *of*).

Word’s grammar checker provides some facility for detection and correction of real-word errors. One of its features is a list of 90 commonly confused words (mainly homophone pairs) which are questioned when they seem to be used in the wrong context. Although it successfully corrected *new (knew)*, as mentioned above, it was not always successful. For example, it did not question the use of *advice* in error for *advise* in the phrase “... could you *advice* me ...” although *advice/advise* are included in its list of confusable words. It can also produce some strange suggestions as illustrated by the phrase “...dumps *witch* are wrong.” *Witch/which* is not included in the commonly confused words list but, using another rule, the grammar checker suggests that this is a subject/verb disagreement and should be corrected to *witch is* or *witches are*!

1.2.4 Dictionaries and false alarms

Rare words or specialist terms are a frequent cause of false alarms - correctly spelled words that are flagged as errors. An example from my samples (flagged by all the spellcheckers) is in this passage:

When you go *scrumping* and when you get caught it is
worth taking the chance of getting some gascoyne's scarlet
apples.

Although s*crumping* is probably familiar to most native English speakers, it should be classed as a rare or obscure word as it would never be used other than in this context. It is not included in the Shorter Oxford Dictionary (nor, incidentally, in the Oxford Dictionary of Modern Slang.) Supplementary dictionaries are often supplied for applications, such as medicine, that use a lot of them. Spellchecker dictionaries are considered in detail in Chapter 4.

1.2.5 Suggestion lists

In the preceding sections a word has been counted as corrected if the target word was found somewhere in the list of suggestions offered for correction, but long lists of often obscure words may not be helpful to a poor speller, even if the required word is among them.

How many suggestions should a spellchecker include in its list? Five or six would seem to be an acceptable maximum. All the spellcheckers produced lists longer than this. Word Perfect had the longest. It made 23 suggestions for the correction of *moter (motor)* (*mother* was the first and the correct word was eighth) and frequently gave 22 suggestions.

Do long lists improve the chances of the correct word being suggested? My survey suggests not. The spellcheckers that generally had shorter lists achieved the best overall correction rates. If they had been restricted to their first suggestion the correction rate would have dropped to between 27% and 41%. But if they were restricted to five, only between 1% and 4% of the corrections would be missed. This suggests that they could usefully reduce the number of words in their suggestion lists. How does a spellchecker decide which word to offer as its first suggestion? Ideally this should be its ‘best guess’ given the context and the syntax. In practice, with the spellcheckers tested, little consideration seems to be given to the order in which the words are presented. Often the correct word appeared second in the list after an obscure first suggestion.

1.3 Conclusion

The spellcheckers' performance in the survey discussed in this chapter demonstrates their reliance on the early techniques described in Section 1.1. The correct word is almost always included in the suggestion list for simple, non-word errors (albeit sometimes buried in a long list). These are the types of error that the simple error algorithm (Damerau, 1964) is designed to deal with so, despite its age, it still seems to be doing its job. It is not, however, designed to deal with multi-errors: around half of those that the spellcheckers found were not corrected. In cases such as the suggestion to correct *probley* (*probably*) to *problem*, the bias towards simple errors is clear. Work needs to be done to develop techniques to deal with more complex spelling errors.

Although context is not required to *detect* non-word errors, it could be useful at the correction stage in improving the spellcheckers' suggestion lists.

The survey confirms findings by Mitton (1987) that real-word errors are a significant problem. The research described in this thesis is concerned with investigating methods to detect and correct them. Unlike non-word errors, which can be detected by dictionary look-up, real-word error detection requires use of context. The next chapter surveys approaches which have been taken to this problem.

Although the samples used in my survey were sufficient to give a flavour of the types of errors produced by dyslexics and the shortcomings of current spellchecking technology in dealing with them, a larger corpus was required before a comprehensive analysis could be made. The compilation of such a corpus is described in Chapter 3.

This chapter has also introduced questions about the type of dictionary a spellchecker needs. Chapter 4 discusses this in more detail and describes the update of an existing electronic dictionary to make it more suitable for the purpose.

Chapter 2: Real-word Error Detection and Correction

The previous chapter showed that the majority of undetected errors are real-word errors where the word produced is in the computer's dictionary but is not the word the user intended. This type of error is largely ignored by most computer spellcheckers as they rely on isolated word look-up to detect misspellings. The detection of real-word errors, on the other hand, requires the spellchecker to make some use of the surrounding context. This chapter examines previous experimental approaches to the problem and the way in which they have informed the direction taken in this research.

2.1 A hierarchy of errors

Kukich (1992) proposes a five-level hierarchy of errors, each of increasing difficulty for a computer spellchecker. At the first level are lexical errors. These are the non-word errors that are by and large detected and at least to some extent corrected by most current spellcheckers. Although there are improvements to be made, this type of error is not dealt with further in this research.

Next come syntactic errors – errors that result in ungrammatical text. Some of these are apparent from the immediate context such as, "could you *advice* (*advise*) me ..." whereas others, such as "The only *weakness* (*weaknesses*) that I think I had were [x and y]..." cannot be detected without using a wider context window. A spellchecker can make use of syntactic information such as that produced by part-of-speech taggers or parsers to detect errors of this type.

Semantic errors – the next level in the hierarchy – cannot be detected by a syntax-based spellchecker as they do not give rise to a syntactic anomaly. However, as the following example (taken from my error corpus) shows, the error word often appears incongruous with the surrounding context.

The cat ... curled up on a sack to *slip* (*sleep*).

To correct this type of error a spellchecker would need some knowledge of word-association – that curling up is more likely to be associated with sleeping than slipping, in this case.

These syntactic and semantic errors are the types of real-word spelling error I am aiming to correct in this research. The final two levels in Kukich's hierarchy – "discourse structure" errors and "pragmatic" errors – although they are mistakes in the text – cannot be classed as spelling errors.

Discourse structure errors give rise to ambiguity in the text. For example, "There were four types of fruit in the salad: oranges, apples and bananas." There is clearly an error somewhere here – we are led to expect a list of four items but are only given three – but what is the mistake? Is it the number (four produced when three was intended) or is there something missing from the list – strawberries, for example? However, although this is difficult to correct, a computer might be able to make some attempt to detect this type of error.

Pragmatic errors, the top level of the hierarchy, represent an error of communication and are not detectable without wider knowledge than that contained in the text. For example, if I gave directions to my office as being at the end of the corridor on the *left* (when it is actually on the *right*) unless the reader knew where my office was they would not recognise the error.

2.2 Syntax-based approaches to error detection

Applications such as part-of-speech taggers or parsers, which are an essential pre-processing stage for many other natural language tasks, often use statistical measures to assign a likelihood to a particular part of speech or grammatical construct in a text. The intuition when such methods are applied to spellchecking is that if there is a low probability for all possible syntactic interpretations of a particular tag sequence or sentence construction this may be indicative of an error. Although these syntax-based methods offer the possibility to detect errors, the correction process needs to be implemented separately. However, detecting real-word errors presents a greater challenge to a computer spellchecker than their subsequent correction. Methods for the correction of non-word errors can be applied to real-word errors once they have been detected and, indeed, are likely to be enhanced by the use of the additional syntactic information available from the detection stage.

2.2.1 Error detection using part-of-speech tagging

The CLAWS part-of-speech tagger, developed at Lancaster in the 1980s and subsequently used for the tagging of the hundred-million word British National Corpus − BNC − (Burnard, 2000), uses bigram part-of-speech tag probabilities to assign the most likely tag in cases where a word has more than one possible part-of-speech (Garside, 1987; Marshall, 1987). Atwell and Elliott (1987) developed an extended version of this tagger, designed to detect real-word spelling errors.

Based on the observation that 'unusual' tag sequences often occurred in the presence of an error, they developed a series of techniques, of decreasing computational complexity, to detect unusual tag pairs. The first of these drew on the findings that the large majority of spelling errors result from one of the four simple error types (Damerau, 1964) introduced in the previous chapter and the application of these rules to the creation of a suggestion list for non-word error correctors (Peterson, 1980; Yannakoudakis & Fawthrop, 1983). To adapt these rules to correct real-word errors, they applied these single-letter transformations to each word in their text at run-time to generate a 'cohort' of similar words. Each word in the cohort was scored by combining the tag probabilities output by CLAWS with a number of other factors: preference for the word that appeared in the text over its alternative cohort members, a high weighting for commonly used words and a low weighting for rare ones, collocational and domain preference. The cohort member achieving the highest score was assumed to be the intended word and, if it differed from the word that had appeared in the text, it was proposed as a correction.

They rejected this approach as impractical to implement for realistic use at the time. The large number of dictionary lookups required to generate the cohorts and the subsequent evaluation of a complex likelihood function proved too computationally expensive, and existing corpora (such as the million-word LOB (Johansson et al., 1986) did not provide sufficient data for accurate measures of collocation and domain preferences. However, faster processors, increased memory and disk capacity and the availability of larger corpora subsequently overcame many of these limitations and several aspects of this method have been incorporated into later research.

The first technique required the program to generate a cohort for each word in the text at run-time. The number of dictionary lookups associated with this could be reduced by storing the cohorts with each dictionary entry. To avoid the large increase in dictionary size that this would entail, their next approach stored just the part-of-speech tags rather than the actual words. Each word in their lexicon, in addition to its own set of tags was assigned a set of 'error' tags - tags that differed from its own tags but belonged to words that might possibly appear in error for it. Using this method, cohorts only needed to be generated if one of these error tags achieved a higher score than that of the word's own tags. They were able to test this on a limited sample of artificial error data using a small prototype lexicon but again concluded that it could not be implemented in a realistic way within the limitations of the existing technology; the large number of alternative tag combinations to consider was still too computationally expensive.

Their final technique used just the likelihoods for the word's own tags. If any of these fell below a predefined threshold (they experimented with different levels for this) the program flagged the word as a potential error. Although this was found to be less accurate than the error tag method – without the error tags, the program sometimes found a plausible tag sequence by mistagging the word following the error – it had the advantage of being able to be to run over a reasonable sized sample of text. In contrast to much of the later research into real-word error checking that used artificially generated errors, Atwell and Elliott compiled a 13,500 word corpus containing just over 500 errors as test data for their program. Their results showed that this method was capable of detecting a useful proportion of the errors and suggested that its performance might be further improved once a practical way to incorporate other factors, such as those in their initial proposal, could be developed.

Stephen Elliott kindly gave me a copy of this Lancaster error corpus and I discuss it further in relation to my own research in Chapter 10.

2.2.2 Error detection using parsing

Atwell and Elliott noted that their tag-bigram based method was only able to detect syntactic errors that were recognisable within a narrow context. Errors that did not meet these criteria were removed from the corpus and stored in a separate 'hard' file. In some cases these were errors that had the same part-of-speech as the target and so would not be detectable by syntactic means, such as:

> ...a *pope* (*pipe*) cleaner can be used when necessary.

In other cases although they were syntactic errors they could not be detected without a wider context, such as:

> ...it was unfortunate that a garden party at the home of our
> chairman and his wife *has* (*had*) to be cancelled due to...

Detection of errors of this second type would only be possible using a more detailed syntactic analysis such as that output by a parser.

The CRITIQUE system (Richardson, 1985), originally known as EPISTLE (Heidorn et al. 1982), was a comprehensive grammar and style checking system, based on a natural language parser, developed at IBM during the 1980's. Its grammar checker was designed to alert the user to such problems as subject-verb disagreement and use of incorrect forms of verbs and pronouns. When run over 350 test sentences, some of which contained errors of this type, the program was found to provide "correct or useful information for 84% of the sentences in the sample". Although this experiment was not aimed specifically at real-word error detection it demonstrates that the program had some facility to detect this type of error. However, this approach to real-word error detection does not appear to have been developed further.

2.3 Using confusion sets to detect and correct errors

The main drawback of a purely syntactic approach to the real-word error problem is that it is limited to errors that cause a syntactic anomaly in the text and so cannot detect errors such as the *pope/pipe* example from Atwell and Elliott's corpus above.

An alternative method, which can be used both to detect and to correct errors and is appropriate for both syntactic and semantic errors, is that of confusion sets. A spellchecker using this approach stores predefined sets of words that are considered to be confusable, such as {*their, there, they're*}. Each time it encounters one of the words in the set it uses rules to decide whether one of the other members of the set would be more appropriate in the context. The rules can make use of any aspect of the text, syntactic or semantic, or any information about the language, such as word frequency. A key question is how the sets of confusable words should be defined.

2.3.1 Simple error correction using word trigrams

One method of creating confusion sets, as used by Mays et al. (1991), is to group together words that differ from each other by a single letter (insertion, omission, substitution or transposition) – this is the method used by Atwell and Elliott (1987) to generate their 'cohorts', described above. The main difference between Atwell and Elliott's cohorts and Mays et al.'s confusion sets is that the latter were created in advance rather than dynamically at run time.

Mays et al. created confusion sets of this type for each word in their 20,000 word vocabulary. The sets varied in size – short words are likely to generate more confusables – up to a maximum of 30 words. To capture semantic as well as syntactic context they used word trigram probabilities – the conditional probability of a word occurring given the two preceding words – derived from a large body of text.

To simulate error correction, they took 100 sentences of correctly spelled text (containing only words in their vocabulary) and from these generated just over eight thousand misspelled sentences by successively replacing each word with each member of its associated confusion set. Each of these misspelled sentences contained just one error. So, using their example, the sentence beginning "I submit that..." would be transformed to "*a* submit that...", "I *summit* that...", among other things, but not "*a summit* that...".

They ran their program over these test sentences and used the stored probabilities to score each sentence and each of its single error alternatives. For the correctly spelled sentences the program could make one of two decisions: accept the sentence as correct or suggest changing it to some other sentence (false alarm). For the error sentences there were three alternatives: flag the sentence as incorrect and propose the correct sentence as an alternative (correct the error), flag the sentence as incorrect but propose some other incorrect sentence as an alternative (detect but not correct the error) or accept the sentence as correct (ignore the error). The words actually appearing in the sentence were given a higher proportion of the overall probability than their alternative confusion set members to represent the expectation that words are more likely to appear correctly spelled than they are to be misspelled. By varying the weighting given to the word seen in the text they were able to detect 76% of the errors and correct 73% at the expense of just one false alarm and were still able to detect 63% and correct 61% while reducing the false alarms to zero.

This suggests that the method of using word trigrams is a potentially useful approach to detecting and correcting real-word spelling errors. However, there are several considerations to be borne in mind: the scalability of the approach, the appropriateness of the confusion sets and the relationship between artificially generated errors and errors that are made by real people.

The number of word trigrams required, even for a 20,000 word vocabulary is enormous – almost eight trillion; assuming that all words in the trigram are distinct this is calculated as 20,000 * 19,999 * 19,998 = 7.9988E+12. For a larger vocabulary – the 72,000 words in the dictionary I am using for this research, for example – the number of trigrams becomes unmanageable. Sparse data is also a problem; even in a large body of text many acceptable combinations of words will not be seen and so will be assigned smoothed rather than actual probabilities which are likely to reduce the program's accuracy. This method would thus seem best suited to specific, high frequency collocations but not practical for all word combinations.

Simple errors, involving the single-letter transformations used to create the confusion sets for this experiment, have been reported to account for 80% of all errors (Damerau, 1964; Damerau and Mays, 1989). However, subsequent research by Mitton (1987), confirmed by my dyslexic error corpus, suggested that the proportion of this type of error in text produced by poor spellers is lower than this; simple-error confusion sets appear to be less appropriate for the type of errors I am aiming to correct.

In the test data used for Mays et al.'s experiment, there was a maximum of one error in each sentence and in each case the intended word was a member of its confusion set. This will not always be the case with real text; users may make more than one error in a sentence, produce misspellings not in our confusion sets and so on.

However, the ability of confusion sets to correct as well as detect errors has made them attractive to many subsequent researchers.

2.3.2 Machine learning techniques

Several machine learning methods that have proved successful for other natural language processing tasks – such as part-of-speech tagging – have been applied to context sensitive spelling correction. Here the problem is regarded as a lexical disambiguation task and confusion sets are used to model the ambiguity between words. Given an occurrence of one of its confusion set members, the spellchecker's job is to predict which member of that confusion set is the most appropriate in the context.

Golding (1995) compared the performance of decision lists and Bayesian classifiers. The latter was found to give better performance and was then improved further by combining it with a trigram part-of-speech method (Golding and Schabes, 1996). Following this Golding and Roth applied a Winnow multiplicative weight-updating algorithm to the same problem with a considerable improvement in accuracy (Golding and Roth, 1996; Golding and Roth, 1999). This led to the development of the SNoW (Sparse Network of Winnows) architecture. Carlson et al. (2001) report achieving a high level of accuracy when applying this method to 256 confusion sets. In contrast to these statistical techniques, Mangu and Brill's (1997) transformation-based learning approach used far fewer parameters although it achieved comparable performance.

Apart from the work by Carlson and Roth (2001), all this research used a small collection of around 18 confusion sets largely taken from a list of commonly confused words in the Random House Dictionary (Flexner, 1983). There are some slight differences between the sets reported on in the different pieces of work but the 'core' sets remain the same. With the exception of {*their*, *there, they're*} (which should perhaps be regarded as the classic confusion set) and {*cite*, *sight*, *site*} these sets are all pairs of words. They include homophones, such as {*peace*, *piece*}, words whose usage is often confused, such as {*accept*, *except*} and potential typos, such as {*begin*, *being*}.

A notable feature of these confusion sets is that they are symmetric; each word in a set is considered to be confusable with every other word in that set and there is no overlap between the sets – each confusable word belongs to just one set. This is in contrast to the sets generated by Mays et al. where, although each member of the confusion set is considered confusable with the headword, the members of the set are not necessarily considered to be confusable with each other and a word may appear as a member of several different sets. This symmetry plays an important part in both the training and testing phases of these machine-learning methods as the sets are both trained and tested as a unit.

During the training phase the program uses a large body of correctly spelled text to learn a set of features – such as patterns of words or part-of-speech tags – that surround the words in its confusion sets. An occurrence of a particular feature in the vicinity of one set member is taken as a positive example for that member and a negative example for the other members of the set. The spellchecker uses these stored features to choose between the confusion set members. At run-time, any occurrence of a confusion set member in the text is regarded as a 'place marker' for a slot that could be filled with any member of its confusion set. The spellchecker then scores each confusable based on how well the features in the surrounding text fit with the features that were associated with it during the training phase. The one with the highest score is assumed to be the intended word and, if it differs from the word that originally appeared in the text, a correction can be proposed.

If the program is deciding simply whether word x or word y is more likely to fill a particular slot in the text, the program will make the same decision whether the word currently filling the slot is the intended word or an error. In this case there will be no difference in the program's decision when the program is tested on correct text to that made when it is tested on text containing (artificial) errors. In other words, if an error is generated by replacing a confusable that is correctly used (and accepted as such by the program) by one of the alternatives in its confusion set, the program must propose the correct word as a replacement. Much of the work referred to above reports results obtained from running the program over correctly spelled text with the assumption that this also indicates its potential performance when used with error data. This performance is measured as prediction accuracy – the proportion of correct decisions out of all decisions made for a confusion set. Many of the programs achieve an accuracy of over 90% while highly developed applications such as SNoW achieve over 95% on average (Carlson et al., 2001).

Golding and Schabes (1996) compared the performance of their Tribayes program with Microsoft Word's grammar checker. For this they used two versions of their text – correct and 'corrupted'. The corrupted text was produced by replacing each word in a confusion set in turn with each other word in its confusion set. For the correct text they measured the number of times each program accepted the word and for the corrupted text how often each program proposed the intended word as a correction. They noted that Tribayes sometimes expressed very little preference for any member of a confusion set as the scores were almost identical. Since it was set to select the word achieving the highest score, no matter how small the margin between the score for the word appearing in the text and the scores for its alternatives, this, on occasion, caused it to propose false corrections when run on the correct version of their text. Word, on the other hand, is programmed to suppress its weaker suggestions so it raised fewer false alarms. In order to compare more fairly between the two programs, they introduced a confidence threshold; Tribayes was set to propose a correction only if the ratio of the probability of the suggested word to the probability of the word that had appeared in the text was above this threshold. Although this had the desired effect of reducing the false alarms in the correct text it also, obviously, reduced the number of errors corrected in the corrupted text. By experimenting with different levels for this threshold they were able to set it at a level that achieved a reasonable balance between these two conflicting outcomes.

Once a confidence threshold is introduced, prediction accuracy can no longer be used alone as a measure of program performance since, in cases where it does not have enough confidence in its decision, the program simply does not make a prediction at all. To factor this in, Carlson et al. (2001) introduce the notion of 'willingness' – the proportion of all occurrences of confusion set members for which the program was confident enough to make a prediction. By implementing a confidence threshold they were able to improve the performance of their program from an average of 95% (with no threshold, hence 100% willingness) to 99% with a willingness of 85% for their 256 confusion sets.

Initially these machine-learning approaches used only a small number of confusion sets, typically about twenty. Carlson et al. address several of the issues involved in scaling-up these methods for use in a practical system by extending the number of confusion sets to 256. However, as they state, this is only a first step as a realistic system would require a coverage of many thousands of words. One of the limitations to achieving this level with these systems is the requirement for symmetric sets with no overlap between the sets. Take, for example, our classic confusion set {*their, there, they're*} on which all the above methods achieve a good level of performance. Suppose we now want to allow for the possibility of *three* being produced in error for *there* (or vice-versa). Where do we put it? It doesn't fit in the existing confusion set for *there* as it seems unlikely to appear when *their* or *they're* is the intended word but if we create the set {*there*, *three*} we now have overlapping sets.

2.4 Using semantics to detect errors

In contrast to Atwell and Elliott's tag-bigram method which detected *syntactic* anomalies, the 'semantic relationship' approach first proposed by Hirst and St-Onge (1998) and later developed by Hirst and Budanitsky (2005), detected *semantic* anomalies but was similarly not restricted to checking words from predefined confusion sets. This approach was based on the observation that the words that a writer intends are generally semantically related to their surrounding words whereas some types of real-word spelling errors are not, such as (using Hirst and Budanitsky's example), "It is my sincere *hole* (*hope*) that you will recover swiftly." Such "malapropisms" cause "a perturbation of the cohesion (and coherence) of a text."

Using their method, malapropism detection is implemented as a two-stage process. In the first stage the program scans the text for likely suspects – words that seem semantically unrelated to other words in the text. Any word not regarded as a suspect at this stage is accepted as correct and not considered further. The second stage then generates a list of possible alternatives for each suspect; if one of these is found to be a better semantic fit than the suspect word, the suspect is flagged as an error and the alternative proposed as a correction.

The semantic relationships between the words in a text are represented using the noun portion of WordNet (Miller et al., 1990). Nouns in WordNet are organised as a lexical hierarchy with each node of the hierarchy representing a synset or grouping of synonymous words. The main organisational principle is hyponymy/hypernymy or the ISA relation. After comparing several methods of measuring the semantic relatedness of words based on traversals of this hierarchy (Budanitsky, 1999; Budanitsky and Hirst, 2005) they concluded that the approach developed by

Jiang and Conrath (1997), combining semantic distance in WordNet with corpus-derived statistical data, offered the best performance for the detection of malapropisms.

The process of offering a list of possible alternatives for suspect words is, as Hirst and Budanitsky note, akin to that of proposing candidate corrections for non-word errors and a spellchecker might well use the same mechanism for both. However, for their experiment they only considered simple error transformations – the type used by Mays et al. to create their confusion sets (described above).

As test data they used 500 Wall Street Journal articles into which they introduced errors at the rate of one every two hundred words, giving around a thousand errors with an average of just under three errors per article. Text produced by dyslexics is likely to contain a higher proportion of errors than this (Pedler, 2001a). As with other artificial error data discussed above, each of the errors introduced was potentially correctable by their system. Again, this will not always be the case with actual error data.

They report recall (the number of errors flagged as a proportion of all errors) and precision (the number of errors flagged as a proportion of the total suspects flagged) for their system, using varying sizes of context, ranging from a single paragraph to the entire text. Larger context size improved precision – more of the words flagged actually *were* errors – but reduced recall – fewer errors overall were detected. The best balance between the two was achieved with the smallest amount of context – a single paragraph – with recall of 50% and precision almost 20%. This, they suggest, indicates that their system "approaches practical usability for malapropism detection and correction". However, malapropisms themselves represent only a small proportion of all real-word errors meaning this method would need to be combined with other approaches for realistic use by a spellchecker. Even then their assessment seems somewhat over-optimistic; the rate of false alarms is high in comparison to the number of errors corrected. Atwell and Elliott (above) report an average 40% recall and 33% precision for their program but are more conservative in their evaluation, rating their results as "reasonably promising". (Although it should be noted that these results are only comparable in a very general sense.)

2.5 Conclusion

This chapter has considered previous experimental approaches to the detection and correction of real-word spelling errors. Syntactic errors, as Kukich (1992) suggested, proved the most tractable since existing natural language applications, such as taggers and parsers, could be adapted to detect them. Correction, if required, could then be implemented using the same method as that used by the spellchecker to produce candidate corrections for non-word errors. Although these methods are limited to correcting errors that result in a syntactic anomaly they have the advantage of not needing to be trained for individual words; new words entering the vocabulary are likely to be covered by existing rules.

Confusion sets can be regarded as predefined suggestion lists. These were originally proposed by Atwell and Elliott (1987) and only rejected as impractical due to the limitations of computing speed and memory at that time. They became a key feature of much subsequent research. They offer the possibility of both detecting and correcting errors but a large number of sets is required if they are to be used for general-purpose spellchecking and they also need to be representative of the types of error actually made by users. In addition to simple errors, as used by Mays et al. to create their confusion sets, they should include other common errors such as homophone confusions. The Random House sets used for the machine learning experiments represent a wider variety of errors but do not cover a large vocabulary and are generally confined to pairs of words (with a few triples as noted above).

A notable feature of the research presented in this chapter is that none of it, apart from the work by Atwell and Elliott, has used real error data; several drawbacks of this have been discussed above. Not only are artificial errors always potentially correctable, there is also no indication that they are representative of the patterns and frequency of errors occurring in unrestricted text.

Several different measures of performance were used to assess the programs: recall and precision (Atwell and Elliott, 1987; Budanitsky and Hirst, 2005); proportion of errors corrected versus number of false alarms (Mays et al. 1991); prediction accuracy (Carlson et al., 2001 etc.). These are discussed further in conjunction with the evaluation of my program (Chapter 10). A common concern when assessing the performance of these programs is the trade-off between accuracy and coverage. A reduction in the number of false alarms means that a greater proportion of the words flagged as errors actually *are* errors – an increase in accuracy – but this generally has the effect of reducing the number of errors detected – a decrease in coverage. The general consensus seems to be that accuracy is the more important of these considerations. Richardson (1985) suggests that false alarms are likely to "...undermine the faith of users in the system". Carlson et al. claim that users' confidence will be increased if the number of false negatives is reduced. To reduce false alarms most programs have factored in the general expectation that the word appearing in the text was more likely be correctly spelled than it was to be an error.

Most previous research into real-word error correction has started with a technique and then applied it to the problem. In this research I start by defining the problem and then investigate techniques suitable for its solution. In order to define the problem, I required a corpus of errors. The compilation of this is described in the next chapter.

Chapter 3: A Corpus of Dyslexic Real-word Errors

The small sample of errors discussed in Chapter 1 gave an indication of the extent of the real-word error problem in text produced by dyslexics but it did not provide enough data for a realistic assessment of a program designed to correct this type of error.

Apart from the work by Atwell and Elliott (1987), discussed in the previous chapter, most research into detecting and correcting real-word errors has been tested on artificial data, produced by introducing errors into correct text and then attempting to correct them, rather than on actual errors made by people. One of the reasons for this is the shortage of actual data. Publicly available error corpora such as the Birkbeck Error Corpus (Mitton, 1985) or the Wikipedia (2003) list of common misspellings are largely lists of pairs of <error, target> words and so not suitable for testing a spellchecking program that requires the use of context. The Birkbeck corpus contains a mix of both non- and real-word errors while the Wikipedia lists contain only non-word errors. One way to simulate real errors would be to substitute errors from these corpora into otherwise correctly spelled text. To be 'realistic' this process would also need to simulate the frequency with which users tended to make particular errors (which can be obtained from the Wikipedia lists). Even then, randomly substituting misspellings for a correctly spelled counterpart, still might not reflect how actual users make errors. To accurately simulate that, we need to know whether the errors are more likely to occur at certain points in a sentence, whether they ever appear in groups of two or more and so on. Questions such as these cannot be answered without knowing the original context in which the error appeared.

To develop a program designed to correct actual errors made by dyslexics, I needed a corpus; since none existed, I needed to create one. As well as providing data to develop and test my program, such a corpus would also enable me to obtain a picture of the pattern and types of error in actual dyslexic text which could help to inform the direction of my research. This chapter describes how my initial corpus[2] was compiled, followed by an analysis of the errors it contains and a consideration of possible methods for their correction.

3.1 Collecting samples of dyslexic text

The samples described in Chapter 1 provided the initial input to my corpus. I then needed to find additional data; obtaining this was harder than I had initially anticipated. I contacted several college disability officers, spoke with various people who worked with dyslexics and posted to a number of bulletin boards and mailing lists. Many people were interested to learn about my research and wanted to know more about its findings; several commented on the difficulty of obtaining this type of data and the usefulness of a resource such as the one I was trying to create but only a few supplied me with actual examples. From these I assembled a corpus of around 12,000 words containing just over 800 real-word errors. These additional sources, with examples from each, are described below.

3.1.1 Dyslexic student

A dyslexic student on our MSc Computing Science course responded to a request on my college home page and supplied me with a number of essays (literature, psychology and sociology) he had produced as an undergraduate.

> The country possessed a capable naval and military force
> as well as advanced industrial homebase which *arised*
> *form* the industrial revolution.
>
> Undergraduate essays

3.1.2 Dyslexic typing experiment

In response to an enquiry to a dyslexic mailing list - http://www.jiscmail.ac.uk/lists/dyslexia.html - Roger Spooner gave me the data from an online typing experiment which he had conducted while researching spellchecking for dyslexics for his PhD (Spooner, 1998). Part of the experiment involved typing dictated sentences. I did not use this as it consisted of multiple misspellings of the

[2] Additional error data assembled for the final testing of my program is described in Chapter 10

same word and did not necessarily represent words that people would actually use. The final part of the experiment involved free writing where the participants were asked to write about their work, interesting people they had met recently, assess their spelling and writing and comment on the experiment. I used the responses to these questions in the corpus. Two examples are given below:

> I *fund* out *thaty* I have a half sister *how* I know little *abouy* except that she has two *childern* and *like out doo activtes.*
>
> I have written *team* papers and *a artcile* that was *publish*.
>
> Typing experiment

3.1.3 Internet sources

I also included some samples from dyslexic bulletin boards and mailing lists and stories written by a dyslexic child which were published on the Internet.

> *i* still *dint kow wat* was *worng* with me. *no one reelly new* how bad my *writig* was
>
> Bulletin Board
>
> He was a tiger and he *shard* the *poeple* and it was a lot of fun to do.
>
> Child's stories

3.1.4 My daughter (again)

Although I did not to want to overweight the corpus with examples of my daughter's misspellings, she was now four or five years older than she had been when I collected the homework samples and at college studying for an IT NVQ, so it seemed reasonable to include some of her later work, especially as there was a noticeable difference in her spelling ability as shown below.

The only *weakness*[3] that I think I had were the *martial* on each slide could have been a bit better and I could have improved the length of the presentation.

IT NVQ

3.1.5 Non- and real-word errors

The samples collected contained a mix of non- and real-word errors and a small proportion of word-boundary errors – split words and run-ons as described in Chapter 1. Table 3.1 summarises the content of this corpus, showing the number of words and sentences in each source and the types of error they contain.

Source	Sentences	Words	Non-word errors	Word-boundary errors	Real-word errors	Total errors
Homework*	67	974	205	28	44	278
Compositions*	61	845	103	8	50	161
Office documents*	67	720	161	6	26	193
IT NVQ	48	732	20	18	51	89
Undergraduate essays	341	6382	421	8	128	557
Typing experiment	61	694	86	7	39	132
Child's stories	11	126	7	0	14	21
Bulletin boards	739	11051	678	77	468	1223
Totals	1395	21524	1681	152	820	2654

Table 3.1: Composition of the entire corpus (* indicates data described in Chapter 1)

I could not be sure to what extent the text that I collected had already been spellchecked and, if checked, whether some of the real-word errors it contained had been generated by the spellchecker – as is likely to happen when poor spellers are presented with long suggestion lists. With a long

[3] This is a type of discourse structure error - the fourth level in Kukich's (1992) hierarchy discussed in Chapter 2. Sophisticated processing would be required to correct it. Even if the subject-verb disagreement *weakness. . were* was detected, it would not be possible to decide whether it should be corrected to *weaknesses . . were* or *weakness . . was* (both grammatically acceptable) without also detecting that the writer continues to describe two *weaknesses*.

list, even if the intended word is included, it may be buried beneath a long list of obscure words. This is no help at all to a dyslexic, or anyone else, who didn't know how to spell the word in the first place. To stop the spellchecker complaining, they may simply resort to selecting the first word in the list, which can result in the transformation of a non-word error into a real-word error. The varying proportion of non-word errors in the different sources suggests that there has been some attempt at error correction in some of them. Non-word errors occur more frequently in the typing experiment than they do in the NVQ and essay examples. The essays were almost certainly spellchecked whereas no spellchecker was available to the participants in the typing experiment and although the bulletin board includes a spellchecking facility, users are likely to pay less attention to spelling when using a bulletin board than they would when writing an essay. However, since this research is only concerned with correction of the real-word errors, I am making the assumption that there would be two phases to the spellchecking process: the first dealing with the non-word errors[4] and the second attempting to detect and correct any real-word errors using the now correctly spelled text. Whether or not real-word errors are introduced by the first phase, it will at least remove the non-word errors so the real-word error checker is left to deal with a passage of correctly spelled text in which the majority of the words are those that the user intended.

3.2 A corpus of real-word errors

3.2.1 Composition

I first marked-up all the errors in the format illustrated below:

> The only **<ERR targ=weaknesses> weakness </ERR>** that I
> think I had were the **<ERR targ=material> martial </ERR>**
> on each slide.

Once this had been done it was a simple matter for a program using Perl regular expressions to extract the errors and their corresponding target word from the corpus. It also enables my

[4] MS Word adequately corrects the majority of those appearing in the examples above.

spellchecker to ignore the target words when checking the text but at the same time to store them for later use to check against its output.

Although including the non-word errors gives the text a distinctive dyslexic flavour, they are just a distraction for a real-word error checker so I next created a sub-corpus, containing only real-word errors, to use for developing and testing my program. To do this I removed any sentences that contained only non-word errors and replaced all non-word errors in the remaining sentences with the intended word – this could be regarded as a simulation of the first stage of the spellchecking process discussed above.

The sub-corpus contained around 12,500 words with a total of 820 real-word errors. Although the majority of these error words occurred just once as errors, a minority occurred repeatedly so that the number of distinct error types was approximately half the number of error tokens. The word occurring most frequently as an error was *there* with 37 instances of incorrect usage, followed by *to* with 25 instances. As a single error word can appear as an error for several different targets – for example, *quit* appears as a misspelling of both *quiet* and *quite* – the total number of distinct <error, target> pairs is higher than the total number of error types. This is summarised in Table 3.2.

Sentences	614
Words	11810
Total errors (tokens)	820
Distinct errors (types)	429
Distinct target|error pairs	493

Table 3.2: Composition of the real word error sub-corpus

3.2.2 Error frequency

The discussion above suggests that users have a tendency to produce certain misspellings consistently. My experience also confirms this; I often catch myself typing *form* when I mean *from*, my daughter regularly produces *theses* in mistake for *these*. Table 3.3 shows the frequency with which the error words occurred in the real-word error sub-corpus; although the majority occur only once, a minority occur repeatedly.

N. Occurrences	N. Errors
>10	10
6-10	9
4 or 5	17
3	25
2	47
1	321
Total error types	**429**

Table 3.3: Frequencies of error types in the corpus

Error words that occur 10 or more times in the corpus are listed in Table 3.4. Many of these appear as an error for more than one target and so contribute to several of the distinct <error, target> pairs. Several of the short, high-frequency words in this list – *to*, *an*, *is*, – appear as errors for four or more different targets which confirms Mitton (1996)'s finding that a high proportion of real-word errors involve this type of word.

Error	Frequency	N. targets
there	37	3
to	25	4
a	20	3
form	19	1
their	18	1
its	17	1
your	17	2
an	13	5
weather	11	1
were	11	2
cant	10	1
is	10	4

Table 3.4: Errors occurring 10 or more times in the corpus

Table 3.5 shows the frequency with which the distinct <error, target> pairs occurred. Again, although most of the pairs occur just once, a minority occur repeatedly. In contrast to the findings for individual errors (Table 3.3), which showed that some words were often produced as a misspelling of several other words, this shows that there are some words which regularly appear as a misspelling of one other word in particular. Pairs such as these are likely to be good candidates for confusion sets; the ten most frequent are listed in Table 3.6

N. Occurrences	N. Pairs
>10	8
6-10	7
4 or 5	13
3	21
2	57
1	387
Total error pairs	**493**

Table 3.5: Frequency of error|target pairs in the corpus

Error|target pair	Frequency
there|their	32
form|from	19
to|too	19
their|there	18
a|an	17
its|it's	17
your|you're	15
weather|whether	11
cant|can't	10
collage|college	9

Table 3.6: Ten most frequent error|target pairs in corpus

Many of these top ten pairs also feature in the small list of sets of 'commonly confused' words used in much of the research discussed in Chapter 2. This confirms that, although the small number of these sets limits their usefulness for a comprehensive effort at real-word error correction, they do at least include some of the errors that users actually make.

Another feature of these 'commonly confused' sets is that they are symmetric – each word in the set is considered confusable with each of the other words. This is true for some, but not all of the most frequent errors listed above, which suggests that symmetric confusion sets may not be the best approach for all sets of confusable words. Table 3.7 lists the pairs from this list in which each word occurs both as an error and as a target and shows that, in all cases, one member of the pair appears as an error significantly more times than the other.

Error|target pair	Count a|b	Count b|a
there|their	32	18
form|from	19	1
to|too	19	3
a|an	17	4
its|it's	17	4

Table 3.7: Symmetric pairs included in the ten most frequent pairs

Three of the errors in Table 3.7 – *to*, *a* and *its* – are omission errors which is consistent with my earlier finding that this is the most common type of simple error made by dyslexics. For the other two pairs, it is the less frequent word that appears more often as an error. This is more notable for the pair <*form*, *from*>, where the former is much less common overall, than it is for the pair <*there*, *their*> where both have a similar high frequency.

3.2.3 Types of error

Homophones

Homophones are often used as the basis of confusion sets. They feature prominently in the sets of commonly confused words used by many researchers and also make up almost all of the 90 pairs of confusable words used by Word's grammar checker. TextHelp Read and Write, a spellchecker designed for dyslexics and included in the survey discussed in Chapter 1 (Pedler, 2001a), incorporates a ‘Homophone Support’ feature. With this switched on, homophones are highlighted in a different colour to the spelling errors when a piece of text is spellchecked. For example, in the following extract (from the Office Documents section of my corpus) it marks the italicised words as homophones and the underlined words as errors.

> No action has *been* taken on dumps *witch are* wrong *but* if
> they *which* to change *the* approch, we will *need* more
> comlecated plans *for* suchg occurancs.

Clicking on one of the homophones produces a list of words which have similar pronunciation from the program's database of commonly confused words. Although this usefully identifies potential word confusion, it is not particularly helpful in terms of real-word error correction as it requires the user to consider a large number of words, many of which are correctly spelled. Additionally, in some cases where the word is an error, the correct word may not be one of its homophones. Of the eight words flagged as homophones in the passage above, only two (*witch* and *which*) are used incorrectly. The list produced for both of these is the same – *which* and *witch*. Although this would allow the correct alternative to be selected for the first, it is no help for the second which should be *wish* (not a homophone of *which*.)

Nevertheless, homophone confusion is certainly something that a dyslexic real-word error checker needs to take into account. Six of the most frequent <error, target> pairs listed in Table 3.6 are homophones, suggesting that words of this type do cause particular problems for dyslexics. In total, 73 (15%) of the distinct error pairs in my corpus are homophones; those that appear more than twice are listed in Table 3.8.

Homophone set	N. Occs
there, their, they're	53
to, too, two	26
its, it's	17
your, you're	15
weather, whether	11
herd, heard	5
witch, which	4
wile, while	3

Table 3.8: Homophone pairs occurring more than twice in the corpus.

Simple errors

Mays et al. (1991) created confusion sets for each word in their dictionary by listing all the other words that differed from it by a single letter insertion, omission, substitution or transposition. An examination of the types of error in my corpus showed that over half of them were simple errors, that is they differed from the correct word in just one of these ways (Table 3.9). This suggests that simple error types could also be useful candidates for a dyslexic real-word error checker to consider although we might want to use fewer words and smaller sets than in the Mays et al. (1991) experiment.

Error Type	N.Errors	Percentage Errors
Omission	144	29%
Substitution	105	21%
Insertion	56	11%
Transposition	11	2%
All simple	316	64%
All error pairs	493	100%

Table 3.9: Proportions of simple error pairs in the corpus

3.2.4 Syntactic comparison

Tagset types

A real-word error sometimes gives rise to a syntactic anomaly, and this can be the basis of error detection. For this to be the case the error and the target must differ in their parts of speech. There are three possibilities to consider:

- Distinct tagsets – Words that have no part of speech tags in common.
- Overlapping tagsets – Words with some but not all not all part-of-speech tags in common.
- Matching tagsets – Words with all part-of-speech tags in common

Table 3.10 shows the number of pairs falling into each group. A syntax-based spellchecker could be expected to perform reasonably with errors falling into the first group and to have some impact on the second although it would clearly not be appropriate for words in the third group which have the same parts-of-speech.

Tagsets	**N.Errors**	**Percentage Errors**
Distinct	324	66%
Overlapping	118	24%
Matching	51	10%
Total error pairs	493	100%

Table 3.10: Count of errors by tagset type

Word class comparison

We have already seen that many of the most frequently occurring error pairs in the corpus (Table 3.6) involve short, high-frequency words. I investigated which particular classes of word were the most problematic by comparing the word class of the error with the word class of the target. For this purpose I grouped the words into five groups: nouns, verbs, adjectives, adverbs and other (function words – conjunctions, determiners, pronouns and so on). Each error or target word may belong to more than one of these classes (28% of the words in the dictionary used have between two and seven part-of-speech tags) which means that these counts are based on <word, class> pairs for each error and target rather than just on the words themselves. For example, using the pair <*form*, *from*>, the error – *form* – is tagged as both a noun and a verb and the target – *from* – is tagged as a preposition so this pair would be counted twice; both as <noun, other> and <verb, other>. Expanding each of the <error, target> pairs in this way gives a total of 1244 <error|class, target|class> pairs. Counts for these pairs are given in Table 3.11.

		Target					
		verb	**noun**	**adjective**	**adverb**	**other**	**Error Total**
Error	**verb**	181	110	57	34	27	409
	noun	153	146	51	37	40	428
	adjective	43	40	24	24	22	153
	adverb	22	19	8	15	29	93
	other	28	20	16	29	68	161
	Target Total	427	336	156	139	186	**1244**

Table 3.11: Comparison of word class for error and target

Inflection Errors

Further investigation of the noun-noun/verb-verb confusions (Table 3.11) found that a large number of these were inflection errors. This corroborates Mitton's (1996) finding that, in a corpus of school leavers' compositions (not those included in my corpus), many errors involved inflected forms.

About a third of the noun targets where the error was also a noun were number errors, almost exclusively a singular noun used in mistake for a plural. Many of these cases were simple omission errors (Table 3.9) resulting from the -*s* being left off the end of the word. Others (such as *virus* for *viruses* or *story* for *stories*) have a slightly more complicated plural form, but again the difference between the error and the intended word occurs right at the end. (The position of the error in the word is considered further in section 3.2.5 below.)

Half of the verb targets where the error was also a verb involved a wrongly inflected form of the verb. Many of these were regular verbs[5] where the error involved the base form with an omitted *-s* (third person singular), *-ed* (past tense, past participle) or *-ing* (present participle).

The remaining verb inflection errors were for irregular verbs. Some of these are completely irregular, such as *be,* while others, such as *tell* are irregular in the particular inflection involved (in this case *told*). Predominantly, however, the irregular inflection errors involved producing the base form of the verb instead of the past tense or past participle. There were three pairs which were both written mistakenly for each other *is/are*, *has/have* and *was/were*. These findings suggest that it might be helpful for a spellchecker to include a method aimed at correcting inflection errors.

3.2.5 Error patterns

Position of first letter error in word

A characteristic of inflection errors is that, even if they are not simple errors, a large proportion of them differ only in the word-ending. I looked at the position at which the first error in the word occurred, using a system described by Wing and Baddeley (1980) – section 1 is the beginning part of the word, 2 left of centre, 3 centre, 4 right of centre and 5 the end part of the word. Following Mitton (1996), I have subdivided the first section into 1A (first letter errors) and 1B (other errors in the early part of the word). Table 3.12 shows the proportion of errors falling into each section for the real-word errors compared to the non-word errors in the corpus. Over half of the real-word errors differ in the last section of the word whereas for non-word errors the position of the first wrong letter is fairly evenly distributed between the middle sections of the word. Conversely, the real-word errors also appear more likely to have an incorrect first letter. Silent first letters - *know*, *now* and the like - are a likely contributory factor here.

[5] Or at least the inflection involved was regular, for example take/takes.

Section	Percentage errors	
	Real-word	Non-word
1A	11%	4%
1B	1%	5%
2	11%	21%
3	10%	30%
4	16%	21%
5	51%	19%
Total = 100%	499	499

Table 3.12: Comparative positions of first wrong letter for real- and non-word errors

Position of error words in sentences

To find out whether errors are more likely to occur in a particular part of the sentence I divided each sentence into sections (in the same way as for words described above) and counted the number of errors occurring in each. This showed the errors to be fairly evenly distributed throughout the sentences with a slight tendency for more to occur towards the beginning so position in sentence is unlikely to be of any particular value to a spellchecker.

3.2.6 Proximity of errors

Real-word error checkers must use context in some way. Syntactic anomaly approaches to error detection, such as those described in the previous chapter, generally use word or part-of speech bigrams or trigrams; one or two words on each side are used to determine the probability of a given word appearing in that context. This will run into difficulty if these words are themselves errors. To assess the extent of this problem in my corpus I looked at the surrounding context for each real-word error. Table 3.13 shows the proportion of the real-word errors with another error (either non- or real-word) within one or two words on each side and suggests that the problem of another error occurring in proximity to a real-word error is one that is likely to cause difficulties for any context-based method of error detection. For a quarter of the real-word errors in the corpus at least one other error occurs within two words to the left or right. In some cases this is another real-word error which compounds the problem. A non-word error will be detected by dictionary look-up and the checker may be able to make an attempt at correcting it and at the very least will be aware that the context that it is considering is unreliable.

	Left	Right	Left & Right	Total
1 word each side	6%	7%	2%	15%
2 words each side	8%	12%	5%	25%
			errors = 100%	840

Table 3.13: Proportion of real-word errors with another error in the immediate context

3.3 Conclusion

At the end of this phase of my research I had produced a corpus suitable for the testing and development of the dyslexic real-word error checker I was aiming to produce. I also hope it will be of use to other researchers at a later date. Resources of this type seem to be scarce and require a fair amount of effort to produce as all the errors need to be manually detected and marked up − if there was a way of reliably automating this process, the problem I am investigating would already have been solved!

As well as providing data for the subsequent development and testing of my program, the corpus gave me a clearer picture of the types of error made by dyslexics which could inform the approach I would take. The large proportion of errors with distinct or overlapping parts-of-speech suggested that a syntax-based approach could be expected to have some degree of success. The confusion set approach remained attractive as it simplifies the process of error detection and provides a ready-made suggestion list. Although this method can be used with both syntactic and semantic rules, it does not seem applicable to inflection errors which, as the foregoing discussion showed, comprised a large number of the errors in my corpus. This is considered further in Chapter 10. Additionally, since the confusion set method is limited to correcting words predefined as being potential errors, a large number of sets would be required if it was to make a realistic impact on the problem. Chapter 5 describes how the collection of sets was created.

Chapter 4: A Spellchecking Dictionary

The dictionary used at the start of my research was CUV2 (Mitton, 1986)[6] – a 'computer-usable' dictionary based on the third edition of the Oxford Advanced Learners' Dictionary of Current English (Hornby, 1974). Its seventy-thousand or so entries had provided adequate coverage for my previous (non-word error) spellchecking work although it would occasionally produce unexpected false alarms – flagging as errors words that were actually correctly spelled. These false alarms largely reflected changes in the language since 1974 when the OALDCE was compiled. For example, *database*, a word in fairly common modern usage not included in CUV2, would have been a specialised term thirty years ago. But CUV2 contained enough words for my purpose and is in text file format so more words can easily be added.

A dictionary containing just words is adequate for non-word error detection but to enable a spellchecker to detect real-word errors other information such as pronunciation, part-of-speech tags and word frequency needs to be included for each entry.

Each entry in CUV2 includes a set of one or more part-of-speech tags each of which is assigned one of three broad frequency categories – very common, ordinary or rare. This is adequate to distinguish between different usages for some words, such as *can* which is far more common as a verb than a noun, but for others the distinction is not clear. For example both noun and verb are marked as common for *form* although it is used far more frequently as a noun. A preliminary investigation, using a small number of confusion sets derived from my corpus (Pedler, 2003a) showed that more precise word frequency information would enable the spellchecker to make a more accurate decision on the correct word for a particular context.

[6] This dictionary and its accompanying documentation (Mitton, 1992) can be downloaded from the Oxford Text Archive: http://www.ota.ahds.ac.uk/

To update the frequency information in the dictionary I used word/tag frequency counts obtained from the written section of the British National Corpus (World Edition), (Burnard, 2000). The BNC is a hundred-million word corpus, compiled between 1991 and 1994, designed to represent a broad range of British English usage in the late twentieth century. The written part, which makes up 90% of the corpus, consists of samples from published texts such as newspapers, academic books and popular fiction, and less formal writing such as essays, letters and memoranda. In addition to obtaining frequency counts for the existing words in the dictionary, I also listed words that occurred in the BNC but were not included in the dictionary. Those occurring frequently were considered for possible inclusion in the updated dictionary (discussed in Section 4.2.4).

Since the C5 tagset used for the tagging of the BNC does not correspond to the tagset used in CUV2, I created a new tagset for each dictionary entry using the C5 tagset. In this chapter I will first consider the requirements of a dictionary for computer spellchecking and then I will describe the dictionary update.

4.1 Dictionaries and spellchecking

4.1.1 Word lists and non-word error detection

The most straightforward and widely used method for a computer spellchecker to detect non-word errors is dictionary look-up (Mitton 1996). For a simple implementation the dictionary need be no more than a word list. The spellchecker then looks up each word in the text to be checked in its list and flags as misspelled any that are not found. The question to be answered at this stage is how many and which words should be included in the list. If there are too few the spellchecker will produce a large number of false alarms – flagging as errors words that are actually correctly spelled. On the other hand, adding words to the dictionary increases the chance of accepting a misspelling. *Wether* is the classic example; a spellchecker with *wether* in its dictionary would correctly accept *the wether ewe* but would fail to get the errors in *wether the wether be fine*. However, a study by

Damerau & Mays (1989) found that when a rare word occurred it was more often the intended word correctly spelled than a misspelling of some other word, which suggests that decreasing the dictionary size disproportionately increases the number of false alarms. Mitton (1996) agrees that a large dictionary is best but suggests some possible exceptions which might need special treatment (such as *cant* and *wont* which are common misspellings of *can't* and *won't*) and points out the danger of including highly obscure words.

Unless space is at a premium it is best to store all inflected forms of a word (for example, *consider*, *considers*, *considered*, *considering*). However, dictionary size can be reduced by 'affix-stripping' where only the stem, *consider* in this case, needs to be stored. The spellchecker then uses rules to remove the suffixes before looking up the word. Prefixes (such as *reconsider*) can also be dealt with in this way. Although this process has the advantage that it will accept words that may not be in the dictionary, such as *resale*, care must be taken not to accept non-words such as *unconsider*. The spellchecker's task is simplified if the dictionary contains all words in full.

4.1.2 Additional dictionary information

A simple wordlist is adequate for non-word error detection (flag any word not in the list as a potential error) and can also be used to produce suggestions for correction by finding words that closely resemble the misspelling (Damerau, 1964, Peterson, 1980, Pollock & Zamora 1984). Suggestion lists for real-word errors (once they have been detected) can be produced in the same way. However, such lists are likely to be long and include many unlikely or inappropriate words; additional information, stored with each dictionary entry, is needed in order to prune such lists.

Word frequency and part-of-speech information can be utilised to promote common words and remove syntactically inappropriate ones. In the latter case the spellchecker also needs to be supplied with a part-of-speech tag table showing how often each tag is followed by each other tag; words resulting in an unrecognized tag combination are then removed from the suggestion list. Mitton (1996) describes a spellchecker that incorporates this process. It is also able to detect a small number of real-word errors but is not as effective as a spellchecker that incorporates full part-of-speech tagging (Atwell and Elliott, 1987).

In addition to homophone errors discussed in the previous chapter, the errors made by poor spellers are often phonetically similar to the intended word, for example, the initial samples used in my survey of spellcheckers (Pedler 2001a) contained *musirls* as a misspelling of *muscles* and one of the spellcheckers suggested *mussels* as a correction. If it had used pronunciation information to incorporate homophones, the intended word – *muscles* – might also have been included in the suggestion list.

As well as tag and pronunciation information, CUV2 includes a syllable count for each word. I am not currently making use of this but it could be of use in correcting errors resulting from the omission of a repeated syllable in the middle of a word - *probably* spelt as *probley* or *remembers* spelt as *rembers* for example - which is a common dyslexic tendency.

The updated dictionary that I produced - CUVPlus[7] - retains the spelling, pronunciation, part-of-speech tags and syllable count from CUV2 and in addition includes the C5 tags with word counts from the BNC for each entry. The next section describes the process of producing this update.

4.2 Updating the dictionary

This section expands the presentation given in Pedler (2003b).

4.2.1 Counting words

The update to the dictionary was based on data obtained from the written section of the BNC (World Edition). The first stage was to count occurrences and part-of-speech tags for each word in CUV2. On the face of it this was a straightforward task. Look up each word in the BNC in CUV2; if the word is found store the part-of-speech and increment the word count. However, although generally word tokens (w-units) in the BNC correspond to orthographic words (Leech & Smith,

[7] Also available from the Oxford Text Archive

2000), there are several cases that need to be given special consideration – hyphenated words, enclitics, multi-word units, proper nouns and abbreviations.

Hyphenation

As there are no fixed rules for hyphenation in English the use of hyphens to form compound words is inconsistent. There are over 5,000 hyphenated entries in CUV2 and around 70 entries for common prefixes, such as *multi-*, *re-* and *un-*. The BNC treats hyphenated words as a single unit and gives them an appropriate tag, for example, *rose-bed* is tagged as a noun, *half-hearted* as an adjective. Both of these words (and many of the other hyphenated entries in CUV2) also appear in their 'solid' form - *rosebed*, *halfhearted* - in the BNC but are not listed as such in CUV2. Conversely, compound words such as *keyring* are entered in 'solid' but not hyphenated form in CUV2 while appearing in both forms in the BNC. The question of how compound words should be entered in a dictionary is discussed further below but at this stage they were simply counted in the form in which they appeared in CUV2.

Enclitic forms

In general a w-unit in the BNC is a sequence of characters delimited by white space. However over sixty contracted forms are broken into their component parts, each of which receives its own tag such as *can't* (*ca*+*n't*), 'twas (*'t*+*was*). These are all shortened forms apart from *cannot* (*can*+*not*). There are also seven trailing enclitics that can be attached to nouns or pronouns, for example *'d* (e.g. *I'd*) and *'ll* (e.g. *they'll*). Leech & Smith (2000) give a complete list. This approach allows the part-of-speech tagger to assign a tag to each grammatical item in the text but it is useful for a dictionary to include common enclitics as entries in their own right. For example, there is often confusion over the spelling of *they're*, *their* and *there* (Pedler 2001b) and the shortened form *they're* is included in CUV2. Contracted forms in the BNC that had entries in CUV2 were recombined and given a compound tag such as PNP+VBB (personal pronoun + present tense of BE) for *they're*, VM0+XX0 (modal auxiliary + negative particle) for *can't*.

Multi-words

In contrast to the enclitic forms where a single orthographic unit is split into two or more w-units, multi-words are combinations of words that logically form a single unit – adverbial phrases such as *a bit*, *in addition to*, complex prepositions such as *in need of*, *save for* and naturalised foreign phrases such as *a posteriori*, *chilli con carne*. Leech & Smith (2000) list almost 700 such combinations which are tagged as a single w-unit in the BNC. Those that also had entries in CUV2 (mainly foreign phrases) were counted and those that didn't were ignored. Over a third of CUV2's multi-word entries do not correspond to the BNC list. A large number of these are place names, discussed further below.

Proper nouns

Words tagged as proper nouns in CUV2 are common forenames and place names. Many of the latter are multi-word entries. Use of the NP0 (proper noun) tag in the BNC is largely confined to personal and geographical names but no proper nouns are processed as multi-word units. This means that a place name such as *Cape Town*, which has its own entry in CUV2, is split into two units in the BNC – *<w NP0>Cape <w NP0>Town*. Multi-word proper nouns such as this could not be matched with their occurrences in the BNC unless each w-unit with an NP0 tag was processed as a special case. If it matched the start of a CUV2 multi-word proper noun the following word would then need to be checked to see if it completed the word. Although this would not have been impossible it seemed an unnecessary complication and would have considerably lengthened processing time. Thus no frequency counts were obtained for the multi-word proper noun entries in the dictionary. An additional consequence was that a large number of words in the dictionary that are not proper nouns in their own right (such as *cape* and *town* from the example above) were recorded as having an NP0 tag which needed to be removed when the tagsets were created.

Abbreviations

The BNC tags abbreviations as if they were written in full with no indication that the word is a shortened form. CUV2 uses a separate abbreviation tag indicating that the word is an abbreviation together with the part-of-speech of its full form. This is useful for text processing programs that need to be able to distinguish between the use of a full-stop as a sentence delimiter and its use as an abbreviation marker. As the updated tagsets use the BNC tags this distinction is no longer made. However, since the CUV2 tags are retained this information can still be retrieved.

Tagging ambiguity

The majority of the w-units in the BNC are given a single part-of-speech tag but in cases where the tagger could not reliably decide between two possible tags for a word an ambiguity tag was assigned. This consists of the two tags in question with a hyphen between them, the first of the two being the preferred tag. For example, a tag AJ0-NN1 indicates an ambiguity between adjective and noun with the preference being for an adjective whereas a tag NN1-AJ0 indicates that the preferred assignment is noun. Such tags have their place in a corpus but are not relevant for a dictionary. In all such cases the first tag was used for the frequency counts.

4.2.2 Ranking the words

After obtaining counts for each word in the dictionary, the list was sorted in descending order of frequency. Not surprisingly, *the*, with over 5.6 million occurrences, was in first place, followed by *of* (2.7 million), *and* (2.3 million) and *a* (1.9 million). As might be expected, a comparison of the first twenty words in my listing corresponds (with slight differences in ranking) to the rank frequency list for the written section of the BNC in (Leech et al., 2001). The counts cannot be directly compared for several reasons; differences in the later version of the BNC used for this work, differences in the treatment of the special cases outlined above, and the combination of some parts-of-speech (such as past participle and past tense) in Leech et al.'s (2001) lists.

Ranking the words by raw frequency may make sense for the first few very common words where there is a clear difference in the number of occurrences but, as Kilgarriff (1997) demonstrates, counts for words beyond the few thousand most frequent are highly dependent on the corpus used. Table 4.1 shows how the difference in frequency between adjacent words decreases as we go down the list of the top 500 words. For the most frequent words the difference is in millions, after 100 words the difference is in thousands, after 200 in hundreds, dropping to less than ten after the 500th ranked word. Rather than include these absolute counts as the frequency information in the dictionary I used the rounded frequency per million figures shown in the fourth column of the table. These retain the differentiation between the first few hundred common words while grouping together words with similar frequencies such as *turned* and *held*.

Ranking	Word	Count	Frequency per million
1	the	5,617,462	64,569
2	of	2,728,483	31,362
3	and	2,350,004	27,012
		...	
99	did	71,130	818
100	one	66,564	765
101	over	65,543	753
		...	
199	men	34,618	398
200	given	34,480	396
201	high	34,302	394
		...	
499	former	16,346	188
500	turned	16,300	187
501	held	16,291	187

Table 4.1: Decreasing difference in frequency with increase in ranking

Rare words

Around 7,000 of the words in CUV were not found in the BNC (over half of these were already tagged as rare); about 10,000 <word, tag> pairs occurred just once and about 6,000 twice only. There is really little difference in the rarity of these words. Some of those not found in the BNC might well occur in another corpus while that corpus might not contain words found in the BNC. They were all given a frequency of -1 to differentiate them from the other words that occurred less than once per million which received a frequency value of 0.

4.2.3 Creating the tagsets

An initial tagset was created for each word by grouping together all tags recorded for the word, together with their frequency. Several stages of refinement were then required before the completed tagsets could be added to the dictionary. The tagging accuracy of the BNC is estimated to be just over 98% (Leech et al., 2001) but even this low level of mistagging resulted in a large number of inappropriate <word, tag> combinations being recorded. For example *there*, which should have two tags – EX0 (existential there) and AV0 (adverb) – initially had 12 additional tags assigned to it, including noun, personal pronoun and modal auxiliary verb. The counts for these superfluous tags were insignificant compared to those for the 'genuine' tags so, for all words that had more than one tag, any tag that accounted for less than 1% of the total occurrences was removed.

Although the parts-of-speech used in CUV2's tagsets do not correspond exactly to those used in the BNC it is possible to map reasonably accurately between the two.[8] The majority of the BNC tags that did not have a corresponding tag in CUV2 were removed. Some exceptions to this were verb participles which are commonly used adjectively (such as *massed* and *matching* which are both tagged more frequently as an adjective than a verb in the BNC but are only tagged as a verb in

[8] A detailed consideration of the differences between the two tagsets is given in the CUVPlus dictionary documentation , Appendix A.

CUV2) and nouns that can also function as adjectives (such as *amateur* which only has a noun tag in CUV2 but is more commonly tagged as an adjective in the BNC).

This process resulted in the removal of all tags for some entries. Many of these were rare words that were presumably not in the 50,000-word lexicon used by the tagger and so had been tagged by other rules. For example, nouns ending in *-er* such as *lounger* and *kroner* had been tagged as comparative adjectives, adjectives ending in *-ed* (e.g. *unprecedented*) as verb past tense or participle, plural nouns (e.g. *intangibles*) as *-s* forms of verbs and so on. These were given C5 tags corresponding to the CUV2 tags with a frequency of -1. Many common nouns also function as proper nouns in some contexts, such as: surnames (*bush, gable, thatcher*); titles (*aunt*, *detective, friar*); parts of place names (*canyon, valley, ocean*); company names (*dell, sharp*) and so on. Proper noun tags were removed from such entries.

Letters of the alphabet were among the most prolifically tagged entries. In CUV2 they are tagged as singular (e.g. *f*) or plural (e.g. *f's*) nouns. The default tag for alphabetic symbols in the BNC is ZZ0 but they are also assigned other tags if more appropriate in the context, Leech & Smith (2000) give several examples. In the updated dictionary I have assigned the ZZ0 tag to all singular letters of the alphabet, apart from those that also function as other parts-of-speech such as *I* (personal pronoun) and *a* (article) which have both tags. Plural letters in the BNC are also often tagged ZZ0 (although sometimes the enclitic is tagged separately). To retain the differentiation between the two forms I decided to use a ZZ2 tag for these (as in the UCREL C6 tagset) which is in keeping with the way plural nouns are tagged (NN2).

Tagsets were created for words in the dictionary that had not occurred in the corpus by mapping the existing CUV2 tags to their corresponding BNC tag. As hyphenated forms in the BNC are always tagged as single units there is no need for a prefix tag but as prefixes have their own entries in CUV2 I created an additional tag – PRE – for these. The completed tagsets were cross-checked with the original CUV2 tags to make sure that all relevant parts-of-speech had been included for each entry.

4.2.4 Adding Entries

Although CUV2 provides an adequate level of coverage for most general-purpose spellchecking tasks, the age of the dictionary means that some fairly common words in modern usage are not included. As well as storing tag frequency counts for all the words included in CUV2, I also stored <word, tag> pairs from the BNC that were not found in the dictionary, together with a count of the number of times they occurred. This enabled me to check for any notable omissions and create new entries for them.

The task of producing a list of words that might usefully be added to the dictionary was potentially enormous. However, not all w-units correspond to words in the lexicographic sense so several types could be excluded from consideration: strings containing digits; units tagged UNC (unclassified); those tagged CRD (cardinal number e.g. *ii*, *twenty-one*) or ORD (ordinal number e.g. *nth*, *twenty-first*); capitalised strings (often abbreviations or acronyms). Enclitic and hyphenated forms were also ignored because of the ambiguities discussed above. Proper nouns and multi-words were considered separately. For the remaining words I considered only those that occurred more than ten times in the BNC which left around 2,000 to inspect. In some ways this choice of frequency was fairly arbitrary but it reduced the list to manageable proportions given the time available and seemed adequate to pinpoint any major omissions.

This shortlist was manually checked to remove unsuitable entries. A non-word – *emailinc* (with 969 occurrences) – was second in the list. This appears as part of an email header and, despite its high frequency, its use is restricted to just seven documents (around 200,000 words in total) consisting of contributions to the Leeds United Football Club mailing list. This demonstrates the way in which a high incidence of use of an unusual word in a restricted domain can skew the overall frequency count. Other entries that would be inappropriate for a spellchecker's dictionary were also removed, for example (with frequency counts in parentheses): misspellings (e.g. *faithfullly* (31)); American-English spellings (e.g. *judgment* (348)); interjections (e.g. *huh* (175), *hmm* (156)); specialist terms (e.g. *unix* (222)); medical terms (e.g. *mucosa* (197), *pancreatitis* (76)); slang (e.g. *caf* (84), *dosh* (23)); abbreviations (e.g. *plc* (228), *mins* (227)).

A large number of 'solid' compound words in the list (e.g. *ceasefire* (107), *holidaymaker* (12), *turnout* (27)) were already in CUV2 in hyphenated form. As discussed earlier, it is difficult to make a consistent decision about how such words should appear in a dictionary but it seems unnecessary to include both. In general hyphenated forms are probably more useful for a spellchecker as it is easier to remove a hyphen than it is to insert it. However, this question was not considered further at this stage. Counts for the hyphenated frequencies of the examples above – *cease-fire* (51), *holiday-maker* (19), *turn-out* (70) – give no clear evidence as to which form is more commonly used. Further investigation might well reveal that it is simply a matter of individual style or preference. Compound words in the list (such as *goodnight* or *lifestyle*) could equally acceptably be hyphenated but have been entered in their solid form.

After this pruning stage 506 words remained in the list. Table 4.2 lists the top ten.

Word/Tag	**N. Occurrences**
organisation (NN1)	1201
organisations (NN2)	733
database (NN1)	403
goodnight (ITJ)	316
workforce (NN1)	291
lifestyle (NN1)	260
wildlife (NN1)	234
accountability (NN1)	210
profitability (NN1)	210
databases (NN2)	205

Table 4.2: Top ten new entries

The top two words - *organisation* and *organisations* - are accompanied by the rest of their 'family' lower down the list – *organised* (127), *organiser* (120), *organisers* (72) *organise* (24) and *organising* (12). This highlighted a general problem with the groups of words that can take an *-ise* or *-ize* suffix. CUV2 only includes *-ize* forms although (as Appendix 3 of the OALDCE remarks) the *-ise* form is equally acceptable. Overall 80 of the words in the list were of this type with several others appearing near to the top – *realised* (163), *recognised* (194), and *privatisation* (181). Similar words, such as *formalize*, *legalize*, have entries in CUV2 but do not appear in their *-ise* form in my BNC list. Altogether there are 200 such groups of words in CUV2. As it would be inconsistent to add entries for some and not others, these were dealt with as a special case and *-ise* entries created for them all. In all cases, apart from *equaliser* (possibly another football-influenced entry), *-ise* forms were less frequently used in the BNC.

Database would have been a specialised term in 1974 when the OALDCE was compiled. Other words in the list are also associated with our move into the information age: *camcorder* (16), *fax* (NN1: 178, VVB: 72), *modem* (32), *modems* (18), *workstation* (137), *workstations* (136). It is also interesting to note the inclusion of gender-neutral words, which these days are regarded as more 'politically correct' – for example, *chairperson* (17) *headteacher* (21) – although they are still outnumbered in the BNC by their gender-specific counterparts (already in CUV2): *chairman* (9379), *headmaster* (1067) and *headmistress* (225).

Proper nouns

Around 400 proper nouns not in CUV2 occurred more than 80 times in the BNC. The majority of these were surnames, the most frequent being *Jones* (492) and *Wilson* (393), which are not appropriate in a dictionary. Forty proper nouns were added to the dictionary. Two of these *Asia* (427) and *Birmingham* (375) were surprisingly omitted from CUV2. Others are places that were not nation states when the previous dictionary was produced – *Bosnia* (249), *Croatia* (149) and *Serbia* (97).

Multi-words

The majority of multi-word entries in CUV2 are place names and naturalised foreign phrases. There were thus no matches for a large number of the BNC multi-words discussed above. Although a spellchecker that uses white space as a word delimiter will process such entries as two words during the detection stage, the corrector may make use of the multi-words, for example it will process "Hong *Kang*" as two words but may propose "Hong *Kong*" as a correction. It may also use these multi-words to correct errors on short function words that are often mistakenly run together (such as *alot* where *a lot* was intended (Mitton, 1996)). Thirty-five such entries were added to the dictionary.

In total 1669 new entries were added to CUVPlus. Pronunciation and CUV2 tags have been added to each of these. For the *-ise* entries these are identical to the existing *-ize* entries. Pronunciations for the multi-word entries are a combination of the pronunciation for the individual words and they have been given CUV2 tags corresponding to their definition in the BNC (for example, *at all* is tagged as an adverb, *out of* as a preposition). For all other entries these fields were created manually.

4.3 Conclusion

The task of updating the dictionary highlighted many of the problems involved in reverse engineering a dictionary from a corpus. There are still improvements that could be made, in particular taking a consistent approach to the compound word entries, the tagging of abbreviations and a more detailed consideration of proper noun entries. However, the end product was an enhanced dictionary with accurate word frequency information and a well-known tagset. The additional entries have improved the coverage of the dictionary but the fact that less than a hundred of the words added had a frequency of more than one per million in the BNC suggests that existing coverage of CUV2, despite its age, is still fairly comprehensive.

Although it took some time and effort to produce, the enhanced version of the dictionary, CUVPlus, is a useful resource which should have lasting value. As well as providing me with the tag frequency information required by my spellchecker, it has also been uploaded to the Oxford Text Archive and has already proved useful in other research – for example, in the development of a text expansion system for disabled users (Willis et al., 2005). Full details of the dictionary, together with the tagsets used, can be found in its accompanying documentation (included as Appendix A).

Chapter 5: Creating Confusion Sets

One of the main attractions of the confusion set approach to real-word error checking is that it offers the possibility of both detecting and correcting the error; once the spellchecker decides that a word is incorrect there is a ready-made suggestion list available. The converse of this is, of course, that it will only detect errors that occur for predefined words and that it can only suggest corrections from a predefined list of possible confusables. Many early experiments used around twenty sets (mainly pairs) of confusable words taken from the Random House list of commonly confused words (Flexner, 1983). Although these sets were adequate to develop and test the proposed methods, they obviously provide insufficient coverage for general purpose spellchecking. Golding and Roth (1996) acknowledge this limitation and comment that "Acquiring confusion sets is an interesting problem in its own right..." Subsequently Carlson et al. (2001) scaled up their approach to use 265 confusion sets containing just over 500 words. They describe this as a first step toward creating a "practical system" which, they suggest, would need "a coverage of thousands of words". This chapter describes the process of creating a collection of almost six thousand confusion sets for use by my spellchecker.

5.1 Creating the sets

How can we create confusion sets that represent the types of error that people actually make? One approach is to derive them from a corpus of errors (Pedler, 2003a). This is not a particularly productive approach for more than a few sets. (I only used eight for my preliminary experiments.) Real-word error data is sparse and difficult to obtain and, beyond a handful of common confusables frequently used in experiments – {*their*, *there*, *they're*}, {*to*, *too*, *two*}, {*your*, *you're*} and the like – the majority of the errors occur just once and many do not seem to be suitable candidates for confusion sets – *fowling* as a misspelling of *following* or *petal* as a misspelling of *petrol*, to take two examples from my corpus.

An alternative is to find a method to automatically generate sets and then use the error corpus as a reference to check how far they are applicable to the types of error we are aiming to correct. This is the approach adopted here.

5.1.1 Listing confusable pairs

Roger Mitton's spellchecker (Mitton, 1996) ranks its suggestion lists using a string to string edit-distance algorithm based on a directed network - often referred to as the Viterbi algorithm in the NLP literature. This assigns a cost to each insertion, deletion or substitution operation required to transform one string into another; the lower the total cost, the more similar the strings. The network has been tuned, using a large collection of non-word errors, to assign a lower score to the type of mistakes that users are likely to make than to those that would be considered unlikely. For instance, using Mitton (1996)'s example, inserting the missing *c* in *sissors* (*scissors*) would have a lower cost than it would in *satter* (*scatter*) on the grounds that people are more likely to omit the *c* from *scissors* than the *c* from *scatter*. To produce an initial list of possible confusables, this program was run over the dictionary, comparing each word with every other word and storing the pairs which scored less than a predefined threshold.

The resulting list contained just over six thousand pairs of words. These were in the form of <*a,b*> word pairs with each pair listed once; thus, for example, the list included the pair <*bad*, *bade*> but not the pair <*bade*, *bad*>. Although the pairs were unique each individual word could occur more than once either as word *a* or as word *b*. *Bad*, for example, appears five times as a word *a*; it is also paired with *bard*, *bawd*, *bed* and *bid*; *write* is a word *b* in the pair <*writ*, *write*> and a word *a* in the pair <*write*, *writhe*>. The order in which the words appear in these pairs and whether they appear as word *a* or word *b* is simply a function of the ordering of the words in the dictionary. At each iteration the program only compares words coming later in the dictionary. Take the *write* example given above, *writ* appears before *write* in the dictionary so the program produces the pair <*writ*, *write*>. It doesn't find any other suitable confusables for *writ* so moves on to consider the next word in the dictionary – *write*. It only checks words coming after *write* in the dictionary so finds the pair <*write*, *writhe*>.

A number of pairs in this initial list were unsuitable for inclusion in confusion sets – proper nouns, prefixes, abbreviations and variant spellings (e.g. <*mama*, *mamma*>, <*whisky*, *whiskey*>). Such pairs, together with those such as <*fain*, *faun*> and <*groat*, *grot*> where both members are rare, were removed programmatically. The list also included some pairs of words which are almost synonymous, such as <*artist*, *artiste*>, <*babes*, *babies*>, <*waggle*, *wiggle*>. Although one of the members of such pairs might be considered more appropriate in a particular context, a computer spellchecker would be unlikely to be able to discriminate between the two. As there was no way of identifying such pairs automatically, they were removed by manually pruning the list.

There were also some notable omissions from the list. Some of these were commonly confused pairs such as <*from*, *form*> (presumably omitted because of the relatively high cost assigned to transpositions), <*cant*, *can't*> and <*were*, *we're*> (apostrophes had not been considered by the list generation program). Additionally, words differing in their first letter had not been considered as possible candidates for the initial list as the first letter of a misspelling is generally correct (Yannakoudakis & Fawthrop, 1983; Pollock & Zamora, 1984). However, although this is a reasonable assumption in general, it does not hold true for words beginning with silent *k*'s or *w*'s, – <*knight*, *night*>, <*know*, *now*>, <*wholly*, *holly*>, <*write, rite*> and so on – which often cause problems for poor spellers. Words such as these were manually added to the list to rectify these omissions.

After several iterations of pruning and addition I rewrote the list with each pair appearing twice, both as an <*a*, *b*> pair and a <*b*, *a*> pair - both <*rite*, *write*> and <*write*, *rite*> were included in this list, for example. This was easier to use since each word was in its alphabetical position as a word *a*. At this point there were around nine thousand pairs in the list.

5.1.2 From pairs to sets

To create the confusion sets from the pairs, each word *a* was taken as a headword and all the word *b*'s with which it was paired became its confusion set; the number of times each word appears as a word *a* represents the number of members in its confusion set; so, for example, *write*, which is listed five times as a word *a* would end up with five members in its confusion set – {*right, rite, writ*, *wright, writhe}*. When the spellchecker encounters *write* in a text it will attempt to decide whether *write* or one of these other words was actually what the user intended. A point to note is that, although *write* is considered to be confusable with each member of its set, the members of the set are not necessarily confusable with each other. So, for example, if the spellchecker encounters *writ* we don't necessarily want it to check whether *right* might be the intended word (in fact in this case we would probably not want it to as it seems an unlikely confusion – *writ*'s confusion set contains just one word - *write*). Here it is the headword that triggers the spellchecking routine and its confusion set defines the alternative words to be considered when this particular headword appears in the text. When one of the confusables itself appears it will be considered in conjunction with its own confusion set.

Unlike the sets used by Carlson et al. (2001) and the earlier work which it extends (Golding and Roth (1996); Golding and Roth (1999)), sets of this type are not symmetric; they are more akin to the type of sets used by Mays et al. (1990). This 'headword: confusion set' structure seems to be more flexible when a large number of words are to be considered. Using symmetric set construction the words in a confusion set are all considered to be confusable with each other and the appearance of any one of them in the text means that all other set members are considered to be possible alternatives. If we used the *write* set discussed above in this way, we would consider all the words *right, rite, writ*, *write*, *wright, writhe,* whenever any one of them appeared in the text, but this is not necessarily what we want to do.

Another feature of non-symmetric sets is that a word can appear as a headword but not as a confusable or vice-versa. This is useful when the confusion is between a highly frequent word and a relatively rare one – <*your*, *yore*> or <*world*, *wold*>, for example. In such cases we will give the spellchecker a lot of (largely unnecessary) work, and also increase the possibility of it making mistakes, if we check whether the rare word was intended every time we encounter the more frequent one. In other words, we would like to check whether *your* was intended when we come across *yore* but not check every occurrence of *your* to see whether *yore* would be more appropriate. To achieve this, I removed all pairs where the word *a* had a per-million frequency <= 1 and the word *b* a per-million frequency of >= 100.

I took a similar approach with confusables containing apostrophes – Mitton (1996) notes that these are commonly omitted but less often inserted. Pairs where word *a* was a contracted form (e.g. *aren't*, *he'll*, *who're*) were also removed. So, for example the final list contained the pairs <*aunt*, *aren't*>, <*hell*, *he'll*> and <*whore*, *who're*> but not the pairs <*aren't*, *aunt*>, <*he'll*, *hell*> or <*who're*, *whore*>. Two exceptions to this were the commonly confused pairs <*its*, *it's*> and <*your*, *you're*> that were included both as <*a* ,*b*> and <*b*, *a*> pairs.

These two stages removed just over a thousand pairs leaving a total of 7876 pairs. Creating sets from these pairs resulted in a total of 5942 headwords with between one and five words in their confusion sets as shown in Table 5.1.

Set size	**N. sets**	**Percentage**
1	4461	75%
2	1063	18%
3	386	6.5%
4	29	0.49%
5	3	0.05%
Total Sets	5942	100%

Table 5.1: Confusion set sizes

The three largest of these sets are shown in Table 5.2 below. (Note that *write*, which originally had five words in its confusion set now only has three as <*write*, *wright*> and <*write*, *writhe*> were among the <frequent, rare> pairs removed in the process described above – thus it does not appear in the list below.)

Headword	**Confusion set**
sit	sat, set, shit, site, suit
ware	war, wear, were, where, wire
were	ware, we're, where, wire, wore

Table 5.2: Three largest confusion sets

Once the list of sets was finalised, each set was appended to the appropriate dictionary entry as shown below:

```
write|0|raIt|J5%|VVI:63,VVB:35|1|right,rite,writ
```

The process of creating sets used here may be similar to the method used by Carlson et al. (2001) which they describe as "using simple edit distance in both the character space and the phoneme space". The majority of their 265 confusion sets were pairs but 20 contained three words and one contained four. Their complete list is not available so I have been unable to compare it to mine. However, their paper gives detailed consideration to 19 of the sets and the majority of these also appear in my list. Those that don't are grammatical/word confusion errors which are not of the type I am aiming to correct - <*among*, *between*>, <*fewer*, *less*>, for example. The notable difference between their sets and mine is that theirs are symmetric.

5.2 Appropriateness for the task

To assess how applicable these confusion sets might be for correcting the types of error made by dyslexics, I compared the generated list with the errors from the dyslexic error corpus. As described in Chapter 3, the corpus contains 493 distinct <error, target> pairs, several of which occur frequently, giving a total of 820 errors overall (Table 3.2). Around 20% of these errors were noun or verb inflection errors which are not suitable candidates for inclusion in confusion sets. This gives three possibilities for the error pairs in the corpus:

- both error and target are members of one of the spellchecker's confusion sets – the spellchecker could be expected to have some success with correcting these errors;
- the error is the headword of a confusion set but the target is not one of the set members – the spellchecker may spot an error here but it will not be able to suggest the correct replacement;
- the error is not a confusion set headword (although the target may be either a headword or a member of another confusion set) – the spellchecker will simply ignore such errors.

A subset of each of the last two groups is where the members of the pair are different inflected forms of the same noun or verb and therefore do not figure in a confusion set, though they could be considered by an inflection checker.

Table 5.3 shows the proportion of the errors in the corpus falling into each of these categories for both error types (considering each error pair once) and error tokens (the overall number of error pairs appearing in the corpus). Overall, almost eighty percent of the error tokens in the corpus would be considered by the spellchecker, although it would only be able to propose the correct replacement for around three-quarters of these errors.

	Types	**Tokens**
In confusion set	44%	58%
Target not in confusion set	27%	20%
(Inflection error	10%	8%)
Error not headword	29%	22%
(Inflection error	13%	9%)
Total (100%)	493	820

Table 5.3: Coverage of corpus errors

The majority of the pairs that the spellchecker could not correct by the methods proposed above occur just once in the corpus. Those occurring more frequently are listed in Table 5.4. The striking thing about this list is the number of short function words it contains and the number of different permutations in which they occur, which again confirms earlier findings that these words are particularly problematic. However the confusion set approach does not seem appropriate for such errors. The most frequently occurring of these pairs - <*a*, *an*> - is an exception to this since the appropriate choice between them depends solely on whether the following word starts with a vowel or a consonant. *The*, the most frequent word in the language appears in second place with four occurrences as a misspelling of *they*. It also appears among the once-only pairs as a misspelling of *that* and *there*, suggesting that users have a tendency to produce *the* in place of other *th*- function words. Attempting to apply the confusion set approach to this collection of words would clearly not be productive. Correct usages of *the* must overwhelmingly outnumber the error usages and to check every occurrence as a potential error would be more likely to raise false alarms than to produce corrections. Function words in general do not seem amenable to the approaches I am proposing.

It is debatable whether some of the words in the list should be considered as spelling errors at all – producing *i* for *it* is clearly a slip; *u* for *your* could be considered 'shorthand' of the type that is used in text messages; *cause* is probably intended as a colloquial version of *because*.

This leaves just three pairs that seem possible candidates for future inclusion in the list of confusables - <*easy*, *easily*>, <*mouths*, *months*> and <*no*, *know*>.

Error not confusable headword		**Target not in confusion set**	
Pair	**Frequency**	**Pair**	**Frequency**
a, an	17	an, a	4
the, they	4	cause, because	3
is, his	2	as, has	2
is, it	2	easy, easily	2
i, it	2	for, from	2
u, your	2	in, is	2
		mouths, months	2
		none, non	2
		no, know	2

Table 5.4: Errors occurring more than once in the corpus but not included in confusion set listing

5.3 Using the sets for spellchecking

At runtime, when the spellchecker finds one of the headwords in the text, it will retrieve the associated confusion set from the dictionary and use some process to decide whether one of these confusables is more likely to be the word the user intended. As noted previously, this process can be based on syntax, semantics or any other aspect of the surrounding text. Although syntax-based methods are only able to make a decision when the headword and confusable differ in their parts of speech, they are the most straightforward to implement and have been shown to have good performance in cases where the error causes a syntactic anomaly (Atwell and Elliott, 1987; Golding and Schabes, 1996).

Previous analysis of the errors in my corpus had shown that a high proportion of them were likely to be amenable to a syntax-based approach; 66% of the <error, target> pairs had distinct part-of-speech tagsets and a further 24% differed in some but not all parts-of-speech, leaving just 10% with identical tagsets that would be indistinguishable syntactically (Table 3.10). A similar comparison of the tagsets of the generated confusable pairs shows that a far smaller proportion (just under a third) have distinct tagsets while almost half have some but not all part-of-speech tags in common and about a quarter have matching tagsets (Table 5.5). These different proportions can largely be accounted for by the number of inflection errors in the corpus which would have contributed to the high count for the distinct tagsets in the corpus error pairs but did not feature in the generated sets, as inflection errors were not considered as candidates for the confusion sets. However, a syntax-based approach would still be appropriate to distinguish between three-quarters of the confusable pairs.

Tagset type	**N. Pairs**	**Percentage**
Distinct	2330	30%
Overlap	3606	46%
Match	1940	24%
Total Pairs	7876	100%

Table 5.5: Tagset types for confusable pairs

The distinction between matching, overlapping and distinct tagets becomes slightly more complicated when there are two or more words in the confusion set. In this case, sets can only be considered distinct if the headword differs in its parts-of-speech from each of the words in its confusion set and each of the words in the set are distinct from each other. In general, the larger the set, the less likely this is to apply.[9] Consider the tags for *far* and its associated confusion set {*fair*, *fare*, *fear*, *fur*}, illustrated in Table 5.6, for example.

Confusable	**AJ0**	**AV0**	**NN1**	**VVB/VVI**
far	●	●		
fair	●	●	●	
fare			●	●
fear			●	●
fur			●	

Table 5.6: Part-of-speech tags for *far* and its associated confusion set

[9] An exception is the five member set for *were*, listed above, where all tagsets *are* distinct. This is largely because *were* itself has just one tag - VBD, past tense of BE - which only belongs to one other word - *was* - and also because the contracted form *we're* has a combination tag - PNP+VBB.

If we pair *far* with each of its confusables individually there are three distinct pairs <*far*, *fare*>, <*far*, *fear*> and <*far*, *fur*> and one overlapping pair – <*far*, *fair*> which suggests that syntax might help with making a decision. However, two of the confusables – *fare* and *fear* – have matching tagsets – both noun and verb – and the noun tags for the other two – *fair* and *fur* – overlap these. A syntax checker alone would be unable select between them. If it came across *far* in the text and decided that a noun or verb tag was more probable than an adverb or adjective, the best it could do would be to flag *far* as an error and propose a list of possible corrections – four if it preferred a noun or two if it preferred a verb. On the other hand, even if it selected one of *far*'s tags – adverb or adjective – it could not accept *far* as correct since *fair* would be equally grammatically acceptable.

The foregoing discussion has not considered how the syntax checker might arrive at its decision. For instance, if frequency was factored in it would prefer *far* (319) to *fair* (20) as an adverb although the distinction is less clear for an adjective – *far* (62), *fair* (78). However, this is not the place for a detailed consideration of the syntax checking process – that will be discussed in Chapter 8. This example is simply intended to illustrate some of the problems that can occur with larger confusion sets. It also demonstrates that even though a syntax checker might not be able to make a final decision for a set such as this, it could at least reduce the number of words left for consideration by some other means. On this basis I planned to implement a two-stage checking process using syntax followed by semantics.

5.4 Conclusion

I had now produced several thousand confusion sets and stored them in the dictionary ready for use by the spellchecker. The method used to create the sets was based on an algorithm tailored to produce accurate suggestions for non-word error correction which therefore might be expected to simulate the types of error that users actually make. Comparison with the corpus showed that they covered around three-fifths of the errors. How appropriate they would prove for their correction could only be assessed once they were used for spellchecking. Before I could do that I needed to prepare the text as described in the next chapter.

Chapter 6: Preparing the Text for Spellchecking

Text tokenisation – the process of splitting a text up into its constituent parts – is an essential pre-processing stage for all natural language applications. At a basic level, English words consist of sequences of characters delimited by white space and sentences are made up of sequences of words starting with an initial capital and ending with a full-stop, exclamation mark or question mark. Applying simple rules such as these is adequate to identify the majority – over 90% – of words and sentences in a text; but dealing accurately with the remainder is a non-trivial task. The accuracy achieved will have an impact on further stages of processing but some tasks are more tolerant of error than others; incorrect identification of sentence breaks will make little difference to the performance of an isolated, non-word error checker but is likely to have an adverse effect on a real-word error checker that needs to consider each word in the context of the sentence in which it occurs.

This chapter first discusses some of the difficulties involved in automatically segmenting text into words and sentences and then describes the way in which my spellchecker implements this process.

6.1 Automatic text segmentation

Text tokenisation is fraught with ambiguity and so difficult to achieve automatically. It is thus not surprising that the tokeniser for the AMALGAM multi-tagger (Atwell et al., 2000) comes with the caveat that this is a task "which really ought to be done by hand – or at the very least the tokenised output should be verified by a human reader." However, although this may be practical, and even desirable, for a small corpus which is to be stored in tokenised format for later processing, it is not an option for a spellchecker that needs to integrate tokenisation into its checking routine. The main areas of ambiguity are full-stops, abbreviations and capital letters (Booth, 1987; Mitton, 1996) which cause difficulties with both word and sentence segmentation.

6.1.1 Words

Each space-delimited string can be regarded as a 'word' in the text. Some of these will be numeric or alphanumeric strings and a few may be odd sequences of non-alphanumeric characters but the majority will be lexical words. A tokeniser can generally remove punctuation characters attached to the start or end of each word and store them as separate text tokens but full-stops and apostrophes need special treatment. Full-stops, although they are more frequently used as sentence terminators, also function as abbreviation markers while apostrophes, in addition to their more frequent usage as an opening single quote, also mark the start of contractions – such as *'em* or *'twas*. In these cases, the tokeniser's task is to distinguish between these two usages and decide whether the punctuation mark should be stripped or left attached.

Words can also contain embedded punctuation. Apostrophes are used to indicate contracted forms – *can't*, *they're, she's* – and the *'s* can also be a possessive marker. Plural possessives – the *students'* essays – have a trailing apostrophe which needs to be distinguished from a closing single quote. A tokeniser will often split word forms such as these into separate syntactic units – *ca* + *n't* or *they* + *'re* for example – which means that each part can be assigned its own part-of-speech tag. However, common contracted forms such as these can often be a source of spelling errors and so words such as *can't* and *they're* should be included as words in their own right in a spellchecking dictionary, as they are in mine. If the spellchecker is to attempt correction of such errors these contracted forms need to be left intact by the tokeniser. On the other hand, enclitic forms may also appear in combinations not found in the dictionary – "I *could've* been a contender", "*Chance'd* be a fine thing" or "*Jim'll* fix it". In such cases, the tokeniser should split the string so that the spellchecker can check each part separately. When an *'s* is appended to a word it may indicate possession, as in "*mum's* work is never done", or be a contraction of *is* or *has* - "*mum's* working hard" or "*mum's* gone out". The BNC tokeniser strips the *'s* in both cases and assigns a POS (possessive) tag to the first and a VBZ to the second whereas the AMALGAM tokeniser splits the contracted *is* or *has* but regards the possessive as part of the word and leaves it attached - although the documentation notes that this is a distinction which can be difficult to make. Although a tokeniser can use rules to deal successfully with many contracted forms there will still be unrecognised slang or dialect usages, particularly in direct speech, which will simply have to be left as a single form. The BNC takes this approach to *ain't*, tagging the whole word as unclassified as no suitable tag could be found for *ai*.

Full-stops, as well as marking the end of abbreviations, can be embedded within them – *U.K.*, *i.e.*, for example. Identifying abbreviations is particularly problematic for a tagger and one of the main causes of misclassified sentence boundaries (discussed further below). An abbreviation list is a partial solution – CUVPlus includes over 300 although acronym and abbreviation dictionaries available on the Internet – such as http://www.acronymfinder.com/ – list more than three million! It would be impractical for a tokeniser to use such a large list and, even if it could, however large the list, it would be unlikely to account for all possibilities. The best approach would seem to be to use a small list of the most common abbreviations combined with morphological rules to attempt to recognise the rest.

The problems here are that for almost every rule we can easily find an exception and that the same abbreviation may appear written in several different ways – *N.A.T.O.*, *NATO* and *Nato* all seem acceptable, for example. As acronyms such as this are generally pronounced as a word – "nay-toe" – rather than being spelled out, there is a tendency to omit the internal periods, and in some cases the acronym may become a word in the language in its own right - such as *radar* (which, as few people now remember, was originally an acronym for *radio detection and ranging*). Nevertheless, in most cases acronyms will appear as a sequence of uppercase letters, optionally including internal or trailing periods so this might suffice for an initial definition. However, many acronyms coincide with common words – *AIDS*, for example – and a distinction needs to be made between the acronym and an uppercase usage of the word - for emphasis or as part of a heading, for instance. In addition, several common abbreviations – *in*. (*inch*) or *no*. (*number*), for example – are also high frequency dictionary words in their own right and are likely to cause problems if they occur at the end of a sentence.

Because of the difficulty of defining rules to recognise abbreviations it may be more productive to dynamically infer them from the text as proposed by Mikheev (2002). A useful feature here is that, at least in fairly formal writing, less common acronyms are generally defined the first time they are used. For example, the acronym *PAP* which appears frequently in the first file of the FLOB corpus (Hundt et al., 1998, discussed further in Section 6.2.5 below) – largely made up of news articles – is written in full the first time it is used – " Singapore's ruling People's Action Party (PAP)...". Future use of *PAP* can now be assumed to be the acronym rather than an upper-case usage of the common noun *pap*. (In practice, the program would assign a probability to this based on whether the word also occurred in its lower-case form in the corpus.) Although such methods have been shown to increase the number of abbreviations correctly identified in large corpora they are likely to be of less use for a spellchecker when it is checking a relatively short document.

Tokenisers may also convert the first word of a sentence to lower-case, unless it is marked as a proper noun in the dictionary although, as the AMALGAM documentation notes, "This rule is not failsafe." Many common nouns are also names – *Bush*, *Thatcher*, *Gates* and so on, as I had noted when updating the dictionary – and so should not be treated in this way. Mikheev (2002) suggests that this problem can be dealt with in a similar way to that for abbreviations described above. Common nouns that appear with an initial capital in mid-sentence can be assumed to be functioning as proper nouns, the proportion of capitalised to non-capitalised mid-sentence usage in the corpus can then be used to assign a common/proper noun probability to such words when they appear at the start of a sentence. Again, this is likely to be of more use when processing a corpus than it is for spellchecking.

The question of hyphenated words was considered while updating the dictionary (Chapter 4). However, although these may cause difficulties for a spellchecker or part-of-speech tagger, they are not a problem for most tokenisers which simply accept the hyphen as an integral part of the word and store the hyphenated form as a single token.

The discussion above suggests that there are three main ambiguities to be resolved:

- the use of the apostrophe to mark possessives, contractions and single quoted expressions;
- identification of abbreviations;
- initial capitalisation of words at the start of a sentence.

Incorrect recognition of the role of an apostrophe is liable to cause difficulties for a tagger or spellchecker when it comes to process the tokenised text but it does not affect the tokeniser's ability to segment the text into sentences. On the other hand, decisions about abbreviations and capitalisation have a direct impact on this, so much so that Mikheev's application treats them as an integral part of the sentence boundary recognition task.

6.1.2 Sentences

Once the tokeniser has split the text into words, it needs to group these words into sentences. The most basic method of doing this is to treat all full-stops, exclamation marks, or question marks (and possibly also colons and semi-colons as the AMALGAM tokeniser does) as sentence delimiters when they are followed by white space and an upper case letter. Although this "period-space-capital letter" rule will apply in the majority of cases, Mikheev reports that 5 - 6 % of the sentence breaks in two large corpora were incorrectly identified using this method.

The main cause of error is, unsurprisingly, the ambiguity of the full-stop, as discussed above, but, as Mikheev notes, exclamation marks and question marks can also be integral parts of words as in *Yahoo!* (the search engine) and *Which?* (the consumer magazine). A method to automatically infer these from the text being tokenised could help with recognition. But although such cases may cause problems, in general it is full-stops that the tokeniser needs to disambiguate. Once it has recognised that a word is an abbreviation (using some combination of the methods discussed above) it needs to decide whether the abbreviation itself is also the last word in the sentence, in which case the full-stop also indicates a sentence boundary, or whether the words following it are a continuation of the same sentence. In most cases it can make this decision based on whether the next word has an initial capital – insert a sentence break if it does, otherwise continue the sentence. However, we need to include an exception to this rule for titles – *Mr*., *Dr*. and the like – which are almost always followed by a proper noun and would not be expected to occur as the last word of a sentence, and we should probably also include *i.e.* and *e.g.* in the list of non-sentence breaking abbreviations.

As the considerations introduced above show, the sentence boundary/initial capital is, as Mikheev comments, a "chicken and egg problem". If we know that a word with an initial capital following a full-stop is a common noun, we can conclude that the full-stop is a sentence terminal. If we know that the full-stop is not sentence terminal, we can conclude that the following word is a proper noun and that the word preceding it is an abbreviation. Although such situations only occur for a small number of sentences, a tokeniser needs to be prepared to deal with them if it is to segment the text accurately into sentences. Developing methods to do this robustly is time-consuming and labour-intensive. Palmer and Hearst (1997) report work (by Mark Wasson and colleagues, unpublished) on the development of a system to recognise special terms and sentence boundaries which took nine staff-months to complete. Although this achieved impressive performance (an error rate of between 0.3% and 1.8% on their test data) the small number of errors remaining suggests that this may well be a problem that cannot be solved with 100% accuracy without human intervention. Nevertheless, I needed to develop a tokeniser that would run automatically and be adequate for the requirements of my spellchecker. The next section describes its implementation.

6.2 Implementation

Dyslexic writing is often poorly punctuated as well as badly spelled; lack of sentence terminal punctuation and unreliable capitalisation means that a tokeniser is likely to have difficulty in accurately recognising all of the sentence breaks. However, as Atwell and Elliot (1987) comment "no system can be expected to cope with highly garbled English input" and I have not attempted to address this problem; if there is no punctuation to indicate the end of a sentence, my program simply assumes that the word it is checking belongs to the current sentence. This does not cause problems with the processing of the real-word error sub-corpus as each sentence extracted from the dyslexic error corpus is stored on a separate line and capital letters and full-stops have been inserted if they were missing.

6.2.1 Requirements

The real-word error checker that I have developed considers each word in the context of the sentence in which it occurs. Thus the first requirement of the tokeniser is that it segments the text into sentences. Each sentence needs to be stored as a sequence of tokens, each representing a word in the sentence. The majority of these will be lexical words but a few will be numeric, alphanumeric or some sequence of non-alphanumeric characters; the tokeniser needs to note this distinction. It also needs to strip and store any leading or trailing punctuation attached to the word. In this implementation, such punctuation is stored as an attribute of the token for the word to which it was attached. This means that it is easier for the semantic checker (Chapter 9) to consider words in a specified window width without including the punctuation in the window although the punctuation is available for use if required.

In addition to creating tokens for the words in the text, the tokeniser needs to recognise marked up errors in the format <ERR targ=targetword> errorword </ERR>, as described in Chapter 3, and store the target as well as the error in the token. The spellchecker will ignore the target words when it is checking the text but the program will use them to assess the correctness of its suggestions.

The spellchecker requires dictionary information for each word. In particular, it needs to know whether the word has a confusion set associated with it as this signals that the word is a potential real-word error. It also needs the set of part-of-speech tags and frequencies that will be used by the syntax checker (Chapter 8). Many of the words, particularly short function words, will occur several times in a text – the real word error corpus contains almost twelve thousand word tokens but only around five thousand word types. To avoid the inefficiency and extra memory overhead of including the dictionary information for each word token, the tokeniser stores this in a 'lexical token' for each word type. Thus, at the end of the tokenising phase, each word has two tokens associated with it – a 'text token' stored with the sentence in which it occurred and a related 'lexical token' containing its dictionary information.

6.2.2 Sentence segmentation

The program starts by reading the file up to the next new-line character. This input line may be a single sentence, a fragment of a sentence or several sentences, depending on the formatting of the file. Although in the majority of cases a new-line character will signal the end of a sentence, the tokeniser does not make this assumption as this will not be the case if the input file has fixed-length lines or if the user has inserted line breaks to control the length of the line (as many inexperienced users of word-processors do).

The tokeniser splits the input line into a sequence of space-delimited strings and/or error tags. It then creates a text token for each 'word' with the attributes shown in Table 6.1.

`type:`	word, numeric, alphanumeric or unclassified
`textStr:`	the word as it appeared in the text
`targ:`	the target spelling if the word was marked as an error
`leadPun:`	any leading punctuation which was attached to the word
`trailPun:`	any trailing punctuation which was attached to the word

Table 6.1: Initial attributes for text tokens

The `type` and `textStr` attributes will be stored for each token – the other attributes may be empty for many of the tokens. The tokeniser does not look up the words in the dictionary at this stage. It simply checks whether a string could correspond to a dictionary word; embedded full-stops, apostrophes and hyphens are considered to be part of the word.

In addition to punctuation characters that legitimately form part of a word, punctuation can mistakenly become embedded if the user omits to type a space after the punctuation mark. Rather than leaving such strings as a single word, the tokeniser splits such strings and stores the punctuation character in the trailing punctuation attribute of the first string. To avoid splitting abbreviations, it only splits on a full-stop if there are more than two letters preceding it.

Once the text token has been created it is stored either as part of the current sentence or as the start of a new sentence. If the last token stored does not end in sentence delimiting punctuation – defined as full-stop, exclamation mark or question mark – the token is stored as part of the current sentence. If the preceding token ends with an exclamation mark or question mark, the token is assumed to be the start of a new sentence if the word has an initial uppercase character, otherwise it is assumed to continue the current sentence. If the preceding token ends in a full-stop and the current word starts with a capital letter, the current token will be stored as the first word of a new sentence unless the previous word was a non-sentence-breaking abbreviation, defined as a title – *Mr.*, *Mrs.* and so on – or *e.g./ i.e.* For all other abbreviations, the program assumes that the full-stop is serving both as an abbreviation marker and sentence terminal. If the current word starts with a lowercase letter and the preceding token ends with a full-stop but is not included in the dictionary, the preceding token is assumed to be an unknown abbreviation and the current token is stored with the current sentence.

At the end of this stage of processing, each word in the text has been stored in a text token in the format described above and the sequences of text tokens have been segmented into sentences.

6.2.3 Word storage

The tokeniser next creates a lexical token containing the dictionary information for each word type in the text and for any confusable words associated with it. Each of these tokens has the attributes shown in Table 6.2.

`tags:`	the set of part-of-speech tags from the dictionary with their associated frequencies
`capFlag:`	the capitalisation flag from the dictionary or assigned by the tokeniser for words not found in the dictionary
`abbrev:`	a flag indicating whether or not the word is an abbreviation
`indict:`	a flag indicating whether or not the word was found in the dictionary
`count:`	the number of times the word occurred in the text being checked. (This will be zero for confusion set members that do not appear in the text.)
`confSet:`	a list of confusables (if any) associated with the word.

Table 6.2: Attributes for lexical tokens

To create these tokens the tokeniser checks each word text token. There are four possibilities:

- the word has already been stored;
- the word is found in the dictionary in the same format as it appeared in the text;
- the word is found in the dictionary but the dictionary format does not match the format of the word in the text;
- the word is not found in the dictionary.

In the simplest case, when the word is already stored, the tokeniser increments the word-count for the existing lexical token and, if the count was zero (indicating that the lexical token had been created for a confusable rather than a word that had previously been seen in the text) checks whether there is an associated confusion set and, if so, stores a list of the confusables in the word's lexical token as well as creating a lexical token with a zero word-count for any confusable that had not previously appeared in the text. The case where the text word matches the dictionary format is also straightforward – create a new lexical token for the word and lexical tokens for any of its confusables that do not already have lexical tokens.

As all words in the dictionary are stored with a lower-case initial letter and abbreviations are stored without embedded punctuation, sentence initial words (assuming they start with a capital letter), proper nouns and abbreviations will appear in the text in a format that does not match the way in which they are stored in the dictionary. The dictionary lookup function converts words into 'dictionary format' (by converting the initial letter to lower case and removing internal periods) before checking them and returns the word in the form in which it has been found in the dictionary. When this does not match the format of the word in the text the tokeniser first checks the capitalisation flag in the dictionary to see whether the word would normally be written with an upper case initial letter. In this case a lexical token is created for the word in dictionary format (i.e. with a lowercase initial letter). If the word would not normally be expected to have an uppercase initial, the program checks whether it is the first word of a sentence. In this case, the program makes the assumption that this is the reason for the capitalisation and creates a lexical token for the lowercase version as above. This will deal incorrectly with surnames such as *Bush* or *Smith,* which are tagged as common nouns in the dictionary, when they appear as the first word of a sentence. However, although such occurrences are fairly frequent in newswire text (which was used for testing the tokenisers discussed in section 6.1 above), they are relatively infrequent in more general text so I have not incorporated a method for disambiguating such occurrences in the program. Conversely, non-function words with an initial capital in mid-sentence are assumed to be proper nouns and are stored with an uppercase initial in the lexical token.

The final category of words is those that are not found in the dictionary at all – hyphens, apostrophes and capitalised words are particularly problematic. Almost any pair (or sequence) of words can be hyphenated and, as discussed in Chapter 4, the use of the hyphen to form compound words is inconsistent. Any hyphenated word in the text which is not found in the dictionary is split on the hyphen(s) and each word is then checked. If all the words are found, the complete word is assigned the tag of the final part. If any part is not found, an unclassified tag is assigned.

Shortened forms – *'em*, *'twas* and so on – are stored in the dictionary with a leading apostrophe but the token retrieved from the text will have had its apostrophe removed by the punctuation stripper. If a token has an apostrophe stored in its `leadPunct` attribute, the word is rechecked in the dictionary with the apostrophe reattached. If it is found, a lexical token is created for the shortened form and the text token modified to include the apostrophe with the word. If the word is not one of the shortened forms included in the dictionary, a lexical token is created for the plain word and it is assigned an unclassified tag.

Although many common shortened forms – *can't*, *they're* and so on – are stored in the dictionary, trailing enclitics – *'d*, *'ll* etc.– can also be attached to other words, as discussed above. In such cases, the enclitic is stripped and the word rechecked. If it is found in the dictionary, it will be stored in the lexical token without the trailing enclitic and the text token will be modified to store the enclitic in a `contract` attribute and the plain word in the `textStr` attribute. When the syntax checker tags the text, it will assign a combination tag to such tokens in the same format as the combination tags stored in the dictionary.

Words written entirely in uppercase may be capitalised out of convention, such as in a header, or for emphasis and in this case will often appear in the dictionary in their lowercase form. However, as noted above, many acronyms which are not listed in the dictionary are themselves dictionary words. Rather than attempt to distinguish such uses, the program simply regards all uppercase words as unknown abbreviations and creates a lexical token for the capitalised form. Words beginning with an uppercase letter are assumed to be unknown proper nouns and stored in the lexical token in this form with a proper noun tag.

During this stage of tokenisation, two additional attributes are stored with each text token as shown in Table 6.3.

`lexStr:`	the word in dictionary format (or the format in which it has been stored in its corresponding lexical token as detailed above).
`contract:`	trailing enclitics stripped from shortened forms not stored in the dictionary

Table 6.3: Additional attributes for text tokens

At the end of the tokenisation phase, the program has split the text into sentences and words. It has also stored dictionary information required by the spellchecker. Any words not found in the dictionary and not classified by any of the rules outlined above could be flagged as potential errors by a non-word error checker. However, as I am making the assumption that non-word errors have already been corrected (as they have been in the sub-corpus), my program makes no attempt to deal with these and simply tags them as unclassified strings.

6.2.4 A tokenisation example

The array of text tokens created to store the following sentence (assuming it to be the first sentence in a file) is shown in Table 6.4:

Mum's <ERR targ=not> note </ERR> here!

	1	2	3
1	`type`: W `textStr`: Mum `lexStr`: mum `contract`: 's	`type`: W `textStr`: note `targ`: not `lexStr`: note	`type`: W `textStr`: here `trailPun`: ! `lexStr`: here

Table 6.4: Text tokens created for example sentence

The sentence is initially split into a sequence of space-delimited strings with error tokens being treated as a single string. Each string is stored as a text token in the sentence array with the attributes listed in Table 6.1 and Table 6.3. Some attributes will not have a value for all tokens and will be set to `undef.` Such attributes have not been included in the table.

The majority of strings (all in this example) will correspond to lexical words and have their `type` attribute set to ‘W’. Numeric strings (1952), alphanumeric strings (P40), or unclassified strings ($%!) will be assigned a `type` of N, AN or U respectively. Any leading and/or trailing punctuation (the exclamation mark attached to *here!* in this example) is then stripped from each string and stored in the `leadPun` or `trailPun` attribute. Contractions attached to the end of non-word strings will be removed and stored in the `contract` attribute. However, for word strings the contraction will only be removed if the contracted form does not appear in the dictionary. In this example the *‘s* is removed from *Mum’s* but if the first word of the sentence was changed to *She’s* the *‘s* would not be stripped as the contracted form - *she’s* has its own entry in the dictionary.

After the punctuation and contractions have been stripped the remaining string, in the format it appeared in the text, is stored in the `textStr` attribute; by recombining this with the `contract` and `leadPun`/`trailPun` attributes the program is able to reproduce the text in its original format if required. The `lexStr` attribute stores the word as it appears in the dictionary and matches the key for the corresponding lexical token (described further below and illustrated in Table 6.5). Both attributes have the same value for all words in this example apart from *Mum* which is stored with its upper case initial in the `textStr` but with a lowercase initial in the `lexStr`.

If a word is marked-up as an error – *note* in this example – the `targ` attribute will be set to the intended word – *not*. This allows the program to check the correctness of its suggestions when run over the error corpus.

Each text token that contains a lexical word (`type` = W) has an associated lexical token stored in a hash. Whereas text tokens are created for each word *token* in the text, lexical tokens are created just once for each word *type*. The `lexStr` attribute of each text token provides the key for lookup in the lexical token hash to enable the program to retrieve the associated dictionary information for each word in the text.

The lexical tokens created for the example sentence above are shown in Table 6.5. All the elements have the same value for `capFlag` (0), `abbrev` (N), `indict` (Y), so these have not been included in the table:. The `confSet` attribute will be `undef` for words that do not have a confusion set associated with them, and this also is not included in the table. Lexical tokens are also created for each member of the confusion set but the count for these will be set to zero (or not incremented if the confusable has previously been seen in the text and already has a lexical token). Confusion sets are not stored for the confusables themselves unless they also appear in the text in their own right.

Key	**Attributes**
hear	`tags`: PRP:4374 `count`: 0
here	`tags`: AV0:563 `count`: 1 `confSet`: hear
mum	`tags:` NN1:44,AJ0:0,ITJ:0 `count`: 1
not	`tags`: NN1:287,VVI:52,VVB:34 `count`: 0
note	`Tags`:NN1:59,VVB:32,VVI:18 `count`: 1 `confSet`: not

Table 6.5: Lexical tokens created for example fragment

6.2.5 Sentence splitting evaluated

I used the million-word FLOB corpus (Hundt et al. 1998) to test the sentence splitter. This corpus, created at Freiburg University, contains a mixture of newspaper articles, book extracts and miscellaneous publications covering a broad range of written British English from 1991 and 1992. It was designed to parallel the earlier LOB corpus (discussed in Section 8.3.4 below). The version of the corpus included in the ICAME corpus CD (ICAME, 1999) is stored in fixed-length lines with some textual mark-up such as paragraph and quotation start and end tags. I had previously re-written this as continuous text with no mark-up and used this ‘plain’ version to test the sentence segmentation program.

After checking several files from the corpus, a few problems came to light. When names were preceded by initials – *G.A. Henty*, for example – the program split the sentence after the second initial. It would be possible to avoid this by defining a sequence of uppercase characters separated by periods and followed by a non-dictionary word as an initial/name combination that should not be split. However, as noted previously, many surnames also appear as common nouns in the dictionary so this rule would not apply to, for example, *G.W. Bush*. Additionally, as for many text tokenization rules, it is easy to think of a counter-example where the sentence *should* be split such as:

They met in downtown L.A. Kerouac had described…

An attempt to deal accurately with such ambiguities would seem to require two passes through the text; the first to mark potentially ambiguous sentence breaks and the second to disambiguate them using, for example, a list of mid-sentence capitalised words that could be assumed to be proper nouns or a more detailed syntactic analysis. However, examples such as these occurred infrequently in the FLOB texts.

As the program was set to check for sentence breaks when it encountered sentence terminal punctuation followed by a word with an initial capital it did not insert a sentence break for ill-formed sentences such as this example (from FLOB):

The result of this clever charade was that for two years my
rates were assessed on the basis of seven feet, eight inches
square of living space. another name.

However, using only sentence terminal punctuation as the basis for sentence segmentation results in an incorrect split after *Rehab*. in this fragment from my error corpus:

The Dept. of Rehab. is going to pay...

A second pass through the text to flag potential sentence initial capitalisation errors could be used to deal with such cases but the current implementation is restricted to a single pass and so runs sentences together if the first word does not start with a capital.

In most running text, line breaks correspond to the end of a paragraph and so will also mark a sentence boundary (although the paragraph itself may contain several sentences). However, as discussed above, I had decided to allow for the possibility of mid-sentence line breaks. This caused the two utterances in the following extract from FLOB to be run together into a single sentence as the first does not end with one of my predefined set of sentence terminals:

> "That is as may be, ma'am," Jake responded, "but I would
> rather know the circumstances so that – "
>
> "So that you may try to order my life as they have done,"
> Clementina finished for him.

This suggests either that it might be better always to regard a line break as indicating the end of a sentence or that additional rules are required to deal with direct speech. Such issues have not been considered in the current implementation.

Apart from a few cases as outlined above, the majority of the sentences in the FLOB files that I checked were split correctly. In FLOB_J, mainly academic writing, there were no errors apparent in the first two thousand or so sentences – a sample illustrating the segmentation of the first 50 sentences of this file is included as Appendix C.

6.3 Conclusion

Tokenisation, although it is an essential pre-processing phase for all natural language processing applications, seems to be a task that is difficult to perform accurately without human intervention. However, the spellchecker has to do this automatically. The method described above should deal adequately with most of its requirements. Modifications can be made at a later stage if inaccuracies in the tokenisation seem to cause particular problems for the spellchecker.

Chapter 7: A Frequency-only Spellchecker as a Baseline

We now have everything in place to begin spellchecking – a dictionary with reasonably accurate word frequencies, a large number of confusion sets, a method for splitting the text into words and a corpus containing real-word spelling errors made by dyslexic writers. This chapter considers ways in which to assess the performance of the spellchecking methods described in the next two chapters. It also sets a baseline against which further developments can be compared.

7.1 Expectations

In the detection phase, the desirable outcomes for a spellchecker are:

- Accept correctly spelled words
- Flag incorrectly spelled words as errors

The undesirable outcomes are:

- Flag correctly spelled words as errors
- Accept incorrectly spelled words as correct

Flagging correctly spelled words produces a false alarm which is annoying for the user and can result in errors being introduced into the text if the program's suggestion is accepted. The task of the spellchecker, therefore, is to minimise these false alarms while at the same time maximising error detection.

Once an error has been detected, the spellchecker needs to produce a list of suggested corrections, ideally with the intended word in first position. One of the attractions of the confusion set approach is that it can combine these two tasks of detection and correction; it decides that a word is an error if one of the alternative words in its confusion set appears to be more likely in the context and then proposes that alternative as a correction. This means that it only suggests one word but, as there are at most five words in the confusion sets, even if it used the whole set, ordered by likelihood, as its suggestion list, it would produce a short list which would likely be more useful than many of the lists discussed in Chapter 1.

However, when a confusable word is an error, the intended word is not necessarily another member of the confusion set and, if the spellchecker is limited to making suggestions from the associated confusion set (which is generally the way in which the confusion set method is implemented), it will not be able to correct the error in such cases. Take, for example, a sequence from my error corpus – "the *lose* of...". Here *lose* is mistakenly produced for *loss* but the confusion set for *lose* – *{loos, loose, louse}* - does not include *loss*. A syntax-based checker (such as the implementation I will describe in Chapter 8), if it is restricted to suggestions from the confusion set, will find that *loose*, is a better syntactic fit than *lose* and so suggest *loose* as a correction; although it flags the error it proposes an incorrect replacement. This should also be regarded as an undesirable outcome as, if the spellchecker's suggestion is accepted, as it often will be by dyslexic users, it simply replaces one misspelling with another.

7.2 Test data

During the development phase, detailed in the next two chapters, I used the real-word error corpus, described in Chapter 3, to test and refine the program. An additional corpus of errors was created for the final testing - described in Chapter 10.

Table 7.1 shows the number of confusables in the corpus. As can be seen, around half of the total words in the corpus are confusables, but the vast majority of them (89%) are correctly used.

	N. Tokens	N. Types
Total words	11809	4856
Total confusables	5838	758
Correct usage	5207	595
Error usage	631	294

Table 7.1: Count of words and confusables in test data

7.3 Establishing a baseline

A convenient way to set a baseline against which to compare the performance of subsequent developments is to establish how often the spellchecker would make the correct decision if it was simply set to select the more frequent confusable. (This will actually be the *most* frequent confusable in a set with more than two members but the spellchecker will always be making a decision between a pair of words – the word it has encountered in the text and the highest frequency word of its associated confusion set.) In other words, in this mode, the spellchecker is simply guessing which confusable to select based on the dictionary frequencies – which for my dictionary represent how often the word occurred in the BNC.

For a pair of words such as <*college, collage*> with dictionary frequencies of 107 and 2 respectively, it has a high probability of making the correct selection – *college* accounts for 98% of the overall occurrences of this pair of words. On the other hand, for a high frequency pair such as <*their*, *there*> where the frequencies are very close – 2772 and 2646 respectively – we would only expect it to be correct about half of the time. Other pairs of words, such as <*from* (4374), *form* (288)> and <*whether* (362), *weather* (58)>, fall between these two extremes.

This "guesses based on the priors" method is used by Golding and Schabes (1996) to set the baseline for assessment of their Tribayes spellchecker. Their program, like much of the other research previously discussed, was tested on correct data and "corrupted" data – data in which a correctly spelled confusable was changed to another member of its set. When errors are artificially introduced in this way the correct word will always be a member of the confusion set, which is not always the case with real errors as the *lose*/*loss* example discussed above shows.

In the case of *lose/loss*, the error will be ignored (*loss* has a higher frequency than *lose* but it is not in the confusion set). In another example from my corpus, "he *shard* the people...", where *scared* is the intended word, the error will be flagged as the single word in the confusion set for *shard* is *shared*. *Shared* has the higher frequency but is an incorrect replacement. There are a number of other similar errors in the corpus. Many of these are inflection errors – *apple* in mistake for *apples* for example; others are typos such as *bet* mistakenly typed for *get*. Thus, for real data, we have a third possibility which does not occur with artificial errors – that of detecting the error but not correcting it. This can also occur when the confusion set consists of more than two words and one of the less frequent words is produced in error for another of the less frequent. For instance, *they're* is also a member of the *{their, there*} confusion set. So, if the spellchecker encountered the sentence "*There* going swimming", where *there* is an error for *they're*, it would flag *there* as an error but propose *their* (the most frequent member of the set) as a correction.

Apart from the set {*their, there, they're*}, all of the 18 sets used in the Golding and Schabes (1996) experiment (which are similar to the sets used in many of the other experiments previously discussed) are pairs of words so the possibility discussed above will never occur.

A 'dumb' spellchecker, simply set to select the highest frequency word whenever it encounters a confusion set member in the text, will raise a false alarm for correct usages of less frequent words and fail to detect errors when a more frequent word is produced as an error for one of its less frequent counterparts. Despite this, it will make the correct decision in the majority of cases since, as Table 7.2 shows, a high proportion (84%) of the correct usages are more frequent confusables while the majority of the errors (62%) occur when a less frequent word is produced in error for a more frequent.

	Correct usage		**Error usage**	
More frequent word	4392	84%	242	38%
Less frequent word	815	16%	398	62%
Total	5207	100%	631	100%

Table 7.2: Frequencies for correct and error usage of confusable words in the test data

However, correctly used confusables form a larger overall proportion of all confusables occurring in the text meaning that the number of false alarms raised (815) will outnumber the errors detected (398). In addition, as discussed above, even when the dumb checker does detect an error it will not always manage to correct it. Table 7.3 shows its performance on the error corpus taking into account the three possible outcomes for errors - flag and correct the error, flag but not correct the error or ignore the error. As can be seen, there are 815 false alarms as opposed to 278 errors corrected. This means that if the spellchecker's suggestion was accepted each time, the text would end up containing more errors than it had done to start with which is clearly not helpful.

Correct usage		**Error usage**		
Accept	**False alarm**	**Flag & correct**	**Flag not correct**	**Ignore**
4392	815	278	111	242

Table 7.3: Initial performance of select-the-most-frequent spellchecker

For the initial run the spellchecker was set to simply select the most frequent word, regardless of how much more frequent it was. However, as we saw earlier, the relative difference in frequencies is much greater for some pairs of words than others. This relative difference in frequency can be used to set a confidence level for the spellchecker. A confidence level of 0.5 is equivalent to simply selecting the more frequent word – a word must occur more than 50% of the time if it is more frequent than the other member of its pair. Setting a confidence level of 0.6 would mean that the program would not suggest changing the word unless it occurred for more than 60% of the total occurrences of the pair, 0.7 would increase this to 70% and so on.

Successively increasing the confidence level in this way reduces the false alarms but also, of course, reduces the number of errors that are corrected. Table 7.4 shows the effect of increasing the confidence level from 0.5 (the default which simply selects the member of the pair with the highest frequency in the dictionary) to 0.99. Even at the maximum level there are still eight false alarms and only 12 of the errors are corrected – this small gain in overall 'correctness' is hardly worth considering. Up to the 0.9 level, the false alarms outnumber the errors corrected.

	Correct usage		**Error usage**		
Confidence Level	**Accept**	**False alarm**	**Flag & correct**	**Flag not correct**	**Ignore**
0.5	4392	815	278	111	242
0.6	4638	569	231	96	304
0.7	4809	398	205	82	344
0.8	4952	255	182	52	397
0.9	5057	150	145	32	454
0.99	5199	8	44	12	575

Table 7.4: Performance of the 'select the most frequent' checker with differing levels of confidence

Although in general the 'select the most frequent' method is not an effective way of correcting real-word errors, it may be appropriate for some pairs where one of the words occurs almost all of the time – such as <*your*, *yore*> for example.

7.4 Conclusion

This chapter has demonstrated that word frequency alone is insufficient for a real-word error checker although it is a factor which needs to be included in the spellchecker's overall decision. However, measuring performance based on word frequency alone provides a useful baseline against which the approaches discussed in the next two chapters can be compared.

Chapter 8: A Syntax-based Spellchecker

Although the confusion set approach is not restricted to correcting syntactic errors, syntax is a useful starting point. A syntax-based method could be expected to have reasonable performance for confusable pairs with distinct syntactic tags – {*advice, advise*} or {*from, form*}, for example; these make up 30% of the pairs in my confusion sets. It might also have some success where the tagsets overlap, such as {*loose, lose*} where both are verbs but only *loose* is also an adjective; these account for a further 46% of the pairs. This leaves just 24% that would have to be dealt with in some other way.

It appears that an effective approach may be to use a syntax checker first followed by semantic or other processing for the cases where it is unable to make a decision and this is borne out by research by Golding and Schabes (1996) which shows that a part-of-speech trigram tag approach outperforms their feature-based approach for confusion sets which differ in their parts-of-speech.

This chapter describes the development of a 'confusion tagger' which, in addition to assigning a part-of-speech tag to each word in the text, uses the tagging algorithm to select the confusion set member with the best syntactic fit. I will first describe the method used to tag the text and then show how this is extended to check confusable words.

8.1 Assigning tags to words

For words, such as *and* or *wood*, which only have a single part of speech (CJC and NN1 respectively), the tagging task is trivial – retrieve the tag from the dictionary. This is the case for 72% of the 72,000 words in cuvPlus. The remaining 28% have between two and seven tags each. In most cases these multi-tagged words more commonly belong to one word class than another. This preference for a particular part-of-speech tag is particularly notable for high frequency words. Take *can* for example. It occurs far and away most commonly as a modal auxiliary verb (VM0) with a dictionary frequency of 1992 per million while occurring only eight times per million as a noun (NN1) and less than once per million with its two verb tags – VVI and VVB. Short, common words such as this – *go* and *do* are other examples – are an extreme but the bias is also apparent in medium frequency words. *Form*, for example, is used more frequently as a noun (281) than as a verb (VVI:52, VVB: 34). How should a part-of-speech tagger decide which tag to assign in these cases?

One approach which has been demonstrated to have a surprisingly high degree of success (Atwell, 1987) is simply to assign the most frequent tag. Charniak et al (1993) found that the correct tag was selected just over 90% of the time by this method. However, this is less remarkable than it seems at first when we consider that the proportion of times many commonly used words are used with their most frequent tag is often higher than this. From the frequencies given above, for example, we can see that *can* is tagged VM0 99% of the time so choosing VM0 for *can* will be right nearly all the time, but it will clearly fail for a good many ordinary sentences. It would not, for example, assign a noun tag in the sequence "a *can* of beans" or a verb tag in "to *can* the fruit".

To make a more informed decision the tagger needs to combine the tag information from the dictionary with knowledge about the likelihood of particular sequences of tags occurring. For instance, given the information that nouns frequently follow an article it could correctly tag *can* as a noun in the first example above. Following the approach taken to obtain the tag transition matrix used by the CLAWS tagger (Garside (1987); Marshall (1987)), this tag sequence information can be captured as tag bigram probabilities derived from a large corpus. When faced with a choice between several tags for a word, the tagger can then use these probabilities to find out how likely each tag is to follow the tag of the preceding word and combine this with the probability of the word itself occurring with each tag to find which tag is most likely to occur in the context.

Although just over three-quarters of the words in my dictionary (CUVPlus, Chapter 4) have just one tag in their tagset, the proportion of words in running text that have a single tag is much smaller than this as many commonly used words have more than one tag. For example, these single-tag words make up just 44% of the 85,000 words in the first file[10] from the FLOB corpus and so can be unambiguously tagged. Of the multi-tagged words, 18% occur sandwiched between two single-tag words, 36% in sequences of between two and five multi-tagged words, while the remainder appear in sequences of up to 16 multi-tagged words in length. The longest of these is shown below, together with the tagset for each of the words with the tags listed in descending order of frequency.

> **...<VBD>was** <AJ0,AV0,NN1>deep <NN1,VVI,VVB>concern <CJT,DT0,AJ0,AV0>that <AT0,AV0>the <AJ0,AV0>new <NN1,VVB,VVI>plan <VM0,NP0,NN1>may <NN1,VVI,VVB>face <AT0,AV0>the <DT0,AJ0,AV0>same <NN1,VVB,VVI>fate <CJS,PRP,AV0>as <AV0,AJC>earlier <NN2,VVZ>attempts <TO0,PRP,AV0>to <VVI,VVB>bring **<NN1>peace...**

It should be noted that one of the causes of the exceptional length of this sequence is the inclusion of the adverb (AV0) tag for *the*. This usage is confined to comparative adjective or adverb phrases such as "*the* more *the* merrier" and since in the overwhelming majority of cases *the* functions as an article, a tagger will make very few mistakes if it simply assigns an AT0 tag in all cases. This will also reduce the length of the multi-tag sequences and is the approach I have adopted with the tagger implemented in this research. Adverbial usage could then be dealt with as a special 'idiom tag' although I have not implemented this.

The tags shown in the sequence above, and used in the rest of this chapter, are from the C5 tagset (used for the tagging of the BNC and incorporated into my dictionary as described in Chapter 4). For the most part they are fairly mnemonic but for the reader who is unfamiliar with these tags and wishes to distinguish accurately between them a complete listing is included in the dictionary documentation (Appendix A).

To decide which tag should be assigned to a word, the tagger needs to find the most likely tag path through a multi-tag sequence using the single-tag words as anchors. For a multi-tag word that occurs immediately after one single-tag word and immediately before another, this is a fairly

[10] FLOB_A.TXT

straightforward process as the number of possibilities to consider is the same as the number of tags for the word. However, longer sequences have many possible paths – for example, there are over five million tag paths through the 16 word sequence given above.

I first explain how the tag bigram probabilities were obtained, and then I describe the tagging algorithm in detail.

8.2 Calculating the tag bigram probabilities

To calculate the bigram probabilities needed by the tagger, I counted the number of occurrences of each tag pair in the written section of the BNC (World Edition). In this section I describe how these counts were used to calculate the conditional probability of each tag occurring given the preceding tag and how the resulting probabilities were then smoothed to remove zero values which could cause problems for the tagger.

8.2.1 Tag pair frequency

There are 61 part-of-speech tags in the C5 tagset used in the BNC, including five punctuation tags and the UNC tag assigned to "words that are not properly considered part of the English lexicon." Two sentence delimiter markers – BOS and EOS – were added to these to mark the beginning and end of sentences. Combining these gives a total of 3843 pairs (BOS can only occur as the first element and EOS as the last).

The most frequent pair was <AT0, NN1> (article followed by singular noun) with over four million occurrences. Twelve tags were followed at least once by every other tag but 279 pairs (7% of the total possible combinations) did not occur. The majority of these are grammatically unacceptable – for example, auxiliary verb combinations such as VHD (*had*) followed by VDI (*do*).

We can assign a probability estimate to each tag pair by calculating the relative frequency with which it occurred in the BNC – the count of each tag pair divided by the total count of tag pairs. This is known as the Maximum Likelihood Estimate (MLE) as it maximises the probability weightings given to the tag pairs that did appear in the BNC while assigning a zero weighting to those that didn't. The P_{MLE} values for the tag pairs are calculated as follows:

Given

N Total count of tag pairs in BNC

$C(t_i,t_j)$ Count of occurrences of tag pair t_i,t_j

$$P_{MLE}(t_i,t_j) = \frac{C(t_i,t_j)}{N}$$

8.2.2 Conditional tag probabilities

The probability estimates calculated above tell us how likely a particular pair of tags is to occur (in the BNC) - $P(t_i,t_j)$ - but what the tagger needs to know, in order to assign a tag to a multi-tag word, is the likelihood of a particular tag occurring following another tag - $P(t_j|t_i)$. This conditional probability is calculated by dividing the total occurrences of the pair, $C(t_i,t_j)$, by the count of occurrences for the first tag, $C(t_i)$, as follows:

$$P_{MLE}(t_j|t_i) = \frac{C(t_i,t_j)}{C(t_i)}$$

Again this is a Maximum Likelihood Estimate so for pairs that did not occur in the BNC the P_{MLE} value will be zero. It would seem reasonable, in a large corpus, to expect that superfluous tag combinations would be seen rather than acceptable ones being unseen so we could simply retain the zero values, assuming that such sequences would never occur in text or that if they did they were an error. However, this has the potential to cause problems for the tagger as it uses the product of the conditional tag pair probabilities to calculate the most likely tag sequence for a sequence of multi-tag words so retaining these zero frequencies would mean that any tag sequence containing an unseen pair would have an overall value of zero. The solution adopted by the CLAWS tagger is to arrange that "a small positive value is associated with any transition that fails to occur in the sample" (Marshall, 1987).

8.2.3 Smoothing

There are several smoothing methods that enable us to assign such a small but non-zero value to the unseen pairs in such a way that the overall probability for all the tag pairs (seen and unseen) still sums to one.

One simple method is just to add one to all the counts – in other words to pretend that we saw each of the pairs once more than we actually did. This results in the zero count pairs now having a value of 1 and we can proceed to calculate the conditional probabilities as above. As Gale and Church (1994) demonstrate, this results in overweighting the unseen pairs. In a detailed comparison of several smoothing methods, Chen and Goodman (1996) suggest that the Good-Turing (Good, 1953) smoothing algorithm gives the best results.

The first stage in Good-Turing smoothing is to group the items (tag pairs in this case) in the dataset by frequency r and count the number of items N_r that occurred with each frequency. For the tag-pair dataset, N_1 = 128 as 128 of the tag pairs occurred just once, N_2 = 75 as 75 pairs occurred twice and so on. Table 8.1 shows the first eight and last two of these 'frequency of frequency' counts for the tag pairs together with the count for the unseen tag pairs, N_0 = 279.

The total number of tag pairs that were seen, N, is calculated as $N = \Sigma\, rN_r$. For this dataset, N = 104479260.

Using this notation, the probability for a tag pair occurring with frequency r can be calculated as $p(r) = \frac{rN_r}{N}$ but this is the MLE that was rejected above as it results in $p(0) = 0$ which is what we are trying to avoid. To overcome this problem, and assign some (small) probability to the unseen tag pairs, the Good-Turing process calculates the total probability for unseen objects as N_1/N (the count of things seen once divided by the total number of things seen). For this dataset this is calculated as 128/104479260 = 1.225e-06. This probability 'mass' is shared between the 279 unseen tag pairs as described further below.

Frequency ***r***	**Frequency of frequency** **N*r***
0	279
1	128
2	75
3	57
4	56
5	42
6	28
7	22
8	48
...	...
4158079	1
4302215	1

Table 8.1: Frequency of frequency counts for tag pairs

By making this adjustment to the probability of the unseen items we are, in effect, increasing the frequency with which we expect them to occur in the future from zero to 0.000001225. In other words, although we don't expect that we will see these particular tag pairs very often, we are not altogether ruling out the possibility of them occurring. To compensate for this, we also need to adjust the frequencies for the tag pairs that were seen by replacing the frequency *r* with an adjusted frequency *r**.

The Turing Estimator, which underlies the Good-Turing smoothing method, calculates *r** as:

$$r^* = (r+1)\frac{N_r + 1}{N_r} \quad \text{where } r \geq 1$$

For *r* = 1, in this dataset, this gives us:

$$1^{*} = 2\text{x}\frac{75}{128} = \frac{150}{128} = 1.17$$

The adjusted *r** frequencies calculated in this way for the first eight frequencies *r* (shown in Table 8.1) are given in Table 8.2.

Frequency *r*	**Frequency of frequency** N_r	**Adjusted frequency** r^*
1	128	1.17
2	75	2.28
3	57	3.93
4	56	3.75
5	42	4.00
6	28	5.50
7	22	17.45
8	48	3.94

Table 8.2: Adjusted frequency r^*

As can be seen from Table 8.2, using the Turing Estimator to calculate *r** for small values of *r* seems reasonable (1^* and 2^* above, for example), but it becomes less reliable as *r* increases (notably 7^* has a much larger value than we would expect and even 3^* seems rather high). This is because although in general as *r* increases N_r decreases, this relationship does not hold true for all values of *r* – the frequency for N_8 is more than double that for N_7 in this dataset, for example. In addition, for high values of *r*, many of the N_r are zero meaning that *r** calculated in this way would also end up with a value of zero (as can be seen from Table 8.1. N*r* for all the frequencies 4158080 to 4302214 is zero in this dataset). To overcome this limitation, Good proposed using smoothed (*r*, N*r*) values for the calculation of *r**.

A simplified version of Good's rather complex algorithm – Simple Good-Turing – was developed by William Gale (Gale and Sampson, 1995). Code for this is available in a number of programming languages from Geoffrey Sampson's resources on the web (Sampson, 2007). I used the C program downloaded from this website (and reproduced in Appendix B) to smooth the tag pair counts.

The input to this program is the frequency of frequency counts as shown above (Table 8.1) and the output is an estimate of the probability of each of the items (tag pairs) that occurred just once, then each that occurred twice, then three times, and so on, as in Table 8.3.

Frequency	Probability
1	8.977e-09
2	1.844e-08
3	2.796e-08
4	3.75e-08
5	4.706e-08

Table 8.3: First five lines of output from Simple Good-Turing frequency estimator

The smoothing program also gives the estimate of the total probability for the pairs that did not occur at all. This is simply N_1/N as described above (1.225e-06 for this dataset). The smoothing algorithm does not suggest how this probability mass should be divided between the non-occurring items but I decided simply to divide it equally between the 279 unseen tag pairs. This seemed a reasonable decision as all of them were considered equally unlikely.

The frequency probabilities output by the program were then assigned to each of the tag pairs that had been seen – tag pairs occurring once were assigned the probability 8.977e-09, those occurring twice 1.844e-08 and so on. These smoothed $P_{SGT}(t_i,t_j)$ probabilities were then used to calculate the conditional SGT tag probabilities as follows:

$$P_{SGT}(t_j \mid t_i) = \frac{P_{SGT}(t_i,t_j)}{P_{SGT}(t_i)}.$$

Note that although all the zero frequency pairs were initially assigned the same probability value, their conditional probability will differ depending on the $P(t_i)$ value; unseen pairs where the first member of the pair is less frequent overall will have a higher conditional probability than those where the first member is more frequent. A comparison of the SGT conditional probabilities with the MLE estimates for a small sample of tag pairs is shown in Table 8.4.

Tag pair	Count	MLE	SGT
AT0,NN1	4302215	0.536180264218224	0.536166713869756
NN2,PNI	995	0.000198174444897446	0.00019815823595266
VDD,PNQ	1	9.88943610435333e-06	9.27499044105613e-06
VDB,VDN	0	0	3.89703034634343e-06
CJS,VDI	0	0	3.85237264879899e-07

Table 8.4: Comparison of MLE and SGT probability estimates

As previously described, the Turing Estimator that forms the basis of the smoothing method employed here, calculates *r** as:

$$r^* = (r+1)\frac{N_r + 1}{N_r} \quad \text{where } r \geq 1$$

Using this formula the value of 1^* for this dataset is calculated as 1.17 (Table 8.2). Using the Turing Estimator, the sign of a closed class is 1* > 1 (Gale and Sampson, 1995). As 1* for this dataset is > 1, this suggests that the class of tag pairs is closed, meaning that all allowable pairs have been seen and that any unseen tag pairs that do occur are likely to be ungrammatical. Thus, although it is undesirable for them to be assigned a zero probability, a less complex smoothing method might have served us just as well. However, the tagger described in the next section uses the SGT smoothed tag-pair probabilities, calculated as described above.

8.3 The tagging algorithm

8.3.1 Calculating tag sequence probabilities

The tagger initially assigns the tag retrieved from the dictionary to all single-tag words. These will form the start and end points of sequences of one or more multi-tag words. To assign tags to the multi-tag words, the tagger selects the tags on the maximum probability path through the sequence. To calculate the probability of a given tag for a word on a particular tag path it retrieves the conditional tag bigram probability for that tag occurring following the tag assigned to the previous word on that path and multiplies it by the probability of the word occurring with that tag. The product of these probabilities gives the overall probability for the complete path. This is calculated as follows:

$$\boldsymbol{T}(w_{1,n}) = \underset{t_{1,n}}{\arg\max} \prod_{i=1}^{n} P(t_i \mid t_{i-1}) P(w_i \mid t_i)$$

The point to note in this equation is the way in which the tag probability for each word is calculated (the final term of the equation). Initially it might seem that we should condition the tag on the word by calculating the relative frequency of each of its tags – in other words, to ask, “Now that we’ve come across this particular word in the text, what is the likelihood of each of its tags occurring?” In this case the final term of the equation would be $P(t_i|w_i)$ which is the value that has been used by some taggers. But the equation above uses $P(w_i|t_i)$. In a comparison of the two approaches, Charniak et al. (1993) report better performance using the equation given above which they describe as “more theoretically pure” (assuming tagging is regarded as a Markov process). In this case we are conditioning the word on the tag by asking, “If we assign this particular tag, what is the likelihood of this word occurring with it?” This method also seems more appropriate when, in addition to assigning tags, we want the tagger to decide between words, as we later shall (Section 8.4). The $P(w_i|t_i)$ values are calculated as:

$$P(w_i \mid t_i) = \frac{C(w_i, t_i)}{C(t_i)}$$

where $C(w_i,t_i)$ is the count of the number of times word w_i occurred with tag t_i and $C(t_i)$ is the total occurrences of the tag t_i with any word.

8.3.2 A tagging example

The tagging process is easiest to understand if we use an example. Take the three word sentence:

They can fly.

The tagger has already assigned tags to the single-tag words and as the first word in this sentence – *They* – has just one tag – PNP – this becomes the anchor tag at the start of the multi-tag sequence *can fly*. As the sentence ends with a multi-tag word, the anchor tag at the end of the sequence is the EOS tag that is appended to the end of each sentence.

The tagger now retrieves the entry for *can* from the dictionary together with the frequencies for each of its tags – VM0, NN1, VVB, VVI. The VVB and VVI tags for *can* occurred less than once per million in the BNC and since frequencies are stored in the dictionary as occurrences per million, this means that the tag frequency recorded in the dictionary for these tags is zero. These zero values, along with the minus-one frequencies which were assigned to the words which occurred once, twice or not at all in the BNC, create problems when it comes to calculating the word-tag probabilities. A useful modification to the dictionary would be to make these values more fine-grained at lower frequencies and possibly to use a smoothing algorithm to assign a value to the unseen words or, alternatively, just to store the raw frequencies. However, this is a task to put on the dictionary wish-list and for the moment the program has to cope somehow with these 0 and -1 values. The solution adopted was to assign values of 0.5 and 0.01 to them respectively at run-time.

After adjusting the tag frequencies in this way, the program continues to consider which of the four tags for *can* is most likely to occur following a PNP tag. It first calculates the $P(can|tag)$ values for each tag. These values are then scaled by dividing by the total of the $P(can|tag)$ values. Although this step is not strictly necessary mathematically, we are interested in proportions rather than actual values at this point and this has the advantage of making the numbers larger and easier to check visually.

The program next retrieves the bigram frequencies which give the likelihood of each tag following the preceding tag. Multiplying these by the word-tag probabilities calculated above gives the probability value for each tag occurring in this context.

A final stage calculates the overall probability of the sequence up to this point by multiplying this value by the overall probability for the sequence of tags that preceded it – this will be 1 for the first tag in the sequence. The complete calculation (illustrated in Table 8.5) for each of the tags of *can* following the PNP tag at the start of the sequence is thus:

$P(can|tag) * P(tag|PNP) * 1.$

Tag	**Count (tag,can)**	**Count(tag)**	**P(can|tag)**	**Scaled P(can|tag)**	**P (tag|PNP)**	**P(tag|PNP) * P(can|tag)**
VM0	1992	12410	0.1605	0.999	0.1223	0.1222
NN1	8	158095	5.0602e-005	0.0003	0.0021	6.6729e-007
VVB	0 (0.5)	13447	3.7183e-005	0.0002	0.0785	1.8172e-005
VVI	0 (0.5)	24925	2.0060e-005	0.0001	0.0183	2.2861e-006

Table 8.5: Calculating the probability for *can* following PNP

8.3.3 Calculating the sequence probability

At this point the tagger could fairly confidently assign a VM0 tag to *can* but since it wants to find the most likely path through the complete sequence of multi-tag words it will not make a final decision until it has reached the end of the sequence. With a short sequence, such as the one we are considering here which has 16 possible paths (four tags for *can* followed by four tags for *fly*), it would be possible to calculate the probabilities for each sequence and then select the one with the maximum value. However, as noted previously, many sequences will be much longer than this and have many more possible paths. What the tagger needs to do is consider the continuation of the most likely sequence at each point while at the same time having the possibility to backtrack to the previous word should that sequence seem unproductive.

As it progresses through the sequence, the tagger stores the path probabilities it has calculated and selects the maximum it has seen so far. It calculates the probabilities for each tag of the succeeding word continuing this path (as described above) and also stores them. If the probability for one of these new paths is greater than the probability of any of the other paths previously stored, the tagger will continue that path, otherwise it will select the most likely of the previously stored paths for continuation. Thus at each stage the program needs to do two things: select the most likely path so far and store the newly calculated probabilities for its continuation. It would be possible to do this by storing the path probabilities in a sorted array but this would require a large sorting overhead each time new paths were added and would not be practical for long sequences. However, provided we have a mechanism for ensuring that the most likely sequence is always in first place, we are not concerned what order the other paths are stored in. What is required is a priority queue, and this can be implemented efficiently using a binary heap (Knuth, 1973). This functions in a similar way to an ordered binary tree except that the only ordering requirement for the heap is that each parent node has a higher value than each of its child nodes but no order is specified between siblings. It can thus be stored simply in a one-dimensional array and each element's array index value can be used to find the position of its parent or child nodes as shown below:

parent node = (i -1) /2

left child = 2 * i + 1

right child = 2 * i + 2

Thus the children of the first element in the array (index 0) are at indices 1 and 2, the children of the element at index 1 are 3 and 4 and so on.

The array is maintained in heap order by moving elements up or down as necessary. When the element with the maximum value is to be extracted from the top of the heap, it cannot simply be removed as its largest child may not be the maximum value on the heap. To maintain the array in heap order when the first element is removed, it is replaced with the tail element which is then moved downwards into its correct position. When new elements are added they are pushed on to the end of the array and then moved upwards if they have a greater probability than their parent node.

The tagger stores the sequence probabilities on a heap as described above. It continues to remove the maximum path so far and store the probabilities for paths continuing from that point until it reaches the end of the sequence for one possible path and this sequence has the highest value on the heap. At this point the tagger terminates and assigns the tags on this path to the words in the sequence. This is easiest to understand if we continue with our example.

To tag the complete sentence “They can fly”, the program first stores the values calculated above on the heap as shown in Table 8.6

Heap element	**Probability**
<PNP>they <VM0>can	0.1222
<PNP>they <VVB>can	1.8173e-005
<PNP>they <NN1>can	6.6729e-007
<PNP>they <VVI>can	2.2861e-006

Table 8.6: Heap storing probabilities for tags of can following PNP

It next calculates the probability for each tag of the next word – *fly* – following a VM0 tag. These calculations are given in Table 8.7.

Tag	**Count (tag|fly)**	**Count (tag)**	**P(fly|tag)**	**Normalised P(fly|tag)**	**P(tag|VM0)**	**P(tag|VM0) * P(fly|tag)**
VVI	22	24925	0.0008	0.5726	0.3760	0.2153
NN1	9	158095	5.6928e-005	0.0369	0.0017	6.4668e-005
VVB	8	13446	0.0006	0.3859	2.4648e-05	9.5126e-006
AJ0	0 (0.5)	71416	7.0012e-006	0.0045	0.0007	3.1918e-006

Table 8.7: Calculating the probabilities for each tag of *fly* following VM0

The tag probabilities calculated above are then combined with the previously calculated probability for the sequence ending with the VM0 tag (0.1222) and the new sequence probabilities stored on the heap as shown in Table 8.8.

Heap element	Probability
<PNP>they <VM0>can <VVI>fly	**0.0263**
<PNP>they <VVI>can	2.2861e-006
<PNP>they <VVB>can	1.8173e-005
<PNP>they <VM0>can <AJ0>fly	**3.9009e-007**
<PNP>they <VM0>can <VVB>fly	**1.1626e-006**
<PNP>they <NN1>can	6.6729e-007
<PNP>they <VM0>can <NN1>fly	**7.9035e-006**

Table 8.8: Heap after storing probabilities for each tag of *fly* following VM0 (new elements added to heap shown in bold)

Although the top element of the heap now contains the complete sentence, because the last word – *fly* – was multi-tag the program appends the EOS tag to the end of the sequence and retrieves the P(EOS|VVI) – (0.0016) – bigram probability. It then calculates the probability of EOS following VVI, multiplies this by the probability of the sequence ending in VVI and stores this on the heap. As shown in Table 8.9, this probability is still higher than the other sequence probabilities already on the heap so this is the final sequence with the maximum path and is used to assign the tags. As the final heap in Table 8.9 shows, only seven of the total 16 possible sequences were ever considered.

One of the features of calculating sequence probabilities in this way is that the overall probability decreases with the length of the sequence. Because the sequence considered above was short (and also partly because the VM0 probability at the start was so high) we have been able to reach the end without having to consider many of the sequences that were assigned a low probability at the outset. With longer sequences these less favoured initial sequences will gradually work their way to the top as their probability becomes greater than that for the longer sequences that have already been considered. To reduce the overhead involved in considering sequences that were unlikely to be productive, I stored the maximum position reached in the sequence in any expansion and discarded the element at the top of the heap if it was more than one word behind.

Heap element	**Probability**
<PNP>they <VM0>can <VVI>fly <EOS>	4.1678e-005
<PNP>they <VVI>can	2.2861e-006
<PNP>they <VVB>can	1.8173e-005
<PNP>they <VM0>can <AJ0>fly	3.9009e-007
<PNP>they <VM0>can <VVB>fly	1.1626e-006
<PNP>they <NN1>can	6.6729e-007
<PNP>they <VM0>can <NN1>fly	7.9035e-006

Table 8.9: The final heap showing the maximum path through the sequence in top position

In the example above, the tagger correctly assigns a VM0 tag to *can* but we have gone to a lot of effort to achieve the same result that the simple ‘select the most frequent tag’ method would have given us. To assess whether this is worthwhile we need to know whether it correctly assigns an NN1 tag to *can* in “a *can* of beans”. The extracts from the tagging process shown below demonstrate that this is the case. As all the letters of the alphabet have entries in the dictionary there are two possibilities for the tagging of *A* at the start of the sequence, ZZ0 (singular letter) or AT0 (article). Table 8.10 shows the heap after calculation of the probability for each of these tags occurring at the start of a sentence. As would be expected, AT0 is the preferred tag.

Heap element	**Probability**
<BOS> <AT0>A	0.0175
<BOS> <ZZ0>A	0.0016

Table 8.10: Heap with probabilities for *A* occurring at the start of a sentence

The tagger now calculates the likelihood of each of the tags of *can* following AT0. This is shown in Table 8.11. (The calculation of the *P*(*can*|*tag*) values is the same as that given above and so has been omitted from this table.) This time we see that, despite the high probability initially assigned to the VM0 tag for *can*, the high probability of a noun occurring after an article makes NN1 the preferred tag in this context. However, in this case, the overall probability for the two word sequence (as shown in the last column of Table 8.11) is lower than for the initial sequence starting with the ZZ0 tag. This is now extracted and the tagger has to go through several more iterations before finally assigning the NN1 tag to *can*.

Tag	**Normalised P(can|tag)**	**P(tag|AT0)**	**P(tag|AT0) * P(can|tag)**	*** P(sequence)**
VM0	0.999	1.7811e-05	1.7799e-005	1.272.e-006
NN1	0.0003	0.5362	0.0002	1.2087e-005
VVB	0.0002	0.0003	6.7970e-008	4.8638e-009
VVI	0.0001	2.4009e-07	2.9985e-011	2.1457e-012

Table 8.11: Calculation of the probability of each of the tags of can following AT0

8.3.4 Tagger assessment

The tagger implemented in this program is relatively unsophisticated; tagging is a more exact science than the preceding description might imply. But although the aim of this research was to correct spelling rather than to produce a state-of-the-art tagger, I needed to check that it was fit for the purpose before proceeding further.

The initial impression, from a visual examination of tagged text output by my tagger, suggested that the tags it assigned were reasonable but I needed to assess its adequacy more formally before extending it to incorporate checking for syntactic real-word errors (as described in the next section). It should be noted, at this point, that this assessment was intended simply to confirm (or otherwise) the tagger's suitability for use by my spellchecker, not to enable me to make claims as to its tagging accuracy.

Although it would have been a relatively straightforward matter to run the tagger over a portion of the BNC and compare the tags output by my tagger with those assigned to words in the corpus, this would not have been a very realistic test as, since the tag bigram probabilities used by the tagger were derived from the BNC, it would have amounted to testing on the training data. To avoid this circularity, I used the tagged version of the LOB corpus (Johansson et al. 1986), included on the ICAME (1999) CD, to test the tagger. This corpus was compiled by researchers from the University of Lancaster, the University of Oslo and the Norwegian Computing Centre for the Humanities, Bergen in the early 1980s. It contains approximately a million words made up of 500 text samples, representing a broad range of British English from 1961.

The tagset used in the LOB corpus (Appendix D) differs from the BNC tagset; not only are the tags themselves different but there are also more tags – excluding punctuation tags there are 139 part-of-speech tags in the LOB tagset compared to just 56 in the C5 tagset used in the BNC. Despite these differences, it was possible to produce a mapping between the two tagsets that was adequate to compare the tagged output from my tagger with the tags in the original corpus.

I ran the tagger over a LOB file (LOBTH_A.TXT), containing approximately eighty-seven thousand words, and compared the tag it assigned to each word to the tag recorded for that word in the corpus. (As the LOB corpus has been manually post-edited to correct errors made by the automatic tagger, it is reasonable to assume that the tags it contains are correct.) Appendix E shows a comparative listing, in vertical format, of the first ten sentences in this file comparing the tags output by my tagger with the LOB tags and Appendix F shows the number of words in the file that were assigned each tag, the number of correct assignments and the number of discrepancies. As the table shows, the tags selected by my program mapped to the corresponding LOB tag for just over 94% of the words in the file. Around a quarter of the discrepancies were proper nouns or single letters, possibly caused by the text segmenter making an incorrect decision for words with initial capitals, and a small proportion may be due to mismatches between the tagsets or inaccuracies in the mapping between them (the distinction between pronouns and determiners, for example). Others appeared to be incorrect tagging decisions. These were often cases where a tagger might be expected to encounter difficulties – incorrectly selecting between nouns and adjectives or past tense and past participle, for example.

Although this small experiment suggested that some refinement might improve the efficiency of the tagger, the performance level seemed reasonable enough for me to continue to the next stage of development.

8.4 The Confusion Tagger

8.4.1 Implementation

The ‘confusion tagger’ is an extension of the part-of-speech tagger described above, but this time, if it encounters a confusable word in the text, the program decides on the most likely tag at that point based on the tags for the word it has seen together with the tags for its associated confusion set words. If the tag selected is more likely to belong to one of the other confusion set members, the word is flagged as an error and the preferred member proposed as a correction.

For example, if it encounters the sequence “...he *wood* rather...” (where *wood* is a misspelling of *would*) it finds that *wood* has an associated confusion set containing *would* and *wooed*. If the program was just tagging, *wood* would be regarded as unambiguous since it only has one tag – NN1 – and the program would simply assign PNP, NN1 and AV0 to each of the words respectively. However, the confusion tagger will consider whether one of *wood*’s confusables is a better syntactic fit. To do this it proceeds in much the same way as the tagger described above except that it calculates the P(word|tag) probabilities for each <word, tag> pair in the confusion set and scales by the overall probability for the confusion set. Table 8.12 shows these calculations.

<Word,Tag>	**Count (tag|word)**	**Count(tag)**	**P(word|tag)**	**Scaled P(word|tag)**	**P(tag|PNP)**	**P(tag|PNP) * P(word|tag)**
wood NN1	52	158095	0.0003	0.0017	0.0021	3.7185e-006
would VM0	2320	12410	0.1869	0.9978	0.1223	0.1220
would VVD	0.5	23440	2.1330e-005	0.0001	0.1717	1.9549e-008
wooed VVN	1	25744	3.8843e-005	0.0002	0.0024	5.0656e-007
wooed VVD	0.5	23440	2.1330e-005	0.0001	0.1717	1.9549e-005

Table 8.12: Calculations for probability of each tag for *wood* and confusables following PNP

The probability values in the last column of Table 8.12 are now stored on the heap as shown in Table 8.13. This shows that a VM0 tag is much more likely than an NN1 to occur following a PNP tag. The sequence <PNP>*He* <VM0>*would* with its associated probability of 0.1220 is extracted from the top of the heap and the probability of an AV0 tag (the single tag for *rather*) is calculated. The bigram probability P(AV0|VM0) is 0.1036. This gives a probability of 0.0126 for the sequence <PNP>*He* <VM0>*would* <AV0>*rather*. Since this is higher than any of the other sequences previously stored on the heap it will appear in the top position again and the confusion tagger will conclude that the word intended in this context is *would* rather than *wood* and offer this as a suggested correction.

Heap element	**Probability**
<PNP>He <VM0>would	0.1220
<PNP>He <VVD>wooed	1.9549e-005
<PNP>He <NN1>wood	3.7185e-006
<PNP>He <VVD>would	1.9549e-008
<PNP>He <VVN>wooed	5.0656e-007

Table 8.13: Heap storing probability of each <word, tag> pair occurring after PNP

8.4.2 Performance

Although a syntax-based approach can generally be expected to make a decision only when the confusables differ in their parts-of-speech, the confusion tagger will sometimes make a choice between two same-tag confusables. This is because the confusion tagger is conditioning the word on the tag so it will pick the word in the set that occurs most frequently with its selected part-of-speech. This means, for example, that it would correct *planed* to *planned* in this fragment from the error corpus. Both are past participles but *planned* has a higher past participle frequency (36) than *planed* (0).

"...a rewrite of the monitor was *planed*."

This is fortuitous rather than intentional and the poor performance of the 'select the most frequent' method discussed in the previous chapter suggests that it is unlikely to be a particularly useful feature. Both *planed* and *planned* can also be tagged as past tense verbs and adjectives but *planned* is used most frequently in all cases so the confusion tagger will make the same selection as 'select the most frequent' for this pair of words, whichever tag it assigns. The situation is slightly different for words with overlapping tagsets such as the {*loose*, *lose*} example given at the start of this chapter. *Lose* has a higher overall frequency and so 'select the most frequent' will always prefer it - changing all correct usages of *loose* to *lose* and ignoring all error usages of *lose*. The confusion tagger will also prefer *lose* when it wants to assign a verb tag – it will correct the error usage of *loose* in "to *loose* speed" but raise a false alarm for a correct usage such as "to *loose* them off". (Both the preceding examples are taken from the BNC.) However, if its selected tag is adjective, adverb or noun it will prefer *loose* since these tags are unique to *loose*. Thus if it came across "a *lose* fit" it would propose *loose* as a correction rather than ignoring the error as the 'select the most frequent' method would do. Conversely, if the usage was correct – "a *loose* fit" – the confusion tagger would accept it rather than raising a false alarm.

On balance, it would seem best to set the confusion tagger so that it only suggests a correction when its selected tag is not in the tagset of the word appearing in the text. Decisions for words with matching tags can be left to the semantic checker (described in the next chapter). Thus for the *planed/planned* example above, where the tagsets are identical, the error would simply be ignored by the syntax checker (and a correct usage of *planed* in for example, "the wood was *planed*", would be accepted if it happened to occur in the corpus).

The possible outcomes from the syntax checking stage for correctly used confusables are:

- accept the word as correct;
- flag it as an error (thus raising a false alarm).

The possible outcomes for errors are:

- flag and correct the error;
- flag but not correct the error;
- ignore the error.

Table 8.14 shows the number of correct and error usages of the confusables falling into each of the categories above when the syntax checker (set to ignore syntactically matching confusables as described above) was run over the dyslexic error corpus. The initial run of the program, shown in the first row of the table assigned the full probability value to all <word, tag> combinations in the confusion set. This means that if one of the confusables was judged to have a higher probability than the word in the original text, even if only fractionally higher, it was proposed as a correction. Although this corrected around half of the errors (column c) while raising a false alarm for just 7% of the correctly used confusables (column b), the overall effect was negative, since the correct usages greatly outnumber the errors. Eighty-nine percent of all occurrences of the confusables in the error corpus are correct usages (Table 7.1) so the number of false alarms outnumbers the errors corrected meaning that, if the spellchecker's suggestion was accepted each time, there would be an overall increase in the number of errors in the text. Column f shows the number of errors remaining in the text – the sum of the false alarms, the ignored errors and the errors that were flagged but miscorrected. Subtracting this from the total of errors that were in the text to start with (644 in this case) gives us a figure for the reduction in the total number of errors in the text (column g). For the initial run with the full weighting for each word in a confusion set, this figure is negative indicating that the text now contains 46 more errors than it had done to start with. Column h shows the percentage 'improvement' in the text – the errors remaining as a proportion of the original errors. This shows that after the initial run there was a 7% *decrease* in the accuracy of the text.

Words appearing in the text are much more likely to be correctly spelled than they are to be errors, and we can factor in this general expectation by handicapping the alternative confusion set members by reducing the weighting given to the words not appearing in the text. The remaining rows of the table show the results of successively reducing this weighting from 50% down to 0.01%. This is also shown graphically in Fig. 8.1.

	Correct Usage (5209)		**Error Usage (644)**			**Improvement**		
Handicap level	**Accept**	**False alarm**	**Flag & correct**	**Flag not correct**	**Ignore**	**Errors remaining (b+d+e)**	**Error reduction (644 – f)**	**Imprvt %**
	a	**b**	**c**	**d**	**e**	**f**	**g**	**h**
1	4843	366	320	63	261	690	-46	-7%
0.5	4956	253	313	51	280	584	60	9%
0.1	5100	109	281	35	328	472	172	27%
0.05	5135	74	267	32	345	451	193	30%
0.01	5189	20	218	24	402	446	198	31%
0.005	5198	11	195	18	431	460	184	29%
0.001	5206	3	139	10	495	508	136	21%

Table 8.14: Syntax checker performance with different weightings

Reducing the weighting in this way means that the program is more likely to accept the word appearing in the text as correct. Although this will have the desired effect of reducing the number of false alarms, it will, of course, also reduce the number of errors corrected. However, as Table 8.14 shows, the rate of decrease for the false alarms is greater than that for the errors corrected. With the level of handicap set at 0.5 there is a small overall improvement in the text but the number of false alarms (253) is still unacceptably high. At the maximum level of handicap (0.001) there are just three false alarms and 22% of the errors are corrected. However, the greatest overall improvement in the text is achieved with the handicap set at 0.01 with 34% of the errors corrected at the expense of 20 false alarms (0.4% of the correctly used confusables). This level thus seems to offer the best balance between maximizing error correction while reducing false alarms.

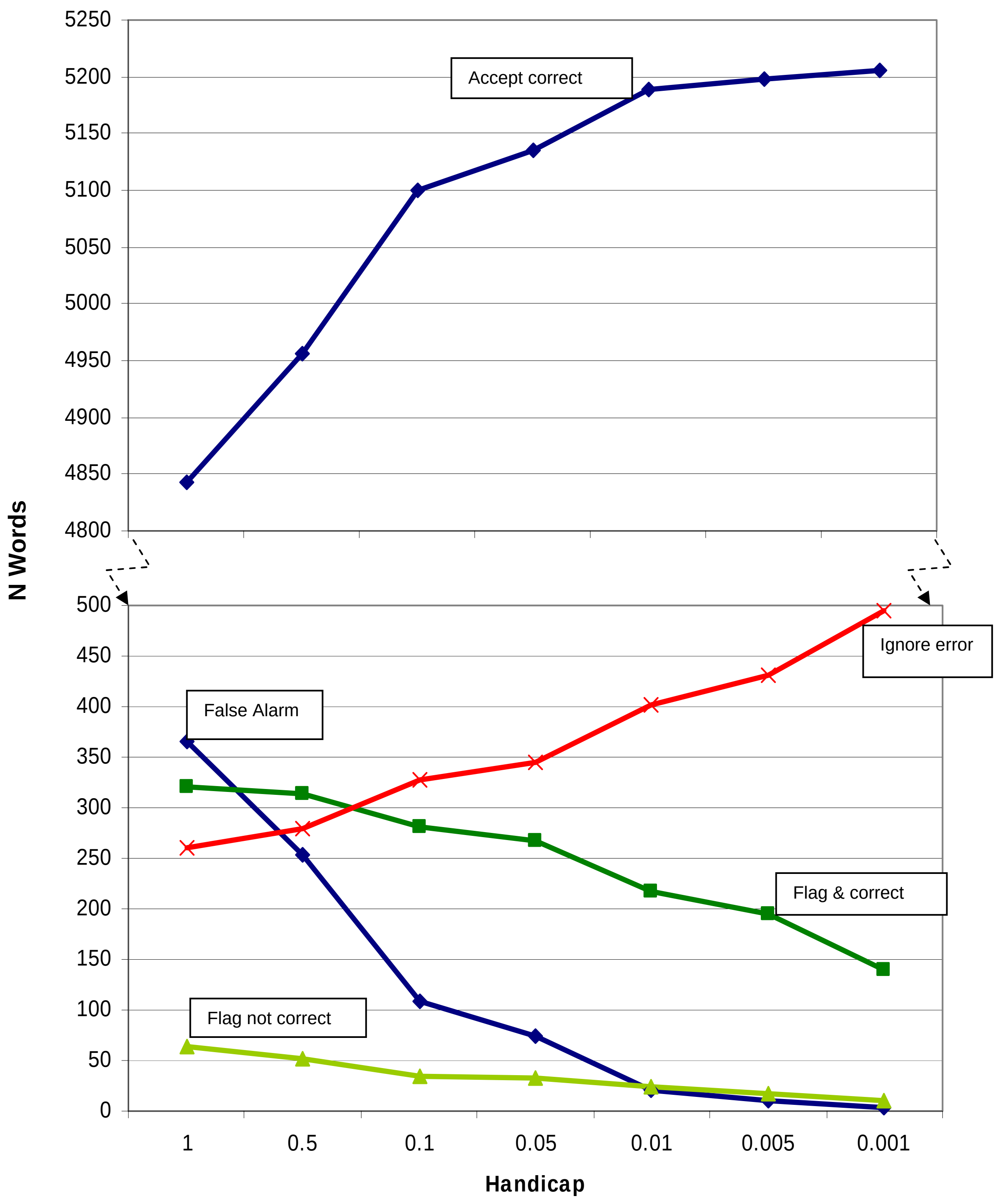

Fig. 8.1: Performance of syntax checker with different levels of handicap

8.5 Conclusion

This chapter has described the implementation of a syntax-based spellchecker that is able to correct errors where the error and target differ in their part-of-speech. With a reduced weighting given to the alternative words – effectively confining the spellchecker to proposing a correction only when it was confident about it – the program was able to correct around a third of the errors appearing in the dyslexic error corpus while producing few false alarms. Experimenting with different levels of ‘handicap’ suggested that 0.01 was the optimum level to achieve a balance between maximizing error correction and minimizing false alarms to obtain the best overall improvement in the ‘correctness’ of the text.

The syntax checker described in this chapter was set to ignore confusables where the alternative word(s) had the same part-of-speech as the word appearing in the text; such words were simply flagged for later consideration by the semantic checker, described in the next chapter. The effect of this is that correctly used ‘semantic’ confusables are accepted by the syntax checker, thus contributing to the total of correct usages accepted (column a, Table 8.14), while error usages are ignored, thus contributing to the total of ignored errors (column e, Table 8.14). In addition to flagging the semantic confusables for which it is unable to make a decision, the syntax checker also assigns a part-of-speech tag to each word in the text. These tagged sentences are then passed to the semantic checker, described in the next chapter.

Chapter 9: Using Semantic Associations

After the syntax checker has run over the text, all the words have been assigned a part of speech and some errors have been corrected. The confusion tagger, as described in the previous chapter, considers not only the part-of-speech tags associated with the word it has encountered in the text but also the tags of any words that are considered confusable with it. If the most likely tag in the context belongs to one of these alternative words and not to the word that was seen in the text, it will propose a correction. On the other hand, if the selected tag belongs to the word that was seen in the text, the syntax checker can only accept the word as correct if the tag does not also belong to one of its confusables. If the word and one or more of its confusables have the tag in common, the syntax checker can either be set to select the word that occurs most frequently with that part-of-speech or to make no decision at all. The first of these approaches may be appropriate where the decision is between a word that is rare and one that is highly common, such as the pair {*wold*, *world*}, but, as shown in Chapter 7, it will result in an unacceptable number of false alarms in cases where both confusables are relatively common. The second approach is simply to flag such cases as confusable and leave them for consideration by a later stage of processing. This is the approach taken here.

At the end of the syntax checking stage, confusables with distinct tagsets have either been accepted as correct or flagged as errors and will not be considered further. Confusables with overlapping tagsets, such as {*loose*, *lose*}, will also have been checked. However, in this case the syntax checker will only have made a decision if its selected part-of-speech is unique to one of the words; if the words have the tag in common, the final decision will be left to the semantic checker. Thus, for example, in the sequence "a *lose* fit", the confusion tagger would consider all the tags for both *lose* and *loose* and assign an adjective tag. Since this is not in the tagset for *lose*, it would at the same time correct *lose* to *loose*; there would be no need for the semantic checker to consider the pair further. On the other hand, the confusion tagger would select a verb tag in the sequence "to *loose* speed" and leave it up to the semantic checker to decide whether *loose* or *lose* was the more appropriate verb in the context. Confusables with matching tagsets are dealt with in a similar way; the confusion tagger assigns a tag based on the word seen in the text and leaves the semantic checker to decide whether this word or one of its confusables is the intended word. So, for example in the sentence "She wrote the appointment in her *dairy*," the confusion tagger would find that *dairy* was confusable with *diary* but as both are nouns it would simply assign a noun tag and flag the possible confusion for later consideration by the semantic checker.

In this example, both the verb – *write* – and the noun – *appointment* – suggest to a human reader that *diary* rather than *dairy* is the intended word. How can we capture these semantic clues in a way that can be used by the spellchecker? This chapter describes an approach using noun co-occurrence.

9.1 The proposed approach

A listing of the nouns that co-occur frequently in the vicinity of a confusable in a large corpus will generally demonstrate a distinct semantic 'flavour'. To illustrate using {*dairy, diary}*, nouns occurring frequently in the BNC near to *dairy* include *product*, *farmer*, *cow* whereas those occurring near to *diary* include *entry*, *appointment*, *engagement*. However, lists such as these can not in themselves be used by a spellchecker to differentiate between the confusables; the lists will be very long and many of the nouns will occur infrequently. Just over 450 different nouns occur within two words of *diary* in the BNC; only 18 of these occur ten or more times while *appointment* and *engagement* occur just five and six times respectively. Over 280 of the nouns occur just once and, although many of them have no particular relationship to *diary*, some of these once-only co-occurrences, such as *chronicle*, *meeting*, *schedule*, have a distinctly *diary* feel to them. In contrast, some words that seem to fit naturally into this category, such as *rota*, do not appear at all. We need to capture this similarity by grouping the nouns and giving an overall score for the group to indicate that a word belonging to it is more likely to appear in the vicinity of *diary* than *dairy*. Armed with this information, when the spellchecker checks a text and encounters say *dairy* it can assess whether the nouns that occur in the vicinity belong in the *dairy* or *diary* group. If they clearly fit in the latter, it can propose a correction.

The approach developed in this research uses WordNet to create such groups for the members of my confusion sets using the nouns that occur in the vicinity of each confusable in the written section of the BNC. This means that in the *diary* example above, the program would make its decision between *dairy* and *diary* based on the co-occurring noun *appointment* although, as noted, the verb *write* might also be helpful in making this decision. However, the WordNet hierarchy for verbs is much broader and shallower than it is for nouns and so does not easily lend itself to creating co-occurrence groupings in the way described below for nouns. For this reason the current semantic checker was developed using just noun co-occurrence.

At run-time, when it encounters one of the confusables, the spellchecker retrieves the nouns from the surrounding text, uses WordNet to ascertain which grouping they belong to and then assigns a score to each member of the confusion set indicating how closely it associates with these groupings. It then decides which member it prefers based on this score.

9.2 Semantic confusable listing

Results from an initial experiment (Pedler, 2005), using noun co-occurrence for 20 pairs of confusables that were either tagged only as nouns, such as {*dairy*, *diary*}, or only as verbs, such as {*carve*, *crave*}, were reasonably promising. The approach was now scaled-up to deal with a larger number of semantic confusables.

9.2.1 Word pairs

There were potentially over five thousand pairs of words that could be checked by the semantic checker – the confusable listings (discussed in Chapter 5) included 1940 pairs of words with matching tagsets and 3606 pairs with some but not all tags in common (Table 5.5). Although the majority of these pairs were nouns or verbs, the list also included a few hundred adjective pairs and around fifty adverb pairs. Adverbs, by definition, are more closely allied to verbs than nouns and although many of them seemed to be the type of error we would like to correct – <*sorely, surely*>, <*warily*, *wearily*> for example – they did not seem suitable candidates for the noun co-occurrence approach and so were removed from the list.

9.2.2 Tagged-word pairs

In contrast to the pairs used for the initial experiment which belonged to a single word class – either noun or verb – around a third of the remaining pairs of confusables belonged to more than one word class ('word class' here being noun, verb or adjective). The majority of these were both nouns and verbs – such as {*plan*, *plane*}, about three hundred were both adjectives and verbs – such as {*hated*, *heated*}, and a few were both adjectives and nouns – such as {*patent*, *patient*}. (In this last example the tagsets overlap as *patent* is also a verb but this possibility will be considered by the syntax checker and if it selects a verb tag the pair will not be passed to the semantic checker.) In addition to these pairs with two classes in common there were also three pairs such as {*bust*, *burst*} that belonged to all three classes – adjective, noun and verb.

When a word belongs to more than one word class the context and frequency with which it appears may differ depending on its class. Take the pair {*plan*, *plane*} for instance, both of which can be either a noun or a verb. Although there is some relationship between the verb *plane* and the noun *plane* when it refers to a woodworking tool, this is not the case when the noun is used in the 'means of transport' sense so we would expect that some nouns that co-occurred with the noun *plane* would not co-occur with the verb. Additionally, although *plane* is relatively common as a noun (with a dictionary frequency of 36 per million), it has a dictionary frequency of zero for its verb usage. (A dictionary frequency of zero actually means that the word occurred less than 90 times in the BNC as the frequencies are per million, as previously described.) In such a case, combining the noun and verb usage would overweight the probability for the verb.

Thus rather than simply considering confusable pairs of words, the semantic checker needs to consider pairs of tagged-words and store the nouns co-occurring with, for example, {*plan*(N), *plane*(N)}, separately from those co-occurring with {*plan*(V), *plane*(V)}. To enable it to do this, the list of confusable pairs was expanded into a list of tagged-word pairs.

9.2.3 Inflected forms

Many of the tagged-word pairs were confusable inflected forms. For example, in addition to the pairs {*spill*(N), *spell*(N)} and {*spill*(V), *spell*(V)} the list also included {*spills*(N), *spells*(N)}, {*spills*(V), *spells*(V)}, {*spilled*(V), *spelled*(V)} and {*spilling*(V), *spelling*(V)}. When the base form of a word is confusable, its inflected forms are also often similarly confusable and likely to produce similar spelling errors. For example, someone who mistakenly records appointments in a *dairy* instead of a *diary* might also suggest that colleagues consult their *dairies*. Such inflected forms will often occur in similar contexts; for example, we can *carve* stone and stone is *carved*; we might eat *lentil* soup or make soup with *lentils*.

Based on this observation, such inflected forms of a particular tagged-word pair can be grouped together under their base form as a single 'lexeme' pair; the lexeme pair {*spill*(V), *spell*(V)}, for example, includes the pairs {*spills*(V), *spells*(V)}, {*spilled*(V), *spelled*(V)} and {*spilling*(V), *spelling*(V)}. Nouns co-occurring in the BNC with any of these forms can be stored together as co-occurrences for the lexeme. At runtime, when the spellchecker encounters an inflected confusable it will use the lexeme co-occurrences to decide which word is the most appropriate in the context and choose the corresponding inflected form as the intended word. So, for example, if it was checking “She *spelled* the water”, it would retrieve the noun co-occurrences associated with the lexemes *spill*(V) and *spell*(V) and (hopefully) find that *water* was more likely to co-occur with *spill* than it was with *spell* and so propose *spilled* as a correction. This approach increases the number of noun co-occurrences for each confusable and also reduces the number of co-occurrence trees required by the spellchecker (Section 9.4.2 below).

9.2.4 Low frequency confusables

The initial experiment had suggested that the proposed approach was unsuitable for low frequency confusables. For example, the pair {*wold, world*} was considered for inclusion in the initial experiment. However, *wold* makes up just 0.4% of the total occurrences of the pair {*wold, world*} in the BNC and only 47 nouns co-occur with it. Some, such as *flower* and *path*, seem to relate to the dictionary definition of *wold* as an "(area of) open uncultivated country; down or moor" but others, such as *bet* and *rice*, seem more puzzling. Further investigation showed that, in fact, many of the occurrences of *wold* in the BNC are real-word errors as the examples below show:

"...variety of *wold* flowers..."

"...the best bet *wold* be..."

"...brown and *wold* rice..."

"...my *wold* as I see it..."

This suggests that a spellchecker would do best if it always flagged *wold* as an error. Indeed, this is what MS Word does, suggesting *would*, *world* and *wild* (which would correct the above errors) along with *weld* and *wood*, as replacements. Similarly, for the pair {*rooster*, *roster*} (also considered for the initial experiment) where there are only 25 occurrences for *rooster* in the BNC and just eight co-occurring nouns, there was simply not enough data to generalise from. These pairs were thus excluded from the initial experiment and similar low frequency lexeme pairs which were unlikely to be suitable for the semantic association approach were also not considered further.

9.2.5 The final list

There were just over two thousand lexemes in the final list (Table 9.1). Initially it might appear that verbs are under-represented as there are over three times as many nouns as there are verbs in this final listing. However, this is to be expected as the base form of each verb can have three inflected forms associated with it whereas each noun has only one (plural) inflection.

Word class	**N. lexemes**
Noun	1471
Verb	453
Adjective	128
Total	**2052**

Table 9.1: Number of lexemes for each word class

9.3 Noun co-occurrence listing

I now listed all nouns occurring within two words before or after each lexeme in the written section of the BNC. In the majority of cases a human reader presented with the resulting lists would have little difficulty in distinguishing between the confusables or in spotting the similarities between their respective sets of co-occurring nouns. For example, the top three co-occurring nouns for *carve* are *stone*, *wood* (both materials that can be carved) and *knife* (a tool that is used for carving). Nouns appearing with a lesser frequency further down the list are clearly related (*oak*, *walnut*, *marble*, *granite*, *tool*, *chisel*). The top three for *crave* are *man* (both craving and being craved), *food* and *success* which again bear the same affinity to *crave* as other words in the list such as *people*, *chocolate* and *attention* and are also clearly different from those co-occurring with *carve*.

9.4 Co-occurrence grouping

9.4.1 WordNet relationships

Nouns in WordNet (Miller et al., 1990) are organised as a lexical hierarchy. The main organisational principle is hyponymy/hypernymy or the ISA relation. For example, using the co-occurrences for *carve* discussed above, *oak* is a hyponym of *wood* and *granite* is a hyponym of *stone*, both *wood* and *stone* are hyponyms of *material*. Thus both *oak* and *granite* are linked through their common hypernym *material*, as illustrated in Fig. 9-1.

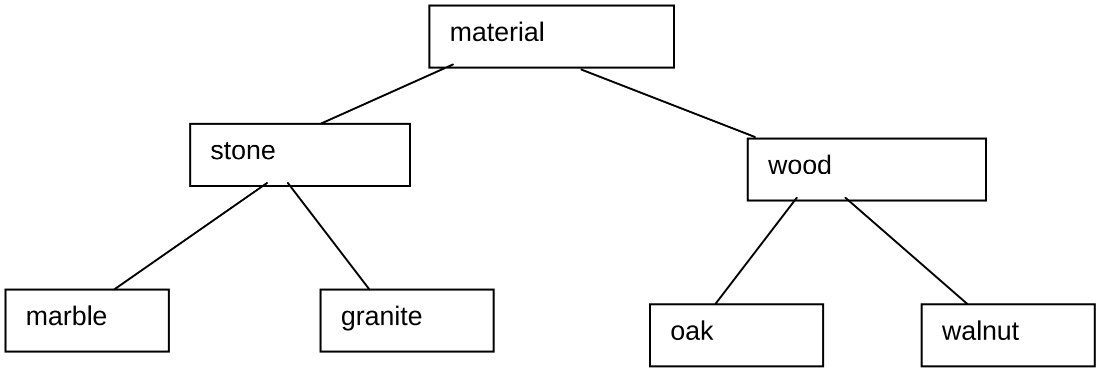

Fig. 9-1: Hypernym grouping of materials that can be carved

However, the WordNet hierarchy represents the relationship between word meanings rather than word forms, with each node of the hierarchy representing a synset or grouping of synonymous words. A word may be part of several synsets, each representing a different sense in which it can be used. There are a total of 12 senses stored for *stone*. Five of these are proper names (e.g. Oliver Stone the film-maker) and can be discounted. The remaining seven are listed below.

stone, rock (countable, as in "he threw a stone at me")

stone, rock (uncountable, as in "stone is abundant in New England")

stone (building material)

gem, gemstone, **stone**

stone, pit, endocarp (e.g. cherry stone)

stone (unit used to measure ... weight)

stone (lack of feeling...)

The sense illustrated in Fig. 9-1 is part of the second {*stone, rock*} synset and appears the most likely to occur in the context of *carve* although all of the first four seem to fit. However, the remaining three do not seem relevant.

The inclusion of slang and informal usage also presents a problem. Resnik (1995) reports obtaining a high similarity rating for the words *horse* and *tobacco*. On investigating this apparent anomaly he found that this had occurred as one of the senses recorded for *horse* is its slang usage for *heroin*, which means that both words can be used in the sense of narcotics.

9.4.2 Creating hypernym trees

We now have lists of words from the BNC that co-occur with, say, *carve*. Next we want to identify a subtree from WordNet's main trees where these words tend to cluster. A subtree in WordNet is a group of synsets that have a common hypernym. Since a word can have several senses and therefore appear in several places in WordNet, and since it was not possible to know in advance which senses best related to their co-occurring confusable, I retained all senses (apart from proper nouns or slang) of the co-occurring nouns in the initial hypernym trees. The assumption was that the most relevant ones would gather together while the others would appear in sparsely populated sections of the tree and could later be removed. Fig. 9-2 and Fig. 9-3 show the final sections of the tree for two of the senses of *stone* discussed above.

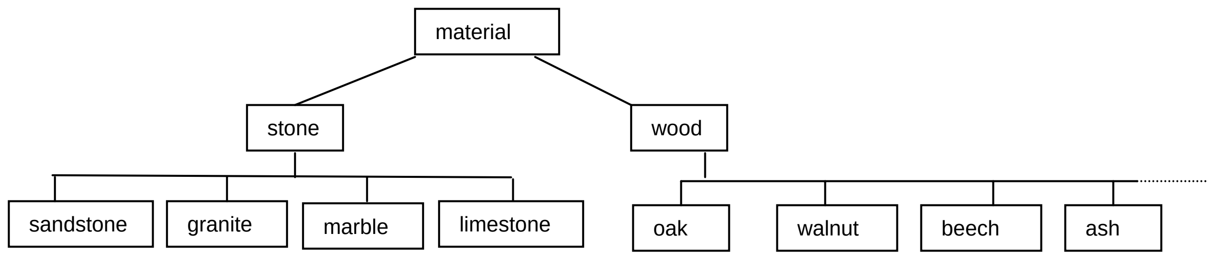

Fig. 9-2: Section of WordNet tree for stone#2

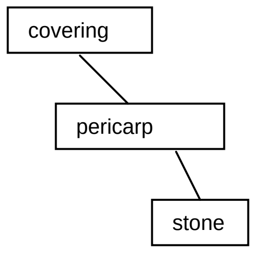

Fig. 9-3: Section of WordNet tree for {stone, pit, endocarp}

Fig. 9-2 shows that not only did the word *stone* itself occur with *carve* in the BNC, but so did *sandstone, granite, marble* and *limestone*, all hyponym senses of one sense of *stone*, similarly *oak, walnut* etc. are all hyponym senses of one sense of *wood*. The *material* node is included since it is the hypernym of both *stone* and *wood* and therefore of a subtree of WordNet that seems to go with *carve*. (The word *material* itself does not actually occur with *carve* in the BNC, though it obviously could do.)

By contrast, no words related to the cherry-stone meaning of *stone* co-occurred with *carve* – neither words in the same synset (*pit, endocarp*) nor words in the synsets of any of its hypernyms (*pericarp* etc.) – so this meaning of *stone* was left as a leaf node at the end of a long branch of isolated hypernyms (Fig. 9-3). This branch of the WordNet tree does not appear to go with *carve* and can now be pruned from the *carve* tree.

Continuing with the *stone* example, we have discarded three senses – those of cherry stone, weight and feeling – leaving four which seem likely co-occurrences – two types of rock, building material and gemstone. The word *stone* occurred 74 times in the vicinity of *carve* in the BNC. We do not know which of the remaining four senses of *stone* was involved in each of these occurrences, so we divide the 74 by four giving each of these nodes a ‘word count’ of 19 (rounded to the nearest whole number).

For a hypernym node however, we want its count to represent how often any of the words in its subtree occurred with the confusable. I therefore summed all the ‘word counts’ (i.e. the counts adjusted in the way just described) for all the words in the subtree and added these to the word count for the hypernym node itself.

Hypernym nodes at higher levels of the tree tend to represent generalised concepts. The node for *entity*, for example, is retained in the *carve* tree not because the word *entity* itself occurred with *carve* but because many of the words in its subtree did. For this reason the initial word count for such nodes will often be zero but as the word counts are propagated up the tree they will accumulate the word counts of all their hyponym nodes.

The final stage in creating a hypernym tree was to convert each of the adjusted word counts to a probability. The probability of each hypernym occurring in the vicinity of a particular confusable is calculated by dividing its word count by the total word count for the tree – i.e. the count stored in the root node of the tree (1714 in the case of *carve*).

Fig. 9-4 illustrates this process for the *material* section of the *carve* tree. Each node shows the initial word count for the hypernym with the summed word counts (rounded to the nearest whole number) in parentheses together with the resulting probability. For example, *stone* (sense 2) has its own word count of 19; to this is added the word counts of *granite, marble* etc., giving 39. Divided by the 1714 co-occurrences for *carve,* this gives a probability of 0.02. As can be seen, the hypernyms *material*, *substance* and *entity* start with an initial word count of zero and then accumulate the word counts of all their hyponym nodes. (These word counts cannot be extrapolated directly from the diagram as not all the hyponym nodes are shown.) As would be expected, the number of co-occurrences (and correspondingly the likelihood of encountering the hypernym or one of its hyponyms) increases as the concepts represented become more general at the higher levels of the tree.

What these figures tell us, in general terms, is that there is a certain probability of encountering, say, *granite,* close to *carve,* a higher one of encountering any sort of *stone*, a yet higher one of encountering some kind of *material*, and so on. The nodes at the higher levels of the tree are too generalised to capture the similarity between the nouns co-occurring with a particular confusable: s*ubstance*, for example, is a hypernym of both *stone* (which is often *carved*) and *food* (which is often *craved*) and so unlikely to be helpful in making a distinction between *carve* and *crave*. Conversely, nodes at the lower levels are too specific; although many different types of stone co-occurred with *carve* in the BNC (Fig. 9-2) there may be others that could equally well be carved but did not appear in the BNC. We are more interested in the frequency with which the some type of stone co-occurred with *carve* than the frequency with which specific types of stone occurred. To capture the level of generality that seemed most useful for the spellchecker, the branches of the hypernym trees stored for use at run-time were truncated at level 4. Thus for the tree illustrated in Fig. 9-4, *stone* and *wood* become leaf nodes.

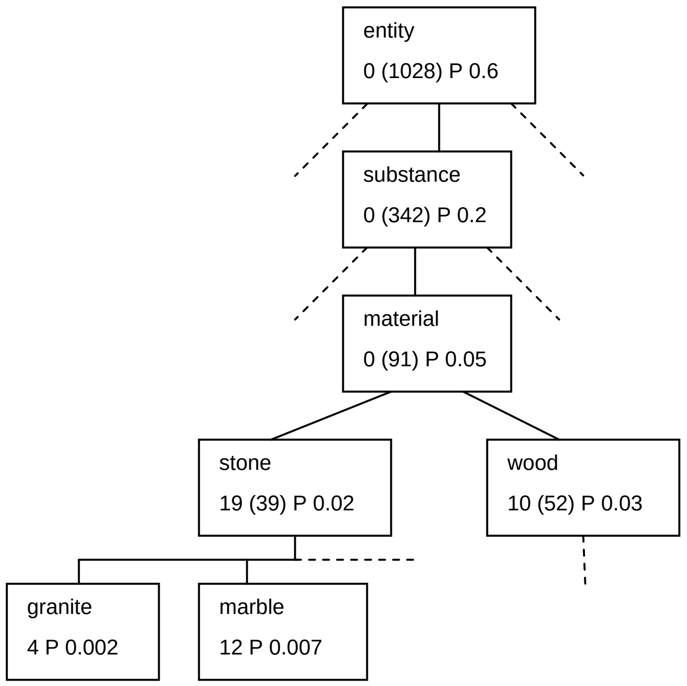

Fig. 9-4: Summed word counts and hypernym probabilities for section of *carve* tree

The hypernym trees created for each individual lexeme were written to file. The program then uses an index to this file to retrieve the trees it requires at run-time.

9.4.3 Merging the hypernym trees

When the spellchecker comes across a confusable word, it needs to decide whether the word it has seen is more likely in the context than one of the other members of its confusion set. For example, given the confusable pair {*carve, crave*}, when it encounters *carve*, it needs to decide whether this is indeed the word the user intended or whether *crave* would be a more appropriate choice.

The 'lexeme' trees described above tell us how likely it is that a particular hypernym will occur in the vicinity of a particular confusable. For instance from the *carve* tree (Fig. 9-4) we see that, given that *some* word in the hypernym tree for *carve* does occur in the vicinity of *carve*, there is a 2% probability of that word being *stone* or one of its hyponyms a 5% probability of it being some type of *material* and so on. The *crave* tree gives the corresponding probabilities for a given hypernym co-occurring with *crave*.

The probabilities stored in these trees tell us the probability of finding, say, *stone,* near *carve* , or, more generally, the probability of a particular hypernym occurring in the vicinity of a given confusable. But what the spellchecker needs to know, when it encounters either *carve* or *crave* in the text, is which of the two confusables is more likely to occur in the context of *stone*. In other words, it needs to know the probability of a particular confusable occurring in the vicinity of a given hypernym. To calculate this 'lexeme-given-hypernym' probability we need to weight each of the 'hypernym-given-lexeme' probabilities in the lexeme trees by the relative frequency with which each hypernym-lexeme pair for the confusion set – {*carve*, *crave*} in this case – occurred with the confusable in the BNC. We can then divide these 'weighted probabilities' by the overall probability for the hypernym occurring with either confusable to obtain a 'relative likelihood' score for each confusable lexeme – this will be a value between 0 and 1 for each confusable such that the sum of the scores is 1. Equation 1 below is used to calculate the weighted probablilities followed by equation 2 to calculate the final relative likelihood scores.

Given:

W a set of words (confusables)

H a set of hypernyms

$p_n(h,w)$ the WordNet probability for hypernym h co-occurring with word w

$f_b(h,w)$ the frequency of the hypernym h co-occurring with word w in the BNC

$p_w(w,h)$ weighted probability for word w co-occurring with hypernym h

$p_r(w,h)$ relative likelihood score for confusable w occurring with hypernym h

1) $$p_w(w,h) = \frac{p_n(h,w) * f_b(h,w)}{\sum_{w \in W} f_b(h,w)}$$

2) $$p_r(w,h) = \frac{p_w(w,h)}{\sum p_w(w,h)}$$

Table 9.2 shows these calculations for three hypernyms from the {*carve*, *crave*} lexeme trees.

w, h	**a** $p_n(h,w)$	**b** $f_b(h,w)$	**c** $\Sigma f_b(h,w)$	**d** $p_w(w,h)$ **a*b/c**	**e** $\Sigma p_w(w,h)$	**f** $p_r(w,h)$ **d/e**
carve, stone	0.02	39	39	1	1	1
crave, stone	0	0		0		0
carve, substance	0.2	342	393	0.174	0.2	0.87
crave, substance	0.2	51		0.026		0.13
carve, foodstuff	0.002	3.4	6	0.0012	0.0055	0.21
crave, foodstuff	0.01	2.6		0.0043		0.79

Table 9.2: Calculation of relative likelihood scores for {*carve, crave*} hypernyms

The *stone* (sense 2: stone, rock (uncountable)) hypernym included in the section of the *carve* tree shown in Fig. 9-4, does not appear at all in the *crave* tree, suggesting that the spellchecker should always prefer *carve* when it sees *stone* or one of its hyponyms in the vicinity of either *carve* or *crave*. In a trivial case such as this, it is actually not necessary to calculate the scores at all – the program can simply assign scores of 1 and 0 directly for hypernyms appearing in one tree but not in the other – but, as the table shows, this is also the result we would get if we *did* do the calculation.

I will now describe in detail the process of calculating the scores for the other examples in Table 9.2 where the hypernym node occurs in both lexeme trees.

Two of the hypernyms for *stone* included in the section of the *carve* tree illustrated in Fig. 9-4 – *material* and *substance* – also appear in the *crave* tree. For the *substance* node (shown in Table 9.2) the probability is almost the same in both trees – 0.1995 for *carve* and 0.1992 for *crave*, rounded to 0.2 in Fig. 9-4 and Table 9.2. This might initially suggest that we are as likely to *crave* a substance as we are to *carve* it. But, as *carve* is much the more frequent member of the pair (*carve* makes up 84% of the total occurrences for the pair {*carve, crave*} in the BNC), it has a larger overall co-occurrence count (1714 compared to just 256 for *crave*), meaning that many more *substance*-related nouns have co-occurred with *carve* (342) than they have with *crave* (51), (column b). Clearly we have to factor this in. We do this by multiplying each hypernym-lexeme probability (column a) by the relative frequency with which the hypernym-lexeme pair occurred in the BNC (column b / column c). This gives us the weighted probability scores (column d), which show us that, as we would expect from the discussion above, *carve* is more likely to occur than *crave* in the vicinity of *substance* or one of its hyponyms. In fact, because the hypernym probability (column a) is the same in each tree, in this case, it simply acts as a 'scaling constant' in the calculation of the weighted probabilities (d) and the final relative likelihood scores (column f) are identical to the relative frequencies of each hypernym-lexeme pair in the BNC – dividing the frequency for each pair (b) by the total occurrences for the pair (c) gives us 342/393 = 0.87 for *carve* and 51/393 = 0.13 for *crave*.

However, this is not the case when the initial hypernym probabilities differ. The more specific *substance* hyponym *foodstuff* (the final example in Table 9.2) also appears in both the *carve* and the *crave* trees (although its branch is not included in Fig. 9-4) and the overall co-occurrence count (column b) is still greater for *carve* (3.4) than it is for *crave* (2.6). But, unlike the *substance* example where the hypernym probabilities were the same for each confusable, *foodstuff* has a far greater probability of occurring with *crave* (0.01) than it does with *carve* (0.002). When this is factored in we now find that there is a 0.78 probability for *crave* co-occurring with some type of foodstuff as opposed to just a 0.22 probability for *carve*; although it is certainly possible to *carve* food – such as the Sunday roast – it seems we are more likely to *crave* it.

The spellchecker calculates these relative likelihood scores at run-time and stores them in a merged lexeme tree which it uses to select between the confusion set members as described in Section 9.5.2 below.

Fig. 9-5 shows the section of the merged tree for the {*carve, crave*} hypernyms discussed above. From it we can see that when the spellchecker comes across a specific kind of *substance* in the vicinity of *carve* or *crave*, it will prefer *carve* if the co-occurrence is a type of material whereas it will prefer *crave* if it is a type of food which is in general what we would want to happen.

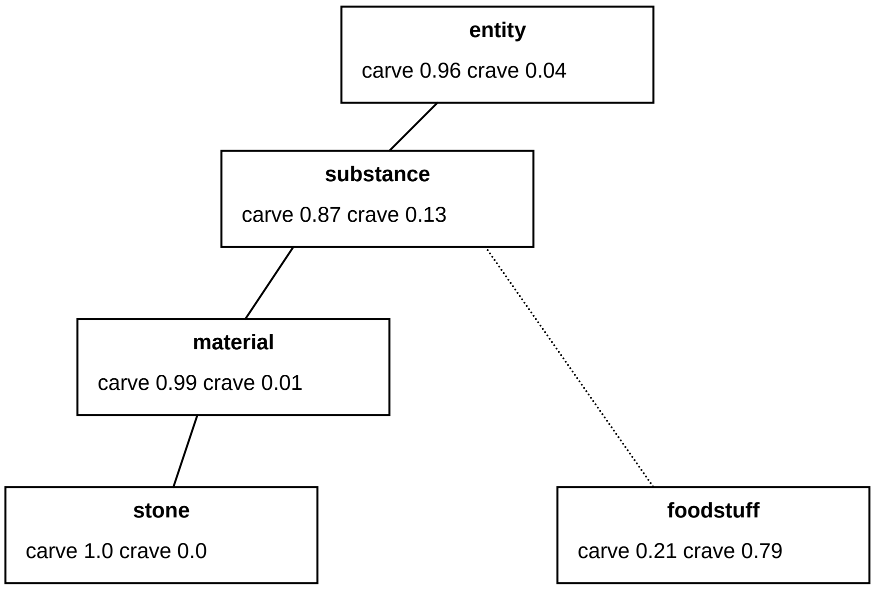

Fig. 9-5: Section of merged *{carve, crave}* tree

9.5 Checking text

9.5.1 Selecting co-occurring nouns

When the program encounters one of the semantic confusables in the text, it first checks for co-occurring nouns (words that had been assigned a noun tag by the syntax checker). If there are none it will be unable to make a decision and so will simply continue through the text until it finds another semantic confusable. The hypernym trees were based on nouns that had co-occurred in a window size of +/- 2 words from the corresponding confusable in the BNC, but a large proportion of the confusables to be checked may not have a co-occurring noun in this vicinity. For instance, only 41% of the confusables in the FLOB corpus used for testing the initial program had a co-occurring noun within +/- 2 words. Increasing the size of the window that the spellchecker uses to retrieve co-occurring nouns means that it is able to check a larger number of confusables; there were noun co-occurrences for 97% of the confusables in the FLOB corpus when the window size was increased to +/- 10. Based on this finding, to maximize the number of confusable occurrences that the program could attempt to correct, the program has been set to check for co-occurring nouns within a window of ten words each side of a confusable (unless it encounters a sentence boundary first). However, the further away a noun is, the less likely it is to be related to the confusable so the scores retrieved for each noun are reduced as their distance from the confusable increases. (The scoring process is described in detail in Section 9.5.4 below.)

Occasionally, even when there is a co-occurring noun within the specified window, that noun will not be included in WordNet so the program will be unable to use it to calculate scores for the confusables. If this is the case for all co-occurring nouns within the ten-word window either side of a confusable, the program again simply makes no decision and continues through the text as described above.

9.5.2 Creating the merged trees

Once the program has found one or more co-occurring nouns that are included in WordNet, it retrieves the confusion set for the confusable from the dictionary and reads the hypernym trees

(Section 9.4.2) for each confusable lexeme from the file. It then creates a merged tree (Section 9.4.3) and uses this to score the confusables. So, for example, if the word to check is *pit* (Noun) with the confusion set {*pat*, *pet, pith*} the program will merge the hypernym trees for *pit, pat*, *pet* and *pith*.

The set of hypernym trees that need to be combined will depend on the word that has appeared in the text. If the word to be checked was the plural noun *pits*, the spellchecker could not use the same merged tree as for the singular noun *pit* since the confusion set for *pits* contains just one word – *pets* – so in this case it would merge just the *pit* and *pet* hypernym trees. (Although *pats* is included as a plural noun in the dictionary, it was excluded from consideration by the semantic checker because of its low frequency. *Pith*, the other member of *pit*'s confusion set is an uncountable noun and so does not have a plural form.) Similarly, although each word in a confusion set is considered confusable with the headword – *pat*, *pet* and *pith* are all confusable with *pit* – the members of a confusion set are not necessarily all confusable with each other. The confusion set for *pet*, for example, does not include *pith*, whereas the confusion set for *pith* contains *path* in addition to *pit*.

9.5.3 Retrieving WordNet senses and hypernyms for co-occurring nouns

The program next lists the WordNet senses and their associated hypernyms for each co-occurring noun. The initial experiment (Pedler, 2005) retrieved these directly from WordNet using the Perl QueryData module (Rennie, 2000) – also used to create the initial hypernym trees (Section 9.4.2 above). This module, once it has been initialised at the start of the program (which takes about ten to fifteen seconds) provides fast and efficient WordNet lookup; the documentation claims that "thousands of queries can be completed every second". However, although the overhead involved in accessing WordNet was not a particular problem in itself, this approach meant that the spellchecker could only be run on a machine that had WordNet installed. To overcome this limitation, and enable the spellchecker to run independently of WordNet, I decided to use QueryData to list each sense and its associated hypernyms (to level four, the deepest level stored in the trees) for each noun in my dictionary that was also included in WordNet. I then wrote these to file and added an index into the file to the dictionary entry for each of the corresponding nouns to enable the program to retrieve the senses/hypernyms it required from this file rather than needing to access WordNet at run-time.

9.5.4 Scoring the confusables

The program has now created a merged lexeme tree (Fig. 9-5) containing the relative likelihood scores for the semantic confusable it encountered in the text and each member of its associated confusion set, and retrieved one or more WordNet senses and their associated hypernyms for each noun that occurred in its immediate context (ten words either side). Using this information, the program now scores each of the confusables with each of the co-occurring nouns and combines these scores to give an overall score for each member of the confusion set. Based on these scores, the program either accepts the confusable it has seen as a correct spelling or flags it as an error. The scores are calculated as follows with explanatory notes for each numbered stage below:

Given:

$C\{c_1,....c_n\}$ Set of confusables
$N\{n_1,....n_m\}$ Set of co-occurring nouns
$\{s_1,....s_p\}$ Set of senses
$\{h_1,....h_4\}$ Set of hypernyms
$d(c_i, n_j)$ distance of co-occurring noun n_j from confusable c_i
$l(<n_j,s_k,h_l>)$ hypernym level for hypernym h_l for sense s_k of noun n_i
$lw(<n_j ,s_k>)$ weighting for sense s_k of noun n_j
$sm(n_j)$ proportional count of matching senses for n_j
$nm(c_i)$ proportional count of nouns co-occurring with confusable c_i
$dw(<c_i,n_j>)$ distance weighting for noun n_j co-occurring with confusable c_i

(1) $l(<n_j,s_k,h_l>) = \max(l)$ where $<c_i,h_l> = <n_j,s_k,h_l>$

(2) $lw(<n_j ,s_k>) = l(<n_j,s_k,h_l>)/4$

(3) $P(c_i|<n_j,s_k>) = P(c_i,h_l)lw(<n_j,s_k>)$ where $<c_i,h_l> = <n_j,s_k,h_l>$

(4) $sm(n_j) = \sum_{k=1}^{p} lw(< n_j , s_k >)$

(5) $dw(<c_i,n_j>) = (11 - d(c_i, n_j))/10$

(6) $P(c_i|n_j) = \sum_{k=1}^{p}(P(c_i \,|< n_j, s_k >)lw(< c_i, n_j >)/sm(n_j))dw < c_i, n_j >)$

(7) $nm(c_i) = \sum_{j=1}^{m} dw(< c_i, n_j >)$

(8) $P(c_i|\{ n_1,....n_m\}) = \sum_{j=1}^{m} P(c_i \mid n_j)/nm(c_i)$

(9) $c_{select} = max(P(C|N))$

(1) The program lists all the WordNet senses $\{s_1,....s_p\}$ for each co-occurring noun $\{n_1,....n_m\}$ and retrieves the hypernyms $\{h_1....h_4\}$ for each sense from WordNet.. For each sense it compares these with the hypernyms stored in the tree for the confusion set $\{c_1,....c_n\}$ to find the deepest level l match for that sense. Some senses will not match at all and are discarded at this point. For those that do match, l is assigned a value between 1 and 4, where 4 represents the more specific senses that match at the deepest level of the tree and 1 represents senses that match only at the more general root node.

(2) Although the senses at the root node are more general and thus less likely to be useful for distinguishing between the members of the confusion set, they have a higher probability of occurring than the more specific senses at lower levels of the tree (Fig. 9-4). As the lower level scores are of more value to the program in making its decisions, level weighting *lw* is calculated as $l/4$ for each matching sense. This will then assign the full value of the score to hypernyms matching at level 4, 75% of the value to those matching at level 3 and so on. (Hypernyms are stored to a depth of 4 in the tree so 4 is the deepest level match.)

(3) Each hypernym node in the merged tree stores the probability value for each member of the confusion set occurring given a noun with that hypernym sense (Fig. 9-5). We define the probability of a particular confusable, c_i occurring given sense s_k of noun n_j as the probability stored for confusable c_i in the hypernym node h_l corresponding to deepest level hypernym match of sense s_k of noun n_j, adjusted by the level weighting $lw(<n_j,s_k>)$for that sense.

(4) The number of matching senses *sm* for noun n_j is calculated as the sum of the level weightings for the matching senses of n_j as senses were matched proportionally as described in (2, 3).

(5) The scores for each noun are weighted according to their distance $d(c_i, n_j)$ from the confusable in the text. This weighting *dw* is calculated as 11- $d(c_i, n_j)$/10. (For a window size of 10 as used in the current implementation.) This thus assigns the full score to collocates ((11-1)/10) = 1), 0.9 of the score to nouns occurring two words away and so on.

(6) The probability of confusable c_i occurring given co-occurring noun n_j is sum of the scores calculated in (3), normalised by the 'sense count' calculated in (4) and weighted by the distance weighting *dw* calculated in (5).

(7) The number of nouns co-occurring with confusable c_i is calculated as the sum of the distance weightings *dw* for nouns co-occurring with c_i, since each noun was matched proportionally according to its distance from the confusable. (This is similar to the calculation of the sense counts in (4)).

(8) The probability of confusable c_i occurring given all nouns $\{n_1,....n_m\}$ that co-occurred with it is the sum of the probability for c_i occurring with each noun as calculated in (6) normalised by the number of nouns matched, *nm*, calculated in (7).

(9) The program selects the confusable with the highest score as its preferred word in the context. If the word that originally appeared in the text matches the confusable with the highest score, the program accepts it as a correct spelling; if the word in the text is different from the confusable with the highest score, it is flagged as an error and the highest scoring confusable is proposed as a correction.

9.5.5 An example

I will illustrate the scoring process with the following example from the FLOB Corpus used for the initial experiment:

Seventeenth century dolls *carved* from wood fetch very high prices...

The program works outwards from the confusable *carved* to identify the nouns *doll*, *century*, *wood* and *price*. It then retrieves all the WordNet senses for each of these – two for *doll*, eight for *wood*, two for *century* and seven for *price* – and lists their hypernyms. It finds a match in the *carve/crave* tree for both senses of *doll*, four for *wood*, one for *century* and six for *price*.

More than one of the matched senses may map to the same hypernym, as can be seen from Table 9.3 which shows the senses retrieved for *wood* and their matching hypernyms. All the matched senses score highly for *carve* but the least relevant sense in this case, that of *wood* as *forest*, only matches at a more general level so only 50% of its score (shown in parentheses in the table) will contribute to the final value assigned for *wood*. Combining the scores and normalising by the number of senses matched (which is the sum of the weighting given to each hypernym match as only a proportion of the sense score has been given to hypernyms at higher levels of the tree) gives a final score of 0.990 for *carve* and 0.010 for *crave* co-occurring with *wood*.

Sense	**Hypernym match**	**Level**	**Weight**	**Score**	
				carve	***crave***
wood (hard fibrous substance)	plant material	4	1	0.993	0.007
forest, wood, woods	collection	2	0.5	0.976 (0.488)	0.024 (0.011)
woodwind, woodwind instrument, wood	instrumentality	4	1	0.992	0.008
wood (golf club)	instrumentality	4	1	0.992	0.008
Sum of scores			3.5	3.465	0.034
Normalised scores				0.990	0.010

Table 9.3: Scores for matching senses of *wood* in the {*carve, crave*} hypernym tree

The program calculates scores in the same way for each of the other co-occurring nouns and weights the score for each according to its distance from the confusable – it gives the full score to any nouns that occurred next to it, 0.9 of the score to those that were two words away and so on.

Table 9.4 shows the final scores for each co-occurring noun in this example with the distance-weighted values in parentheses. These scores are then summed and normalised by the number of nouns matched (the sum of the distance weightings) to give a final score, in this case, of 0.776 for *carve* and 0.224 for *crave*, correctly preferring *carve*.

			Score	
Co-occurrence	**Distance**	**Distance weight**	***carve***	***crave***
doll	1	1	0.899	0.101
century	2	0.9	0.432 (0.388)	0.568 (0.512)
wood	2	0.9	0.990 (0.891)	0.010 (0.009)
price	6	0.5	0.768 (0.384)	0.232 (0.116)
Sum of scores		3.3	2.562	0.738
Normalised scores			0.776	0.224

Table 9.4: Final scores for each co-occurring noun

9.6 Testing

The dyslexic error corpus (Chapter 3) was used to test the program. There are almost two thousand confusables in this corpus that have some or all part-of-speech tags in common with one or more members of their associated confusion set and which therefore might be passed to the semantic checker - Table 9.5. However, not all of them will in fact be passed on. In the majority of cases the tagsets overlap which means that, if the syntax checker selects a tag that is unique to one of the words, they will not be considered further by the semantic checker. In addition, even when the syntax checker has assigned a tag that one or more of the confusables have in common, the confusables will only be passed to the semantic checker if there is an associated hypernym tree.

Overlapping	1626
Matching	246
Both overlapping and matching	56
Total	1928

Table 9.5: Types of confusables in the error corpus for possible consideration by semantic checker

After the syntax checker had been run over the error corpus (with the handicap for the words not appearing in the text set at 0.01 as described in the previous chapter) there were just over 600 confusables flagged for consideration by the semantic checker. The majority of these words were correctly used as might be expected. Although all these words were flagged for consideration, the semantic checker was unable to make a decision in around 10% of cases as there were no co-occurring nouns within the +/- 10 word window used by the program. Table 9.6 shows the number of words flagged for consideration by the semantic checker and the number of these that would actually be checked by the program as there were co-occurring nouns associated with them.

	All	**No co-occurring nouns**	**Checked**
Correctly used	529	50	479
Errors	115	14	101

Table 9.6: Number of confusables in the error corpus flagged for consideration by the semantic checker

The program was initially set to select the word with the highest score and then run at varying 'confidence levels'. The confidence level was implemented as a 'handicap' in the same way as for the syntax checker by successively reducing the weighting given to the score for the alternative word from 1 down to 0.01. Table 9.7 shows the results of running the semantic checker over the dyslexic error corpus using these varying levels of handicap. As noted above, there were no co-occurring nouns for about 10% of the confusables flagged for consideration. In these cases, if the word was correctly used it would be accepted and if it was an error it would be ignored. Thus these words, although they are not actually considered at all will contribute to the proportion of words for which the checker performs 'correctly' for correct usages and 'incorrectly' for error usages. Since this figure is a constant at each handicap level it has not been included in the counts given in Table 9.7 below.

	Correct Usage (479)		**Error Usage (101)**			**Improvement**		
Hcap level	**Accept**	**False alarm**	**Flag & correct**	**Flag not correct**	**Ignore**	**Errors remaining (b+d+e)**	**Error reduction (101 – f)**	**Imprvt %**
	a	**b**	**c**	**d**	**e**	**f**	**g**	**h**
1	412	67	44	13	44	124	-23	-23%
0.5	445	34	42	9	50	93	8	8%
0.2	465	14	38	4	59	77	24	24%
0.1	469	10	36	4	61	75	26	26%
0.05	474	5	26	2	73	80	21	21%
0.01	479	0	12	0	89	89	12	12%

Table 9.7: Results of running semantic checker on error corpus

The table shows the number of correct and error usages considered by the semantic checker falling into each of the categories as described for the syntax checker in the previous chapter. As with the syntax checker, although the largest proportion of errors is corrected when the full weighting is given to all the words being considered, this results in an unacceptable level of false alarms and an overall increase in the number of errors in the text. Reducing the weighting given to the alternative words to 0.5 gives a small overall improvement in the text but the greatest overall improvement is achieved when this is set to 0.1. It is possible, with the handicap set at 0.01, to reduce the false alarms to zero and still correct a small proportion of the errors (a total of 12 corrections) but it seems better to accept a small number of false alarms while at the same time correcting a larger proportion of the errors. This is illustrated graphically in Fig. 9-6.

Although setting the handicap to 0.1 maximizes the improvement made by the program for this corpus, the improvement is only slightly reduced at 0.2 which achieves a better rate of error correction at the expense of a small increase in false alarms. This makes it tempting to try a further level of handicap, such as 0.15, set somewhere between these two levels. However, far fewer errors were considered by the semantic checker than by the syntax checker and optimizing the performance of the semantic checker for this corpus is likely to have the effect of over-fitting the program to this data rather than suggesting an optimal level to set for unseen text. Based on these considerations, I opted to select 0.2 as the handicap level for the semantic checker in the evaluation described in the next chapter.

9.7 Conclusion

There are almost two thousand words with matching or overlapping part-of-speech tagsets in the dyslexic error corpus (Table 9.5). Around a third of these were flagged by the syntax checker for further consideration by the semantic checker; the syntax checker will already have made a decision for many of the overlapping sets and only passes words to the semantic checker if they have an associated hypernym tree. For these reasons, the semantic checker is only applicable for about 12% of the total errors in the corpus. However, it managed to correct 38% of these, which represents a modest but useful contribution.

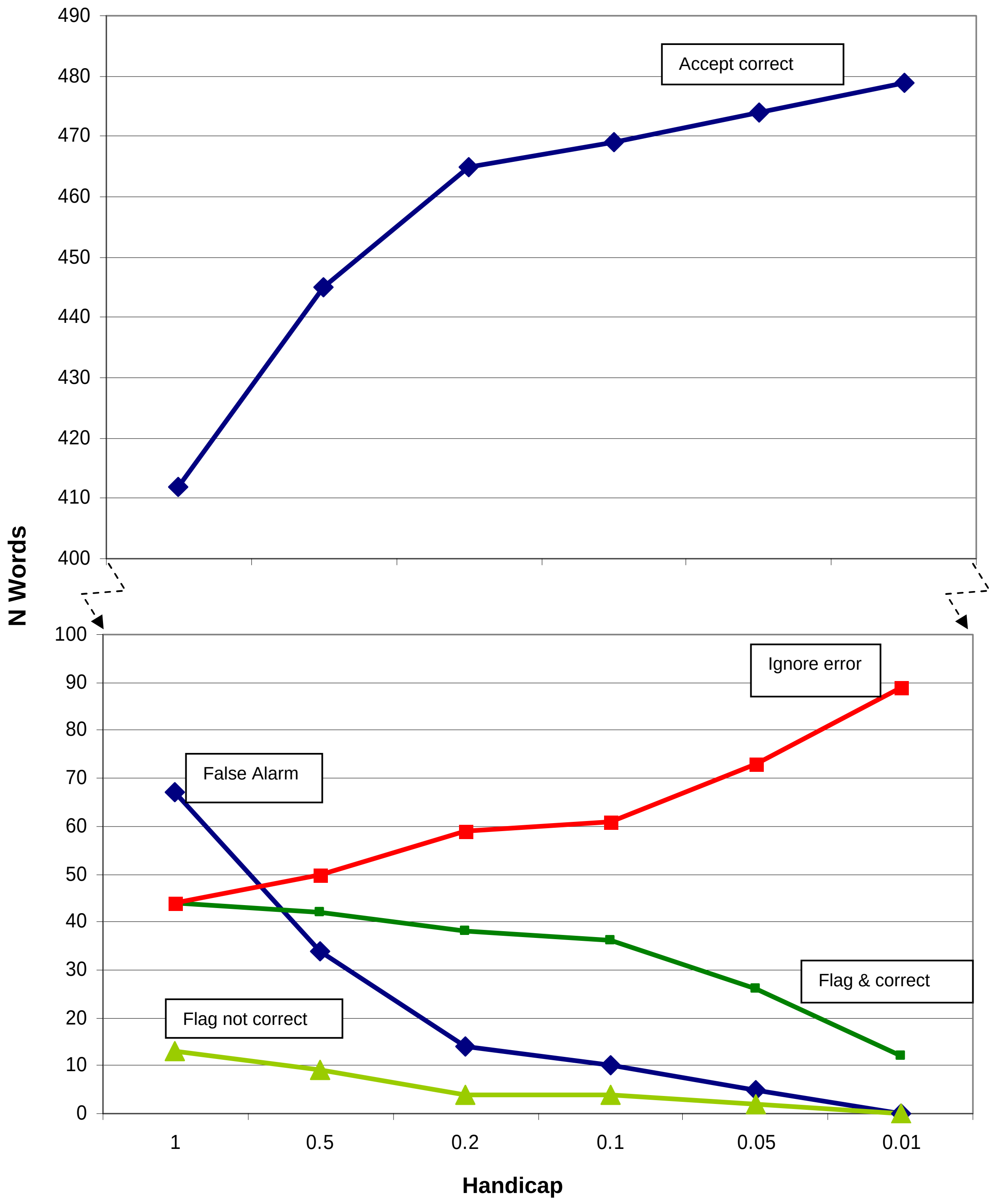

Fig. 9-6: Performance of semantic checker with different levels of handicap

Chapter 10: Evaluation

The spellchecker developed in this research relies on its built-in confusion sets both to detect and to correct errors. Thus it will only consider a word as a potential error if it has a confusion set associated with its dictionary entry and it can only suggest the intended word as a correction for an error if that word is a member of its confusion set. Because of this inherent limitation of the confusion set approach, a large number of confusion sets are required if it is to be used for general purpose spellchecking; this spellchecker uses almost six thousand. However, even when using a large collection such as this, there will be errors that remain undetected or that are miscorrected either because the error itself has not been defined as confusable or because, although the error has a confusion set associated with it, the intended word is not among its members.

Most previous work using confusion sets has focused on developing techniques for selecting the correct word in a given context using a limited number of confusion sets (generally pairs). Carlson et al. (2001) address the issue of scaling up the approach and use 265 confusion sets (over 500 confusable words) but, by their own admission, this still falls far short of the number of confusion sets needed to make a realistic impact on the problem. In addition to using only a small number of sets, these previous experiments have not attempted to correct real errors; they have been tested either on correctly spelled text or on text into which errors have been artificially introduced.

However, when a spellchecker is tested on text containing real errors – such as the error corpora used for the experiment reported in this chapter – the errors cannot be so neatly controlled. In this situation there are two factors that influence the program's correction rate – its ability to detect errors in the first place (which depends on the confusion sets) and its ability to propose the correct word once it has identified a potential error (which depends both on the confusion sets and on the efficiency of its decision-making mechanism). Some errors will remain undetected simply because the word involved is not included among the program's confusion sets and so they will not even be considered as potential errors. Errors that do have a confusion set associated with them will at least be considered but will also go undetected by the program if it decides that the error is more likely to be the correct word in the context than any of the alternative members of the confusion set. In both cases the program's error detection rate is reduced but the cause of the failure is different for each. The first case demonstrates a limitation of the confusion set approach; short of defining every word in the dictionary as potentially confusable with every other word (which is clearly impractical) it is impossible to ensure that all possible errors are included in a program's confusion sets. Additionally some errors, such as inflection errors, do not seem to be suitable candidates for inclusion in confusion sets. The second case, where the program is alerted to the possibility of an error but wrongly accepts the word as correct, can be either a deficiency in the confusion sets – the intended word is not included and the error is more likely in the context than any of the alternatives offered - or a shortcoming of the scoring mechanism used by the spellchecker. This is summarized in Table 10.1.

	Ignore	**Detect but miscorrect**	**Correct**
Error not in confusion set	✓		
Error in confusion set but target not	✓	✓	
Error and target in confusion set	✓	✓	✓

Table 10.1: Summary of possible decisions for real-word errors

In this chapter I consider both the performance of the spellchecker that I have implemented in this research and the overall effectiveness of the confusion set approach to spellchecking.

10.1 Error corpora

The main aim of this research was to develop a spellchecker that would correct real errors in actual text produced by dyslexics. A realistic assessment of its ability to perform this task required real error data; my efforts to produce the dyslexic error corpus described in Chapter 3 demonstrated the difficulties of acquiring this. The sub-corpus of real-word errors that has been used throughout the development of the program described in previous chapters was derived by selecting sentences containing real-word errors from the initial corpus. Although this corpus could not in the strict sense be regarded as training data, as the tag bigram and word co-occurrence data used by the spellchecker was derived from the BNC, it was used during program development – for example to assess the relevance of the confusion sets and to set the handicap level for the words not appearing in the text – which must, inevitably, have introduced a bias toward correcting the errors it contains. Therefore, for the final assessment of the program presented in this chapter, in addition to assessing the overall performance on the original dyslexic error corpus, I also report results obtained from two additional, unseen error corpora.

The collection of real-word error samples has been an on-going task throughout the course of this research. I gathered sentences containing real-word errors from assignments submitted by my students and also from work that my daughter gave me to proofread. This resulted in a corpus containing almost three thousand words and just under 200 real-word errors. Although I had no way of knowing whether the students were dyslexic, undoubtedly a number of them were and the errors collected appear to be of a similar nature to the errors in the original corpus. In addition to this, Roger Mitton supplied me with a large corpus of school leavers' compositions (not including those incorporated into the original corpus). These compositions were written by pupils of varying ability; in some the overall level of spelling is poor, suggesting that the writer may well have been dyslexic, whereas in others the spelling is generally good with just the occasional error. However, as it was not possible to make an accurate judgement as to which (if any) of the writers were dyslexic, I simply extracted any sentences that contained real-word errors and included them in the Compositions error corpus. Another difference between these two corpora is that the students' work was originally word-processed whereas the compositions were transcribed from handwritten originals. Some common types of dyslexic error, such as writing *b* for *d* (for example, *bark* for *dark*) are probably less likely to occur when the text is typed.

Table 10.2 gives a breakdown for each of these error corpora. The Compositions corpus is the largest of these with almost twenty thousand words and over one thousand errors. However, since all the compositions are about the same subject – memories of primary school – the vocabulary is rather restricted and repetitive.

	Dyslexic	Students	Compositions
Sentences	614	192	852
Words	11810	2661	19179
Real-word errors	830	199	1049

Table 10.2: Composition of error corpora

10.1.1 Errors and confusable words in the corpora

Table 10.3 shows the proportion of real-word errors that the program can be expected to correct in each corpus. The detectable errors are the proportion of all real-word errors that will be considered by the program (the error is the headword of a confusion set) and the correctable errors the proportion of all real-word errors that can be corrected (the error is the headword of a confusion set and the intended word is a member of its associated confusion set). As this table shows, a rather high proportion (between a quarter and a third) of the errors are undetectable even with over 6000 confusion sets.

	Dyslexic		Students		Compositions	
	N	%	N	%	N	%
Detectable errors	644	78%	136	68%	738	70%
Correctable errors	481	58%	94	47%	380	36%
Real-word errors	830	100%	199	100%	1049	100%

Table 10.3: Detectable and correctable errors in error corpora

Although the occurrence of a confusable is the cue for the spellchecker to investigate, most occurrences of the confusables are correct (even in these sentences selected because they contain at least one real-word error). Table 10.4 shows the proportions of correct and error usage of these confusables in each corpus.

	Dyslexic error		Students		Compositions	
	N	**%**	**N**	**%**	**N**	**%**
Correct usage	5210	89%	1055	89%	9449	93%
Error usage	644	11%	136	11%	738	7%
All confusables	5854	100%	1191	100%	10187	100%

Table 10.4: Correct and error usage of confusables in error corpora

Table 10.3 shows the number of errors in each corpus that we would like the spellchecker to correct and the proportion of these that it can be expected to correct using our confusion sets. Table 10.4 shows the number of confusion set headwords – appearing either correctly used or as errors – that the spellchecker will consider as potential errors. In both cases, these are counts of word *tokens*. However, since many of the words occur more than once, the count of word *types* is lower than this. The ratio of types to tokens (Table 10.5) gives us a 'variety index' for the text; a higher ratio indicates more variation in the vocabulary used, whereas a lower one indicates that many words were repeated.

The table shows this ratio for the correctly used confusables and for the errors appearing in each corpus. The errors are further broken down into non-confusable errors (words that do not have a confusion set associated with them), non-correctable (the error is a confusion set headword but the target not in its confusion set) and correctable (the error is a confusion set headword and the target is a member of its confusion set).

	Dyslexic error			**Students**			**Compositions**		
	Tok	**Type**	**%**	**Tok**	**Type**	**%**	**Tok**	**Type**	**%**
Correct Usage	5210	607	12%	1055	229	22%	9449	677	7%
Error usage All	830	493	59%	199	148	74%	1049	540	51%
Non-confusable	186	145	78%	62	57	92%	311	207	67%
Non-correctable	163	131	80%	43	38	88%	358	183	51%
Correctable	481	217	45%	94	53	56%	380	150	39%

Table 10.5: Type:token ratio for confusables and errors in error corpora

The low type:token ratio for the correctly used confusables in the Compositions corpus is unsurprising in the light of earlier observations about the repetitive nature of this corpus. However, this ratio is also significantly lower for the correctly used confusables than it is for the errors in each of the corpora whereas if the errors were purely random we would expect it to be about the same. The reason for this is that there are a number of very common words among the confusables and, as a word is more likely to be correctly spelled than it is to be an error, these words appear many more times as correct usages than they do as errors – *were*, for example, occurs 162 times in the Compositions corpus and only 26 of these occurrences are errors.

If we were considering the performance of a non-word error checker that was dealing with each word in isolation, we would only be interested in error types as each occurrence of a particular error would be dealt with in the same way. However, a real-word error checker is dealing with each word in the context in which it occurs and so may make a different decision for the same confusable word appearing in a different context. For this reason, the results reported below use token rather than type counts.

10.2 Spellchecker performance

10.2.1 Error correction

Table 10.6, Table 10.7 and Table 10.8 show the results obtained from running the spellchecker over each of the error corpora described above. The first row of each table (Select most frequent) gives the results obtained from the baseline select-the-most-frequent method described in Chapter 7. This is included as a baseline performance measure against which to compare the performance of the confusion-set checker. The combined syntax and semantic checker was then run over each corpus both with and without a handicap applied to the alternative word(s) in the confusion set. The second row (Non-handicapped score) gives the results when all words in the confusion set are considered equally likely and the program makes its selection based on the one achieving the highest overall score. The third row (Handicapped score) gives the results when the expectation that the word seen in the text will be correct is factored in by handicapping the scores of the alternative words. The handicaps used were those assessed to be optimum during the development phase – 0.01 for the syntax checker and 0.2 for the semantic checker.

There are just two possibilities for the correctly used confusables:

- Correctly accepted;
- Erroneously flagged (false alarm).

For the errors there are four:

- Correct error – the program flagged the error and proposed the intended word as a replacement;
- Flag but miscorrect – the program flagged the error but the proposed correction was not the intended word;
- Accept error – the error was considered by the program but wrongly accepted as correct;
- Ignore error – the error was not considered by the program as it did not have an associated confusion set.

Since the ignored errors are never considered by the program their number remains constant for each scoring method. From an end-user's perspective the accepted and ignored errors amount to the same thing – errors that remain silently uncorrected. However, to enable us to assess the program's performance both for all errors and for the errors that it has been designed to correct we need to consider them separately.

	Correct usage (5210)		**Errors (830)**			
Scoring method	**Accept correct**	**False alarm**	**Correct error**	**Flag but miscorrect**	**Accept error**	**Ignore error**
Select most frequent	4406	804	292	107	245	186
Non- handicapped score	4781	429	359	70	215	186
Handicapped score	5177	33	256	28	360	186

Table 10.6: Error correction – dyslexic error corpus.

	Correct usage (1055)		**Errors (199)**			
Scoring method	**Accept correct**	**False alarm**	**Correct error**	**Flag but miscorrect**	**Accept error**	**Ignore error**
Select most frequent	904	151	63	24	49	63
Non- handicapped score	956	99	71	16	49	63
Handicapped score	1046	9	38	7	91	63

Table 10.7: Error correction – students

	Correct usage (9449)		Errors (1049)			
Scoring method	**Accept correct**	**False alarm**	**Correct error**	**Flag but miscorrect**	**Accept error**	**Ignore error**
Select most frequent	8211	1238	186	114	438	311
Non- handicapped score	8800	649	248	127	363	311
Handicapped score	9398	51	135	38	565	311

Table 10.8: Error correction – compositions

As the results above show, using the confusion sets, even with the non-handicapped scores, both reduces the false alarms and increases the number of errors corrected compared to the baseline select-the-most-frequent algorithm. However, without the handicapping, the false alarms still outnumber the errors corrected which is clearly unacceptable. Introducing the handicap reduces the number of false alarms while at the same time correcting a proportion of the errors.

Table 10.9 shows total errors corrected as a proportion of all errors (the overall performance as it would appear to an end-user), of the detectable errors (the errors that are considered by the spellchecker) and of the correctable errors (the errors that can be corrected using the confusion sets).

Although the program corrects almost a third of all errors in the original dyslexic error corpus, this proportion drops to 19% and 13% respectively in the Students and Compositions corpora. This would be expected as the proportion of correctable errors is lower in these corpora (Table 10.3). For the errors that it has been designed to correct the program performs reasonably well, correcting between around a third to half of all correctable errors. Performance on the correctly used confusables also seems acceptable with less than 1% false alarms. However, these figures in themselves are not an adequate measure of the spellchecker's performance.

	Dyslexic		**Students**		**Compositions**	
Total corrected	256		38		135	
out of:						
All errors	830	31%	199	19%	1049	13%
Detectable errors	644	40%	136	28%	738	18%
Correctable errors	481	53%	94	40%	380	36%
False alarms	33		9		51	
out of:						
Correctly used confusables	5210	0.6%	1055	0.8%	9449	0.5%

Table 10.9: Errors corrected as a proportion of all errors, detectable errors and correctable errors, compared to proportion of false alarms (Handicapped score)

10.2.2 Overall improvement

If the program's decision is incorrect even for just a small proportion of the correctly used confusables there will be a large number of false alarms. However, provided the program corrects more errors than it raises false alarms, there will be a net improvement in the 'correctness' of the text. This suggests that the key measure of spellchecker performance is the overall improvement in the text after the program has run over it. This choice of measure seems particularly relevant for dyslexics who are likely to accept most of the spellchecker’s suggestions.

If we assume that the spellchecker's suggestion is always accepted, then subtracting the corrected errors from the number of original errors and adding the false alarms gives us the number of errors remaining in the text after it has been spellchecked. The difference between this and the number of errors originally in the text gives the decrease in the total number of errors. Taking this 'error decrease' as a proportion of the number of original errors gives us the percentage improvement in the text. (Note that this counts the flagged but miscorrected errors as no better than the ignored, which might be a bit hard on the spellchecker.) These figures are given for each of the error corpora in Table 10.10. The table shows this improvement percentage both as a proportion of all errors in the text and as a proportion of the detectable errors.

	Dyslexic	**Students**	**Compositions**
Original errors (all)	830	199	1049
Original errors (detectable)	644	136	738
Errors corrected	256	38	135
False alarms	33	9	51
Errors remaining	421	107	654
Error decrease	223	29	84
Overall improvement (%) (All errors)	27%	15%	8%
Overall improvement (%) (Detectable errors)	35%	21%	11%

Table 10.10: Overall improvement (Handicapped score)

Estimating the overall improvement in the correctness of the text after the spellchecker has been run over it can be regarded as a practical measure of the usefulness of the spellchecker. Although it is obviously a useful yardstick of spellchecker performance, it does not appear to have been used to assess other work. This is presumably because most of the previous context sensitive text correction work using confusion sets has been tested on correctly spelled text (where, on the assumption that the text is 100% correct, any incorrect decisions made by the program would result in an overall worsening of the text). Evaluation in this previous work has focused on the ability of the program to make correct decisions rather than on its practical use as a spellchecker.

10.2.3 Recall and precision

Recall and precision, which has its origins in the field of information retrieval, is a commonly used metric for the evaluation of natural language processing tasks.

Recall measures the 'completeness' of retrieval. In terms of spellchecking this tells us the proportion of the errors in the text that were detected, calculated by dividing the number of errors detected by the total number of errors in the text. For example, if there were 100 errors in a text and a program detected 90 of them it would have a recall of 90/100 = 90%. The recall value is not affected by the number of false alarms that are produced; a program that flagged every word in a text as an error would achieve a recall of 100%.

Precision measures the accuracy of retrieval. For a spellchecker this represents the proportion of words flagged as errors that actually were errors, calculated by dividing the number of errors detected by the total of words flagged. In this case, using the same 100 error example, if, in addition to correctly detecting 90 of the 100 errors in the text the program had also produced 30 false alarms the total of words flagged would be 120 and the program's precision would be calculated as 90/120 = 75%. The precision value is not affected by the number of errors remaining undetected; a program that only detected a small number of the total errors but never raised any false alarms would achieve a precision of 100%.

The two measures need to be taken together for an overall assessment of a spellchecker. High recall coupled with low precision indicates that although a large proportion of the errors were detected, there were also a large number of false alarms whereas low recall coupled with high precision indicates that although the program was generally correct when it flagged an error, a large number of errors went undetected. Ideally we would like a program to achieve a high score for both recall and precision indicating that a large proportion of the errors were detected at the expense of very few false alarms. However, this is an ideal situation as there is a trade-off between recall and precision. To improve recall we need to relax the rules used by the program thus making it more willing to flag a word as an error. This is likely to cause it to raise more false alarms. To improve precision we need to tighten the rules, making the program less willing to flag a word as an error. This means that the words it flags are more likely to be errors but that fewer of the errors overall will be flagged.

Dyslexics have little confidence in their ability to spell and make a large number of mistakes; if the spellchecker flags a word as an error they are likely to assume that the spellchecker is right, even when this is not the case. This will result in correctly spelled words turning into real-word errors and an overall worsening of the text. Thus for a spellchecker designed for use by dyslexics, it is better to make an accurate decision for a small but useful proportion of the errors and produce few false alarms than it is for it to flag a large proportion of the errors at the expense of a large number of false alarms. This means sacrificing some recall in order to achieve greater precision which is the effect of the handicap used by my program (and the various confidence threshold measures used by others).

Recall and precision scores for my spellchecker when run over each corpus (using the handicapped scores) are given in Table 10.11. Recall has been calculated both as a proportion of all errors in the corpus and as a proportion of the detectable errors. The first of these shows the program's overall efficiency at detecting errors while the second shows how effective it is at detecting the errors included in its confusion sets. Precision has been calculated both for detection (the proportion of words flagged that were errors) and correction (the proportion of words flagged that were errors and for which the intended word was proposed as a correction).

	Dyslexic	**Students**	**Compositions**
Total errors	830	199	1049
N. Detectable errors	644	136	738
N. Words flagged	317	54	224
N. errors detected	284	45	173
N. errors corrected	256	38	135
Recall (all)	34.2%	22.6%	16.5%
Recall (detectable)	44.1%	33.1%	23.4%
Precision (detection)	89.5%	83.3%	77.2%
Precision (correction)	80.7%	70.3%	60.3%

Table 10.11: Recall and precision

10.2.4 Comparative recall and precision

The only previous research that has made a realistic attempt to correct actual errors is that of Atwell and Elliott (1987). They compiled a corpus of around 13,500 words containing just over 500 errors. This Lancaster corpus was derived from four sources: a collection of word-processed documents from ICL's Knowledge Engineering Group; the Gill Corpus, compiled by Dr Gill of Warwick University to test an automatic brailling machine – they obtained this from the Oxford Text Archive though it seems no longer to be available there; the *Manual of Information* for the LOB corpus that lists all the errors found in the original texts (some but not all of these were subsequently corrected in the electronic version); a collection of business-style documents that had been typed by ITEC trainees on work experience. Stephen Elliott kindly made a copy of this corpus available to me which has enabled me to make some comparison between the recall and precision scores reported in their work with that achieved by my spellchecker when run over the same corpus. However, there are several difficulties in making a comparison between these two pieces of work.

Almost a quarter of the errors in the Lancaster corpus involve an omitted possessive apostrophe – *boys* in error for *boy's*, for example – and a number of these occur for proper nouns, including over 30 occurrences of *Croziers* in error for *Crozier's* in the Gill corpus sample. Apart from the six occurrences of *it's* as an error for *its* (or vice-versa), none of these errors can be detected by my program and although we would like a program to be able to correct errors of this type they are more akin to grammar errors than real-word spelling errors and not natural candidates for a confusion-set approach. A further 19% of the errors in the Lancaster corpus are inflection errors which again are not amenable to the confusion-set approach. As the proportions of detectable and correctable errors in this corpus (Table 10.12) show, my program would be expected to perform poorly on this corpus; only half of the errors in this corpus are included in my confusion sets and less than half of these are actually correctable.

	N	**%**
Detectable errors	221	44%
Correctable errors	89	18%
Real-word errors	502	100%

Table 10.12: Detectable and correctable errors in Lancaster corpus

Not only are many of the errors in the Lancaster corpus of a different nature to those that my program was designed to correct, they also make up a smaller proportion of the total words in the corpus – 4% compared to between 5% and 7% in my corpora (Table 10.2). This suggests that, in a sense, their corpus represents the types of error made by relatively good spellers rather than the errors of poor spellers which constitute the bulk of my corpora.

In addition to the differences in the corpora, there are also differences of implementation to take into account. Atwell and Elliott's spellchecker was not limited to correcting predefined confusable words although, since it used tag-bigram probabilities to detect errors, it was confined to syntactic errors that were recognisable within a narrow context. Errors that did not meet these criteria were removed from the corpus and stored in a separate 'hard' file. For example, the error in this fragment has the same part-of-speech as the target:

> ...a *pope* (*pipe*) cleaner can be used when necessary.

and in this a wide context is needed to detect the error:

> ...it was unfortunate that a garden party at the home of our
> chairman and his wife *has* (*had*) to be cancelled due to...

In order to detect errors, their program used the likelihood of the occurrence of a particular tag pair. If the tag pair probability returned by the CLAWS tagger fell below a predefined threshold, the word was flagged as an error. They experimented with two threshold levels; increasing the level improved precision but reduced recall (as would be expected from the discussion above). They report recall and precision scores for a lower and higher threshold (the exact measure for each of these is not important here) for each component of their corpus, excluding the 'hard' errors and also for the hard errors which, as they comment, would only be detected by chance factors. In their research they give scores for each component of their corpus separately. However, for comparison with my program, the scores have been aggregated. This means that the lower recall and precision scores for the hard (as compared to the non-hard) errors reduces the overall scores achieved by Atwell and Elliott's program. On the other hand, the recall and precision scores achieved by my program are reduced by the high proportion of errors that are undetectable using my confusion sets

(Table 10.12). Therefore it seems that including all errors in the results achieved by both programs allows the most direct comparison between the two pieces of work.

Table 10.13 compares the recall and precision scores achieved by both programs for error detection of all errors in the Lancaster corpus using Atwell and Elliott's higher threshold and the handicapped scores for my program. In each case these are the scores that optimise the precision level achieved by the program thus reducing the number of false alarms produced.

	Atwell & Elliott (higher)	**Pedler (handicap)**
N.Errors	502	502
Total flags	608	51
Errors flagged	202	44
Recall	40%	9%
Precision	33%	72%

Table 10.13: Comparative recall and precision scores for Lancaster corpus

As the table shows, Atwell and Elliott's program flags far more of the words and so achieves a higher recall but my program achieves a higher accuracy (better precision).

10.2.5 Prediction accuracy

Prediction accuracy is a measure of the program's ability to select the correct word from a set of alternatives in a given context expressed as the proportion of correct decisions out of all decisions made for confusable words in a text. This is the method used to evaluate the performance of most of the previous experiments with confusion sets described in Chapter 2, many of which achieve an accuracy level above 90%. However, although this is a fairly high level of performance, Carlson et al (2001) observe that a level of 98 - 99% is required for a practical system and report achieving this level on average for the confusion sets used in their experiment.

Table 10.14 shows the prediction accuracy achieved by my spellchecker (using the handicapped scores) for each corpus calculated as the proportion of correct decisions (correct words accepted plus errors corrected) out of all decisions made (the total number of confusable words in each corpus).

	Dyslexic	**Students**	**Compositions**
Accept correct	5177	1046	9398
Correct Error	256	38	135
Total correct	5433	1084	9533
All decisions (100%)	5854	1191	10187
Prediction accuracy	93%	91%	94%

Table 10.14: Prediction accuracy (handicapped scores)

Initially this seems to be a respectable figure although it falls short of the suggested threshold proposed by Carlson et al., mentioned above. However, although prediction accuracy can be used to measure a program's ability to select the correct word from a group of alternatives in a given context (which is the way in which it has been applied in the other confusion set experiments discussed previously) it does not tell us much about its ability to correct errors. For example, a program with a prediction accuracy of 95% would make five incorrect decisions for 100 confusables encountered. If these 100 confusables also happened to contain five errors the program could never actually make any improvement to the overall correctness of the text; if it corrected all the errors it would have to make an incorrect decision for five of the correctly spelled words thus negating the effect of correcting the errors by introducing an equal number of false alarms. Conversely, if it accepted all the correctly spelled words it would have to make an incorrect decision for all the errors and leave them uncorrected. If however, there were a larger number of errors in the text (say 10) the errors corrected would always outnumber the false alarms if the program had a 95% prediction accuracy.

Thus the prediction accuracy of a spellchecking program must be higher than the proportion of correctly spelled confusables in the text it is checking if it is to make any impact on the errors. For instance, text containing no errors is 100% correct. In order not to introduce errors the spellchecker would also need to have a prediction accuracy of 100%. However, if the text was only 90% correct a prediction accuracy of over 90% must result in some errors being corrected. Since in all cases the prediction accuracy of my program is higher than the 'correctness value' of the corpora (Table 10.4) it corrects more errors than it introduces false alarms.

10.2.6 Performance measures compared

The performance measures discussed above are summarised in Table 10.15. This table uses the recall and precision figures for the errors corrected since these provide a better comparison with the prediction accuracy and improvement figures both of which reflect the proportion of errors corrected.

	Dyslexic	**Students**	**Compositions**
Prediction accuracy	93%	91%	94%
Recall (correction)	40%	28%	18%
Precision (correction)	81%	70%	78%
Improvement (%)	37%	21%	11%

Table 10.15: Performance measures summarised (handicapped scores)

The program achieves a prediction accuracy of over 90% for each corpus and by this measure performs best on the Compositions corpus. However, it actually makes a much smaller improvement for this corpus than for the other two. This suggests that, when derived from real error data, prediction accuracy is not in fact a very useful statistic. None of the experiments that use this as a measure of performance have used real error data; they have either been run over correctly spelled text or text where errors have been artificially introduced by replacing a correct word with one of the members of its confusion set.

10.3 Non-correctable errors

Around a half of the errors in the corpora used for this evaluation cannot be corrected using the collection of confusion sets created for this research (Table 10.16). These can be subdivided into:

- Undetectable – errors that do not have a confusion set associated with them and so will not even be considered by the spellchecker as potential errors.
- Uncorrectable – errors that, although they have a confusion set associated with them and may be flagged as errors by the spellchecker, cannot be corrected as the intended word is not a member of the confusion set.

These non-correctable errors demonstrate a shortcoming of using confusion sets for general-purpose spellchecking. Most previous experimental confusion set work has been evaluated using correct text or artificial error data created by replacing a correctly spelled confusable with one of the members of its confusion set, and therefore has not needed to address the problem of non-correctable errors as any errors that were introduced into the text would, by definition, be correctable.

	Dyslexic		**Students**		**Compositions**		**All**	
	N	**%**	**N**	**%**	**N**	**%**	**N**	**%**
Non-correctable (all)	349	42%	105	53%	669	63%	1123	54%
Undetectable	186	22%	62	31%	311	30%	559	27%
Uncorrectable	163	20%	43	22%	358	34%	564	27%
All errors	830	100%	199	100%	1049	100%	2078	100%

Table 10.16: Proportions of non-correctable errors in the corpora

This section considers possible approaches that might be applicable to detecting and correcting these non-correctable errors.

10.3.1 Inflection errors

Over a third of the undetectable or uncorrectable errors are inflection errors (Table 10.17). The majority of verb inflection errors involve the base form of a verb appearing as an error for one of its inflected forms (*seem* instead of *seemed*, *move* instead of *moving*, for example) but some also occur for other forms (e.g. *remained* in error for *remaining*, *requires* for *required*). Similarly most, but not all, of the noun inflection errors occur when the singular form of a noun is used where the plural form was intended.

	Dyslexic		Students		Compositions		All	
	N	**%**	**N**	**%**	**N**	**%**	**N**	**%**
Inflection errors	144	41%	27	26%	293	44%	464	41%
Non-correctable (all)	349	100%	105	100%	669	100%	1123	100%

Table 10.17: Inflection errors as proportion of non-correctable errors

The high proportion of inflection errors suggests that a method to detect and correct this type of error would be useful. Although inflection errors do not seem to be appropriate candidates for confusion sets, it might be possible to treat them as a type of 'meta confusion set' by creating a confusion set containing all inflected forms each time a noun or verb appeared in the text. As the majority of the inflection errors occur when a base form is produced in error for an inflected form, it might be sufficient to implement this only when a singular noun or the base form of a verb was encountered. This could run either as a separate stage of the spellchecking process or be combined with the confusion tagger. Some handling of inflection errors is an obvious line of future development.

10.3.2 Other errors not included in confusion sets

The majority of the non-inflection undetectable errors occurred once only although there were more repeated errors in the Compositions corpus as illustrated by the type:token ratios shown in Table 10.18.

	Dyslexic	**Students**	**Compositions**	**All**
N. Tokens	205	78	376	659
N. Types	159	70	257	486
%Types	78%	90%	68%	74%

Table 10.18: Ratio of types to tokens for non-inflection undetectable errors

The assessment of the confusion set coverage in Section 5.2 discussed the undetectable errors occurring in the Dyslexic error corpus. The undetectable errors in the Student and Compositions corpora follow a similar pattern. Again, many errors occur on short function words. In the Compositions corpus *a* appears 24 times as a misspelling, occurring for five different targets – *at* (7), *an* (6), *and* (4), *as* (4), *I* (3) – and *the* occurs 14 times for seven different targets – *to* (5), *them* (2), *then* (2), *they* (2), *he* (1), *she* (1), *there* (1). Errors of this type seem particularly problematic because of the high frequencies of the words involved and the wide variety of intended targets.

The most frequently repeated undetectable error is *use* as an error for *used*, occurring 83 times in the Compositions corpus. The high incidence of this is presumably largely because of the subject matter of the corpus. It is easy to imagine poor spellers (or even reasonably good spellers who weren't paying attention) producing *use to* when writing about what they *used to* do in primary school. It also suggests that perhaps some types of inflection error should be considered for inclusion as confusion sets.

Some once-only errors in the Students corpus seem puzzling and may well be errors introduced by the user making an incorrect selection from a spellchecker selection list for the correction of a non-word error, for example:

diapers (*disappears*)

guesses (*guests*)

prepuce (*purpose*)

recurred (*required*)

There are others that seem similar to the types of error already included in the confusion sets such as:

find (fine)

luck (lack)

meter (*matter*)

throw (through) (the confusion set for *through* contains *threw* but not *throw*)

Although adding such words to the existing confusion sets simply because they occurred once in a corpus of errors does not seem to be a particularly productive approach, further analysis of the type and patterns of the uncorrectable errors in these corpora could help with refining and developing the confusion sets for future versions of the spellchecker.

10.4 Confusion sets assessed

The non-correctable errors demonstrate some drawbacks with the confusion set approach: the spellchecker is only able to correct errors for words included in its confusion sets and confusion sets themselves do not seem to be appropriate for some types of error, such as inflected forms of the same word or short function words. A more generalised syntax-based approach would seem to be required in such cases.

Other types of non-correctable errors might be considered for inclusion in the spellchecker's set of confusion sets. Take this miscorrected sequence from my error corpus for example, "the *lose* of...". Here *lose* is mistakenly produced for *loss* but the confusion set for *lose* – {*loos*, *loose*, *louse*} – does not include *loss*. The syntax checker finds that a noun tag is more appropriate than a verb tag in this context and selects *loose* (the highest frequency noun – as in "on the *loose*") as a being a better syntactic fit. As *loos* and *louse* are also nouns, this selection results in a possible semantic confusion so {*loose*, *loos*, *louse*} are flagged for further consideration by the semantic checker. At the end of this process, *loose* is still the preferred word; although the spellchecker has flagged the error it has proposed an incorrect replacement.

A simple solution could be to add *loss* to *lose*'s confusion set. The facility to add confusables to the spellchecker's previously defined sets seems somewhat akin to that of adding words to a custom dictionary which is a feature of most spellchecking applications. A disadvantage of this scheme is that the confusion sets would tend to grow large and large sets are a nuisance at the detection stage. Even when confusables are correctly used, as they generally are (Table 10.4), the spellchecker needs to consider all alternatives before accepting them; the larger the sets, the more work it has to do.

An alternative approach might simply be to treat the confusion sets as a detection mechanism. If the spellchecker decided that another member of the confusion set fitted better than the word it had encountered in the text, it would invoke its ordinary correction routine (the same one as it would use for the correction of non-word errors) to produce an ordered list of candidate corrections (which would probably, though not necessarily, include the confusable which triggered the correction routine in the first place).

If a confusion set simply functions as a signal to the spellchecker that the word it is considering is possibly an error, we don't need to try to include every potential misspelling in the set. In fact, rather than increasing the set sizes we can probably reduce them. As well as decreasing the processing required for correctly spelled words, this might also improve the correction rate – for instance, if *loose* was the only word in the confusion set for *lose* this would be sufficient to detect the error in "the *lose* of...". The corrector might well include *loss* (as well as *loose*, *loos* and *louse*) among its suggestions and the spellchecker would thus at least have a chance of proposing the correct replacement.

This approach seems to have some potential for future development of the spellchecker.

10.5 Syntactic and semantic approaches compared

The spellchecker developed in this research uses both syntactic and semantic information to make its decisions. The syntax checker is the first phase and, in addition to correcting some of the errors, assigns part-of-speech tags to each word in the text and flags potential semantic errors for consideration by the semantic checker.

One advantage of a syntactic approach is that it can be implemented using a minimal amount of linguistic information; the version I have developed requires only tag bigram probabilities (derived from the BNC) and the <word, tag> frequencies from the dictionary. This means that new confusion sets, provided their tagsets differ, can be incorporated without the need for additional training. Conversely, the semantic approach implemented here requires considerable effort both to store and group the co-occurring nouns for each confusable during the training phase and to retrieve and merge the hypernym probabilities at run-time. It is also only appropriate for words that occur fairly frequently in the BNC; for words that occur less than around a hundred times there are insufficient noun co-occurrences to generalise from as discussed in Chapter 9. In addition to this, even when a word does have a hypernym tree associated with it, there may be no co-occurring nouns when it appears in text being checked in which case the semantic checker will be unable to make a decision.

Although the hypernym co-occurrence approach implemented here shows some promise as a way of capturing the differences between words by grouping their co-occurring nouns and makes some impact on the errors at run-time by contributing about 14% of the total errors corrected, considerable refinement is required before it could be implemented as part of a practical spellchecking application.

10.6 The trade-off between accuracy and coverage

From an end-user's perspective, a spellchecker performs well if it corrects the majority of the errors while at the same time producing a minimum number of false alarms. However, it is difficult to achieve this ideal. In common with other research discussed in Chapter 2, my spellchecker factors in the expectation that the word appearing in the text is more likely to be correctly spelled than it is to be an error by handicapping the alternative words. This has the effect of reducing the number of false alarms but also, inevitably, reduces the number of errors corrected.

For the evaluation reported on in this chapter, I have set this handicap at the levels assessed to be optimum – 0.01 for the syntax checker (Chapter 8) and 0.2 for the semantic checker (Chapter 9).

This minimizes the false alarms while still correcting a useful proportion of the errors and achieves a reasonable level of improvement in the overall accuracy of the text, which is the preferred measure of spellchecker performance for this evaluation (discussed in Section 10.2.2).

The consensus of opinion (as discussed in Section 2.5) is that users will have more confidence in a spellchecker that, although it may ignore many errors, produces few false alarms than in one that flags the majority of the errors but also flags a large number of the correctly spelled words as errors. For a spellchecker developed for dyslexic users this is probably the best approach. They expect to make a large number of errors and are likely to accept the spellchecker's suggestion in the majority of cases. If there are a large number of false alarms this will result in an overall worsening of the text. However, as Carlson et al. (2001) and others have suggested, the confidence level can be set at runtime according to the user's preference. A user who wanted to be sure that the majority of the errors in the text had been flagged and who was confident enough to ignore the false alarms could achieve this by reducing the handicap set for the alternative words.

10.7 Future development

Although the spellchecker developed in this research has achieved some measure of success in detecting and correcting the real-word errors in the corpora used for this evaluation (Table 10.15), it has also suggested several lines of future development.

Inflection and function word errors, neither of which seems appropriate for a confusion-set approach, form the greatest proportion of the errors that are not dealt with by the current spellchecker. Since these are generally syntax errors, developing the syntax checker to incorporate them seems to be the most productive next stage.

Using confusion sets for detection rather than correction might reduce the proportion of 'detected but not corrected' errors as we would no longer be restricting the spellchecker to making suggestions from a predefined list.

Improving the syntax checker could be expected to improve the overall performance of the spellchecker but would not deal with errors that have matching parts-of-speech. Although the current hypernym tree implementation is too slow and laborious, using WordNet senses to capture semantic context still seems an attractive approach that might usefully be developed further to deal with errors of this type.

10.8 Contribution

This research has produced the following original contributions, including several resources of value to the wider research community:

- An updated electronic dictionary – CUVPlus. This dictionary, which has been uploaded to the Oxford Text Archive, is freely available for use for research purposes and is already being used by other researchers. The inclusion of more precise part-of-speech tag frequencies, based on word frequency in the BNC, represents a significant improvement on the previous version in which part-of-speech tag frequency for each word was simply classed as rare, ordinary or common, with the majority being ordinary. The addition of the C5 tagset, as used in the BNC, means that the dictionary now also uses a widely recognised tagset which is familiar to the research community.
- An annotated corpus of dyslexic real-word spelling errors. Although this corpus is still relatively small, containing just over 12,000 words, no resource of this type was previously available. I plan to continue with the development of this resource. The corpus should be of use for both educational and linguistic research and I intend to make it more widely available in the near future.
- A large collection of confusion sets. Much research into real-word error correction relies on sets of confusable words but there is no general consensus as to the most effective way to produce these. The almost six thousand sets of confusable words that I used for this research were produced using a distance measure tuned on real error data and represent a

significant increase on the 256 sets used by the largest-scale implementation of this method to date (Carlson et al., 2001). As well as being a larger collection, these sets are also non-symmetric which makes them more flexible for large-scale use.

- The use of real error data for the development and testing of the program. This enabled me to address the problems of correcting actual errors made by dyslexic writers which are less predictable than the artificial errors introduced into the text used for the assessment of much of the other research in this area.
- The semantic error correction algorithm. Although the current implementation is rather slow and cumbersome, it demonstrates that semantic association, using WordNet, has potential for the correction of errors of this type.
- The 'overall improvement' measure used for the evaluation. This has not been used previously to assess spellchecker performance. It is shown to be a more practical measure of the effectiveness of the spellchecker than other previously used measures to which it is compared.

References

Atwell, E. 1987 'Constituent-likelihood Grammar'. In R. Garside, G. Leech & G. Sampson (eds.), *The Computational Analysis of English*. Longman, London, pp. 57-65.

Atwell, E. and Elliott, S. 1987 'Dealing with ill-formed English text'. In R. Garside, G. Leech & G. Sampson (eds.), *The Computational Analysis of English*. Longman, London, pp. 120-138.

Atwell, E., Demetriou, G., Hughes, J., Schiffrin, A., Souter, C. and Wilcock, S. 2000 'A comparative evaluation of modern English corpus grammatical annotation schemes'. *ICAME Journal* 24 pp. 7-23.

Booth, B. 1987 'Text input and preprocessing'. In R. Garside, G. Leech & G. Sampson, (eds.),*The Computational Analysis of English.* Longman, London, pp. 97-109.

British Dyslexia Association <http://www.bdadyslexia.org.uk/research.html>

Budanitsky, A. 1999 *Lexical semantic relationship and its application in natural language processing*. Technical Report CSRG390, University of Toronto <http://citeseer.ist.psu.edu/budanitsky99lexical.html>

Budanitsky, A. and Hirst, G. 2001 'Semantic distance in WordNet: An experimental, application-oriented evaluation of five measures'. *Workshop on WordNet and Other Lexical Resources, in the North American Chapter of the Association for Computational Linguistics NAACL-2001*, Pittsburgh, PA. <http://citeseer.ist.psu.edu/budanitsky01semantic.html>

Burnard, L. 2000 *Reference Guide for the British National Corpus (World edition)* <http://www.natcorp.ox.ac.uk/docs/userManual>

Carlson, A.J., Rosen, J. and Roth, D. 2001 'Scaling Up Context Sensitive Text Correction'. *Proceedings of the National Conference on Innovative Applications of Artificial Intelligence* pp. 45 – 50. <citeseer.ist.psu.edu/budanitsky01semantic.html>

Charniak, E., C. Hendrickson, N. Jacobson, and M. Perkowitz. 1993 'Equations for part-of-speech tagging'. *Proceedings of the eleventh national conference on artificial intelligence Washington, D.C., American Association for Artificial Intelligence*, pp. 784--89. <http://citeseer.ist.psu.edu/charniak93equations.html>

Chen, S.F. and Goodman, J. 1996 'An empirical study of smoothing techniques for natural language modelling'.. *ACL-96*, Santa Cruz, CA. pp.310-318.

Damerau, F.J. 1964 'A Technique for Computer Detection and Correction of Spelling Errors' *Communications of the A.C.M.* 7 pp. 171-6.

Damerau, F.J. and Mays, E. 1989 'An Examination of Undetected Typing Errors' *Information Processing and Management* 25 (6):. 659-64.

Flexner, S.B. (ed.) 1983 *Random House Unabridged Dictionary*, Random House, New York. 2nd Edition.

Gale, William A. and Church, Kenneth W. 1994 'What's Wrong with Adding One?' In Oostdijk, N and de Haan, P. (Eds.) *Corpus-based Research into Language*. pp. 189-198. Rodopi, Amsterdam.

Gale, William A. and Sampson, G. 1995 'Good-Turing Frequency Estimation Without Tears' *Journal of Quantitative Linguistics* 2: pp. 217-37.

Garside, R. 1987 'The CLAWS word-tagging system' *In The Computational Analysis of English*. R. Garside, G. Leech & G. Sampson, (Eds)., Longman, London, pp. 30-41.

Golding, A.R. 1995 'A Bayesian Hybrid Method for Context-sensitive Spelling Correction'. *Proceedings of the Third Workshop on Very Large Corpora*, pp. 39-53.

Golding, A.R. and Roth, D. 1996 'Applying Winnow to Context-sensitive Spelling Correction'. *Machine Learning: Proceedings of the 13th International Conference*, pp. 182-190.

Golding, A.R. and Roth, D. 1999 'A Winnow based approach to context-sensitive spelling correction'. *Machine Learning* 34 (1-3) pp. 107-30

Golding, A.R. and Schabes, Y. 1996 *Combining Trigram-based and Feature-based Methods for Context-sensitive Spelling Correction.* Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics, pp. 71-78.

Good, I.J. 1953 'The population frequencies of species and the estimation of population parameters'. *Biometrika* 40: 237-264.

Heidorn, G.E., Jensen, K., Miller, L.A., Byrd, R.J., Chodorow, M.S. 1982 *The EPISTLE text-critiquing system* IBM Systems Journal 21:3 pp.305-326

Hill, A 2005 *Scandal of secret school exclusions* Observer 11December 2005, <http://observer.guardian.co.uk/uk_news/story/0,6903,1664699,00.html>

Hirst, G and St-Onge, D. 1998 'Lexical chains as representations of context for the detection and correction of malapropisms'.. In: Christiane Fellbaum (ed), *WordNet: An electronic lexical database*. The MIT Press, Cambridge, MA. pp. 305-332.

Hirst, G and Budanitsky, A. 2005 'Correcting Real-Word Spelling Errors by Restoring Lexical Cohesion' *Natural Language Engineering*, 11(1): 87-111.

Holbrook, D. 1964 . *English for the Rejected.* Cambridge University Press.

Hornby, A.S. 1974 *Oxford Advanced Learner's Dictionary of Current English* (3rd Edition). Oxford University Press.

Hundt, D., Sand, A. and Siemund, R. 1998 *Manual of Information to accompany the Friebug-LOB Corpus ('FLOB')* <http://knht.hit.uib.no/icame/manuals/flob>

ICAME 1999 *The ICAME Corpus Collection on CD-ROM, version 2* <http://icame.uib.no/newcd.htm>

Jiang, J.J and Conrath, D.W. 1997 'Semantic similarity based on corpus statistics and lexical taxonomy'. *Proceedings of International Conference on Research in Computational Linguistics.*

Johansson, S., Atwell, E., Garside, R. and Leech, G. 1986 *The Tagged LOB Corpus Users' Manual* <http://khnt.hit.uib.no/icame/manuals/lobman>

Kilgarriff, A. 1997 *Putting Frequencies in the Dictionary.* International Journal of Lexicography 10 (2). pp 135-155

Knuth, D.E. 1973 *Sorting and Searching: The Art of Computer Programming Vol. 3.* Addison-Wesley.

Kukich, K. 1992 'Techniques for Automatically Correcting Words in Text'. *Computing Surveys* 24 (4) 377-439.

Leech, G. and Smith, N. 2000 *Manual to accompany The British National Corpus (Version 2) with Improved Word-class Tagging* UCREL, Lancaster

Leech, G., Rayson, P. and Wilson, A. 2001 *Word Frequencies in Written and Spoken English* Longman, London.

Mangu, L. and Brill, E. 1997 'Automatic Rule Acquisition for Spelling Correction'. *Proceedings of the 14th International Conference on Machine Learning (ICML 97)*, pp. 187-194.

Marshall, I. 1987 'Tag selection using probabilistic measures' In *The Computational Analysis of English* R. Garside, G. Leech & G. Sampson, (Eds)., Longman, London, pp. 42-56.

Mays, E., Damerau, F.J. and Mercer, R.L. 1991 'Context Based Spelling Correction'.. *Information Processing and Management,* 25 /5 pp. 517-22.

Mikheev, A. 2002 'Periods, Capitalized Words, etc'. *Computational Linguistics* 28(3): 289-318.

Miller, G., Beckwith, R., Fellbaum, C., Gross, D., and Miller, K. 1990 *Five papers on WordNet.* CSL Report 43, Princeton University.

Mitton, R 1985 *Birkbeck Spelling Error Corpus.* <http://ota.ahds.ac.uk/texts/0643.html>

Mitton, R. 1987 'Spelling Checkers, Spelling Correctors and the Misspellings of Poor Spellers'. *Information Processing and Management* **23** (5) 495-505.

Mitton, R. 1992 *A description of a computer-usable dictionary file based on the Oxford Advanced Learner's Dictionary of Current English*. Oxford Text Archive.

Mitton, R. 1996 *English Spelling and the Computer* Longman, London.

Nancholas, J. 2005 *Signage for an Inclusive Library.* MA Dissertation, London Metropolitan University

Nisbet, P., Spooner, R., Arthur, A., Whittaker, P. and Wilson, A.. 1999 *Supportive Writing Technology.* Call Centre, Edinburgh, University of Edinburgh.

Palmer, D. D. and Hearst, M.A. 1997 'Adaptive Multilingual Sentence Boundary Disambiguation'. *Computational Linguistics*, 23(2):241-269.

Pedler, J. 1996 *Computer Correction of Dyslexic Spelling Errors*. MSc Computing Science Report, Birkbeck, London University.

Pedler, J. 2001a 'Computer Spellcheckers and Dyslexics - a Performance Study'. *The British Journal of Educational Technology*, 32/1, pp. 23-38.

Pedler, J. 2001b 'The detection and correction of real-word spelling errors in dyslexic text'. *Proceedings of the 4th Annual CLUK Colloquium*, pp.115-119.

Pedler, J. 2003a 'The contribution of syntactic tag collocation to the correction of real-word spelling errors'. *Proceedings of the 6th Annual CLUK Colloquium.*pp.129-133.

Pedler, J. 2003b 'A corpus-based update of a 'computer-usable' dictionary' *Proceedings of the Eighth International Symposium on Social Communication* pp. 487-492.

Pedler, J. 2005 'Using semantic associations for the detection of real-word spelling errors' In: *Proceedings from The Corpus Linguistics Conference Series, Vol. 1, no. 1 Corpus Linguistics 2005*, <http://www.corpus.bham.ac.uk/PCLC>.

Peterson, J.L. 1980 'Computer Programs for Detecting and Correcting Spelling Errors'. *Communications of the A.C.M.* 23 (12): 676-87

Pollock, J.J. & Zamora, A. 1984 'Automatic Spelling Correction in Scientific and Scholarly Text'. *Communications of the A.C.M.* 27 (4): 358-68.

Rennie, J. 2000 *WordNet::QueryData: a {P}erl module for accessing the {W}ord{N}et database.* <http://people.csail.mit.edu/~jrennie/WordNet>

Resnik, P. 1995 'Using Information Content to Evaluate Semantic Similarity in a Taxonomy'. *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI).*

Richardson, S.D. 1985 *Enhanced Text Critiquing using a Natural Language Parser.* IBM, Research Report RC 11332(#51041).

Sampson, G. 2007 *Downloadable Research Resources.* <www.grsampson.net/Resources.html>

Spooner, R 1998 *A spelling aid for dyslexic writers.* PhD thesis, University of York.

Tresman, S. and Cooke, A 2006 *The Dyslexia Handbook 2006.* British Dyslexia Foundation

Willis, T., Pain, H. and Trewin, S. 2005 'A Probabilistic Flexible Abbreviation System for Users with Motor Disabilities' *Proceedings of Accessible Design in the Digital World.* <http://www.accessinthedigitalworld.org/2005>.

Wikipedia 2003 *Lists of common misspellings* <http://en.wikipedia.org/wiki/Wikipedia:List_of_common_misspellings>

Wing, A.M. and Baddeley, A.D. 1980 'Spelling errors in handwriting: a corpus and distributional analysis'. In Uta Frith (ed.) *Cognitive Processes in Spelling*, Academic Press pp. 251-85.

Yannakoudakis, E.J. & Fawthrop,D. 1983 'The Rules of Spelling Errors' *Information Processing and Management* 19 (2): 87-99.

Appendix A: Documentation to accompany CUVPlus

This appendix is a reproduction of the dictionary documentation available for download from the Oxford Text Archive.

Jennifer Pedler
School of Computer Science and Information Systems,
Birkbeck College, Malet St, London WC1E 7HX, UK.
jenny@dcs.bbk.ac.uk

October 2002

1. Introduction

In 1992 Roger Mitton produced an updated version of a 'computer usable' dictionary called CUV2 (Mitton, 1992), itself derived from CUVOALD (Mitton, 1986) his version of the Oxford Advanced Learner's Dictionary of Current English (OALDCE) (Hornby, 1974). CUVPlus is an enhanced version of CUV2.

Main enhancements:

- More precise, corpus-based, word frequency information
- An additional part-of-speech tagset using tags from UCREL C5 tagset (as used in the BNC World Edition).
- Over 1,600 new entries giving increased coverage.

The main motivation behind the update was the inclusion of more accurate frequency information. CUV2 assigns one of three broad frequency categories - very common, ordinary or rare - to each part-of-speech. This is adequate to distinguish between different usages for some words, such as *can*, which is marked 'very common' as a verb and 'ordinary' as a noun, but for others the difference is not clear. For example both noun and verb are marked as common for *form* although it is used far more frequently as a noun. My research into computer spellchecking showed that it would be useful if this type of distinction could be made. To obtain frequency counts for the words in CUV2 I used the written section of the BNC (World Edition), which comprises approximately 80 million words tagged with their part-of-speech. The tagset used in CUV2 does not correspond exactly to the UCREL C5 tagset used in the BNC, for example CUV2 tags nouns as countable and uncountable whereas the C5 tagset does not make this distinction. Conversely, the C5 tagset has three tags for conjunctions - subordinating, co-ordinating and the subordinating conjunction that - while CUV2 has just one. For this reason a new tagset was created for each entry in CUVPlus using the C5 tags. This also has the advantage of making the dictionary compatible with the BNC.

CUV2's 70646 entries have provided adequate coverage for most of my spellchecking work. However, it occasionally produces unexpected false alarms - flagging as errors words that are actually correctly spelled. This largely reflects changes in the language since 1974 when the OALDCE was compiled. For example *database*, a word in fairly common modern usage not included in CUV2, would have been a specialised term thirty years ago. To improve the coverage of the dictionary additional entries were created for words occurring frequently in the BNC but not found in CUV2.

This document describes the differences between CUVPlus and CUV2. CUV2 is available from the Oxford Text Archive and full details can be found in its accompanying documentation (Mitton, 1992).

2. Format of entries

CUVPlus contains 72060 ASCII ordered entries, one per line in text format. Each entry comprises six pipe-separated fields:

spelling | capitalisation flag | pronunciation | CUV2 tags | C5 tags+frequency | syllable count

Example lines are shown below:

```
bourne|2|bU@n|K6$|NN1:0|1

india|1|'IndI@|Nm%|NP0:50|3
there|0|De@R|Pu*,W-%|EX0:2166,AV0:480,ITJ:0|1
```

The spelling, pronunciation, CUV2 tags and syllable count remain unchanged since CUV2. Full details of these fields can be found in (Mitton, 1992). The CUV2 tagset is also included as an appendix to this document. The CUV2 tags were retained in CUVPlus as they often provide information about the word not available from the C5 tags. (See summary of differences between CUV2 and C5 tagsets in Section 8.)

3. Frequency information

The frequency information added to each of the part-of-speech tags in CUVPlus is the rounded frequency per million based on counts obtained from the written section of the BNC. These counts roughly correspond to the listing for the written section of the BNC in (Leech et al., 2001) but they cannot be directly compared for several reasons; differences in the later version of the BNC used for this work, differences in the treatment of special cases such as hyphenated forms, enclitics and multi-word units, and the combination of some parts-of-speech (such as past participle and past tense) in Leech et al.'s (2001) lists. Using these counts rather than the raw frequency retains the differentiation between the first few hundred common words while grouping together lower-ranked words which are little different in either frequency or usage.

Around 7,000 of the words in CUV2 were not found in the BNC (over half of these were already tagged as rare); about 10,000 word/tag pairs occurred just once and about 6,000 twice only. There is really little difference in the rarity of these words. Some of those not found in the BNC might well occur in another corpus while that corpus might not contain words found in the BNC. They have all been given a frequency of -1 to differentiate them from the other words that occurred less than once per million which have a frequency value of 0.

4. Part-of-Speech tags

An initial tagset was created for each word in CUV2 that had occurred in the BNC by grouping together all tags recorded for the word, together with their frequency count. These included a large number of mistaggings that had to be removed before the completed tagsets could be added to the dictionary. For example *there*, which should have two tags - EX0 (existential there) and AV0 (adverb) - initially had 12 additional tags assigned to it, including noun, personal pronoun and modal auxiliary verb. The tagsets were then cross-checked with the original CUV2 tags to make sure that any rare usages that had not occurred in the BNC were accounted for. By and large BNC tags that did not have a corresponding tag in CUV2 were removed. Some exceptions to this were verb participles which are commonly used adjectively (such as *massed* and *matching* which are both tagged more frequently as an adjective than a verb in the BNC but are only tagged as a verb in CUV2) and nouns that can also function as adjectives (such as *amateur* which only has a noun tag in CUV2 but is more commonly tagged as an adjective in the BNC). In such cases although the C5 tagset records both usages, the CUV2 tagset has not been updated to reflect this as shown below:

```
massed|0|m&st|Jc%,Jd%|AJ0:2,VVN:1,VVD:0|1
matching|0|'m&tSIN|Jb%|AJ0:8,VVG:4,NN1:3|2
```

```
amateur|0|'&m@t@R|K6%|AJ0:13,NN1:4|3
```

CUV2 includes entries for prefixes such as *multi*- and *un*- which have their own prefix tag. The C5 tagset does not include a prefix tag and as hyphenated forms in the BNC are always tagged as single units there is no need for prefixes to be tagged in their own right. In CUVPlus these have been assigned a PRE tag and currently have no frequency information as shown below:

```
multi-|0|'mVltI-|U-%|PRE:-|2
un-|0|Vn-|U-%|PRE:-|1
```

In the BNC inflected forms are broken into their component parts each of which receives its own tag. For example *they're* is tagged *<w PNP> they<w VBB>'re*. CUVPlus includes entries for common enclitics such as this and they have been assigned combined tags as shown below:

```
they're|0|De@R|Gf%|PNP+VBB:108|1
```

Letters of the alphabet are assigned a singular or plural noun tag in CUV2. The C5 tagset uses the tag ZZ0 for alphabetical symbols. In the BNC plural letters (e.g. *f's*) are sometimes tagged as a single unit with the ZZ0 tag and at other times broken into two separate units as for the enclitic forms described above. In CUVPlus the ZZ2 tag (as used in the UCREL C6 tagset) has been assigned to such entries. This is illustrated below:

```
f|4|ef|Ki$|ZZ0:30|1
f's|4|efs|Kj$|ZZ2:0|1
```

Tagsets were created for words in the dictionary that had not occurred in the corpus by mapping the existing CUV2 tags to their corresponding BNC tag. These were assigned the frequency value -1.

5. Additional entries

Initial candidates for new entries were words occurring more than ten times in the BNC excluding: strings containing digits; units tagged UNC (unclassified); those tagged CRD (cardinal number e.g. *ii*, *twenty-one*) or ORD (ordinal number e.g. *nth*, *twenty-first*); capitalised strings (often abbreviations or acronyms); enclitic and hyphenated forms. Proper nouns and multi-words were considered separately. This resulted in a shortlist of around 2000 words. This was manually checked to remove unsuitable entries: non-words (e.g. *emailinc* (969); misspellings (e.g. *faithfullly* (31)); American English spellings (e.g. *judgment* (348)); interjections (e.g. *huh* (175), *hmm* (156)); specialist terms (e.g. *unix* (222)); medical terms (e.g. *mucosa* (197), *pancreatitis* (76)); slang (e.g. *caf* (84), *dosh* (23)); abbreviations (e.g. *plc* (228), *mins* (227)). A large number of 'solid' compound words in the list (e.g. *ceasefire* (107), *holidaymaker* (12), *turnout* (27)) were already in CUV2 in hyphenated form. These were also removed from the list. New entries were created for other compounds (such as *goodnight* and *lifestyle*) in 'solid' form although they could just as well be hyphenated. The question of how compound words should be entered in a dictionary is considered further in Section 7 below.

After this pruning just over 500 words were left in the list. The ten highest ranked are listed below:

Word/Tag	**N. Occurrences**
organisation (NN1)	1201
organisations (NN2)	733
database (NN1)	403
goodnight (ITJ)	316

workforce (NN1)	291
lifestyle (NN1)	260
wildlife (NN1)	234
accountability (NN1)	210
profitability (NN1)	210
databases (NN2)	205

The top two words - *organisation* and *organisations* - highlighted a general problem with the groups of words that can take an *-ise* or *-ize* suffix. CUV2 only includes *-ize* forms although (as Appendix 3 of the OALDCE remarks) the *-ise* form is equally acceptable. Overall 80 of the words in the list were of this type. There are 200 such groups of words in CUV2. Many, such as *formalize*, *legalize*, did not appear in their *-ise* form in my BNC list. To maintain consistency *-ise* entries were created for all of these resulting in just over a thousand new entries.

Many nouns appeared in the list in either singular or plural form but not both (such as *deckchairs* (12) and *stockmarket* (23)). In such cases additional entries have been included for the lower frequency form as well (*deckchair* (7), *stockmarkets* (4)). Some of the plural nouns in the list already had singular entries in CUV2 that were tagged as taking no plural (for example: *determinations*, *sediments*). In such cases the CUV2 tags have been updated to indicate that a plural is acceptable.

CUV2 contains around 2,500 proper noun entries - forenames, countries, cities and so on. Although there are a great many more that could be included I did not have time to consider this in detail. From a list of around 400 proper nouns not in CUV2 which occurred with a frequency of more than one per million in the BNC forty were added to the dictionary. Two of these *Asia* (427) and *Birmingham* (375) were surprisingly omitted from CUV2. Others are places that were not independent states when the previous dictionary was produced such as *Bosnia* (249), *Croatia* (149) and *Serbia* (97). The majority of the others were surnames, which are not appropriate in a dictionary.

There are just over 400 entries in CUV2 that consist of more than one word, the majority of these are place names and naturalised foreign phrases (such as *New York*, *hors d'oeuvres*). The BNC treats many combinations of words that logically form a single unit - adverbial phrases such as *a bit*, *in addition to*, complex prepositions such as *in need of*, *save for* and naturalised foreign phrases such as *a posteriori*, *chilli con carne* - as multi-word units and assigns a single tag to the complete phrase. For a complete list of multi-word units in the BNC see the BNC tagging manual (Leech & Smith, 2000). Many of these occur with high frequency in the BNC (the most frequent being *per cent* (already in the dictionary in 'solid' form) and *as well* which occur over 5,000 times each). As text-processing programs that use white space as a word delimiter will often not make use of such entries I was dubious whether these would be useful additions. Again, I did not have time for detailed consideration but decided to include 35 of the most common, consisting of short function words that are often mistakenly run together (such as *alot* where *a lot* was intended (Mitton, 1996)) that would be useful for spellchecking purposes.

In total 1669 new entries have been included in CUVPlus. Pronunciation and CUV2 tags have been added to each of these. For the *-ise* entries these are identical to the existing *-ize* entries. Pronunciations for the multi-word entries are a combination of the pronunciation for the individual words and they have been given CUV2 tags corresponding to their definition in the BNC (for example, *at all* is tagged as an adverb, *out of* as a preposition). For all other entries these fields were created manually.

6. Other changes from CUV2

6.1. Capitalisation

As upper-case come before lower-case letters in ASCII ordering, entries for words such as *Nice* (the place name) and *nice* (the adjective) are widely separated in CUV2 although in a paper dictionary they would appear together. To overcome this problem all initial letters have been converted to lower-case in CUVPlus and a code, indicating whether or not the word should start with a capital, inserted as the second field in the line.

This code can have one of four values as shown in the following examples:

0 initial letter always lower-case

```
person|0|'p3sn|K6%|NN1:254|2
```

1 initial letter always upper-case

```
fred|1|fred|Nl%|NP0:19|1
```

2 initial lower-case but there is also an upper-case entry with the same spelling but different pronunciation

```
job|2|dZ0b|J4$,K6*|NN1:209,VVB:0,VVI:0|1
```

3 initial upper-case but there is also a lower-case entry with the same spelling but different pronunciation

```
job|3|dZ@Ub|Nl$|NP0:0|1
```

4 initial letter upper or lower-case depending on usage but pronunciation the same in both cases.

```
kitty|4|'kItI|K8%,Nl%|NP0:2,NN1:1|2
```

kitty can be either a common or a proper noun. Initial capitalisation will depend on the sense in which it is being used.

Many abbreviations consist entirely of upper case letters (e.g. *AGM*, *GMT*). For words such as these only the initial letter has been converted to lower case, which results in some rather odd-looking entries as shown below. It also means that they do not appear in their absolute alphabetical position. This is perhaps not the best way of dealing with this type of entry but it maintains consistency with the other entries.

```
aGM|1|,eIdZI'em|Y>%|NN1:3|2
gMT|1|,dZi,em'ti|Y~%|AV0:8|3
```

It is useful to note at this point that the code in this field only deals with capitalisation, it does not explicitly mark spellings that appear more than once in the dictionary. Homographs, if both would normally be written with an initial lower-case letter will have two entries, both with the code 0 in the second field as, for example:

```
wind|0|waInd|J5%,K6%|NN1:73,VVI:3,VVB:2|1
wind|0|wInd|H0%,M6%|NN1:73,VVI:3,VVB:2|1
```

6.2. Diacritics

In CUV2 diacritic characters - acute, grave, umlaut, cedilla, circumflex and tilde - precede the letter that they mark. This presents several problems. They do not appear in their correct alphabetical position - this is particularly extreme for words beginning with a diacritic which all appear grouped together at the beginning of the dictionary. More importantly, they never appear written in this way in normal text and so cannot be looked up automatically (by a spellchecker, for example). Finally, in many cases, as these foreign words become naturalised into English they are often written without the diacritics and are thus more appropriately entered into the dictionary in their unaccented form. On balance, if they were to be included at all it seemed more useful to enter them without diacritics so these have been removed from all entries in CUVPlus. Some examples are shown below:

CUV2	**CUVPlus**
`_eclair`	`eclair`
`t^ete-`a-t^ete`	`tete-a-tete`
`f"uhrer`	`fuhrer`
`fa<cade`	`facade`
`se~nor`	`senor`

6.3. Verb pattern fields

Entries for verbs in the OALDCE include verb pattern fields designed to show the learner of English how to form correct sentences. These were retained in the entries for CUV2 as the fourth field in the line. For entries other than verbs this field was left blank as shown in the two examples below:

```
CUV2      form|fOm|J0*,M6*|2A,2C,6A,14,15A|1
CUV2      cat|k&t|K6%||1
```

I decided to omit this field from CUVPlus, partly because I have never found a use for it but largely for consistency reasons. There is now no need for empty fields in some entries and the time consuming task of creating verb patterns for new entries was avoided. The examples below show the above two entries as they appear in CUVPlus.

```
CUVPlus   form|0|fOm|J0*,M6*|NN1:287,VVI:52,VVB:34|1
CUVPlus   cat|0|k&t|K6%|NN1:36|1
```

7. Further considerations

This update has achieved its main aim of adding more accurate frequency information to the dictionary. The new entries have increased the coverage which should lead to fewer false alarms when it is used for spellchecking (although the fact that less than a hundred of the words added had a frequency of more than one per million in the BNC suggests that existing coverage of CUV2 was fairly comprehensive). However, there were several issues, some of which arose during the course of the work, which I did not have time to consider in detail. In particular:

- Compound word entries
  In the current version some of these are entered in hyphenated form and others in 'solid' form. In use they are often written in both ways although there is possibly an increasing tendency to omit the hyphen. There seems little point in a dictionary including both forms but it would be best to take a consistent approach. In general hyphenated forms are probably more useful for a spellchecker (or any other text-processing program) as it is easier to remove a hyphen than it is to insert it.
- Abbreviations
  In the current version abbreviations have had their initial capital converted to lower-case in the same way as all other entries. This is possibly not the right approach when subsequent letters are also upper-case. Additionally, as the C5 tagset does not include a specific abbreviation tag it

is necessary to use both sets of tags to determine whether a word is an abbreviation. This information is required to disambiguate between the use of a full stop as an abbreviation marker and a sentence delimiter.

- Proper nouns
  Some proper noun entries have been added to the current version, rectifying some omissions from CUV2 and reflecting major geographical/political changes since that dictionary was produced. A more comprehensive approach requires a clear definition of the categories of proper noun to be included and, for cities, countries and the like, the use of a gazetteer rather than a general purpose corpus to obtain the data.
- Multi-word units
  A rather ad hoc approach has been taken to these in the current version, largely because they are not particularly useful when the dictionary is used by a spellchecker. A clear idea of the ways in which they might be used would help to define clearer criteria for inclusion.
- Prefixes
  There is currently no frequency information included with these entries. It would be useful to include this as it would give an indication as to how commonly they were used to form other words.
- Homophones and homographs
  It would be useful to add flags for these to indicate that there is another entry with the same pronunciation or spelling.

Further details of the development of CUVPlus can be found in (Pedler, forthcoming).

8. Summary of differences between CUV2 and C5 tagsets

This section provides a summary of the differences between the two tagsets used in CUVPlus as an aid for users who wish to combine them to obtain further information about a word. Details of the C5 tagset as used in the BNC can be found in the BNC Users Reference Guide (Burnard, 2000). The CUV2 tagset is described in the documentation that accompanies the dictionary (Mitton, 1992). Both tagsets are also included as appendices to this document.

8.1. Verbs

Apart from *be*, *do* and *have*, which have their own sets of tags the BNC categorises verbs as either lexical or modal auxiliary. CUV2 divides verbs into four main groups - anomalous, transitive, intransitive, both transitive & intransitive. The anomalous verbs include the primary auxiliaries be, have and do, as well as the modal auxiliaries (e.g. can, may) and the semi-modals (e.g. dare, need). The other categories correspond to the lexical verbs in the C5 tagset. This means that additional information about the verb's usage - whether or not it takes an object is available from the CUV2 tags. Both tagsets further divide the main verb categories into infinitive, third person present, past tense and so on but CUV2 also includes inflection information with the tag for the infinitive form. For the modal auxiliaries in the C5 tagset no differentiation is made between past and present forms of these verbs whereas the CUV2 tagset does make this distinction. For example *can* and *could* are both simply tagged as modal auxiliaries in the BNC whereas in CUV2 *can* is tagged anomalous irregular and *could* is tagged anomalous past tense.

8.2. Nouns

The C5 tagset has four noun tags - common noun neutral for number, singular common noun, plural common noun and proper noun. Apart from proper nouns these do not correspond to the four main tags used in CUV2 - countable, uncountable, both countable and uncountable and proper noun. In CUV2 these are further subdivided into singular and plural with additional information about the way the noun inflects. CUV2 also further subdivides proper nouns into forename, country, town and other whereas in the BNC they all use the same tag.

8.3. Adjectives

Both tagsets make the distinction between general, comparative and superlative adjectives but in addition the CUV2 tags categorise them as predicative/attributive and indicate the way in which the inflections are formed.

8.4. Adverbs

The C5 tagset has 3 tags for adjectives - general adverb (not included in either of other two categories), adverb particle and wh- adverb. The CUV2 tagset has 4 - not interrogative or relative, interrogative, relative and adverbial particle. Apart from the particles there is no general correspondence between these two sets of tags and as the majority of adverbs in CUV2 fall into the 'not interrogative or relative' category little extra information is gained from using the CUV2 tags for these entries.

8.5. Pronouns

The C5 tagset splits these into determiner pronouns and pronouns using a total of seven tags - possessive determiner-pronoun, general determiner-pronoun, wh- determiner-pronoun, indefinite pronoun, personal pronoun, wh- pronoun and reflexive pronoun. This is far more fine-grained than the CUV2 tagset which classes pronouns as not interrogative or relative, interrogative, and relative with the majority of them falling into the first category.

8.6. Articles

The C5 tagset has one tag for all articles, definite and indefinite whereas CUV2 makes the distinction between definite and indefinite.

8.7. Prepositions

Both tagsets use a single tag for prepositions, apart from *of* which has its own unique tag in the C5 tagset. However, there is often some discrepancy between words tagged as prepositions in the BNC and those tagged as such in CUV2. For example, *out* and post *both* appear tagged as prepositions in the BNC (and thus have a PRP tag in CUVPlus) but have no preposition tag in CUV2. *Aslant* and *mid* have a preposition tag in CUV2 but do not appear tagged as prepositions in the BNC. In CUVPlus these also have preposition tags mapped from the existing CUV2 tags.

8.8. Conjunctions

CUV2 has a single tag for all conjunctions whereas the C5 tagset divides them into three categories - co-ordinating conjunction, subordinating conjunction, and the subordinating conjunction *that*.

8.9. Interjections

Both tagsets use a single tag for interjections.

8.10. Infinitive marker to

To, when used as the infinitive marker, has its own tag in the C5 tagset . It is the only word tagged as a particle in CUV2 thus the effect is the same in both tagsets.

8.11. Abbreviations

Abbreviations in the BNC are tagged as if they were written as full forms whereas CUV2 explicitly tags abbreviations and further categorises them into the type of abbreviation - singular noun, plural noun, both singular and plural noun, uncountable noun, title, proper noun and other. To determine whether a word is an abbreviation it is essential to use the CUV2 tags.

Acknowledgements

Thanks to George Mitton for producing and keying in the pronunciations and CUV2 tags for the new entries in CUVPlus.

References

Burnard, L. (Ed) (2000) *The British National Corpus Users Reference Guide*

Hornby, A.S. (1974) *Oxford Advanced Learner's Dictionary of Current English* (3rd Edition). Oxford University Press, 1039 p.

Leech, G., Rayson, P. and Wilson, A. (2001) *Word Frequencies in Written and Spoken English* Longman, London, 304 p.

Leech, G. and Smith, N. (2000) *Manual to accompany The British National Corpus (Version 2) with Improved Word-class Tagging* UCREL, Lancaster

Mitton, R. (1986) 'A partial dictionary of English in computer-usable form'. *Literary and Linguistic Computing* 1 (4) pp. 214-5

Mitton, R. (1992) *A description of a computer-usable dictionary file based on the Oxford Advanced Learner's Dictionary of Current English*. Oxford Text Archive.

Mitton, R. (1996) *English Spelling and the Computer* Longman, London, 207 p.

Pedler, J.(forthcoming) 'A corpus-based update of a 'computer-usable' dictionary' To appear in *Proceedings of the Eighth International Symposium on Social Communication* Jan 2003

Appendix 1

CUV2 Tagset

Word Class	First Character		Second Character	
VERBS	G	Anomolous	0	inflects like work
	H	Transitive	1	wish
	I	Intransitive	2	love
	J	Trans & Intrans	3	apply
			4	abet
			5	irregular
			a	3rd pers sing pres tense
			b	present participle -ing
			c	past tense
			d	past participle
			e	other part of verb
			f	contraction pronoun + anom. verb
			g	contraction anom verb + not
			h	contraction anom vb, other
NOUNS	K	countable	6	Plural like cat
	L	Uncountable	7	fox
	M	Count & uncount	8	pony
	N	Proper noun	9	Pl same as sing, like Salmon
			@	No plural
			i	sing form
			j	plural form
			k	pl. but acts like sg. e.g. economics
			l	proper, forename e.g. Sandra
			m	proper, country etc. e.g. Scotland
			n	proper, town e.g. Scunthorpe
			o	other e.g. Saturn
ADJECTIVES	O		A	no -er or -est form
			B	+r, +st, like subtle
			C	+ er, +est, like light

			D	y to ier, iest, like heavy
			E	irregular comp &/or sup
			p	predicative
			q	attributive
			r	comparative form
			s	superlative form
			t	can be attached by hyphen e.g. bellied
ADVERBS	P		u	not interrogative or relative
			v	interrogative
			w	relative
			+	adverbial particle
PRONOUNS	Q		x	not interrogative or relative
			y	interrogative
			z	relative
OTHER	R	Definite article	-	
	S	Indefinite article	-	
	T	Preposition	-	
	U	Prefix	-	
	V	Conjunction	-	
	W	Interjection	-	
	X	Particle	-	
	Y	Abbreviation	>	sing noun
			)	plural noun
			]	both sing & plural
			}	uncountable noun
			:	title
			=	proper noun
			~	other abbreviation
	Z	Not classified	-	

Frequency codes (third character)

*	very common
%	ordinary
$	rare

Appendix 2

C5 Tagset

(Excluding punctuation tags)

Tag	Description
AJ0	Adjective (general or positive) (e.g. *good*, *old*, *beautiful*)
AJC	Comparative adjective (e.g. *better*, *older*)
AJS	Superlative adjective (e.g. *best*, *oldest*)
AT0	Article (e.g. *the*, *a*, *an*, *no*)
AV0	General adverb: an adverb not subclassified as AVP or AVQ (see below) (e.g. *often*, *well*, *longer* (adv.), *furthest*.
AVP	Adverb particle (e.g. *up*, *off*, *out*)
AVQ	Wh-adverb (e.g. *when*, *where*, *how*, *why*, *wherever*)
CJC	Coordinating conjunction (e.g. *and*, *or*, *but*)
CJS	Subordinating conjunction (e.g. *although*, *when*)
CJT	The subordinating conjunction *that*
CRD	Cardinal number (e.g. *one*, *3*, *fifty-five*, *3609*)
DPS	Possessive determiner-pronoun (e.g. *your*, *their*, *his*)
DT0	General determiner-pronoun: i.e. a determiner-pronoun which is not a DTQ or an AT0
DTQ	Wh-determiner-pronoun (e.g. *which*, *what*, *whose*, *whichever*)
EX0	Existential there, i.e. there occurring in the *there is* ... or *there are* ... construction
ITJ	Interjection or other isolate (e.g. *oh*, *yes*, *mhm*, *wow*)
NN0	Common noun, neutral for number (e.g. *aircraft*, *data*, *committee*)
NN1	Singular common noun (e.g. *pencil*, *goose*, *time*, *revelation*)
NN2	Plural common noun (e.g. *pencils*, *geese*, *times*, *revelations*)
NP0	Proper noun (e.g. *London*, *Michael*, *Mars*,

	IBM)
ORD	Ordinal numeral (e.g. *first*, *sixth*, *77th*, *last*) .

PNI	Indefinite pronoun (e.g. *none*, *everything*, *one* [as pronoun], *nobody*)
PNP	Personal pronoun (e.g. *I*, *you*, *them*, *ours*)
PNQ	Wh-pronoun (e.g. *who*, *whoever*, *whom*)
PNX	Reflexive pronoun (e.g. *myself*, *yourself*, *itself*, *ourselves*)
POS	The possessive or genitive marker *'s* or '
PRF	The preposition of
PRP	Preposition (except for *of*) (e.g. *about*, *at*, *in*, *on*, *on behalf* of, *with*)
TO0	Infinitive marker *to*
UNC	Unclassified items which are not appropriately considered as items of the English lexicon
VBB	The present tense forms of the verb BE, except for *is*, *'s*: i.e. *am*, *are*, *'m*, *'re* and *be* [subjunctive or imperative]
VBD	The past tense forms of the verb BE: *was* and *were*
VBG	The -ing form of the verb BE: *being*
VBI	The infinitive form of the verb BE: *be*
VBN	The past participle form of the verb BE: *been*
VBZ	The -s form of the verb BE: *is*, *'s*
VDB	The finite base form of the verb DO: *do*
VDD	The past tense form of the verb DO: *did*
VDG	The -ing form of the verb DO: *doing*
VDI	The infinitive form of the verb DO: *do*
VDN	The past participle form of the verb DO: *done*
VDZ	The -s form of the verb DO: *does*, *'s*
VHB	The finite base form of the verb HAVE: *have*, *'ve*
VHD	The past tense form of the verb HAVE: *had*, *'d*
VHG	The -ing form of the verb HAVE: *having*

VHI	The infinitive form of the verb HAVE: *have*
VHN	The past participle form of the verb HAVE: *had*
VHZ	The -s form of the verb HAVE: *has*, *'s*
VM0	Modal auxiliary verb (e.g. *will*, *would*, *can*, *could*, *'ll*, *'d*)
VVB	The finite base form of lexical verbs (e.g. *forget*, *send*, *live*, *return*) [Including the imperative and present subjunctive]
VVD	The past tense form of lexical verbs (e.g. *forgot*, *sent*, *lived*, *returned*)
VVG	The -ing form of lexical verbs (e.g. *forgetting*, *sending*, *living*, *returning*)
VVI	The infinitive form of lexical verbs (e.g. *forget*, *send*, *live*, *return*)
VVN	The past participle form of lexical verbs (e.g. *forgotten*, *sent*, *lived*, *returned*)
VVZ	The -s form of lexical verbs (e.g. *forgets*, *sends*, *lives*, *returns*)
XX0	The negative particle *not* or *n't*
ZZ0	Alphabetical symbols (e.g. *A*, *a*, *B*, *b*, *c*, *d*)

Additional tags used in CUVPlus

PRE	Prefix (e.g. *multi-*, *un-*)
ZZ2	Plural alphabetical symbol (e.g. *f's*, *p's*)

Appendix 3

Pronunciation characters

Vowels

`i` ....as in..........	*bead*
`I`	*bid*
`e`	*bed*
& (ampsnd)............	*bad*
`A`	*bard*
`0` (zero)	*cod*
`O` (cap O)...............	*cord*
`U`	*good*
`u`	*food*
`V`	*bud*
`3` (three)	*bird*
`@`	"*a*" in *about*

Consonants

`N` ....as in	*sing*
`T`..........	*thin*
`D`..........	*then*
`S`..........	*shed*
`Z`..........	*beige*
`tS`	*etch*
`dZ`	*edge*

`p t k b d g`

`m n f v s z`

`r l w h j`

Dipthongs

`eI`........as in..........	day
`@U`........	go
`aI`........	eye
`aU`........	cow
`oI`........	boy
`I@`........	beer
`e@`........	bare
`U@`........	tour

R-linking (the sounding of a /r/ at the end of a word when it is followed by a vowel) is marked `R`

eg `fAR` for "*far*"

(compare "*far away*" with "*far beyond*"

Primary stress: apostrophe e.g. `@'baUt` ("about")

Secondary stress : comma e.g. `,&ntI'septIk`

Plus-sign as in "*courtship*" and "*bookclub*" `'kOt+Sip   'bUk+klVb`

Compound words

When the spelling contains a space and/or a hyphen, the pronunciation has one also,

e.g. *above board* `@,bVv 'bOd`   *air-raid* `'e@-reId`

Appendix B: Code for Simple Good-Turing Frequency Estimator

```
 *
 *
 *      Simple Good-Turing Frequency Estimator
 *
 *
 *      Geoffrey Sampson, with help from Miles Dennis
 *
 *      School of Cognitive and Computing Sciences
 *      University of Sussex, England
 *
 *      http://www.grs.u-net.com/
 *
 *
 *      First release:  27 June 1995
 *      Revised release:  24 July 2000
 *
 *
 *      Takes a set of (frequency, frequency-of-frequency) pairs, and
 *      applies the "Simple Good-Turing" technique for estimating
 *      the probabilities corresponding to the observed frequencies,
 *      and P.0, the joint probability of all unobserved species.
 *      The Simple Good-Turing technique was devised by William A. Gale
 *      of AT&T Bell Labs, and described in Gale & Sampson,
 *      "Good-Turing Frequency Estimation Without Tears" (JOURNAL
 *      OF QUANTITATIVE LINGUISTICS, vol. 2, pp. 217-37 -- reprinted in
 *      Geoffrey Sampson, EMPIRICAL LINGUISTICS, Continuum, 2001).
 *
 *      Anyone is welcome to take copies of this program and use it
 *      for any purpose, at his or her own risk.  If it is used in
 *      connexion with published work, acknowledgment of Sampson and
 *      the University of Sussex would be a welcome courtesy.
 *
 *      The program is written to take input from "stdin" and send output
 *      to "stdout"; redirection can be used to take input from and
 *      send output to permanent files.  The code is in ANSI standard C.
 *
 *      The input file should be a series of lines separated by newline
 *      characters, where all nonblank lines contain two positive
integers
 *      (an observed frequency, followed by the frequency of that
frequency)
 *      separated by whitespace.  (Blank lines are ignored.)
 *      The lines should be in ascending order of frequency.
 *
 *      No checks are made for linearity; the program simply assumes that
the
 *      requirements for using the SGT estimator are met.
 *
 *      The output is a series of lines each containing an integer
followed
 *      by a probability (a real number between zero and one), separated
by a
 *      tab.  In the first line, the integer is 0 and the real number is
the
 *      estimate for P.0.  In subsequent lines, the integers are the
 *      successive observed frequencies, and the reals are the estimated
 *      probabilities corresponding to those frequencies.
 *
 *      The revised release cures a bug to which Martin Jansche of Ohio
 *      State University kindly drew attention.  No warranty is given
 *      as to absence of further bugs.
 *
 *
 */


 #include <stdio.h>
 #include <math.h>
 #include <ctype.h>
 #include <stdlib.h>
 #include <string.h>

 #define TRUE   1
```

```
#define FALSE  0
#define MAX_LINE       100
#define MAX_ROWS       6215
#define MIN_INPUT      5

int r[MAX_ROWS], n[MAX_ROWS];
double Z[MAX_ROWS], log_r[MAX_ROWS], log_Z[MAX_ROWS],
               rStar[MAX_ROWS], p[MAX_ROWS];
int rows, bigN;
double PZero, bigNprime, slope, intercept;

int main(void)
       {
       int readValidInput(void);
       void analyseInput(void);

       if ((rows = readValidInput()) >= 0)
               {
               if (rows < MIN_INPUT)
                       printf("\nFewer than %d input value-pairs\n",
                                       MIN_INPUT);
               else
                       analyseInput();
               }
       return(TRUE);
       }

double sq(double x)
       {
       return(x * x);
       }

int readValidInput(void)
       /*
        *      returns number of rows if input file is valid, else -1
        *      NB:  number of rows is one more than index of last row
        *
```

```
         */

        {
        char line[MAX_LINE];
        const char* whiteSpace = " \t\n\v\f\r";
        int lineNumber = 0;
        int rowNumber = 0;
        const int error = -1;

        while (fgets(line, MAX_LINE, stdin) != NULL && rowNumber <
MAX_ROWS)
                {
                char* ptr = line;
                char* integer;
                int i;

                ++lineNumber;

                while (isspace(*ptr))
                        ++ptr;  /* skip white space at the start of a
line */
                if (*ptr == '\0')
                        continue;
                if ((integer = strtok(ptr, whiteSpace)) == NULL ||
                                (i = atoi(integer)) < 1)
                        {
                        fprintf(stderr, "Invalid field 1, line %d\n",
                                        lineNumber);
                        return(error);
                        }
                if (rowNumber > 0 && i <= r[rowNumber - 1])
                        {
                        fprintf(stderr,
                      "Frequency not in ascending order, line %d\n",
                                        lineNumber);
                        return(error);
                        }
                r[rowNumber] = i;
                if ((integer = strtok(NULL, whiteSpace)) == NULL ||
                                (i = atoi(integer)) < 1)
                        {
                        fprintf(stderr, "Invalid field 2, line %d\n",
                                        lineNumber);
                        return(error);
                        }
                n[rowNumber] = i;
                if (strtok(NULL, whiteSpace) != NULL)
                        {
                        fprintf(stderr, "Invalid extra field, line %d\n",
                                        lineNumber);
                        return(error);
                        }
                ++rowNumber;
                }
        if (rowNumber >= MAX_ROWS)
                {
                fprintf(stderr, "\nInsufficient memory reserved for
input\
                        values\nYou need to change the definition of\
                        MAX_ROWS\n");
                return(error);
                }
        return(rowNumber);
        }

 void findBestFit(void)
        {
        double XYs, Xsquares, meanX, meanY;
        double sq(double);
        int i;

        XYs = Xsquares = meanX = meanY = 0.0;
        for (i = 0; i < rows; ++i)
                {
                meanX += log_r[i];
                meanY += log_Z[i];
                }
        meanX /= rows;
        meanY /= rows;
        for (i = 0; i < rows; ++i)
                {
                XYs += (log_r[i] - meanX) * (log_Z[i] - meanY);
                Xsquares += sq(log_r[i] - meanX);
                }
        slope = XYs / Xsquares;
        intercept = meanY - slope * meanX;
        }

double smoothed(int i)
        {
        return(exp(intercept + slope * log(i)));
        }

int row(int i)
        {
        int j = 0;

        while (j < rows && r[j] < i)
                ++j;
        return((j < rows && r[j] == i) ? j : -1);
        }

void showEstimates(void)
        {
        int i;

        printf("0\t%.4g\n", PZero);
        for (i = 0; i < rows; ++i)
                printf("%d\t%.4g\n", r[i], p[i]);
        }

void analyseInput(void)
        {
```

```
int i, j, next_n;
double k, x, y;
int indiffValsSeen = FALSE;
int row(int);
void findBestFit(void);
double smoothed(int);
double sq(double);
void showEstimates(void);

bigN = 0;
for (j = 0; j < rows; ++j)
        bigN += r[j] * n[j];
PZero = n[row(1)] / (double) bigN;
for (j = 0; j < rows; ++j)
        {
        i = (j == 0 ? 0 : r[j - 1]);
        if (j == rows - 1)
                k = (double) (2 * r[j] - i);
        else
                k = (double) r[j + 1];
        Z[j] = 2 * n[j] / (k - i);
        log_r[j] = log(r[j]);
        log_Z[j] = log(Z[j]);
        }
findBestFit();
for (j = 0; j < rows; ++j)
        {
        y = (r[j] + 1) * smoothed(r[j] + 1) / smoothed(r[j]);
        if (row(r[j] + 1) < 0)
                indiffValsSeen = TRUE;
        if (! indiffValsSeen)
                {
                x = (r[j] + 1) * (next_n = n[row(r[j] + 1)]) /
                                (double) n[j];
                if (fabs(x - y) <= 1.96 * sqrt(sq(r[j] + 1.0) *
                                next_n / (sq((double) n[j])) *
                                (1 + next_n / (double) n[j])))
                        indiffValsSeen = TRUE;
                else
                        rStar[j] = x;
                }
        if (indiffValsSeen)
                rStar[j] = y;
        }
bigNprime = 0.0;
for (j = 0; j < rows; ++j)
        bigNprime += n[j] * rStar[j];
for (j = 0; j < rows; ++j)
        p[j] = (1 - PZero) * rStar[j] / bigNprime;
showEstimates();
}
```

Appendix C : Sample sentence splitting

Input text from FLOB_J

Having defined the quantities that are normally measured in a nuclear reaction we here outline the typical experimental procedures which are followed for studying the symbolic reaction A(a, b)B. No details are given of the apparatus other than to mention very briefly the underlying physical principles. Details of low energy nuclear physics apparatus are given, for example, in Burcham (1988) and of high energy elementary particle apparatus in a book in this series by Kenyon (1988).

Referring to Fig. 5.3, charged ions of the particle a are produced in some form of accelerator (described later in this section) and, by use of bending magnets for example, will emerge with a particular energy. These ions then pass through a collimator in order to define their direction with some precision and strike a target containing the nuclei A. As the beam particles move through the target they will mainly lose energy by ionizing target atoms and so, if precise energy measurements are to be made, a thin target must be used. This, however, increases the difficulty of the experiment since few interactions will take place. Choice of target thickness is clearly a crucial decision in planning an experiment.

The reaction product particles b move off in all directions and their angular distribution can be studied by detecting them after passage through another collimator set at a particular angle . Various types of detector are used (discussed later) - sometimes in combination - and these can determine the type of particle as well as its energy. But experimenters have to contend with many complications of interpretation, impurities in targets and, not least, the stability of their apparatus. In the end, detailed information becomes available about , d/d and their energy dependence for the reaction under study.

Most important for nuclear reaction studies are Van de Graaff accelerators in which ions are accelerated in an evacuated tube by an electrostatic field maintained between a high voltage terminal and an earth terminal, charge being conveyed to the high voltage terminal by a rotating belt or chain. In early forms of this accelerator, positive ions from a gaseous discharge tube were accelerated from the high voltage terminal to earth. But, in modern 'tandem' accelerators, negative ions are accelerated from earth to the high voltage terminal where they are then stripped of some electrons and the resultant positive ions are further accelerated down to earth potential. The effective accelerating potential is thus twice the potential difference in the machine. High flux proton beams with energies up to around 30MeV can be produced in this way. The machines can also be used to accelerate heavy ions such as 16O.

At higher energies use is generally made of orbital accelerators in which charged particles are confined to move in circular orbits by a magnetic field. At non-relativistic energies the angular frequency of rotation , known as the cyclotron frequency, is constant depending only on the strength of the field. In a cyclotron, the particles rotate in a circular metallic box split into two halves, known as Ds, between which an oscillating electric field is maintained. Its frequency matches and so the particle is continually accelerated. In a fixed magnetic field the orbital radius increases as the energy increases and, at some maximum radius, the particles are extracted using an electrostatic deflecting field. However, as the energy becomes relativistic (remember), decreases with energy and it becomes necessary to decrease steadily the frequency of the oscillating electric field with energy to preserve synchronization.

Such a machine is known as a synchrocyclotron and protons with energies in the region of 100 MeV have been produced in this way.

For energies higher than this gigantic magnets would be needed and so the approach is to accelerate bunches of particles in orbits of essentially constant radius using annular magnets producing magnetic fields which increase as the particle energy increases: This energy increase is provided by passing the particles through radio-frequency cavities whose frequency also changes slightly as the particles are accelerated to ensure synchronization. Such devices are called synchrotrons and can be physically very large. For example, the so-called Super Proton Synchrotron (SPS) at CERN (Geneva) has a circumference around 6 km and can produce protons with energies up to around 450 GeV. LEP (the Large Electron-Positron Collider) has a circumference of 27 km and accelerates electrons (and positrons in the opposite direction) to energies of 60 GeV or more. Finally, the Superconducting Super Collider (SCC), which uses superconducting magnets, and which is being built in the USA, has a circumference of 87 km and will produce proton and antiproton beams with energies 20 000 GeV!

Electrons can also be accelerated in synchrotrons but, because of their small mass, large amounts of energy are radiated (synchrotron radiation) owing to the circular acceleration. At energies beyond a few GeV this loss becomes prohibitive and use has to be made of linear accelerators in which electrons are accelerated down a long evacuated tube by a travelling electromagnetic wave. The Stanford Linear Accelerator (SLAC) in the USA, for example, is around 3 km long and can produce pulses of electrons with energies up to 50 GeV.

Although in the early days much use was made of ionization chambers, for example the Geiger counter (section 1.4), the detectors currently in use for nuclear physics experiments are usually either scintillation counters or semiconductor detectors or some combination. The former are developments of the approach of Rutherford, Geiger and Marsden (section 1.3) using the scintillations produced in a ZnS screen to detect -particles. Various scintillators are in current use such as NaI activated by an impurity (usually thallium for detection of -particles), or some organic material dissolved in a transparent plastic or liquid. The scintillations are detected by a photomultiplier tube producing a pulse of photo-electrons. The size of the pulse - the pulse height - gives a measure of the energy of the incident particle.

Semiconductor detectors depend on an incident particle or photon exciting an electron from the valence band to the conduction band. The resultant increase in conductivity - a conduction pulse - then produces a signal which is processed electronically and which enables the energy of the incident radiation to be measured.

In the field of very high energy physics, considerable use is made of bubble chambers and wire chambers. The former follows on from the Wilson cloud chamber and consists essentially of a large chamber, possibly several metres in diameter, containing liquid (e.g. hydrogen, helium, propane, ...) near its boiling point. The chamber is expanded as charged particles pass through it, leading to the formation of bubbles, as a result of boiling, along the particle tracks which can be stereo flash photographed. The lengths of the tracks and their curvature in a magnetic field enable particle lifetimes, masses and energies to be deduced.

Wire chambers consist of stacks of positively and negatively charged wire grids in a low pressure gas. An incident charged particle ionizes the gas and acceleration of the resultant electrons near the anode wires leads to further ionization and an electrical pulse. The physical location of the pulse can be determined electronically so that track measurements can be made. Using an applied magnetic field to bend the tracks again enables information to be obtained about the properties of the detected particle.

In the previous chapter some understanding of nuclear structure has been achieved in terms of a nuclear model in which nucleons move around fairly independently in a potential well. To give some intuitive understanding of nuclear reaction processes we stay with this description of the nucleus and follow a very illuminating discussion given by Weisskopf (1957).

Output from sentence splitter

Snum 1

Having defined the quantities that are normally measured in a nuclear reaction we here outline the typical experimental procedures which are followed for studying the symbolic reaction A(a, b)B.

Snum 2

No details are given of the apparatus other than to mention very briefly the underlying physical principles.

Snum 3

Details of low energy nuclear physics apparatus are given, for example, in Burcham (1988) and of high energy elementary particle apparatus in a book in this series by Kenyon (1988).

Snum 4

Referring to Fig. 5.3, charged ions of the particle a are produced in some form of accelerator (described later in this section) and, by use of bending magnets for example, will emerge with a particular energy.

Snum 5

These ions then pass through a collimator in order to define their direction with some precision and strike a target containing the nuclei A.

Snum 6

As the beam particles move through the target they will mainly lose energy by ionizing target atoms and so, if precise energy measurements are to be made, a thin target must be used.

Snum 7

This, however, increases the difficulty of the experiment since few interactions will take place.

Snum 8

Choice of target thickness is clearly a crucial decision in planning an experiment.

Snum 9

The reaction product particles b move off in all directions and their angular distribution can be studied by detecting them after passage through another collimator set at a particular angle.

Snum 10

Various types of detector are used (discussed later) - sometimes in combination - and these can determine the type of particle as well as its energy.

Snum 11

But experimenters have to contend with many complications of interpretation, impurities in targets and, not least, the stability of their apparatus.

Snum 12

In the end, detailed information becomes available about, d/d and their energy dependence for the reaction under study.

Snum 13

Most important for nuclear reaction studies are Van de Graaff accelerators in which ions are accelerated in an evacuated tube by an electrostatic field maintained between a high voltage terminal and an earth terminal, charge being conveyed to the high voltage terminal by a rotating belt or chain.

Snum 14

In early forms of this accelerator, positive ions from a gaseous discharge tube were accelerated from the high voltage terminal to earth.

Snum 15

But, in modern 'tandem' accelerators, negative ions are accelerated from earth to the high voltage terminal where they are then stripped of some electrons and the resultant positive ions are further accelerated down to earth potential.

Snum 16

The effective accelerating potential is thus twice the potential difference in the machine.

Snum 17

High flux proton beams with energies up to around 30MeV can be produced in this way.

Snum 18

The machines can also be used to accelerate heavy ions such as 16O.

Snum 19

At higher energies use is generally made of orbital accelerators in which charged particles are confined to move in circular orbits by a magnetic field.

Snum 20

At non-relativistic energies the angular frequency of rotation, known as the cyclotron frequency, is constant depending only on the strength of the field.

Snum 21

In a cyclotron, the particles rotate in a circular metallic box split into two halves, known as Ds, between which an oscillating electric field is maintained.

Snum 22

Its frequency matches and so the particle is continually accelerated.

Snum 23

In a fixed magnetic field the orbital radius increases as the energy increases and, at some maximum radius, the particles are extracted using an electrostatic deflecting field.

Snum 24

However, as the energy becomes relativistic (remember), decreases with energy and it becomes necessary to decrease steadily the frequency of the oscillating electric field with energy to preserve synchronization.

Snum 25

Such a machine is known as a synchrocyclotron and protons with energies in the region of 100 MeV have been produced in this way.

Snum 26

For energies higher than this gigantic magnets would be needed and so the approach is to accelerate bunches of particles in orbits of essentially constant radius using annular magnets producing magnetic fields which increase as the particle energy increases: This energy increase is provided by passing the particles through radio-frequency cavities whose frequency also changes slightly as the particles are accelerated to ensure synchronization.

Snum 27

Such devices are called synchrotrons and can be physically very large.

Snum 28

For example, the so-called Super Proton Synchrotron (SPS) at CERN (Geneva) has a circumference around 6 km and can produce protons with energies up to around 450 GeV.

Snum 29

LEP (the Large Electron-Positron Collider) has a circumference of 27 km and accelerates electrons (and positrons in the opposite direction) to energies of 60 GeV or more.

Snum 30

Finally, the Superconducting Super Collider (SCC), which uses superconducting magnets, and which is being built in the USA, has a circumference of 87 km and will produce proton and antiproton beams with energies 20 000 GeV!

Snum 31

Electrons can also be accelerated in synchrotrons but, because of their small mass, large amounts of energy are radiated (synchrotron radiation) owing to the circular acceleration.

Snum 32

At energies beyond a few GeV this loss becomes prohibitive and use has to be made of linear accelerators in which electrons are accelerated down a long evacuated tube by a travelling electromagnetic wave.

Snum 33

The Stanford Linear Accelerator (SLAC) in the USA, for example, is around 3 km long and can produce pulses of electrons with energies up to 50 GeV.

Snum 34

Although in the early days much use was made of ionization chambers, for example the Geiger counter (section 1.4), the detectors currently in use for nuclear physics experiments are usually either scintillation counters or semiconductor detectors or some combination.

Snum 35

The former are developments of the approach of Rutherford, Geiger and Marsden (section 1.3) using the scintillations produced in a ZnS screen to detect -particles.

Snum 36

Various scintillators are in current use such as NaI activated by an impurity (usually thallium for detection of -particles), or some organic material dissolved in a transparent plastic or liquid.

Snum 37

The scintillations are detected by a photomultiplier tube producing a pulse of photo-electrons.

Snum 38

The size of the pulse - the pulse height - gives a measure of the energy of the incident particle.

Snum 39

Semiconductor detectors depend on an incident particle or photon exciting an electron from the valence band to the conduction band.

Snum 40

The resultant increase in conductivity - a conduction pulse - then produces a signal which is processed electronically and which enables the energy of the incident radiation to be measured.

Snum 41

In the field of very high energy physics, considerable use is made of bubble chambers and wire chambers.

Snum 42

The former follows on from the Wilson cloud chamber and consists essentially of a large chamber, possibly several metres in diameter, containing liquid (e.g. hydrogen, helium, propane,...) near its boiling point.

Snum 43

The chamber is expanded as charged particles pass through it, leading to the formation of bubbles, as a result of boiling, along the particle tracks which can be stereo flash photographed.

Snum 44

The lengths of the tracks and their curvature in a magnetic field enable particle lifetimes, masses and energies to be deduced.

Snum 45

Wire chambers consist of stacks of positively and negatively charged wire grids in a low pressure gas.

Snum 46

An incident charged particle ionizes the gas and acceleration of the resultant electrons near the anode wires leads to further ionization and an electrical pulse.

Snum 47

The physical location of the pulse can be determined electronically so that track measurements can be made.

Snum 48

Using an applied magnetic field to bend the tracks again enables information to be obtained about the properties of the detected particle.

Snum 49

In the previous chapter some understanding of nuclear structure has been achieved in terms of a nuclear model in which nucleons move around fairly independently in a potential well.

Snum 50

To give some intuitive understanding of nuclear reaction processes we stay with this description of the nucleus and follow a very illuminating discussion given by Weisskopf (1957).

Appendix D: LOB Tagset

Reproduced from LOB manual available at;

http://khnt.hit.uib.no/icame/manuals/lobman/LOBAPP4.HTM

Appendix 4: List of tags

Definitions are followed by references to sections in the manual where the tags are discussed. The figures in the righthand column give the total frequency of the tags in the corpus. Ditto tags are given within parantheses; the first word in the examples carries an ordinary tag. For more examples of ditto-tagged sequences, see 7.2.

!	exclamation mark	1.030
&FO	formula 7.22	1.220
&FW	foreign word 7.21	3.111
(	left bracket: ([	2.903
)	right bracket)]	2.975
*’	begin quote: *’ *" 2.6	10.191
**’	end quote: **’ **" 2.6	9.976
*_	dash 7.24	3.930
,	comma 7.24	54.548
.	full stop 7.24	50.288
...	ellipsis	665
:	colon 7.24	1.937
;	semicolon 7.24	2.514
?	question mark	2.584
ABL	pre-qualifier (*quite, rather, such*) 7.12	1.032
ABN	pre-quantifier (*all, half*) 7.12	2.833
ABX	pre-quantifier/double conjunction (*both*)	675
AP	post-determiner (*few, fewer, former, last, latter, least, less, little, many, more, most, much, next, only, other, own, same, several, very*) 7.12	8.860
(AP"	*a few, a little*	448)
AP$	*other’s*	21
APS	*others*	272
APS$	*others’*	2

AT	singular article (*a, an, every*) 7.12	25.906
ATI	singular og plural article (*the, no*) 7.12	70.219
BE	*be* 7.5	7.187
BED	*were*	3.427
BEDZ	*was*	10.685
BEG	*being*	907
BEM	*am, 'm*	636
BEN	*been*	3.116
BER	*are, 're*	4.856
BEZ	*is, 's*	12.165
CC	coordinating conjunction (*and, and/or, but, nor, only, or, yet*) 7.14 - 7.15	36.919
(CC"	*as well as*	282)
CD	cardinal (*2, 3,* etc; *two, three,* etc; *hundred, thousand,* etc; *dozen, zero*) 7.17	12.956
CD$	cardinal + genitive	7
CD-CD	hyphenated pair of cardinals 7.17	304
CD1	*one*, 1 7.17	3.364
CD1$	*one's*	62
CD1S	*ones*	105
CDS	cardinal + plural (*tens, millions, dozens*)	263
CS	subordinating conjunction (*after, although,* etc)	18.583
(CS"	*in that, so that,* etc	850)
DO	*do* 7.5	2.005
DOD	*did*	1.174
DOZ	*does*	618
DT	singular determiner (*another, each, that, this*) 7.12	9.030
DT$	singular determiner + genitive (*another's*)	1
DTI	singular or plural determiner (*any, enough, some*)	3.349
DTS	plural determiner (*these, those*)	2.462
DTX	determiner/double conjunction (*either, neither*) 7.12	376
EX	existensial *there* 7.10	2.794

HV	*have* 7.5	4.998
HVD	*had, ‘d*	5.499
HVG	*having*	382
HVN	*had* (past participle)	284
HVZ	*has, ‘s*	2.916
IN	preposition (*about, above,* etc) 7.13, 7.15	123.440
(IN"	*as to, in spite of*, etc	1.216)
JJ	adjective 7.3 - 7.4, 7.8 - 7.9, 7.11	63.877
(JJ"		25)
JJB	attribute-only adjective (*chief, entire, main* etc) 7.8	3.578
(JJB"		6)
JJR	comparative adjective 7.9, 7.11	1.960
(JJR"		1)
JJT	superlative adjective 7.9, 7.11	1.040
(JJT"		1)
JNP	adjective with word-initial capital (*English, German*, etc)	3.137
MD	modal auxiliary (*‘ll, can, could,* etc)	14.861
NC	cited word 7.23	370
NN	singular common noun 7.4, 7.6, 7.7	148.759
(NN"		20)
NN$	singular common noun + genitive 7.6	1.574
NNP	singular common noun with word-initial capital (*Englishman, German*, etc)	532
NNP$	singular common noun with word-initial capital + genitive	21
NNPS	plural common noun with word-initial capital	782
NNPS$	plural common noun with word-initial capital + genitive	7
NNS	plural common noun 7.6, 7.7	50.838
(NNS"		2)
NNS$	plural common noun + genitive	488
NNU	abbreviated unit of measurement unmarked for	

	number (\0*hr*, \0*lb*, etc) 7.19	2.625
(NNU"	per cent	386)
NNUS	abbreviated plural unit of measurement (\0*gns*, \0*yds*, etc)	52
NP	singular proper noun 7.7	34.797
NP$	singular proper noun + genitive	2.479
NPL	singular locative noun with wordinitial capital (*Abbey*, *Bridge*, etc) 7.7	1.952
NPL$	singular locative noun with word-initial capital + genitive	15
NPLS	plural locative noun with word-initial capital	102
NPLS$	plural locative noun with word-initial capital + genitive	1
NPS	plural proper noun 7.7	406
NPS$	plural proper noun + genitive	28
NPT	singular titular noun with word-initial capital (*Archbishop*, *Captain*, etc) 7.7	6.039
(NPT"		2)
NPT$	singular titular noun with word capital + genitive	197
NPTS	plural titular noun with word-initial capital	215
NPTS$	plural titular noun with word-initial capital + genitive	4
NR	singular adverbial noun (*January*, *February*, *etc; Sunday*, *Monday*, *etc; east*, *west*, *etc; today*, *tomorrow*, *tonight*, *downtown*, *home*) 7.10	2.916
NR$	singular adverbial noun + genitive	48
NRS	plural adverbial noun	84
NRS$	plural adverbial noun + genitive	0
OD	ordinal (*1st*, *2nd*, *etc; first*, *second*, etc)	2.069
OD$	ordinal + genitive	0
PN	nominal pronoun (*anybody*, *anyone*, *anything; everybody*, *everyone*, *everything; nobody*, *none*, *nothing; somebody*, *someone*, *something; so*) 7.12, 7.14	2.581
(PN"	no one, some one	120)
PN$	nominal pronoun + genitive	10

(PN$		1)
PP$	possessive determiner (my, your, etc) 7.12	17.004
PP$$	possessive pronoun (mine, yours, etc)	204
PP1A	personal pronoun, 1st pers sing nom (*I*)	7.600
PP1AS	personal pronoun, 1st pers plur nom (*we*)	3.129
PP1O	personal pronoun, 1st pers sing acc (*me*)	1.555
PP1OS	personal pronoun, 1st pers plur acc (*us, 's*)	696
PP2	personal pronoun, 2nd pers (*you, thou, thee, ye*)	4.137
PP3	personal pronoun, 3rd pers sing nom+acc (*it*)	10.507
PP3A	personal pronoun, 3rd pers sing nom (*he, she*)	13.160
PP3AS	personal pronoun, 3rd pers plur nom (*they*)	3.685
PP3O	personal pronoun, 3rd pers plur acc (*him, her*)	3.784
PP3OS	personal pronoun, 3rd pers plur acc (*them, 'em*)	1.715
PPL	singular reflexive pronoun	1.257
PPLS	plural reflexive pronoun, reciprocal pronoun	464
(PPLS"	each other, one another	140)
QL	qualifier (*as, awfully, less, more, so, too very,* etc)	5.375
QLP	post-qualifier (*enough, indeed*)	283
RB	adverb 7.10 7.11	35.353
(RB"	at last, in general, etc	1.781)
RB$	adverb + genitive (*else's*)	6
RBR	comparative adverb 7.10 - 7.11	1.375
RBT	superlative adverb 7.10 - 7.11	103
RI	adverb (homograph of preposition: *below, near,* etc)	571
RN	nominal adverb (*here, now, there, then,* etc)	4.332
RP	adverbial particle (*back, down, off,* etc) 7.10, 7.13	8.700
TO	infinitival *to* 7.13	15.842
(TO"	*in order to, so as to*	268)
UH	interjection 7.18	1.113
VB	base form of verb (uninflected present tense, imperative, infinitive) 7.5	32.679
(VB"		3)

VBD	past tense of verb 7.3	24.679
VBG	present participle, gerund 7.4	12.979
VBN	past participle 7.3	27.031
VBZ	3rd person singular of verb	6.918
WDT	WH-determiner (*what, whatever, whatsoever, interrogative which, whichever, whichsoever*) 7.16	2.118
(WDT"		1)
WDTR	WH-determiner, relative (*which*) 7.16	4.405
WP	WH-pronoun, interrogative, nom+acc (*who, whoever*)	149
WP$	WH-pronoun, interrogative, gen (*whose*)	8
WP$R	WH-pronoun, relative, gen (*whose*)	293
WPA	WH-pronoun, nom (*whosoever*)	1
WPO	WH-pronoun, interrogative, acc (*whom, whomsoever*)	6
WPOR	WH-pronoun, relative, acc (*whom*)	214
WPR	WH-pronoun, relative, nom+acc (*that, relative who*)	3.448
WRB	WB-adverb (*how, when*, etc) 7.16	5.076
XNOT	not, n’t 5.3	7.454
ZZ	letter of the alphabet (e, pi, x, etc) 7.25	1.349

Appendix E: Mapping between BNC and LOB tagsets

This is the mapping used for the evaluation of the tagger described in Chapter 8:.

Each of the tags in the BNC (C5) tagset (apart from POS (possessive marker) and UNC (unclassified)) is mapped to one or more of the LOB tags. Where more than one LOB tag matches, the tags are separated by | (pipe).

As the LOB tagset is more fine-grained than the BNC tagset, a single BNC tag can subsume a 'family' of LOB tags. For example, the DT0 (general determiner pronoun) BNC tag matches the three post/pre-quantifier tags ABL, ABN and ABX in the LOB tagset. In cases such as this where the first two letters are adequate to identify a match with the entire group, only these two letters are shown in the table.

The descriptions, in the third column are those given for the BNC tags.

BNC Tag	**LOB Tag**	**Description (BNC tag)**
AJ0	JJ|JNP	Adjective (general or positive) (e.g. *good*, *old*, *beautifu*l)
AJC	JJR	Comparative adjective (e.g. *better*, *olde*r)
AJS	JJT	Superlative adjective (e.g. *best*, *oldest*)
AT0	AT	Article (e.g. *the*, *a*, *an*, *no*)
AV0	RB|ABL|NR|RN|QL	General adverb: an adverb not subclassified as AVP or AVQ (see below) (e.g. *often*, *well*, *longer* (adv.), *furthest*.
AVP	RP	Adverb particle (e.g. *up*, *off*, *out*)
AVQ	WRB	Wh-adverb (e.g. *when*, *where*, *how*, *why*, *wherever*)
CJC	CC	Coordinating conjunction (e.g. *and*, *or*, *but*)
CJS	CS	Subordinating conjunction (e.g. *although*, *when*)
CJT	CS	The subordinating conjunction *that*
CRD	CD	Cardinal number (e.g. *one*, *3*, *fifty-five*, *3609*)
DPS	PP	Possessive determiner-pronoun (e.g. *your*, *their*, *his*)
DT0	DT|AB|AP	General determiner-pronoun: i.e. a determiner-pronoun which is not a DTQ or an AT0

BNC Tag	**LOB Tag**	**Description (BNC tag)**
DTQ	WD|WP	Wh-determiner-pronoun (e.g. *which*, *what*, *whose*, *whichever*)
EX0	EX	Existential there, i.e. there occurring in the *there is* ... or *there are* ... construction
ITJ	UH	Interjection or other isolate (e.g. *oh*, *yes*, *mhm*, *wow*)
NN0	NN	Common noun, neutral for number (e.g. *aircraft*, *data*, *committee*)
NN1	NN|NNP|NPT|NR	Singular common noun (e.g. *pencil*, *goose*, *time*, *revelation*)
NN2	NNS|NNPS	Plural common noun (e.g. *pencils*, *geese*, *times*, *revelations*)
NP0	NP	Proper noun (e.g. *London*, *Michael*, *Mars*, *IBM*)
ORD	OD	Ordinal numeral (e.g. *first*, *sixth*, *77th*, *last*) .
PNI	PN	Indefinite pronoun (e.g. none, everything, one [as pronoun], nobody)
PNP	PP	Personal pronoun (e.g. I, you, them, ours)
PNQ	WP	Wh-pronoun (e.g. who, whoever, whom)
PNX	PP	Reflexive pronoun (e.g. myself, yourself, itself, ourselves)
POS	Not matched	The possessive or genitive marker 's or '
PRF	IN	The preposition of
PRP	IN	Preposition (except for of) (e.g. about, at, in, on, on behalf of, with)
TO0	TO	Infinitive marker to
UNC	Not matched	Unclassified items which are not appropriately considered as items of the English lexicon
VBB	BEM|BER	The present tense forms of the verb BE, except for is, 's: i.e. am, are, 'm, 're and be [subjunctive or imperative]
VBD	BED|BEDZ	The past tense forms of the verb BE: was and were
VBG	BEG	The -ing form of the verb BE: being
VBI	BE	The infinitive form of the verb BE: be
VBN	BEN	The past participle form of the verb BE: been
VBZ	BEZ	The -s form of the verb BE: is, 's
VDB	DO	The finite base form of the verb DO: do
VDD	DOD	The past tense form of the verb DO: did
VDG	VBG	The -ing form of the verb DO: doing

BNC Tag	LOB Tag	Description (BNC tag)
VDI	DO	The infinitive form of the verb DO: do
VDN	VBN	The past participle form of the verb DO: done
VDZ	DOZ	The -s form of the verb DO: does, 's
VHB	HV	The finite base form of the verb HAVE: have, 've
VHD	HVD	The past tense form of the verb HAVE: had, 'd
VHG	HVG	The -ing form of the verb HAVE: having
VHI	HV	The infinitive form of the verb HAVE: have
VHN	HVN	The past participle form of the verb HAVE: had
VHZ	HVZ	The -s form of the verb HAVE: has, 's
VM0	MD	Modal auxiliary verb (e.g. will, would, can, could, 'll, 'd)
VVB	VB	The finite base form of lexical verbs (e.g. forget, send, live, return) [Including the imperative and present subjunctive]
VVD	VBD	The past tense form of lexical verbs (e.g. forgot, sent, lived, returned)
VVG	VBG	The -ing form of lexical verbs (e.g. forgetting, sending, living, returning)
VVI	VB	The infinitive form of lexical verbs (e.g. forget, send, live, return)
VVN	VBN	The past participle form of lexical verbs (e.g. forgotten, sent, lived, returned)
VVZ	VBZ	The -s form of lexical verbs (e.g. forgets, sends, lives, returns)
XX0	XNOT	The negative particle not or n't
ZZ0	ZZ	Alphabetical symbols (e.g. A, a, B, b, c, d)
ZZ2	ZZ	Plural alphabetical symbol (e.g. f's, p's)

Appendix F: Comparative Tagger output

The table below gives a comparison of the tags output by my tagger and the tags assigned in the LOB corpus for the first ten sentences of the LOBTH_A.TXT file (Section 8.3.4).

The Flag column contains 'C' where the tag assigned by my program matched the tag in the LOB file and 'E' where there was a discrepancy.

	My Tag	**Lob Tag**	**Flag**
Snum 1			
Stop	NP0	VB	E
electing	VVG	VBG	C
life	NN1	NN	C
peers	NN2	NNS	C
Snum 2			
By	PRP	IN	C
Trevor	NP0	NP	C
Williams	NP0	NP	C
Snum 3			
A	AT0	AT	C
move	NN1	NN	C
to	TO0	TO	C
stop	VVI	VB	C
Mr	NP0	NPT	C
Gaitskell	NP0	NP	C
from	PRP	IN	C
nominating	VVG	VBG	C
any	DT0	DTI	C
more	AV0	AP	E
labour	AJ0	NN	E
life	NN1	NN	C
peers	NN2	NNS	C
is	VBZ	BEZ	C

	My Tag	**Lob Tag**	**Flag**
to	TO0	TO	C
be	VBI	BE	C
made	VVN	VBN	C
at	PRP	IN	C
a	AT0	AT	C
meeting	NN1	NN	C
of	PRF	IN	C
labour	AJ0	NN	E
MPs	NP0	NPTS	C
tomorrow	AV0	NR	C
Snum 4			
Mr	NP0	NPT	C
Michael	NP0	NP	C
Foot	NP0	NP	C
has	VHZ	HVZ	C
put	VVN	VBN	C
down	AVP	RP	C
a	AT0	AT	C
resolution	NN1	NN	C
on	PRP	IN	C
the	AT0	ATI	C
subject	NN1	NN	C
and	CJC	CC	C
he	PNP	PP3A	C
is	VBZ	BEZ	C
to	TO0	TO	C
be	VBI	BE	C
backed	VVN	VBN	C
by	PRP	IN	C
Mr	NP0	NPT	C
Will	VM0	NP	E
Griffiths	NP0	NP	C

	My Tag	**Lob Tag**	**Flag**
MP	NN1	NPT	C
for	PRP	IN	C
Manchester	NP0	NP	C
Exchange	NP0	NP	C
Snum 5			
Though	CJS	CS	C
they	PNP	PP3AS	C
may	VM0	MD	C
gather	VVI	VB	C
some	DT0	DTI	C
left-wing	AJ0	JJB	C
support	NN1	NN	C
a	AT0	AT	C
large	AJ0	JJ	C
majority	NN1	NN	C
of	PRF	IN	C
labour	AJ0	NN	E
MPs	NP0	NPTS	C
are	VBB	BER	C
likely	AJ0	JJ	C
to	TO0	TO	C
turn	VVI	VB	C
down	AVP	RP	C
the	AT0	ATI	C
Foot-Griffiths	NP0	NP	C
resolution	NN1	NN	C
Snum 6			
Abolish	NP0	VB	E
Lords	NP0	NPTS	C
Snum 7			
Mr	NP0	NPT	C
Foot's	NP0	NP$	C

	My Tag	**Lob Tag**	**Flag**
line	NN1	NN	C
will	VM0	MD	C
be	VBI	BE	C
that	CJT	CS	C
as	CJS	CS	C
labour	AJ0	NN	E
MPs	NP0	NPTS	C
opposed	VVD	VBD	C
the	AT0	ATI	C
government	NN1	NN	C
bill	NN1	NN	C
which	DTQ	WDTR	C
brought	VVD	VBD	C
life	NN1	NN	C
peers	NN2	NNS	C
into	PRP	IN	C
existence	NN1	NN	C
they	PNP	PP3AS	C
should	VM0	MD	C
not	XX0	XNOT	C
now	AV0	RN	C
put	VVN	VB	E
forward	AV0	RB	C
nominees	NN2	NNS	C
Snum 8			
He	PNP	PP3A	C
believes	VVZ	VBZ	C
that	CJT	CS	C
the	AT0	ATI	C
House	NP0	NPL	C
of	PRF	IN	C
Lords	NP0	NPTS	C

	My Tag	**Lob Tag**	**Flag**
should	VM0	MD	C
be	VBI	BE	C
abolished	VVN	VBN	C
and	CJC	CC	C
that	DT0	CS	E
labour	NN1	NN	C
should	VM0	MD	C
not	XX0	XNOT	C
take	VVI	VB	C
any	DT0	DTI	C
steps	NN2	NNS	C
which	DTQ	WDTR	C
would	VM0	MD	C
appear	VVI	VB	C
to	PRP	TO	E
prop	VVB	VB	C
up	AVP	RP	C
an	AT0	AT	C
out-dated	AJ0	JJ	C
institution	NN1	NN	C
Snum 9			
Since	CJS	IN	E
labour	AJ0	NN	E
life	NN1	NN	C
peers	NN2	NNS	C
and	CJC	CC	C
peeresses	NN2	NNS	C
have	VHB	HV	C
been	VBN	BEN	C
created	VVN	VBN	C
Snum 10			
Most	AV0	AP	E

	My Tag	**Lob Tag**	**Flag**
labour	AJ0	NN	E
sentiment	NN1	NN	C
would	VM0	MD	C
still	AV0	RB	C
favour	VVI	VB	C
the	AT0	ATI	C
abolition	NN1	NN	C
of	PRF	IN	C
the	AT0	ATI	C
House	NP0	NPL	C
of	PRF	IN	C
Lords	NP0	NPTS	C
but	CJC	CC	C
while	CJS	CS	C
it	PNP	PP3	C
remains	VVZ	VBZ	C
labour	NN1	NN	C
has	VHZ	HVZ	C
to	TO0	TO	C
have	VHI	HV	C
an	AT0	AT	C
adequate	AJ0	JJ	C
number	NN1	NN	C
of	PRF	IN	C
members	NN2	NNS	C

Appendix G: Results of tagger assessment

For each tag in the C5 tagset, shown in column 1, the table shows the total number of words that were assigned that tag by my tagger (column 2), the number of those tags that matched the tags assigned in the LOB file LOBTH_A.TXT (column 3), the number where there was a discrepancy (column 4), the discrepancies as a .percentage of the total assignment of that tag (column 5) and the discrepancies for that tag as a percentage of the total tags (column 6).

Tag	Total tags	Match	Non-match	% Discrepancy tag	% Discrepancy non-matches
AJ0	5762	5115	647	11.23%	11.22%
AJC	163	127	36	22.09%	0.62%
AJS	106	104	2	1.89%	0.03%
AT0	8449	8385	64	0.76%	1.11%
AV0	3284	2878	406	12.36%	7.04%
AVP	628	583	45	7.17%	0.78%
AVQ	114	110	4	3.51%	0.07%
CJC	2719	2709	10	0.37%	0.17%
CJS	1073	559	514	47.90%	8.91%
CJT	732	630	102	13.93%	1.77%
CRD	796	750	46	5.78%	0.80%
DPS	1181	1181	0	0.00%	0.00%
DT0	1531	1440	91	5.94%	1.58%
DTQ	422	422	0	0.00%	0.00%
EX0	197	197	0	0.00%	0.00%
ITJ	11	9	2	18.18%	0.03%
NN0	221	217	4	1.81%	0.07%
NN1	13638	13206	432	3.17%	7.49%
NN2	4437	4368	69	1.56%	1.20%
NP0	9775	8591	1184	12.11%	20.53%
ORD	453	189	264	58.28%	4.58%
PNI	172	120	52	30.23%	0.90%
PNP	3050	3048	2	0.07%	0.03%
PNQ	259	259	0	0.00%	0.00%
PNX	59	59	0	0.00%	0.00%
PRF	2743	2721	22	0.80%	0.38%
PRP	8227	7794	433	5.26%	7.51%
TO0	1272	1260	12	0.94%	0.21%
VBB	373	366	7	1.88%	0.12%
VBD	1177	1177	0	0.00%	0.00%
VBG	81	81	0	0.00%	0.00%
VBI	603	603	0	0.00%	0.00%
VBN	263	263	0	0.00%	0.00%
VBZ	909	907	2	0.22%	0.03%
VDB	68	68	0	0.00%	0.00%

Tag	**Total tags**	**Match**	**Non-match**	**% Discrepancy tag**	**% Discrepancy non-matches**
VDD	70	70	0	0.00%	0.00%
VDG	14	14	0	0.00%	0.00%
VDI	39	39	0	0.00%	0.00%
VDN	29	28	1	3.45%	0.02%
VDZ	24	24	0	0.00%	0.00%
VHB	235	235	0	0.00%	0.00%
VHD	346	345	1	0.29%	0.02%
VHG	34	34	0	0.00%	0.00%
VHI	154	154	0	0.00%	0.00%
VHN	33	28	5	15.15%	0.09%
VHZ	377	377	0	0.00%	0.00%
VM0	1116	1114	2	0.18%	0.03%
VVB	689	593	96	13.93%	1.66%
VVD	2296	2123	173	7.53%	3.00%
VVG	1137	995	142	12.49%	2.46%
VVI	1707	1687	20	1.17%	0.35%
VVN	2324	2067	257	11.06%	4.46%
VVZ	621	565	56	9.02%	0.97%
XX0	487	487	0	0.00%	0.00%
ZZ0	571	9	562	98.42%	9.74%
ZZ2	1	0	1	100.00%	0.02%
Totals	**87252**	**81484**	**5768**		**6.6%%**