EMOS ✓ ✓x
1/7/13

GN

Q ?HX?

The Use of Orthographic and Lexical Information for Handwriting Recognition

by

Cynthia Joyce Wells

This thesis has been submitted to the Council for National Academic Awards in partial fulfilment of the requirements for the degree of Doctor of Philosophy

(This work was conducted at Nottingham Polytechnic Department of Computing)

June 1992

The Use of Orthographic and Lexical Information for Handwriting Recognition

Cynthia Joyce Wells

Doctor of Philosophy

Abstract

This thesis details the research work undertaken by the author from July 1987 to May 1992 concerning the automatic recognition of handwriting by computer. The emphasis has been the development of algorithms and data structures which facilitate the use of orthographic and lexical information to resolve the ambiguity present in handwriting recognition systems.

Good recognition performance is ultimately linked not only to the capacity of the system to extract and compare good feature sets, but also to the integration of context and knowledge in the different processing stages. Hence the current study forms the first part of a contextual recognition system. It describes a number of levels of analysis for handwriting recognition, which begin with the application of orthographic information. Letters do not combine arbitrarily to form words, so letter string combinations produced by a pattern recogniser can be checked for acceptability. Such a process is known as a lexical check and requires a list of words or lexicon against which to compare alternative candidate strings.

A list of words can be acquired from a standard dictionary. However the words must be stored in a suitable data structure which can easily be searched to check the existence of candidate letter strings. This structure should preferably be searchable in real time and have only modest memory requirements so that it could be part of a recognition system on a personal computer. These aspects are considered within this thesis. The advantages and disadvantages of a number of methods are introduced and compared. The lexical analysis system described can interface to higher level linguistic constraints which aid the recognition process.

In the future it is hoped that the computer could be a "notepad" style portable. The interface to such computers should mimic conventional pen and paper instead of using a keyboard. Hence efficient and reliable handwriting recognition will necessarily form an important part of this new technology.

Acknowledgments

I would like to thank the following (in no particular order) for their assistance and encouragement over the past five years.

Lindsay Evett, Frank Keenan, Tony Rose, Warren Smith, Paul Whitby, other colleagues at Nottingham Polytechnic, members of my family, and especially to Mike Flynn for his unfailing support (especially with the arduous task of proof reading).

Thanks also to my students for providing some "light" relief !

Special thanks are due to my supervisors Bob Whitrow and Lindsay Evett. To Bob for offering me the opportunity to study for a research degree and to Lindsay for reading and commenting constructively on many versions of this thesis.

Frank Keenan developed the indexing system described in § 4.2.3, and the set of flags used in § 4.4.3, in conjunction with myself.

Mike Flynn suggested the hash table data structure for the inverted look-up in § 5.4.3, and provided many helpful discussions regarding the algorithms (described in § 3.2.5) for constructing a directed acyclic word graph.

The sections of this thesis which have been published are:

- § 3.2.3.2, § 3.3.1, § 3.3.3, § 3.4.1 and § 3.4.2 (published as Wells et al, 1990a);
- § 3.2.3.2, § 3.3.1 and § 3.3.2 (published as Wells et al, 1990b);
- § 3.3.4 (presented and published as Wells et al, 1989);
- § 5.3 (which is published as Wells et al, 1991).

This research was funded by the European Commission under the ESPRIT initiative. Firstly as part of project 295 — The Paper Interface, continued in project 5204 — Papyrus, and utilised in project 5203 — Intrepid.

Copyright

For my mother, Edna Wells

Table of Contents

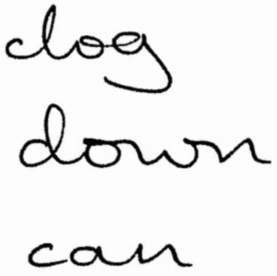
Introduction

But Eeyore was saying to himself, "This writing business. Pencils and what-not. Over-rated, if you ask me. Silly stuff. Nothing in it."

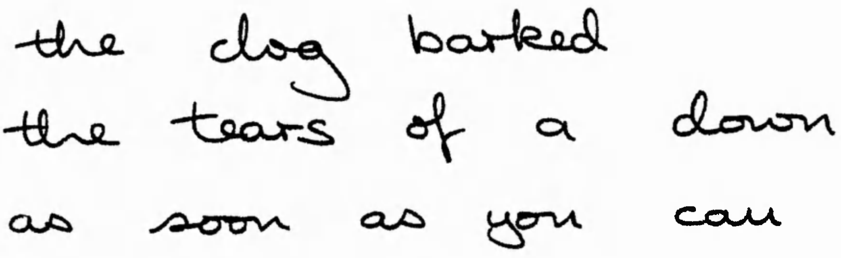
'Winnie-the-Pooh', by AA Milne.

This thesis details research undertaken by the author in the Department of Computing at Nottingham Polytechnic over almost five years. The work has been funded by the European Commission under the ESPRIT initiative. The subject of the research is the automatic recognition of handwriting by computer. In particular, it details a system which takes output from a pattern recogniser in the form of alternative characters, and applies orthographic and lexical information in order to discriminate between these alternative characters.

Handwriting, or script recognition is a difficult task due to the inherent ambiguity within the input. For example, do the following words say *dog* or *clog, clown* or *down* ? The word *can* really looks more like *cau.*



However, when these words occur within a meaningful context, they can easily be read and understood by humans. For example :



Handwriting contains many similarly shaped characters which must be distinguished from each other to achieve effective recognition. Some easily confusable characters pairs are U-V, C-L, a-d, n-h, o-a, and c-e. Upper and lower case letters are often written the same, such as C-c, K-k, and O-o, the distinguishing factor here is the character size relative to the line spacing. For P-p and Y-y it is the position of the characters relative to the baseline. Letters can also be confused with digits such as O-0, I-1, l-1, Z-2, S-5, g-9 and G-6. Cases such as I-l-1 and O-0 are often written identically and therefore only distinguishable in context.

The problem is much worse when characters run together. For example, the samples of handwriting given above are written cursively (or “joined-up”). If characters are printed separately, it is relatively simple to find where one character finishes and the next one begins. For cursive writing, the data must first be segmented into characters before recognition can take place. One of the most difficult tasks is to decide on appropriate segmentation points. The most commonly applied method for recognition is by matching input script with a database of previously collected “template” characters.

Good recognition performance is ultimately linked not only to the capacity of the system to extract and compare good feature sets, but also to the integration of context and knowledge in the different processing stages. Hence the current study forms the first part of a contextual recognition system. It describes a number of levels of analysis in a handwriting recognition system. These begin with the application of orthographic information. Letters do not combine arbitrarily to form words, so letter string combinations produced by a pattern recogniser can be checked for acceptability. Non-occurring sequences can be discarded. The aim is to reduce the pattern level ambiguity until only allowable words remain. This can be achieved by comparing the candidate letter strings with a list of n-grams[1]. N-grams are legal letter string combinations collected from a dictionary or a corpus of text.

In this way strings containing non-occurring sequences of letters will be rejected. Unfortunately the remaining strings are not always words. Candidate letter strings can be compared with a list of words instead of n-grams, and this method guarantees lexical output. Such a process is known as a lexical check. The required list of words, or lexicon, can be acquired from a standard dictionary in machine readable form. This thesis is concerned with aspects such as the storage of a lexicon for ease of searching. A number of alternative data structures are discussed and compared.

Ambiguity remains because most word positions give rise to a number of alternative candidate words. However words do not combine arbitrarily to form phrases and sentences, so higher level constraints can be applied to reduce the remaining ambiguity. For example this can be syntactic and semantic information to ensure more meaningful results, especially for running text. Other sources of information are discussed in the current study, along with details of their integration. The additional information gained from higher level knowledge must be combined in some meaningful way in order to contribute best to the recognition system.

[1] A gram is essentially a sequence of letters where n is the length of the gram. Hence bi-grams (or di-grams) are where $n = 2$, tri-grams are where $n = 3$ and so on.

It should be noted that although the system described often refers to the particular data for on-line cursive script recognition, the techniques used are equally applicable to other forms of recognition. This can be on-line or off-line recognition of hand-printed characters, or of machine-printed characters using optical character recognition. All of these situations produce character-level ambiguity which must be reduced to achieve good recognition performance.

In the future it is hoped that there will be a new type of computer interface which should mimic the conventional pen and paper interface. Such "notepad" machines are currently being introduced into the market, and it is expected that they will become an effective alternative to keyboards in applications where handwriting will prevail. Effective and reliable handwriting recognition will necessarily form an important part of this new technology.

Chapter One

Review

1.1 Introduction

Computerised document handling is now common in every activity within business life. The continuously falling price and increasing power of desk-top computers has led to their widespread availability and use. To take full advantage of the facilities offered by these systems, keyboard skills must be acquired by users who wish to interact efficiently with the machines. Objections are frequently raised by affected personnel, and methods to input documents automatically are thus highly desirable. Moreover, the introduction of notepad style computers will necessitate non-keyboard input. Two natural modes of communicating with computers are via spoken and handwritten input. For such input to be possible, the speech or handwritten data must be "recognised" by the computer and translated into digital text representation (such as ASCII characters).

The human information processing system generally has few problems with spoken or written language, even when the stimulus is noisy or ambiguous. We learn to filter out background noise when listening to a particular voice, accents provide little difficulty, and even heavy dialects can be understood with some practice. Reading takes years of learning and practice by the human child, who already has an established linguistic and cognitive system. However, the skilled human reader has few problems reading text in many different fonts, including new ones, or with handwritten material, including unfamiliar handwriting. Badly formed characters and even illegible words can be understood in context because human readers use their knowledge of language and the world to guide their processing. Similar problems in isolated characters or words are not so easy to read, because there is little or no surrounding context (Mitchell, 1982).

For automatic recognition of both script and speech the information physically present in the signal is not sufficient for unambiguous identification of words. Higher level knowledge can improve recognition performance by helping to choose between alternative characters. For written language this involves information about how letters combine to form words, or orthography. For spoken language phonological information is required to understand how individual speech sounds (phonemes) combine and affect each other within words and across word boundaries. Additional knowledge about how words may combine to form sentences, and about how sentences are put together to produce text will also be necessary to resolve remaining ambiguity at the lexical level. This involves information about syntax, semantics, discourse structure, pragmatics and knowledge of the world.

Present systems for the input of text into a computer mainly use keyboards. Trained keyboard users can type much faster than humans write by hand, but speech is much faster than either of these for the majority of people. Trained writers of shorthand can however transcribe speech faster than keyboard entry (Leedham, 1990). In the future it is envisaged that all three methods will be available as alternative input techniques for computers, perhaps including shorthand as a fourth. Already existing documents can be input using optical character recognition (OCR). This may be printed text or handwritten.

There are situations in which one technique will be preferable to all others, and which of these is most suitable depends on the particular situation. For example, people who are not trained in keyboard use would find either speech or script recognition systems much faster and easier to use. However in noisy or quiet environments, both keyboard and speech input may be unsuitable. For example for note-taking in lectures and seminars, amongst noisy machinery and other background conversations speech input would be difficult, and in quiet environments (for example in libraries) both speech and keyboard input are inappropriate. For security reasons it may be better to write (or use a keyboard) to avoid being overheard, and for medical doctors who have automated systems, speech is considered unsuitable in front of the patient and others in a hospital ward, and keyboard input is socially unacceptable. Patients are used to doctors writing while they are talking, but typing would be disconcerting. A script recognition system would also be more easily adapted to foreign languages (if the same alphabet is used), than would a speech recognition system which would require completely different information about the phonology of other languages. Interestingly, there have been rapid developments in the recognition of oriental languages (e.g. Japanese, Chinese and Korean). The large character set used by such languages (a few thousand) makes keyboard use unwieldy, and the uniformity of writing style makes recognition relatively simple (Tappert et al, 1990).

Handwritten data is input to a computer via an electronic tablet which accurately captures x,y coordinate information of pen-tip movement. Such tablets first became available in the late 1950's and precipitated considerable activity in on-line handwriting recognition. The recent advances in technology have combined tablets and flat displays to bring input and output together on the same surface, known as electronic paper. This can simply be described as a flat panel display that is written on with a stylus (pen). As the stylus moves in contact with the display, pixels are illuminated to leave a trail of "electronic ink" on the writing surface. Tablets are now much more accurate than before, and computers are more compact and powerful. Combined with better recognition algorithms we now see the advent of the pen-based or "notepad" computer.

This follows naturally from small "laptop" personal computers which are still keyboard-based. See Higgins and Ford (1991b) for an excellent review of pen-driven interfaces.

For the reasons explained above, the interest in script recognition systems has been expanding in recent years. The automatic recognition of handwritten words and digits is an important but difficult task that now has a large literature (see Harmon, 1972; Tappert et al, 1990). In some systems the input script is restricted to upper case unconnected letters, or lower case unconnected characters. Systems coping with cursive script are fewer and on the whole less accurate than their unconnected character counterparts because recognition of cursive script is much more difficult. Individual letters are subject to more variation in cursive writing, depending upon the letters preceding and following it (Eldridge et al, 1984; Wing, 1979), and because it is not clear where one letter finishes and the next begins, a stage of segmentation is usually employed, which introduces more ambiguity.

Tappert (et al, 1990) cites eleven experimental handprinting recognition systems, and twenty-one commercial handprinting recognition systems (sixteen use opaque tablets, and five use integrated tablet/LCD devices). Tappert also lists four experimental cursive script recognition systems, although literature suggests that there are others which he does not mention, or that have appeared since his paper was published. Most existing recognition systems concentrate on the pattern recognition process, and have not utilised the substantial amounts of available context. Typically context is used only in the form of spelling correction information to compensate for errors in character recognition.

The goal of pattern recognition is to map the set of initial representations (data) into a set of interpretations (names). For handwriting this means matching sequences of x,y coordinates with the characters they represent. Pattern recognisers are often quoted as producing correct results approximately 90% of the time. Some systems are better than this, but given the ambiguous nature of the input, recognition results are never going to reach 100% even for trained, writer-dependent systems. Most writers have a range of shapes for a given letter depending upon the particular letter context in which it occurs. All writers make occasional slips when letters are mis-formed, omitted entirely and so on (Wing 1979; Ellis, 1979). The only way for correct recognition to be achieved in such situations is by the use of additional contextual information.

Systems that have employed more contextual information are text recognition systems (OCR) which have also made use of error correction techniques. Examples of these applications (speech, script and text recognition systems) will be discussed in the following sections. Sources of linguistic information required by a recognition system will also be investigated.

1.2 Speech recognition

Research into speech recognition has traditionally taken priority over research into script recognition. At a first glance, it may be thought that methods for script and speech recognition would be the same, because both are attempting to process natural language. However where written language is letter-based, spoken language is phoneme-based (Crystal, 1987). A phoneme is the smallest unit of speech sound, and the correspondence between phonemes and letters is not a direct one-to-one relationship. Pattern recognition for speech recognition presents different problems from that for script recognition, one of the main problems for speech is identifying word boundaries from a continuous sound signal (Fallside and Woods, 1985).

The majority of past and current speech recognition systems have used whole word based methods to identify what has been spoken and have been very restricted. Because of this, successful systems tend to be speaker dependent, allowing only isolated words, and have small vocabularies (Holmes, 1988). The aim would ultimately be for a speaker independent system, allowing continuous speech, with a large vocabulary, and this has only recently become a practical possibility as speech recognition methods have shifted to phoneme-based systems, using transitional probabilities and methods such as hidden Markov models to allow sequences of phonemes found in English (or whatever language is being considered) and reject others. The problem is still that the location of beginnings and ends of words are never certain. Consequently most of the speech recognition research has concentrated on the pattern recognition level, and references to the use of higher level information of the language are mainly theoretical.

In fact a large vocabulary system accepting natural language is simpler to obtain for written language than for spoken, because the recognition units (i.e. letters) appear to be easier to identify, and identifying word breaks is not such a problem. Letter databases needed for matching are smaller, but give a much larger number of potential candidates. Technological advances over recent years have made script input systems more viable, and the improvements in script recognition are greater than the equivalent for speech.

1.3 Script recognition

Handwriting recognition is performed either on-line or off-line. The former means that the machine recognises the writing while the user writes, and is also known as dynamic or real-time recognition, although the recognition will lag behind the writer to a certain extent. This may be one or two characters in most commercial systems (Tappert et al, 1990), but need only be fast enough to keep up with the writing. On-line handwriting recognition requires some kind of digitising data tablet to capture the script as it is written. These typically have resolutions of 200 points per inch, a sampling rate of 100 points per second, and an indication of pen-down.

For recognition systems which use electronic paper (tablet and display combined) there remains a user-interface problem of exactly when the display of script should be removed for the recognised text to appear. This process should be unobtrusive to the user, but it is not clear how best this should be done. If each word is displayed as soon as it is recognised, the display will be changing disconcertingly whilst the user is writing the next word. The user may wish to see the previous few words, or the current sentence, or perhaps more than one sentence of handwriting before the display changes.

In contrast, off-line handwriting recognition is performed after the writing is completed, and involves aspects of computer vision via an optical scanner to convert the image of the writing into a bit pattern. This is similar to optical character recognition (OCR) which has concentrated mostly on machine-printed characters, although there has been some effort on handwriting as well. The use of higher-level techniques in OCR are similar to those in dynamic script recognition, but ambiguity is generally less at the letter level, although merged or overlapping characters present more of a problem.

The advantage of on-line data capture is that temporal or dynamic information about the handwriting can also be collected. This may be information such as the number of strokes used, the order in which the strokes are written, the direction of the writing, and even the speed of the writing for each stroke. The use of this kind of information by handwriting recognisers can improve their accuracy. Little learning (on the part of the user) is needed to use an on-line recognition system because it seems just like real pen and paper. However the main disadvantage seems to be that current digitising tablets are not quite so comfortable and natural to use.

Data collected by this method are usually in the form of x,y co-ordinates, and are passed into a pattern recogniser, which will aim to produce characters, or a number of candidate characters as output, usually by matching against a database. Input is coded and a decision is made about the possible characters it represents (Frishkopf and Harmon, 1961; Munson, 1968; Tappert, 1982; Wright, 1989). Pattern recognition techniques that have most often been applied to script include spatial analysis methods (where strokes are coded by a numbering system on a grid) which are easy to implement but are only suitable for unconnected characters, and will be user dependent in order to keep the database of character codings small and accuracy high. Topological feature based methods detect and code straight lines and the orientation of strokes. They also identify curves, pen-ups, dots and cross strokes, and can be applied at the single character level and for complete word analysis. Vector chain coding techniques (e.g. Freeman, 1961), which code six or eight directions of strokes (see chapter 2 for more details) are also often used for pattern recognition.

Even if the pattern recognition stage is highly reliable, there will be some instances of error and ambiguity, especially if the script is untidy, illegible, or if the slope of writing is extreme. In a word like *pack* for example, it is unlikely that a pattern recogniser, no matter how accurate, will be able to say if the correct sequence of letters is *pack, paclc,* or *padc*. Consider also the word *minimum* and the candidate letters a recogniser might produce for it. Each curve in the script could form part of many different letters (e.g. *n, u, m, w, v* and *i*), and the number of alternative strings these letters would combine to form will be very large. Therefore some form of subsequent processing (often called post-processing) is necessary to improve recognition rates.

The most common type of contextual information to be implemented in script recognition is that of the surrounding letters (Ehrich and Koehler, 1975; Goshtasby and Ehrich, 1988), or orthography. Orthographic information can be taken advantage of to improve recognition. Letters do not combine arbitrarily to form words. For example, the number of four letter combinations of the alphabet is $26^4 = 456,976$. The number of four letter words, taken from a dictionary of about 14,000 words, is 1,323. This is about 0.3% of the total number of possible combinations. For a pattern recognition system which outputs alternative candidates for each letter position in a word, this fact can be exploited to rule out letter combinations which are not allowable in English. There are various ways in which this can be done. For example, a common method in the script recognition literature has been to use n-grams (Riseman and Hanson, 1974; Ehrich and Koehler, 1975; Higgins and Whitrow, 1984; Whitrow and Higgins, 1987), or letter transitional probabilities such as the Viterbi algorithm (Hull, Srihari and Choudhari, 1983), or Markov modelling (Raviv, 1967; Neuhoff, 1975; Farag, 1979) to rule out illegal strings of letters, or to select the most likely letter combinations. However, this does not exploit redundancy to the full. In the case of n-grams, there are 13,166 legal quad-grams of the total possible combinations (from the above 14,000 word dictionary), a reduction to 3% compared to 0.3% for words. If only the most likely combinations are used, from estimated probabilities, there will also be a significant probability of error. This is because if the most likely candidate is not the correct one, this method will give an uncorrectable error which will be propagated if transitional probabilities are combined. Even if positional n-grams are used (Shinghal, Rosenberg and Toussaint, 1978), they will not be as successful as the words themselves (Wells et al, 1990a; Ford and Higgins, 1990).

1.3.1 Systems for handwriting recognition

Script recognition systems are traditionally most heavily concerned with the problem of pattern recognition. For a comprehensive review of pattern recognition methods, see Tappert, Suen and Wakahara (Tappert et al, 1990). A range of methods have been applied to the recognition of printed characters, or unconnected handwriting, especially for oriental alphabets as well as for English. There are a number of commercial systems currently available on the market which recognise handprint, some of which are quite successful for careful writing within specified boxes (quoted recognition rates of up to 95%), and very successful if the system has been trained in a writer-dependent mode. Further details of printed character recognition systems are not included here in order to concentrate on cursive handwriting systems.

The recognition of cursive script is much more difficult because several characters can be written with a single stroke. Consequently there have been fewer serious efforts towards obtaining effective solutions. Moreover such efforts have been restricted to lower case English, and have concentrated on two main approaches. Firstly there is the whole word approach, whereby shape and pattern recognition procedures attempt to match directly with complete words. The second and more common approach involves breaking or segmenting each cursive word into parts.

It should be noted that it is often difficult to compare different recognition systems as they are reported because differences in input data and equipment can affect performance. Details of the exact nature of the writing tested, such as the size of the script, the quality of script, the speed it was written, the pen type, tablet resolution and so on are often sadly lacking in research papers. The hardware used for data collection can also give rise to differences in recognition performance. Standards for describing such information should be defined so that systems can be more meaningfully assessed and compared.

1.3.1.1 Recognition of whole words

Studies of whole word recognition attempt to recognise a word as a single entity by examining certain global features of the word. Such approaches have been applied to cursive words of English (Frishkopf and Harmon 1961; Harmon, 1962b; Earnest, 1962; Farag, 1979; Brown and Ganapathy, 1980), but to achieve any degree of accuracy the vocabulary which can be recognised is very small. Pattern recognition procedures used for this approach are mostly identical or similar to those applied to separate characters. For example the system developed by Earnest (1962) extracts a few primitive measures such as differential vertical extent, closed loops and a count of horizontal centreline crossings. These features are matched against a stored dictionary (approximately 10,000 words) of similar word encodings. Five different subjects were asked to write 100 test words selected at random from the dictionary. A CRT light-pen was used for input of script, and possible answers were lists of about 20 words. The correct word was included about 60% of the time, although it was found in the first position of the list of alternatives only 18% of the time.

Another study (Farag, 1979) used elastic matching with eight direction codes to establish accurate recognition, unfortunately for only ten cursively written key words. More recently, O'Hair and Kabrisky (1991) have presented a method for recognising whole words as single symbols, although their system is applied to the off-line recognition of printed text. The technique uses Fourier transforms, and reportedly correctly recognises at least 5000 words using 24 various font styles, including cursive ones.

Current thinking is that the whole word recognition approach is not viable for the more general problem with large vocabularies (Tappert et al, 1990). However it may be very effective in situations where the number of words likely to be written is severely restricted.

1.3.1.2 Segmentation methods

More commonly applied recognition techniques split cursive words into a number of smaller parts. This process is known as segmentation, and methods vary considerably resulting in individual strokes, characters, or some unit which is usually less than a character. Sequences of strokes or stroke segments are used to identify characters. Recognition of these resulting segments utilise techniques similar to those used in systems for unconnected characters.

One early study of connected handwriting (Mermelstein and Eden, 1964) segmented the input script into upstrokes and downstrokes by segmenting at points of minimum velocity. Practically unique letter specification was obtained from only the downstrokes of the writing, and there is other evidence to suggest that most of the information in cursive writing is in the downward portions of the writing. The upward portions of writing serve mainly as ligatures to join characters together. Other studies have analysed cursive words on a letter-by-letter basis (Frishkopf and Harmon, 1961; Harmon 1962a) where segmentation was based on an estimate of letter width, and letters identified by reference to stored features such as cusps, closures, retrograde strokes and so on.

Elastic curve matching has also been applied to cursive script recognition. Letter segmentation and recognition were effectively combined into a single operation by Tappert (Tappert, 1982) who matched segments against stored letter prototypes and evaluated recognition at all possible segmentations. Later Tappert went on to use a loose segmentation method to cut script into sub-strokes in regions identified as possible ligatures (Tappert, 1988). This was carried out on-line, although a similar method has been employed in an off-line study (Bozinovic and Srihari, 1989).

Much research effort is still being expended into different segmentation methods and a number of new techniques have recently been reported (Wright, 1989; Kadirkamanathan and Rayner, 1990; Teulings et al, 1990; Higgins and Ford, 1991a). Wright's system for cursive script recognition has efficient low-level processing but relies on a dictionary and higher level linguistic processing. The system is quoted as correctly recognising 94% of characters of a data set of 112 people's writing. Another approach is investigating a stochastic method for segmentation inspired by simulated annealing (Teulings and Schomaker, 1991). The use of neural networks for handwriting recognition is also currently being researched (Schomaker and Teulings, 1990; Morasso and Pagliano, 1991; Skrzypek et al, 1991).

1.3.1.3 Discussion

The highest level of information used in processing to date has been some form of lexical look-up. The most common use of context has been some measure of how letters may legally combine to form words. There is one major difference between the various methods for exemplifying this information. This is whether or not the information is represented statistically. Information about how letters combine is extracted from some source text or lexicon. This information may then be represented statistically; in terms of the frequency of occurrence of combinations of letters, or in terms of the probability that some letter is preceded by some combination of a number of other letters (transitional probabilities); or non-statistically in terms of whether or not some combination of letters occurs in the source. This difference leads to different methods of operation and potential for success.

Examples of different systems will illustrate these points. It is difficult to compare the success of different systems, for reasons including the following:

- differences in performance may be due to changes in technology;
- coding methods may be constrained by methods of input and by the amount of main memory available to a system;
- the vocabulary of recognisable input differs widely;
- constraints imposed on input may differ;
- the number of writers for whom the system will operate effectively varies;
- the amount of training a system has received, both absolutely and in terms of the number of writers will influence its success;
- systems are rarely tested in a comparable way, for example in terms of actual input, number of writers, size of vocabulary and so on;
- apart from estimates of computational efficiency, success can only be assessed by considering how they cope with language in a principled way;
- none of the systems to be reported has been tested in a way which would allow evaluation of their linguistic effectiveness.

1.3.2 The use of context to aid handwriting recognition

Early systems were highly constrained by technology. Advances in this area have led to the development of more realistic script recognition systems. An early system that produced some success was that of Sayre (1973). The input to Sayre's system was provided by an earlier project (Frishkopf and Harmon, 1961). It was received in the form of x,y co-ordinates of discrete points. This data comprised alphabets and handwritten phrases of unsegmented form from various writers. The input was cursive, lower case script.

The system stored the form any letter may take within the sample, and indicated all the different letters of which a segment of any description may be a part. The input strings were segmented in the following way. The input cursive line data was first filled in to a consistent thickness. This was because coding of the data was in terms of its concentration, so that consistency was necessary to avoid spurious calculations. Baselines were then established against which letter elements could be categorised. These baselines divided the data into lower, middle and upper horizontal regions. Vector changes relative to these regions were established and categorised according to type and position. In this way, possible segmentation points were established and types of segment identified. These segments were then checked against the reference database for possible identification checking both letter and letter string patterns. Alternative identifications were produced which were then reduced by post-processing techniques. Sayre intentionally did not include time and movement information in his encoding scheme. He argued that since human readers do not use such information it is not necessary for successful recognition. He did not place any constraints on input.

Sayre used two types of context to aid recognition. One involved using whole word shape to help determine segment positions and to allow for letter shape variations within words. This does not require any higher level knowledge. The second form of context does. Sayre used di-gram and tri-gram statistics to rule out implausible letter string combinations. It was intended initially to use statistics reflecting the frequency of occurrence of letter combinations, together with letter probabilities from the pattern recogniser. Sayre decided against this method. This was because, he discovered, that when there is a borderline choice between two letters based on the output from the recogniser, the frequency statistics will favour one of the alternatives and permanently remove the others from consideration, even if they are correct. For example, if the pattern recogniser produces ‘a’ and ‘o’ with equal probability, the choice between them will be based on frequency statistics. The input word *far* would never be correctly recognised, since the sequence ‘fo’ is approximately three times more common in English than the di-gram ‘fa’. In other cases, statistics would not be able to decide. ‘po’ and ‘pa’ are of almost equal probability, so that statistics could never decide between *pod* and *pad*, for example. Even when information from the pattern recogniser favours one answer, bias from the statistics may lead to an incorrect result.

To overcome this problem, Sayre did not use probability information. Rather he ruled out only very infrequent letter combinations, based on di-gram and tri-gram statistics. This method does not always yield a unique result. However, it is much more likely to produce the correct answer, even if this answer is one of several alternatives. Sayre states that in most cases, there are less than four resulting alternative letter strings, and usually one. He suggests that when there are alternatives, these could be reduced in several ways: by listing permissible combinations of words in phrases; by listing allowable input sentences; by ruling out infrequent words; and by using grammatical criteria. These suggestions appeal to higher level knowledge or suggest a fudge which would severely limit the scope of the system. If the system does not produce a result, letters may be changed. If nothing works, the input is considered illegible.

Sayre's paper illustrates the main problems with the statistical approach. Some other examples of this approach are given, and then some systems are considered which extend non-statistical methods.

Ehrich and Koehler (1975) also produced a script recognition system which did not use real time information in its coding scheme, although they did use some sequence information. Their input was via a graphics tablet. Spatial data points were collected. Data was compressed to reduce noise. This was done by only sampling at fixed distances between points. Size and slant constraints were employed, and writers were asked to use only particular forms of letters. Horizontal baselines were established. Letter features were extracted with respect to these, and compared to reference sets of features. The pattern recogniser generated sets of letters which could have occurred. Some substitution sets were stored to anticipate common confusions. From this information, alternative letter strings were generated by combining the alternative letters.

Some of these strings could be ruled out because the linking information between the letters contradicted them. Others could be ruled out because they were illegal strings by reference to binary di-grams. Binary di-grams[2] give non-statistical information about letter co-occurrence. The dictionary used by this system consisted of 300 seven letter words. Only letter strings which appeared in this dictionary were considered to be correct. While this system may produce more than one letter string as a result, this did not happen often. When it did, various nearest match strategies could be used to select among them.

A popular method for comparing coded input against reference prototypes is that of elastic matching (also known as dynamic programming). This method has had some

[2] A binary di-gram d(i,j) is a 26×26 binary matrix that corresponds to letter positions i and j such that $i \neq j$, in a dictionary of words of fixed length, l. So d(k,r) = 1 if and only if some word in the dictionary contains the character a(k) in position i and a(r) in position j, where a is the alphabet. All alternatives are zero. Thus these di-grams record their occurrence in a lexical source with position information.

success in speech recognition and its application to script recognition is described by Tappert (1982; see also Wong and Fallside, 1985). From input data via a graphics tablet, a sequence of parameter vectors are produced to represent a word. These parameter vectors contain estimated strings of letters. Operating on a word at a time using letter prototypes and allowing any letter to follow any letter, elastic matching is used to find the prototype sequence which best fits the input word vector. This technique is insensitive to minor perturbations of input letter shapes. Explicit letter segmentation is not performed. Rather, elastic matching permits evaluation of all possible segmentations and simultaneously obtains some optimised combination of segmentation and recognition. Basically, a graph of all possible letter-segment combinations is set up and an optimum path through the graph is calculated. This pathfinding is augmented by limiting segmentation shapes and by using di-gram statistics. Segmentation was inhibited at points which were definitely not segmentation points; with the aim of avoiding segmentation errors and speeding computation since this limits the possibilities.

Di-gram frequency information in the form of di-gram weights (penalties) derived from di-gram transition probabilities were employed. On making the transition from one prototype to another a di-gram weight corresponding to the appropriate letter pair was used in the optimisation metric, to influence the path chosen. In this way, the highest transition probability would have more weight in the optimisation procedure. The transition probabilities used were based on a text containing 10,000 letters, with zero probabilities converted to a small value to enable computation. This number of letters is approximately 1,700 words and is a relatively small sample set. Tappert gives no other information about the source text used.

This use of probability information is susceptible to the problems of using statistical information outlined earlier and described by Sayre. No lexical checking was carried out.

Burr (1983) also used a dynamic programming technique to recognise handwritten script. He similarly used whole word shape constraints to aid recognition. Input was lower case handwritten print via a graphics tablet. Reference shapes were stored for each user, who input the 26 letters of the alphabet for this purpose. A vector of 26 numbers was computed for each unknown letter. Each number represented the difference in shape between the unknown and each reference letter. For an unknown word of n characters, an $n \times 26$ shape matrix would result. This shape matrix was then compared to a stored dictionary of words to find the word most consistent with the shape information. This was based on correlation between the shape matrix and the shapes of words. The words were stored in various sub-dictionaries containing vocabulary words and suffixes. The aim was to extend this approach to include prefixes [3].

Burr's method of dictionary search was a complicated one. The results presented are not detailed enough to assess the success of the method. An attempt was made to take advantage of the morphological structure of words to cut down on storage space and search time. However, it is not clear from the information presented by Burr how well he has been able to cope with the irregular morphological structure of English. The dictionary was partitioned by word length. Words were stored sequentially as their

[3] Burr states that suffixed forms of words can be derived by rule. This is not strictly true, since there are many exception forms in English, and there are many pseudo-affixed words. These points are not discussed.

equivalent ASCII code. To minimise memory costs, words were stored as their roots with suffixes stored in various sub-dictionaries. These sub-dictionaries are stored on disk and are read in as required. An input word is initially tested for the presence of a suffix. If good evidence for one is found, then the remainder of the word is tested for the root. The rationale for this approach is that there are few suffixes relative to words. The shape matrices are then tested against those of the appropriate sub-dictionaries to find the best match, either between the unknown word and a dictionary entry, or a composite for suffixed forms. The system allows for whole word matches as well as composite matches, so that if the suffix analysis does not produce a result, the whole word match is chosen.

This method seems over-complicated for lexical checking of pattern recognition output. This is because suffixation in English is not a regular, rule-based system. Burr does not give sufficient detail to enable a reasonable evaluation of its effectiveness.

A cursive script recognition system which incorporates efficient lexical look-up and is described in sufficient detail is that of Srihari and Bozinovic (1987). This system would appear to be the most comprehensive to date, and is reasonably successful. The only constraint on writer input was that some care was taken over legibility. Input was off-line. After an initial normalisation procedure which standardised the data, the data was segmented and coded to produce possible letters and letter strings. These were checked against a lexicon as they were produced so that unacceptable letter strings could be ruled out as early as possible. The possible letters and letter strings had associated probabilities, based on properties of the pattern recognition system. These probabilities were not used to rule out alternatives, rather to rank them in terms of their likelihood. This rating system starts with the likelihoods of the beginnings of letter strings. These beginnings, or prefixes, are checked against a lexicon, which is constructed in such a way that prefixes which will not produce legal words can be identified and pruned from the list. Following pruning, the prefix with the consequent highest likelihood is expanded, producing a new prefix. This is then sent for lexical checking and the ratings re-computed. This continues iteratively over the possible candidates so that letter strings approach the length of the input string, illegal possibilities are ruled out and likelihoods re-computed to produce ordered resultant candidate words. This process usually produces the correct result. Occasionally the correct word would not be the highest rated.

The method of lexical representation used was one which allowed efficient representation and search. The lexicon was represented as a trie[4]. See figure 1.1, and Srihari, Hull and Choudhari (1983) for further details of this trie.

The advantages of this representation of the lexicon are that :

(i) its structure naturally fits the search algorithm and hypothesis expanding rules;

(ii) it has convenient knowledge representation by storing various information in its nodes;

(iii) it has storage space savings due to numerous identical initial word segments.

[4] This is a tree-shaped data structure, each node of which is an ordered pair (l,e) where *l* is a letter and *e* is a boolean flag for the end of a word. The root of the trie is the only exception in that its initial entry is the empty string.

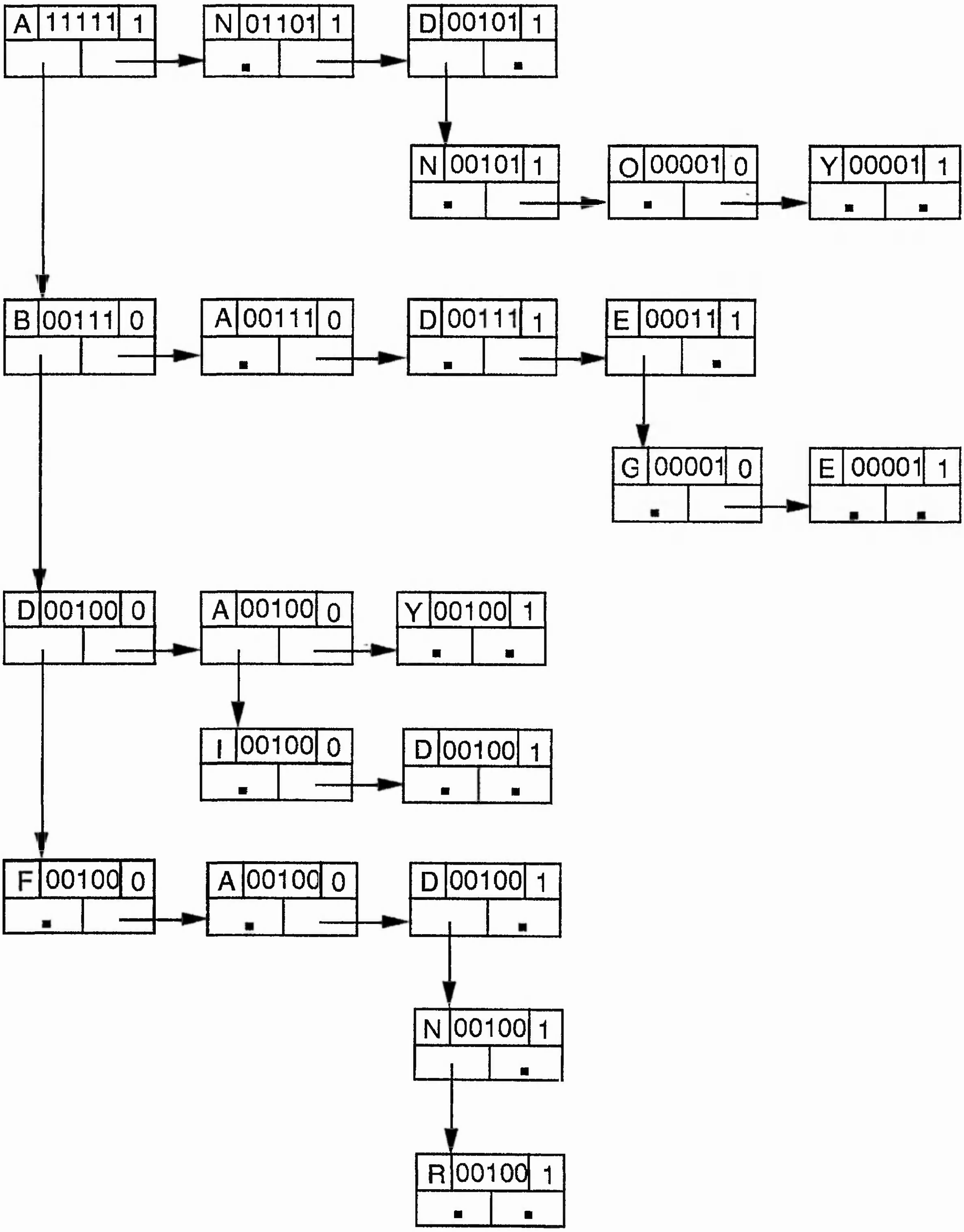


Figure 1.1 — Trie structure used by Srihari et al (1983) for the lexicon *a, an, and, ann, annoy, bad, bade, badge, day, did, fad, fan, far.*

Lexical checking here has the advantage that knowledge about how letters may legally combine by position is represented succinctly, and that any output from it will be of real words. Srihari and Bozinovic (1987) tried the system with two different sized lexicons. One contained 710 words, the other 7,800 words. Performance with the larger lexicon showed some deterioration. It is not clear in what way performance was considered to have deteriorated. This effect may have been reversed if a larger set of test words had been used.

1.3.3 Summary and conclusions

A representative selection of the literature on script recognition has illustrated the main approaches to the problem, and the main problems to be resolved. Systems have used information about the orthography of words to select output from a pattern recogniser in various ways. Statistical information about n-grams has been applied to the output to select a single most likely letter string result. The problems with this approach are that when the correct output is not the most likely, an error is bound to occur. Also, this approach does not guarantee a word as output, even though parts of the letter string will contain legal combinations of letters. Another approach has used n-gram information in a non-statistical way. In these cases, letter string combinations are only ruled out if they do not occur in the sample source. This approach may lead to the output of more than one letter string, and again does not guarantee that these letter strings will be words.

A third approach was considered to exhibit the least disadvantages. This involved some form of lexical checking. Here at least the output is guaranteed to be an acceptable word, although more than one may result. While methods of choosing amongst alternatives when they occur (which is not often) may be based upon statistical measures of likelihood (such as word frequency; see Kucera and Francis, 1967) the problem remains that the correct word may be rejected.

It is important to note that the source of information, either for the extraction of n-grams or for the creation of a lexicon, requires careful consideration. The source should reflect the nature of the to-be processed material, in both its statistical make-up and content. Otherwise candidates which are acceptable may be rejected because they do not occur in the source, and alternatives which are not acceptable may be accepted because they appear in the source and provide a good fit, even though they are not characteristic of the material being processed. If a lexicon contains many words which occur rarely or never in the input material, this will introduce costs of storage and search, although if a lexicon is too small, words will be rejected. The selection of source material will limit the information which can be processed, and may introduce unnecessary overheads if it does not fit material to be processed well (Sampson, 1989).

Most authors suggest that ambiguity remaining after initial lexical processing would be reduced by applying higher level context, such as syntax and semantics. None to date have attempted to do this. It was argued earlier that only the use of higher level knowledge will enable any additional success. Language has too much inherent ambiguity to enable solely bottom-up processing. While this is a notoriously difficult problem, it was argued that some progress can be made.

1.4 Text recognition

The problems for text recognition are different from those of script recognition. Text recognition refers to the recognition of machine printed characters by scanning documents. This type of recognition is by definition off-line, and is often called optical character recognition (OCR). It is considered briefly here because researchers have concentrated more on the use of context, although none have ventured beyond the lexical level. Many of the methods involve error correction techniques to resolve recognition errors. This is also a problem for script recognition, although it has rarely been considered.

Three approaches are generally considered for the application of contextual information in the field of text recognition. These are Markov, dictionary and hybrid methods (combinations of the other two methods).

The Markov methods represent the bottom-up approach, and they model English text as a Markov process which allows transition probabilities to be assigned to various letter combinations or n-grams. A dictionary of legal words is usually used to calculate the probabilities, but they can be calculated dynamically from the text being processed. Generally, as the order of the n-gram and the number of constraints is increased the accuracy of the results will be improved, but at the expense of computational resources. All permutations of transitional probabilities arising out of the character recognition are calculated to give an associated probability for a given string. The requirement is to maximise this probability, and thus obtain the most likely string. A number of methods have been proposed to achieve this aim, including the Viterbi algorithm (Viterbi, 1967; Neuhoff, 1975; Shinghal and Toussaint, 1979a), probabilistic relaxation (Goshtasby and Ehrich, 1988), and the Recursive Bayes algorithm (Raviv, 1967; Shinghal, Rosenberg and Toussaint, 1978). However it has been claimed that Markov probabilities do not give significant reductions in word error rate, and can even lead to an increased error rate (Riseman and Ehrich, 1971; Riseman and Hanson, 1974; Hanson, Riseman and Fisher, 1976).

Dictionary look-up techniques represent the top-down approach, and involve verifying the input word by matching it with a dictionary word. In the simplest form the word will only be verified if it exists in the dictionary. More complex techniques have been developed for approximate string matching, allowing for possible spelling or recognition errors in the input word (Hall and Dowling, 1980; Kashyap and Oommen, 1984).

The advantage of using hybrid methods is that high-order Markov dependencies do not have to be introduced because accuracy is ensured by using the dictionary. The computational complexity of the problem is therefore reduced without sacrificing performance. Examples of the application of some hybrid methods are discussed in the following section.

1.4.1 Hybrid methods

Shinghal and Toussaint (1979b) used a combined bottom-up and top-down approach to using context in text recognition. This paper exemplifies the main bottom-up approach used in this field, and also uses higher-level context. They point out that bottom-up statistical methods are efficient from a computational point of view, but exhibit poor error correcting capabilities. Dictionary look-up methods give impressive error correction but require much greater storage and computation. They develop a combination of these two approaches which combine the advantages while minimising the disadvantages.

The knowledge of the statistical structure of English is used in conjunction with a Modified Viterbi Algorithm to obtain an optimal letter string from a number of alternatives. The Viterbi algorithm combines information about the probability of a letter being correct given knowledge of the performance of an input device, with transitional probabilities to select the most likely letter string. The algorithm is modified, since only some set of the higher probability letters are selected as candidates to be chosen between, instead of choosing between all possible candidates. Note that this method is open to the criticisms made of statistical methods above. Even if the higher probability strings are included in the computation, as opposed to just the highest, there remains a built in possibility of error.

Shinghal and Toussaint estimated their transitional probabilities from a corpus of English text containing 531,445 words. Uni-gram probabilities were used. Shinghal, Rosenberg and Toussaint (1978) compared the use of uni-gram and di-gram probabilities both position dependent and independent. They found that performance improved as the amount of context increased, so that word position dependent n-gram probabilities gave better performance. Beyond a certain point however, the amount of improvement tailed off with increase in context. Longer grams have greater storage requirements too, so there is a trade-off between performance and storage.

The dictionary look-up algorithm was based on that of Bledsoe and Browning (1966). Input words were coded by a feature vector sequence. The words in the dictionary were also coded in this way. Input feature vectors were compared to those of the dictionary words, and the word with the best match was chosen as the recognised word. The dictionary contained 11,603 words. The dictionary method gives much greater error correction performance, with greater storage requirements and computational cost.

The combined algorithm was called the Predictor-Corrector Algorithm (PCA). This combined the two approaches to reduce the computational costs of the dictionary algorithm while maintaining its advantages. In this case, the dictionary was partitioned by word length and, for any length, was sorted by the vector sequences of the words in descending order. In this case, only a fraction of the words from any sub-list need to be searched to find the best match. The number of words to be searched is an heuristic decided upon by the user.

The PCA recognises a word using the modified Viterbi algorithm. The result is then checked in the dictionary. This means that fewer words need to be checked in the dictionary. If the word exists in the dictionary it is taken to be the answer. Otherwise, the vector scores of the words in the neighbourhood of the result are calculated and the word with the closest score is taken as the answer. This method assumes that no two words in the dictionary have the same length and the same vector score. So far this assumption has not been violated.

Hull, Srihari and Choudhari (1983) compared performance of two types of bottom-up and top-down (or hybrid) algorithms. One was the PCA as just discussed. The other was an algorithm which integrated bottom-up and top-down knowledge sources (Srihari, Hull and Choudhari, 1983). This second method checked letter strings in a lexicon while selection was made amongst the possible alternative letter strings (see Srihari and Bozinovic, 1987, discussed earlier). This allowed illegal letter strings to be ruled out at an early stage. Results showed that this algorithm required less time and memory than the PCA. This was partly because of the structure of the lexicon used which enabled efficient storage and search, and cut off non-productive alternatives at an early stage.

Srihari and his colleagues decided on their methods partly from consideration of human performance on reading tasks. Hull (1986, 1987) has taken this approach further and used an analysis of stages involved in the human reading process to motivate his choice of methods for lexical look-up and comparison. He reports 96% correct recognition for 12,600 words, and the approach is at least as successful, if not more so, as any reported so far. In his approach, selection of neighbourhoods from a lexicon for search are made on the basis of gross visual similarity. Finer comparisons are then made. While these comparisons consider visual features, and the human literature suggests that at this stage visual information is not involved (Evett and Humphreys, 1981), Hull's considerations have led him to adopt a neighbourhood approach which is also suggested by the cognitive science literature (Hull, 1986). He considers that the use of syntax and semantics will be necessary for any further improvement.

Text recognition has the same remaining problem as handwriting recognition. That is, once candidate words have been selected, frequently there is more than one allowable candidate. While it seems that there may often be only one candidate, especially for longer words, there will be more than one on a significant number of occasions. The only way in which this ambiguity can be resolved is by appealing to higher levels of information such as syntax, semantics, pragmatics and general knowledge. The grammar of the language can be used to restrict word combinations because they do not combine arbitrarily to form sentences. Knowledge of how word meanings combine at the sentence level can rule out grammatically correct, but semantically implausible sentences. Knowledge of topic of discourse could also aid selection of candidates.

1.5 Sources of information

In the use of n-grams or a dictionary check to rule out letter sequences that do not appear in English, the information needed is a list of English words, or perhaps a large corpus of text from which to extract such a list and/or the list of n-grams for whichever values of *n* are necessary. Higher levels of information (for example syntax and semantics) must also have some source(s) for this information (Evett et al, 1989; Keenan and Evett, 1989). Most of this required data is found in a dictionary, and over the past few years more paper dictionaries have become available in machine-readable form. For example Webster's 7th Collegiate Dictionary (W7) (G and C Merriam and Co., 1963), Longman's Dictionary of Contemporary English (LDOCE) (Procter, 1978), Collins English Dictionary (CED) (Hanks, 1979), Oxford Advanced Learners

Dictionary of Current English (OALDCE) (Hornby, 1988), and soon the complete Oxford English Dictionary (OED) (Oxford, 1989).

Computational lexicographers have advocated the use of machine-readable dictionaries (MRDs) for many uses such as spelling correction, lexical analysis, thesaurus construction, machine translation and so on (Amsler 1984). Machine-readable dictionaries are often used for natural language understanding and processing systems (Boguraev and Briscoe, 1987; 1988). Currently available dictionaries vary in size and content (Amsler 1984, Lesk 1986). Table 1.1 compares the five MRDs listed above.

Table 1.1 — Sizes of available machine readable dictionaries

Dictionary	Mbytes	Headwords (in 1,000s)	Bytes/headword
OALDCE	6.6	21	290
W7	15.6	69	226
CED	21.3	85	251
OED	350.0	304	1,200
LDOCE	14.0	55	254

Many machine-readable dictionaries are now available to academics for research purposes, although these are usually in the form of copies of typesetting tapes, and have no accompanying software for operating on the data. Conversion from typesetting version to a usable format or database of the dictionary is a lengthy process (Weiner, 1985; Keenan, 1989; Peterson, 1982; Boguraev et al, 1987; Neff et al, 1988).

Walker (1986) has used the LDOCE (amongst other machine-readable texts) to build a text subject assessment system and a concept elaboration system by using the basic definitional information from the dictionary, involving coding of topic or domain information.

Lesk (1986) gives a study of text in definitional parts of dictionary entries and compares currently available machine-readable dictionaries with OED entries which are generally much longer and include many quotations, which he requires for his sense dis-ambiguation system. Lesk explains that “the idea is to select the correct sense of a word by counting overlapping words between the sense definitions and the definitions of the other words in nearby context ... it depends for its success on having fairly long and informative definitions. Thus it is likely to work much better with the OED than with shorter dictionaries”. He goes on to say that “lexically based computer research is growing rapidly. More and more researchers in natural language processing are investing their efforts in dictionaries and lexicons, and efforts are being made to use machine-readable dictionaries instead of constructing lexicons from scratch”.

1.6 Conclusions

The preceding sections have reviewed past and current approaches to the application of lexical context to handwriting and text recognition. These various approaches have a number of problems. For example, whole word recognition techniques are only effective for domains with restricted vocabularies. Segmentation based techniques produce alternative characters and therefore introduce more ambiguity. It has been established that contextual information is necessary in addition to a pattern recogniser. Some systems have employed letter level and word level information, in the form of n-grams or a limited word look-up.

The aim of the present system is to improve the recognition rates of a cursive script recogniser by implementing techniques using linguistic information for a large vocabulary. Any higher level processing must begin with orthography, hence this study forms the first part of a contextual system. Lexical, syntactic and semantic information needed for the system can be obtained from a machine-readable dictionary.

The following chapters expand on the practical application of reducing the ambiguity produced from a pattern recogniser. This can be achieved by using both n-gram look-up and a lexical check. Techniques for effectively implementing a lexical check by machine are discussed and compared. Other possible requirements of a script recognition system are investigated, including the combination of information from various sources and levels of analysis. The notion of what constitutes a lexical unit is also addressed. Further problems the recognition system may encounter (for example misspelled words) are considered, and possible solutions to these problems are proposed. Additional information may be required to provide such solutions, and data structures for the effective storage and retrieval of this information are investigated.

Chapter Two

Pattern Recognition

2.1 Introduction

The initial stage of a script recognition system involves a pattern recogniser. This stage does not fall within the scope of this thesis, but the interface to the recognition process does form an important part of this work. The output produced by the pattern recogniser becomes the data that must be accepted by the first stage of this system. As script is written, sequences of x,y co-ordinates are collected, coded, and matched to a database. This produces a number of character candidates for each letter position of each word of handwriting processed.

Taken together, these character candidates produce a number of letter string candidates for any one word. For short words, the number of candidate strings produced can number in the hundreds, usually number in the thousands for longer words, and can reach millions for words over about twelve letters long. The majority of these candidate strings will be nonsense strings, not English words, and therefore need to be rejected. Given some output from the pattern recogniser, all possible candidate strings should be generated. These strings can then be checked for allowability. Unacceptable strings will be rejected, and acceptable ones are stored for further processing.

2.2 *The Pattern Recogniser*

The pattern recognition process will be discussed here briefly. A sample of handwriting is collected (see figure 2.1 for example). The writing can be unconnected characters (e.g. *pack*), connected characters, also known as cursive script (e.g. *extra*), or some mixture of these two forms (e.g. *bags*). The recognition system tries to emulate a "normal" writing situation such as pen on paper. Script is written using a digitising tablet (the "paper") and a pen or stylus. To see the script that has been written, such tablets can be overlaid with ordinary paper, and the pen can have an ink-filled ballpoint centre. Alternatively, with the advances in technology, so called "electronic paper" devices are now available. Such devices combine a tablet with a computer screen, usually as an LCD display, and the pens are untethered. Ink is not needed because the LCD display shows the script as it is being written.

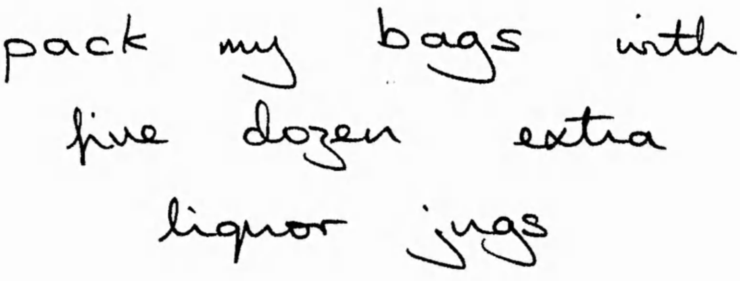


Figure 2.1 — Typical sample of handwriting

The tablet used in the initial experiments captures data by sensing (electromagnetically) the movement of the stylus across its surface. Data is collected in the form of x,y co-ordinates, along with 'pen-up' and 'pen-down' signals. The resolution of the tablet is 1000 points per inch. The sequences of co-ordinates are transformed into vector chain codes (Freeman, 1961) whereby changes of direction in the strokes of the pen are converted into a numbered code. Figure 2.2 shows the numbered directions.

The chain codes are reduced to five, or fewer codes. A chain code can of course be fewer than five codes because certain letters are formed by strokes of fewer than five directions. For example the letter *l* (one direction) or the letter *v* (two directions). Figure 2.3 shows a possible Freeman encoding of a handwritten letter *a*.

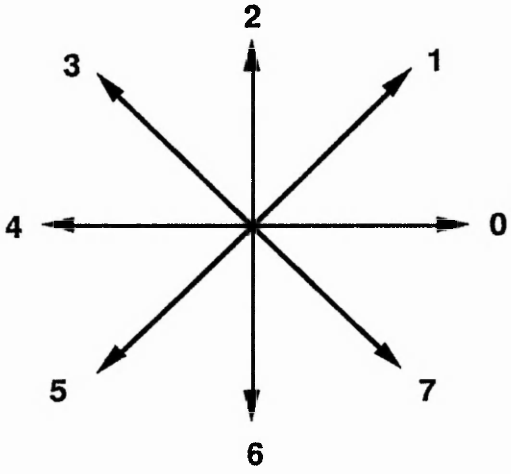


Figure 2.2 — Freeman vector numbering scheme

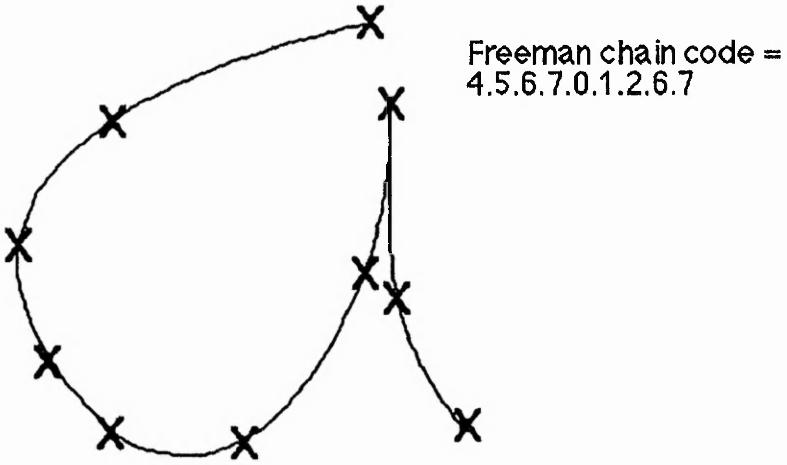


Figure 2.3 — Freeman coding of letter *a*

The pattern recognition has two phases, acquisition and recognition. Firstly samples of handwriting must be acquired to train the database, and secondly this database will be used to attempt recognition of some new samples of script. The acquisition phase can be on-going in order to add new examples of characters to the database. A selection of samples of different people's handwriting were collected and encoded. For the acquisition phase, these Freeman codes are stored in a database which covers each letter of the alphabet, for each person's handwriting. For recognition, as script is written, the Freeman code for each character is compared with the contents of the database, and those entries which match the current code are output as candidate letters, together with a measure of confidence (an integer between 0 and 100) as a guide to how close the match was.

The recogniser used by the present system gives up to six candidate letters per letter position of input script. If the script is connected (cursive), the sequence of x,y co-ordinates from the tablet must first be segmented so that comparison with the Freeman database can take place. This process of segmentation is complex and will not be discussed here, nor will other processes involved in the pattern recognition programs, as they do not fall within the scope of this thesis. See Wright (1989) for further details.

2.3 Data format

As co-ordinates are collected, encoded and recognised, matched characters must be presented to further stages of analysis in some agreed format. Alternative characters must be represented, along with alternative segmentations such as *cl* or *d, lc* or *k*. An example of handwriting such as the word *pack* shown in figure 2.4 typically produces data similar to that given in figure 2.5 from the pattern recogniser.

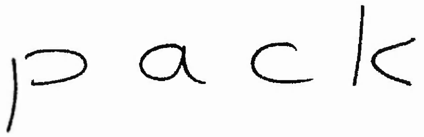


Figure 2.4 — Sample of handwritten word

```
0  :99 [1 7 ]
1 l:99 [2 ]
2 j:62 o:19 s:17 [3 ]
3 a:100 d:73 q:29 n:25 [4 8 ]
4 c:74 a:42 o:40 e:29 n:18 [5 9 ]
5 l:99 i:75 [6 ]
6 c:75 l:44 i:23 [10 ]
7 p:94 [3 ]
8 d:57 [6 ]
9 k:74 t:57 [10 ]
10  :99 []
```

Figure 2.5 — Sample data for handwritten word *pack*

The lines of data are of the form "line_number alternative_characters [destinations]", and the alternative characters are each of the form "reference_character : confidence". The line "0 :99 [...]" indicates the start of a word, and "... :99 []" indicates its finish. The numbers in square brackets give one or more destination numbers, which refer to the line numbers of the sets of candidate letters which can follow at the next character position. For example "0 :99 [1 7]" means that the following position is either line 1 or line 7. Line 7 is in turn followed by line 3 and so on. Each candidate character has an associated measure of confidence, as described earlier.

Combining the character candidates across a complete word gives a number of candidate strings. Taking only the highest priority candidate letters from the data in figure 2.5, the pattern recogniser would find six candidate strings for this word, namely *ljaclc, ljack, ljadc, paclc, pack,* and *padc*. Taking all candidate letters from the data will give many more strings, in this case a total of 1,204.

Calculating the mean of probabilities for each of the candidate letters across the length of the strings (for example *ljaclc* = (99 + 62 + 100 + 74 + 99 + 75)/6 = 84.83), and ordering the strings on the basis of these results gives table 2.1 below.

Table 2.1 — Candidate strings with probability scores

Candidate string	Probability
paclc	88.4
pack	85.5
ljaclc	84.8
ljack	81.8
padc	81.5
ljadc	78.6

These results must be improved upon in some way, so that *paclc, ljaclc* and so on are rejected, but *pack* is accepted. Although the pattern recogniser gives higher confidences to some characters, these confidences should not be relied upon too heavily. The correct character may often have a lower confidence because it was not such a good match with the Freeman database. This could be for many reasons: for example the database does not contain a good example of that particular form of letter, or the character may have been badly written. Consequently all character candidates should be considered in any further processing.

2.4 Representation of data

The data input to our system is the output from the pattern recogniser as discussed above. Each sample of script collected begins with an optional header. The header contains information about the writer (e.g. sex, age, handedness etc), the date the sample was collected, the style of writing used (e.g. lower case, upper case, cursive), and one or more sentences indicating the script which was written by the subject. Figure 2.6 shows an example file header.

```
{*
$ 040892 f 37 r
# packmybagswithfivedozenextraliquorjugsbothwizenedmenquicklyju
    dgedfour sharpvixens
! packmybagswlih-.flvc-dozc-nc-rallquor]ugs.bolh-wlzc-nc-dmc-nqulck
    ly]udgc -d[-oursharpvlns
& numonics
% lower
*}
```

Figure 2.6 — Typical header information

A graph structure was chosen as an efficient way of storing all the candidate letters, after Whitrow and Higgins (1987). The structure of a graph is such that it will allow checking and searching of pathways between letters from any point in it, in either direction. The graph also gives an economical means of representing a large amount of information in an implicit and flexible format, and provides a record of the input information should further checking become necessary at a later stage.

Each node of the graph contains a candidate letter, two integers, and two arrays of pointers. The first integer is the letter's associated confidence (taken directly from the data, e.g. *d* has a confidence of 57), and the second integer represents its rank compared to the alternative candidate letters at the same letter position (e.g. *k* has rank 1, and *t* has rank 2). The first array of pointers points to the candidate letters in the next position within the word, and the second array points to the candidate letters in the previous position. The arrays hold a variable number of pointers, and any unused ones are set to null. Figure 2.7 shows the structure of each node in the graph, and figure 2.8 represents a graph for the data for the input word *pack* as presented above (figures 2.4 and 2.5).

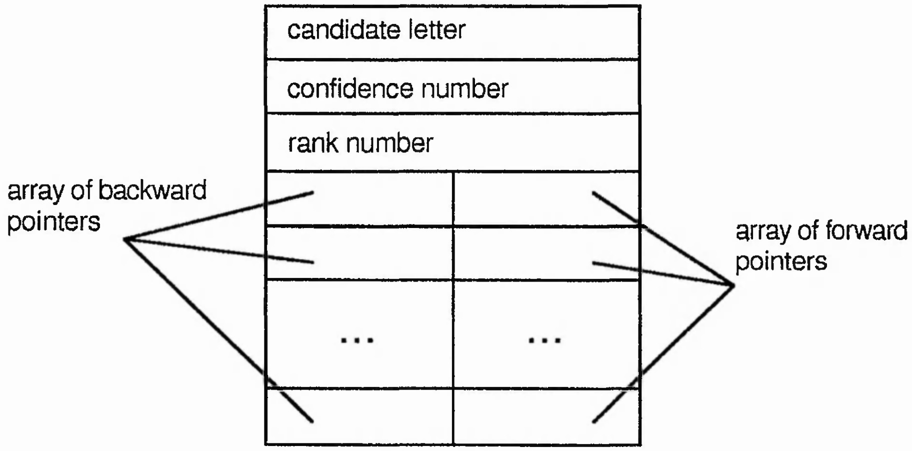


Figure 2.7 — Detail of graph node

The links in figure 2.8 represent both forward and backward pointers, but the confidences and ranks are not displayed for simplicity.

2.5 *Generation of candidate strings*

For the purposes of traversing the graph network to generate all possible candidate strings, the probability correct information is not used, and the worst possible cases are considered. To generate all candidate strings, begin from the start node of the graph (*), take the first route from it (i.e. the first of the forward pointers), and note the letter at the node reached as the first letter of the candidate string. Again take the first route from this node, noting the letter at this next node as the second letter in the candidate string. This process is repeated until the end node of the graph is reached (**), at which point a complete candidate string has been generated (e.g. from figure 2.8, this would have found *ljadc*).

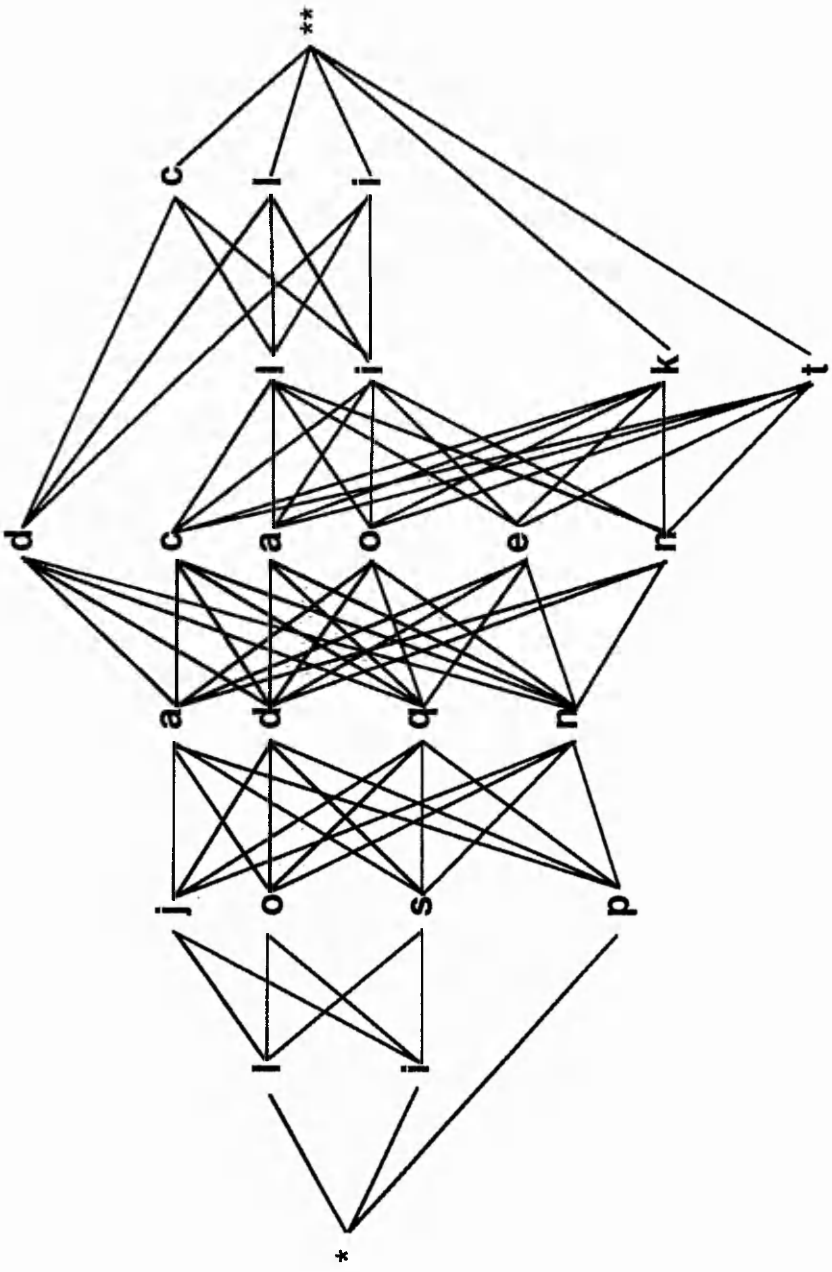


Figure 2.8 — Graph structure of candidate letters

Now follow the first backwards pointer (i.e. back to the node we have just come from, e.g. from the ‘c’ node back to the ‘d’ node), and take the next route forwards again (to the ‘l’ node, giving another complete candidate string *ljadl*). Repeat this process (through *ljadi*) until all routes from that node have been followed (now followed each of ‘c’, ‘l’ and ‘i’ routes from the ‘d’ node). Then take one step backwards again (to the ‘a’ node) and down the next route forwards (to the ‘c’ node), and so on down the first route from each node, until the end of the graph is again reached (i.e. the complete candidate string *ljaclc*). The stages of this method are repeated as described, until all possible paths are found, ending with the candidate string *pnnt*, giving 1,204 candidate strings in total i.e. *ljadc, ljadl, ljadi, ljaclc, ljacll, ljacli, ljacic, ljacil, ljacii, ..., pnnk, pnnt*. This process is complicated to describe, but when expressed computationally it becomes a neat recursive procedure. In fact due to the nature of recursion the backwards pointers are not required. See flowchart D1 in Appendix D.

The number of candidate strings generated can be huge, for example a twelve letter word with five candidate letters at each letter position gives 244,140,625 complete candidate strings. See Appendix B for further examples.

2.6 *Checking for allowable strings*

As previously discussed in chapter one it is possible to use orthographic information to rule out those candidate strings which are not allowable in English. Two techniques which have been used to check strings of characters are n-gram look-up, and a lexical check (whole word look-up). The efficacy of these two techniques will now be compared.

Using n-grams, those candidate strings which are found to exist in the list of n-grams are then stored in a list. Those which do not exist are effectively discarded because they are not stored in the list, however they could be regenerated from the graph should that become necessary at some future stage. One reason for this would be if the original script contained spelling errors, so the “correct” word could not be found as a legal English word. Thus a reduced list of candidate strings is produced.

2.6.1 The use of n-grams

A gram is essentially a sequence of letters where *n* is the length of the gram. Hence letter sequences of length two are 2-grams (bi-grams or di-grams), of length three are 3-grams (or tri-grams) and so on. The use of n-grams requires two stages: firstly the acquisition of lists of n-grams, and secondly candidate strings are checked against the lists as part of the recognition process. For the acquisition phase, the lists of n-grams were derived from a machine readable dictionary (MRD). Each dictionary entry is divided up into sequences of the required gram length. For example to compile a list of 3-grams, one and two letter entries would be ignored, and three letter entries stored as they are. Dictionary entries more than three letters long are divided into their 3-gram components. Table 2.2 shows some examples.

Table 2.2 — Example words with the 3-grams they produce

Word	3-grams
happily	hap app ppi pil ily
in	—
table	tab abl ble
zoo	zoo

The whole dictionary is processed in this way, a frequency count for each gram is collected, and each gram has a flag which is set to true if it occurs as a separate word (i.e. dictionary entry length equals required gram length). Table 2.3 illustrates some 3-gram entries.

Table 2.3 — Example 3-grams and their frequency of occurrence

Gram	Frequency	Word flag
ard	574	false
zoo	42	true
veb	1	false
two	33	true

For the recognition phase, a candidate string can either be compared with a list of grams of the same length as itself, or it can be divided into sequences of shorter grams (in the same way as described above for the acquisition phase) and compared with a list of grams of this shorter length. If all these separate grams occur then the candidate string can be said to be allowable. If at least one of the separate grams does not occur, the candidate string is not allowable, and is therefore discarded. The frequency count collected for each gram can be used as a way of deciding which grams to include if rarer items are to be excluded. They could also be used to calculate a measure of likelihood correct for accepted candidate strings.

2.6.2 *Experiments using n-grams*

The list of n-grams collected was first built into a binary tree structure for ease of searching before comparison takes place (see chapter three for further discussion of memory structures). It is interesting to note that the majority of processing time is taken up by building the tree of grams, the traversing of the graph and comparison with the tree was negligible. Two different lists were compiled, and the same data set run against them. The first list was a set of n-grams taken from a short dictionary (11,795 words), and the second was a list of n-grams taken from a larger dictionary (71,279 words). Both lists were formed from the Medical Research Council Psycholinguistic Database (Wilson, 1987) by excluding repetitions, rare usages, plurals and derived forms. Tables 2.4 to 2.6 show the results from various experiments[1].

[1] If there are 100 candidate strings: 10 allowable = 90% reduction, 1 allowable = 99% reduction. Processing time includes time taken to build list of grams into binary tree.

Table 2.4 — Results from using n-grams taken from a shorter dictionary

Gram size	Length of list	Processing time (mins:secs)	% reduction of list of candidates	No. of words tested
2	469	0:03		
3	3916	0:26		
4	13166	1:38	94.5	7
5	18934	2:46	99.3	3
6	18212	2:54		

Table 2.5 — Results from using n-grams taken from a longer dictionary

Gram size	Length of list	Processing time (mins:secs)	% reduction of list of candidates	No. of words tested
2	577	0:04		
3	6527	0:46		
4	35412	4:58	86.2	7
5	83493	13:16	98.4	3
6	107011	18:49		

Table 2.6 — Additional results from using n-grams taken from a shorter dictionary

Gram size	Length of candidate string	% reduction of list of candidates	No. of words tested
2	4	38.7	7
3	4	80.5	7
2	5	55.1	3
3	5	92.9	3
4	5	98.7	3

From tables 2.4 and 2.5 it can be seen that as gram size increases, processing time increases (because a larger tree takes longer to build), but the shorter lists are comparatively quick to build, and give the best reduction. The long list of grams tended to allow candidate strings that an average reader would not accept as real words, so there appears to be good evidence for restricting the dictionary.

The major problem with the use of n-grams is that the candidate strings remaining after look-up are not guaranteed to be words. This is an undesirable situation because a user of a script recognition system would not expect to write a word, and have it recognised as something which is not a word. Hence experiments utilising a lexical check were carried out.

2.6.3 Experiments using a lexical check

The above n-gram experiment was repeated using only those grams which were flagged as being a word. The list of words were taken from a shorter dictionary (21,211 words).

Table 2.7 — Results from using a list of words

Gram size	Length of list	Processing time (mins:secs)	% reduction of list of candidates	No. of words tested
3	441	0:02		
4	1323	0:10	98.5	7
5	1775	0:14	99.7	3
6	2318	0:20		

Much better reduction is given by the lexical check of candidate strings compared with the n-gram results. More sensible results are achieved because the output strings are guaranteed to be words. Processing times are also much reduced due to the lists of words being shorter than the lists of grams. The use of n-grams was therefore discontinued.

The main problem in the past for systems using lexical checks has been the amount of memory available for the representation of the word list. With recent advances in technology, even personal computers now have much increased memory capacity, so a lexical look-up technique which uses a large vocabulary (say, anything over 20,000 words) is now feasible. However there is still the problem of exactly what to include in the word list. Methods for effectively representing the word list in memory to provide fast look-up times in a reasonably sized structure will be discussed in chapter three.

2.7 Conclusions

The output from the pattern recogniser is poor, and requires further processing to improve. Methods commonly used for such processing involve using transitional probabilities (for example the Viterbi algorithm or Markov modelling), using information about how letters combine (for example n-grams), using lexical look-up, or combinations of these.

Statistical methods involve selecting one "correct" answer and thus have a built-in margin of error, and the use of n-gram information does not guarantee the output to be words. We have shown that a lexical look-up is more effective than n-grams in terms of reduction of candidate strings. Not only does a lexical check give a greater reduction in the number of candidates, but also a more useful reduction because it guarantees the output will be words (see also Higgins and Ford, 1989).

The lexical look-up technique is preferable to statistical methods since it does not have a built-in error rate and guarantees lexical output. The limitation of this method is that an input word may not be included in the look-up vocabulary), however this is unavoidable. This particular problem also exists for statistical methods since they sample from the language and assume a reliable distribution.

The pattern recogniser and lexical check are currently two separate stages in the recognition system. The number of candidate strings even for short words is ridiculously large, and many of the letter sequences in such strings contain non-occurring sequences. The pattern recognition output could be improved by checking some (or all) letter sequences against either lists of n-grams (bi-grams would be fastest) or even against words. Thus unacceptable strings could be rejected at an earlier stage, leaving fewer for the lexical look-up to check. This process would therefore become much faster. Taken to its natural conclusion, the stages of pattern recognition and lexical checking could become interactive processes. The two techniques could be combined in some suitable fashion to provide a fast pattern recogniser with lexical output.

Chapter Three

Word Recognition

3.1 Introduction

In chapter two we established that the most effective way of reducing the ambiguity produced by the pattern recogniser is by a lexical look-up which discards candidate strings that are not allowable in English. The chosen lexicon or word list must be represented in computer memory, but there are numerous methods which could be employed to do this, in order that it can easily be searched. However each of these methods has advantages and disadvantages, and the suitability for a particular situation will depend on many factors. For example:

- the ease of construction of the data structure;
- the speed of construction of the data structure;
- the ease of searching the data structure;
- the speed of searching the data structure;
- the amount of memory used by the data structure;
- the ease of alteration of the data structure (i.e. adding and deleting items);
- the efficiency of representation of the data so that its particular features are succinctly expressed, and can easily be retrieved and analysed.

Not all of these factors will apply in every situation, and some will be more important than others. In the case of a lexicon or word list, especially within the context of a script recognition system, the speed of searching the list is paramount, especially for unsuccessful searches, in order to obtain almost immediate recognition. This also means that the ease of searching is very important, but in a commercial package running in a single workstation environment, the amount of memory used should be kept to a minimum. Efficient update of the data structure may be important if the word list is to be changed frequently — such as adding technical terms and proper nouns. The initial construction of the data structure is of little importance to the user since it is rarely performed.

3.2 Alternative data structures

Looking at possible data structures for representing such a word list, there are many standard methods (Knuth, 1973; Wirth, 1976; Korsh, 1980; Standish, 1980; Claybrook, 1983; Amsbury, 1985; Stubbs and Webre, 1985), for example lists (stacks, queues), trees (binary, B, multi-way), graphs and hash tables to name a few.

3.2.1 List structures

The simplest structure in which to store a lexicon would be an array. However this is a static structure, it is necessary to know in advance how many items will be stored in order to allocate the correct amount of memory. Not knowing the number of items will result in wasted space due to over allocation. Each element of the array could be a fixed length character string, but this would be quite wasteful of memory as this fixed length would have to be enough to fit the longest word in the list. As the majority of the words would be shorter than this, there would a large amount of wasted space. The words could be stored contiguously in an array, so each element of the array would in effect be just a character, and each word would have to be terminated with a dummy character, for example a null character. This would be a memory efficient structure (one byte per letter, plus one byte for the null, for each word), however searching such a structure would be sequential, and therefore slow, of O(n), where *n* is the number of words in the lexicon. The whole array would need to be searched to establish whether or not a given word was stored, unless the words are stored in alphabetical order. In this case a sequential search would be of $O(^n/_2)$, on average only half of the array would be searched, for both successful and unsuccessful searches. This could be speeded up to $O(\log_2 n)$ if a binary search technique were employed.

Instead of beginning a search at the start of the structure (i.e. first element of the array), the binary search technique begins at a mid point. The item stored at this mid-point is compared with the item being searched for. If the searched-for item is less than the mid-point (for words this means the searched-for item comes earlier in the alphabet than the item at the mid-point), then the search continues in the first half of the structure, if it is greater, the search continues in the second half of the structure. Whichever half is to be searched, a mid-point is chosen again, and the process is repeated. At any stage, if the comparison shows that the items are equal, then the search terminates successfully. An unsuccessful search terminates when the end points of the section of the list being searched are in fact next to each other, and neither matches the searched-for item. Search times are of $O(\log_2 n)$ on average, and will be the worst case for all unsuccessful searches (Knuth, 1973).

The lexicon storage structure can be created dynamically rather than statically. This means that it is not necessary to know the total number of items in advance, because each item is a separately allocated piece of memory. This type of structure is known as a linked list, because each item is linked to the next by a pointer. Linked lists can be used to implement stacks and queues. A stack works on the principle of "last in first out" (LIFO). New items are pushed onto the top of the stack, and an item is popped off the top of the stack to retrieve it. The first item stored becomes the bottom of the stack. A queue follows the principle "first in first out" (FIFO). The first item stored becomes the head of the queue and new items are added onto the end, or tail of the queue.

Neither stacks nor queues are appropriate to store a list of words because of their FIFO and LIFO storage and access principles, however the linked list structure can be used.

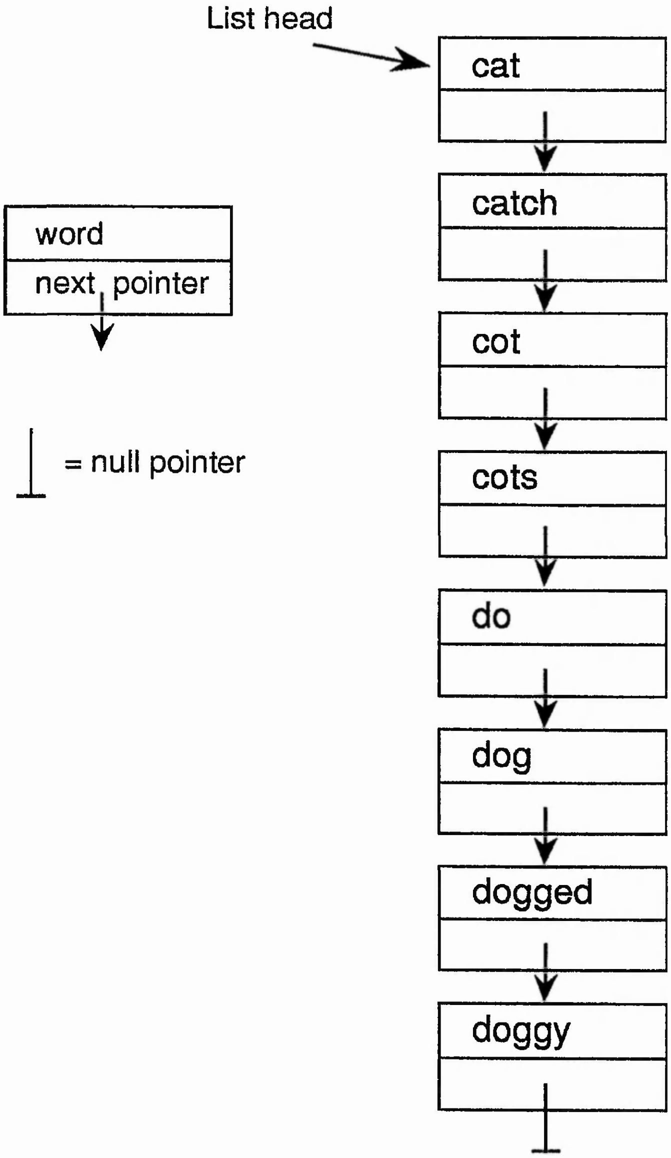


Figure 3.1 — Linked list representation of a limited lexicon

Figure 3.1 shows a representation of an ordered linked list for the small lexicon *cat, catch, cot, cots, do, dog, dogged, doggy*. Each node in the structure contains one word and one pointer to the next word in the list. This data structure is extremely simple to implement, but the search times are large, O(n) for an unordered list, O($^n/_2$) for an ordered list, because the search must be sequential. The construction of such a list is very fast if built from an already alphabetically ordered list, because new items are simply added onto the end of the list. It is much slower, O(n^2), if the words are not already ordered because new items must be inserted into the correct position in the list. This means the structure must first be searched to establish the correct position for insertion.

3.2.2 Binary tree structures

The binary search technique described earlier can be used with a dynamically allocated structure. This structure is known as a binary tree, and neatly encodes the mid-point information needed for searching. It has faster search times than the simple linked list, but greater memory overheads because each node has two pointers. See figure 3.2.

The searching and building of the binary tree structure are both straightforward, and the word list becomes ordered due to nature of the tree. For each node, every word in its left sub-tree precedes the word at the parent node in the alphabet, and every word in its right sub-tree follows it in the alphabet. Either pointer can be null, and leaf nodes in the tree are those nodes at which both the left and right pointers are null. This is known as a random binary search tree. When searching the binary tree, a comparison is made between the word at the root of the tree and the string being searched for. If they are not equal, either the left sub-tree is searched, or the right is searched, depending upon whether the required string precedes the root word in the alphabet, or follows it. This process continues until the required string is found in the tree, or until a leaf node is reached, which means that the required string does not exist in the tree. Search times are approximately O(log$_2$ n).

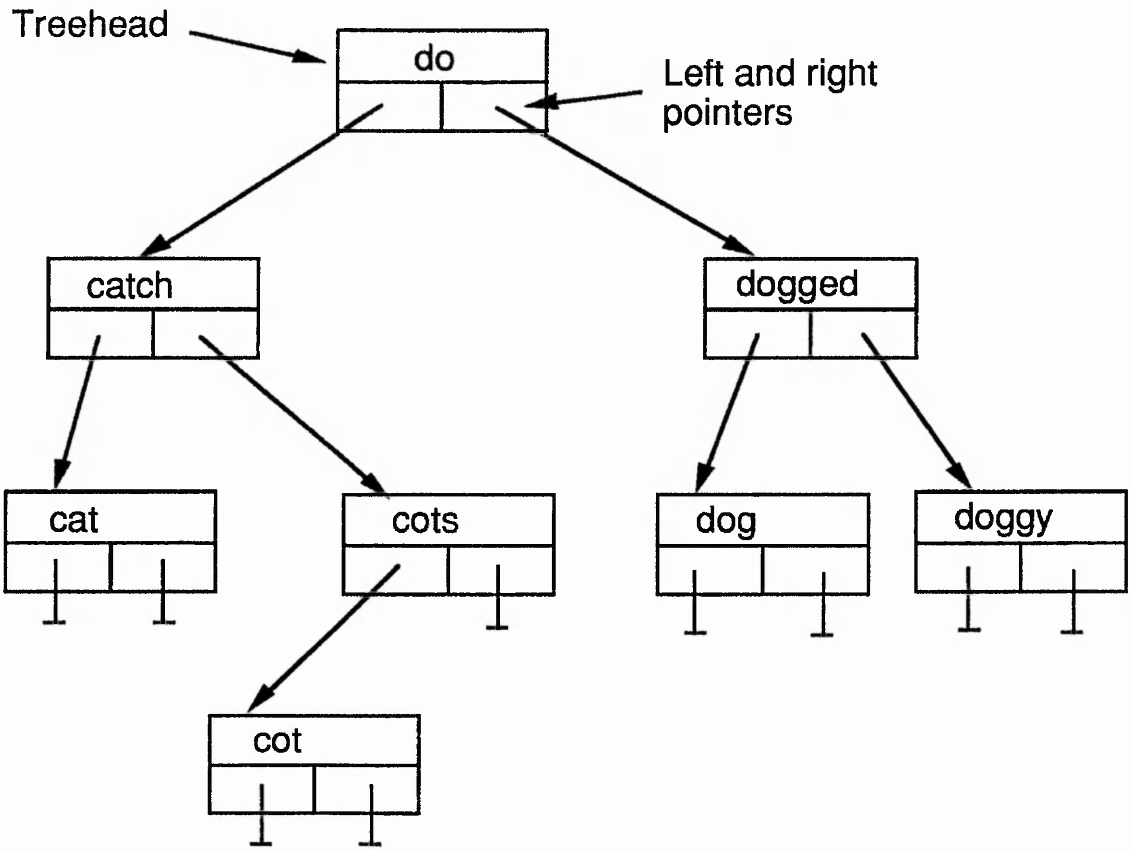


Figure 3.2 — A binary tree representation of a lexicon

Binary trees can easily become unbalanced. A balanced tree is one in which the number of levels hanging from its left branch is very close to the number of levels hanging from its right branch. For a perfectly balanced tree, also known as an AVL tree[1], the height of the two branches should differ by at most one level. Search times are quicker,

[1] after Adel'son-Vel'skii and Landis, two Russian mathematicians (see Knuth, 1973)

approaching O($\log_2$ n), if a tree is balanced, but the construction time and complexity of such a tree is greater than that of the random binary search tree which is not checked for balance. A perfectly balanced binary search tree is fastest to search, but is complex to maintain (Knuth, 1973; Wirth, 1976; Standish, 1980; Claybrook, 1983; Stubbs and Webre, 1985).

The random binary search tree is easiest to implement, but care must be taken in the construction of the word list. The first item in the list is taken as the root of the tree, so if it is built from an alphabetically ordered word list, a degenerate tree will result, where all the left pointers will be null (Knuth, 1973, p426; Claybrook, 1983, p95). This is the equivalent of a linear list, but more wasteful of space. A "zig-zag" tree structure can result from building from a list such as: *cat, doggy, catch, dogged, cot, dog, cots, do.* The left and right pointers will alternately be null. Degenerate trees can be avoided simply by "disordering" the list and ensuring that the first item is from approximately the mid-point of the lexicon. This disordering was performed by reversing each string, sorting the list alphabetically, and reversing each string again. The middle item was then moved to the top of the list. The binary tree in figure 3.2 was constructed from the resulting list: *do, dogged, catch, dog, cots, cat, cot, doggy.*

3.2.3 Multi-way tree structures

Multi-way trees are trees which have more than two pointers leading from each node. They are faster to search than binary trees (Fredkin, 1960; Knuth, 1973), and for this particular application it is apparent that a 26-way tree would be the most efficient structure in order to take full advantage of the alphabet. Unsuccessful searches will always be faster in a multi-way tree than in a binary search tree, which suits our purpose as the majority of strings being searched for will not be in the word list. For example, of the 1,204 candidate strings from the data given earlier (§2.3), only 3 strings are allowable, namely *pack, pact* and *pant.*

3.2.3.1 B-trees

B-trees are a form of multi-way tree where the growth of a tree is restricted (Bayer and McCreight, 1972; Knuth, 1973). A B-tree is balanced, in other words the tree is symmetrical so that all paths through it are the same length, so all leaf nodes appear on the same level, and carry no information. A B-tree is said to be of order m when every node has at least $^{m}/_{2}$ children, and at most m children, except the root and the leaf nodes. The root has at least two children, unless it is a leaf. Insertions are quite simple in B-trees, every leaf corresponds to a place where a new insertion might happen, however deletions are slightly more complicated. B-trees are very good for storing numbers (for example ORACLE the relational database stores its indexes as B-trees), but it is not clear that they are particularly useful for storing words.

3.2.3.2 26-way tree

The 26-way tree is a multi-way tree with 26 pointers leading from each node. Searching is simple, efficient and fast, as is the construction of the tree, however the memory overhead is vast. Each node contains one letter rather than one word, thus allowing for the study of sub-word letter sequences, but to make the searching algorithm most efficient, all 26 routes from every node in the tree must be allowed for, and on average most of these are wasted. Using two experimental word lists of 14,769 items and 79,065 items, the mean number of routes used is $^{1}/_{26}$, and just less than $^{1}/_{26}$ respectively. Each node has a flag which is set if that letter is an end of word (see figure 3.3).

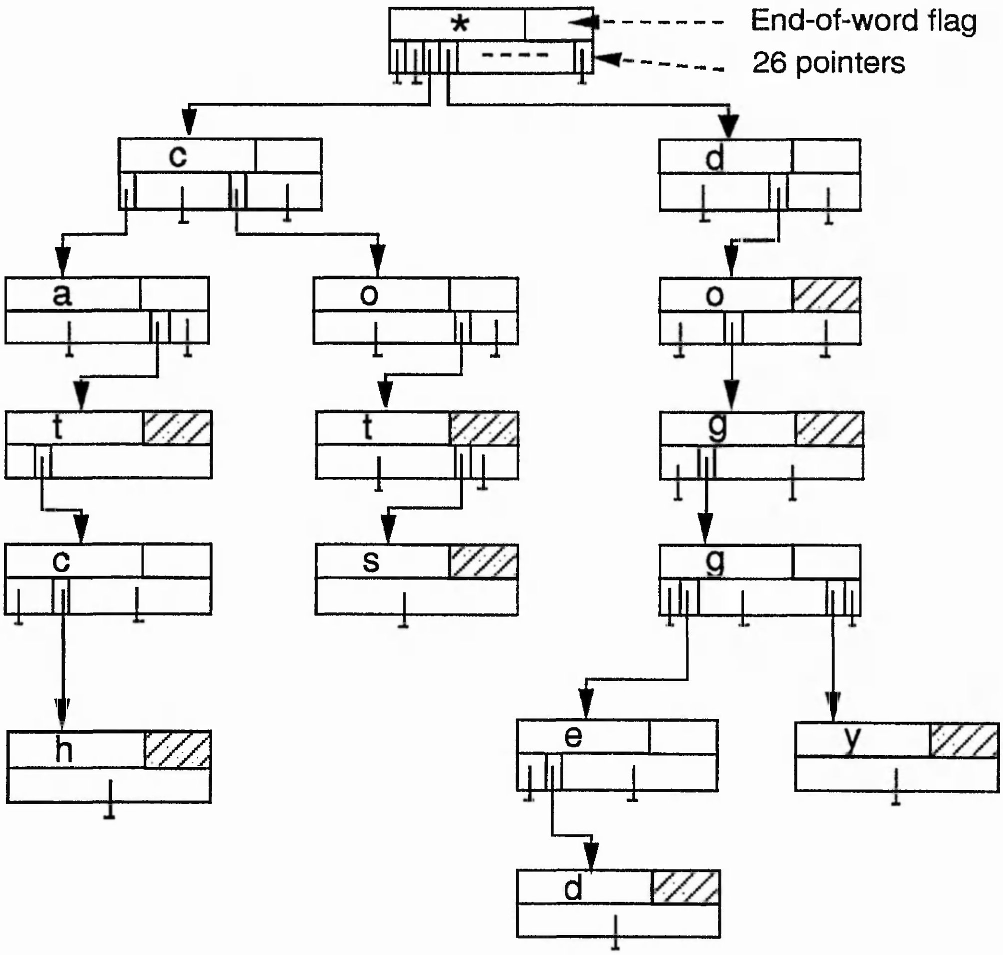


Figure 3.3 — A 26-way tree representation of a lexicon

Obviously this memory overhead can be greatly reduced by only allocating space for those routes which are used (i.e. an n -ary tree where n is variable between 0 and 26), for example typical routes from ‘z’ would be ‘e’, ‘i’ and ‘o’. However, when searching for a particular route from the ‘z’, a linear search must be employed to establish whether it exists or not, which slows down the search time compared with the 26-way tree where all 26 routes are allocated.

3.2.3.4 Trie structures

Another type of multi-way tree is often known as a trie (from the word "retrieval", but pronounced as "try" to distinguish it from "tree" in speech) and was first introduced by Fredkin (Fredkin, 1960; Knuth, 1973). A trie takes advantage of the redundancy of common prefixes, and is essentially an m-ary tree, because each node specifies an m-way branch. There are many alternative implementations of trie memory, in fact the 26-way tree just described could be said to be a type of trie. Figure 3.4 shows a trie (for the lexicon *cat, catch, cot, cots, do, dog, dogged, doggy*) after Knuth.

	1	2	3	4	5	6	7	8	9	10
A		3								
B										
C	2			catch						
D	7									
E										dogged
F										
G								9	10	
...										
0		5					8			
P										
Q										
R										
S						cots				
T			4		6					
...										
Y										doggy
Z										
—				cat		cot		do	dog	

Figure 3.4 — A trie for a limited lexicon after Knuth

Starting with the first position of the word being searched for, the trie is checked in column 1. If the word begins with 'c' or 'd', the trie is followed to the address (column number) specified, hence words beginning with 'c' go to column 2, and with 'd' go to column 7. Words beginning with other characters are (in this example) not represented. The second letter of the word is now checked. Stored words in column 2 represent 'ca' and 'co', again no other sequences are allowed. The addresses are followed until words become unique. For example in column 4 are found the words ‘catch’ and ‘cat’. Figure 3.5 shows a binary trie for the same lexicon, after Srihari (Srihari et al, 1983) who implemented a trie to represent a lexicon, but only for 1,724 words (see figure 1.1).

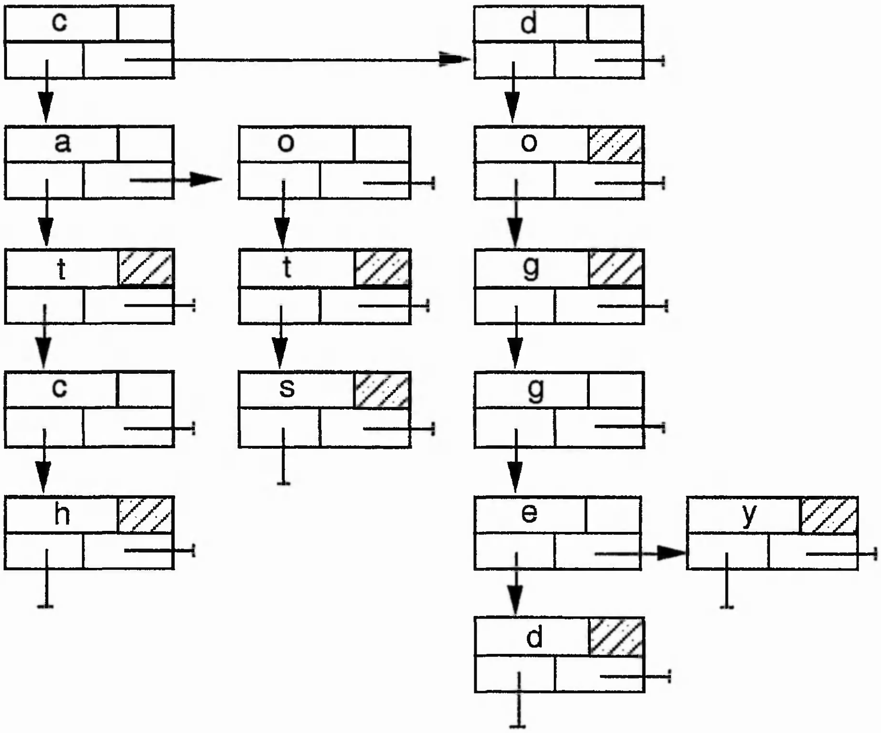


Figure 3.5 — A binary trie for a limited lexicon

This method of storage is an inefficient use of memory compared to a binary search tree or to the original word list, but searching is much faster for both the acceptance and rejection of a given string. Knuth has shown that this will take on average O($\log_2$ n / $\log_2$ m) iterations, where *n* is the size of the lexicon and *m* is the order of the trie. A trie neatly allows the study of sub-word letter sequences, i.e. it allows the immediate determination of whether or not an initial sub-string is valid. A binary trie is different from the binary tree described earlier because the two pointers leading out of a node have different functions, and there is one letter per node, rather than one word per node. The first pointer leads to other possibilities at that position in a word (alternative, brother or sibling pointers), and the second one leads to possibilities for the next position in a word (son or child pointers). There must also be an end of word flag.

3.2.4 Tree compression

It is important to keep the amount of memory required by any structure to a minimum. There are many ways in which this can be achieved. So-called compression methods generally apply to particularly wasteful areas of the structure, and reduce, or compress, these areas to use smaller amounts of memory (Knuth, 1973).

It is easy to see that tree structures representing lexicons (such as those just described) are wasteful of memory because on average there is only one route leading from any node. For every word ending in the letter sequence *-ing*, there is a separate set of three nodes in the tree, one for the *i*, which points to the node for the *n*, which in turn points to the node for the *g*. In a lexicon of just over 70,000 words nearly 6,000 end in the sequence *-ing*. It is a similar situation for the ending *-ed,* and there are also large numbers of words ending with *-er,* and *-tion*. A large amount of memory could be saved if each occurrence of the same sequence of letters used the same set of nodes in the tree structure. For example, in a tree where each nodes takes 16 bytes, compressing the *-ed* endings would save 196448 bytes. If *-ed, -ing, -er* and *-tion* are all compressed, the tree representing just over 70,000 words would be reduced from 2,628,928 bytes to 1,972,416 bytes.

However, such reductions are only possible if the word ending forms a unique route in the tree, and not if the ending is part of other words. For example *walked, walking* and *walker* could be compressed, but *alarming* could not, because it forms part of *alarmingly*. Compressing this occurrence of *-ing* would lead to the false impression that *walkingly* is a word.

It is not just at the ends of words where compression methods can be applied. Words such as *aardvark* use a number of nodes which are not shared with any other words. Ideally these unique sequences of nodes could be reduced into one single node (in a similar way to the mixed-method tree described in § 3.3.2). However such hybrid techniques mean the searching algorithm is not quite so straightforward, which is a major disadvantage for this type of application.

3.2.5 Directed acyclic word graphs

It is not just at the ends of words where common letter sequences can make use of the same nodes in a tree structure. This can be taken advantage of at any point within a word. A tree can be constructed where this process is taken to its optimal conclusion, so that all possible common letter sequences are shared. This structure is actually a graph rather than a tree, and is known as a Directed Acyclic Word Graph (dawg).

The dawg is the most efficient implementation in terms of memory (Blumer et al, 1985; Appel and Jacobson, 1988; Elliman and Lancaster, 1990). Appel and Jacobson stored 94,240 words in 175 Kbytes. A diagram showing a dawg structure of the lexicon used so far would in fact be no different from the trie in figure 3.5. However using the lexicon *car, cars, cat, cats, do, dog, dogs, done, ear, ears, eat, eats* (after Appel and Jacobson), the features of the dawg can be clearly demonstrated, especially when compared with the equivalent trie. Figure 3.6 shows a trie representation of the above lexicon, figure 3.7 shows the dawg, and figure 3.8 shows a dawg represented as a finite state recogniser of the same lexicon. This is how Appel and Jacobson represent it.

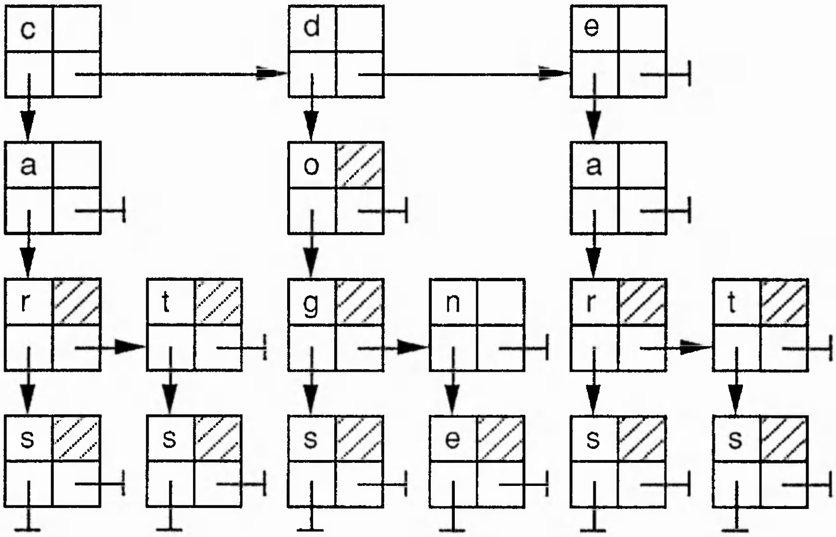


Figure 3.6 — Trie representation of Appel and Jacobson lexicon

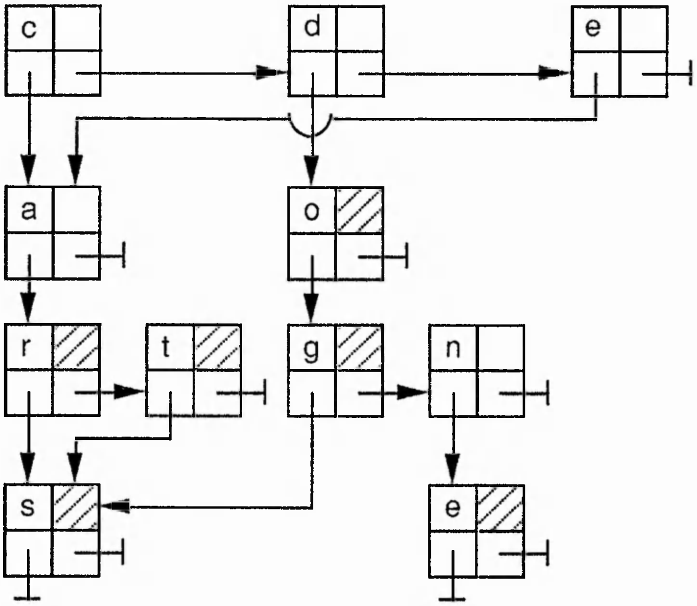


Figure 3.7 — Dawg of Appel and Jacobson lexicon

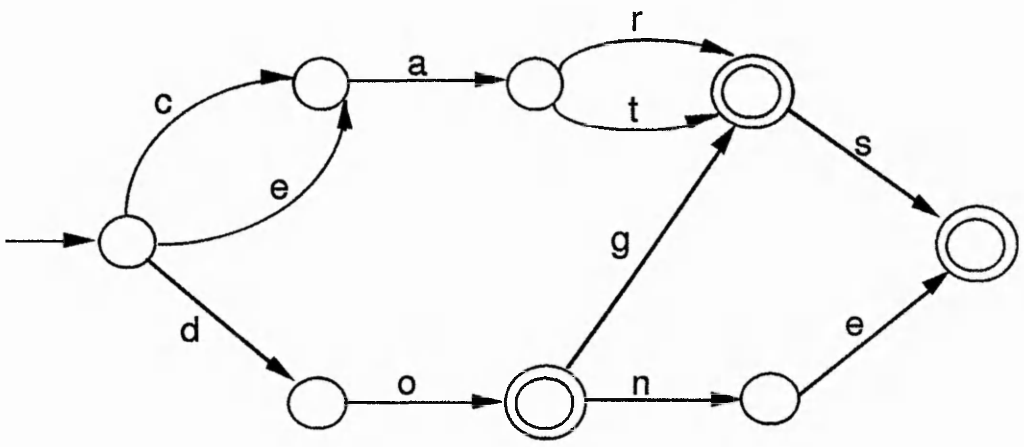


Figure 3.8 — A dawg represented as a finite-state recogniser (Appel and Jacobson, 1988)

A dawg is basically a trie where all equivalent sub-tries (i.e. identical patterns of acceptable word endings) have been merged. Appel and Jacobson represent their dawg as a finite-state recogniser of the lexicon. Nodes of the graph are the states of the finite-state machine, edges of the graph are the transitions of the machine, and terminal nodes are the accepting states. The language of a finite-state recogniser is the set of words that it will accept. For any language there will be many different finite-state recognisers. In particular there will be one with a minimum number of states, which is the one represented by the dawg.

Taking the node in figures 3.6 and 3.7 to be 10 bytes (2 chars and 2 pointers, assuming 4 bytes for a pointer, i.e. a 32 bit word length machine), the trie would use 180 bytes (18 nodes @ 10 bytes each), whereas the dawg would use only 110 bytes (11 nodes). Search times should be the same because the same node structure can be used, but building times would probably be much greater for the dawg. This is because the structure is more complex, and additional searches would need to be made to check whether paths already exist. For the trie, the current structure is searched as far as possible following the sequence of letters in the new word to be added. When the required path does not exist, new nodes are allocated to build it. To add a new word into the dawg, the current structure would need to be searched both forwards and backwards to establish whether or not the required paths exist, which would slow down the building algorithm (see § 3.3.3).

The dawg could probably be further reduced by replacing common suffixes such as *-ing, -tion,* and *-ed* with a single node (as discussed with reference to tree structures in §3.2.4). Given the nature of the dawg, it could also be made two-way, or bi-directional. This would be searchable in either direction, which could be useful in some parallel application for string searching, or just allowing for reverse searching because the end of the string is more restrictive. However due to the effective doubling of the number of paths through the dawg, making it bi-directional would increase its size (Appel and Jacobson, 1988).

3.2.6 Hashing

There is a class of popular storage and search methods known as hashing (hash encoding) or scatter storage techniques (Knuth, 1973; Aho et al, 1983; Cooper and Clancy, 1985; see also §5.4.3). An arithmetical calculation is performed on a particular item (in this case a word), thereby computing a function (known as the hashing function) which gives the location of the item (and any associated data) in a table — the hash table. This method is applied both for storing items in the table, and for searching for items in a previously stored structure. Hashing functions often indicate the same hash table entry. Although such clashes are difficult to avoid, straight-forward methods can be employed to resolve them.

Hashing techniques are very fast, but do not allow the study of sub-word letter sequences, which is particularly important in this case, as explained in §3.3 below.

The optimum size of hash table can only be determined if the approximate number of items to be stored is known.

3.3 Implementations

Several types of tree structures were implemented for comparison: a binary tree (§ 3.2.2), a 26-way tree (§ 3.2.3.2), a reduced memory tree (§ 3.3.1), two linked list mixed-method trees (§ 3.3.2), one using the 26-way tree structure, and the other using the reduced memory structure, a binary trie (§ 3.2.3.3) and a directed acyclic word graph (§ 3.2.5). The binary tree structure was found to be very straightforward to implement and use, but slow in look-up times, so multi-way trees were investigated. It was thought that the fastest look-up would be achieved using a 26-way tree, so that was implemented, and found to be extremely efficient, but the memory overhead was too large for practical purposes.

Other multi-way trees were not implemented due to increase in search times (e.g. B-trees, § 3.2.3.1) over the 26-way tree. Hash coding methods (§ 3.2.6) were also not implemented as they do not allow for the study of sub-word letter sequences. The part 26-way tree, part linked list method was implemented for comparison purposes, and because it appeared to be a simple way of reducing the amount of memory necessary. The trie structure requires less memory than the 26-way tree, and look-up times are also fast. See § 3.3.4 and § 3.3.5 for results of comparisons of implementations.

Using the binary tree structure, a successful search is when a node in the tree is reached containing the candidate string. If a leaf node is reached, the search is unsuccessful. Using the 26-way tree, a search is successful if each of the candidate letters in the string can be followed to the next one, and if the last letter in the string is flagged for end-of-word in the tree. This means that if at any stage the required pointer to the next letter does not exist, the search is unsuccessful.

Table 3.1 — Comparing candidate strings with the lexicon from figure 3.3

String	Allowable in tree ?
cat	yes
cad	no
dog	yes
dogg	no

Therefore the search can be terminated before the complete candidate string is generated from the graph of data (see §2.4 and figure 2.8). Each letter can be individually checked against the tree to see if that particular path exists. For example, if the candidate string starts with the letter *b*, check from the head of the tree to see if the 2nd of the 26 pointers is set. If it is, follow the route down the tree to the 'b' node, and get the next letter from the graph of data. Let's say the next letter is *e* (i.e. candidate string *be*) — look down the 5th of the 26 pointers from the 'b' node to see if it is set, and so on, until the end of the candidate string is reached. If at any point the required pointer in the tree is null, then that candidate string and all candidate strings beginning with those letters cannot be allowable, because no words in the tree begin with that sequence of letters. The next candidate letter at the present position in the graph is then tried. This procedure continues until the end of the graph is reached. A complete candidate string is only allowable if the end-of-word flag is set at the node in the tree for the last letter of the string (see flowchart D2 in Appendix D).

The following data gives 24 candidate strings, namely:

> *catc, cats, cadc, cads, cotc, cots, codc, cods, oatc, oats, oadc, oads, ootc, oots, oodc, oods, atc, aats, aadc, aads, aotc, aots, aodc, aods.*

```
0  :99 [1 ]
1 c:89 o:71 a:45 [2 ]
2 a:85 o:29 [3 ]
3 t:74 d:42 [4 ]
4 c:52 s:13 [5 ]
5  :99 [ ]
```

These candidate strings are compared against the 26-way tree of words (figure 3.3). As each candidate string is generated, its successive letters are tested for existence in the tree. If all such letters are found, the algorithm checks whether the resulting string is a word (i.e. the end-of-word flag is set at the tree-node reached by the search). This process results in only one allowable string: *cots*. These comparisons are illustrated in the table 3.2 below.

Table 3.2 — Incremental comparison of candidate strings with the lexicon in figure 3.3

String generated by graph	Exists in tree?	Is it a word?
c	yes	–
ca	yes	–
cat	yes	–
catc	yes	no
cats	yes	no
cad	no	–
ce	no	–
co	yes	–
cot	yes	–
cotc	no	–
cots	yes	yes
cod	no	–
o	no	–
a	no	–

An example word *supercilious*, compiled from simulated data (244,140,625 complete candidate strings) was tested using the old method of complete string look-up (7 hours and 48 minutes processing time), and the new letter by letter look-up (13 seconds processing time) — these figures include the time taken to build the tree structure as well as searching for the candidate strings. However on short words, the difference in processing times was not so great. A test sentence from data from the recogniser (*both wizened men quickly judged four sharp vixens*) took 17 seconds to build and search the tree using the old method, whereas the new method took 13 seconds.

3.3.1 Reduced memory method

An alternative method was devised which includes the advantages of the 26-way tree method, but reduces memory considerably (see figure 3.9). Instead of having 26 pointers to indicate the next letters in words, the new method uses a 32-bit integer, 26 bits of which are set (i.e. = 1) if that letter is allowable, and not (i.e. = 0) if that letter is not allowable. The end of word flag is the most significant bit, leaving five bits unused in which to encode the letter for that particular node.

> e.g. For words beginning with *c* – allowable routes are *a,e,h,i,l,o,r,u,y* and *z*, which gives 10 pointers associated with the 'c' node.
>
> 00001011000100100100100110010001
>
> which is :
>
> 0 (end of word flag), 00010 (character 'c' encoding) and
>
> 11000100100100100110010001 (26 bit indicators)

Each node also has one pointer which if used, points to a variable length array of pointers to nodes (each node is 32 bits + 1 pointer), so from the above example, there would be 10 elements in the array because 10 out of 26 bits are set. (See figure 3.9).

When searching for a particular string, it is immediately apparent at any node, whether the required route from that node exists or not by checking the relevant bit of the 26 flags. If and only if it does, that route is followed, which means counting how many of the 26 bits are set up to and including the required one, to establish which member of the pointer array to follow to the next level in the tree. It is this count which increases the search time from the original 26-way tree, but the decrease in memory usage is so great as to out weigh this slower search. For example, if the search is for the string *chess*, *h* is the 3rd bit of the 26 to be set, so follow the 3rd pointer of the 10 element array to reach the 'h' node on the level below, and so on until the entire string is found, at the 2nd 's' node, where the word flag is set.

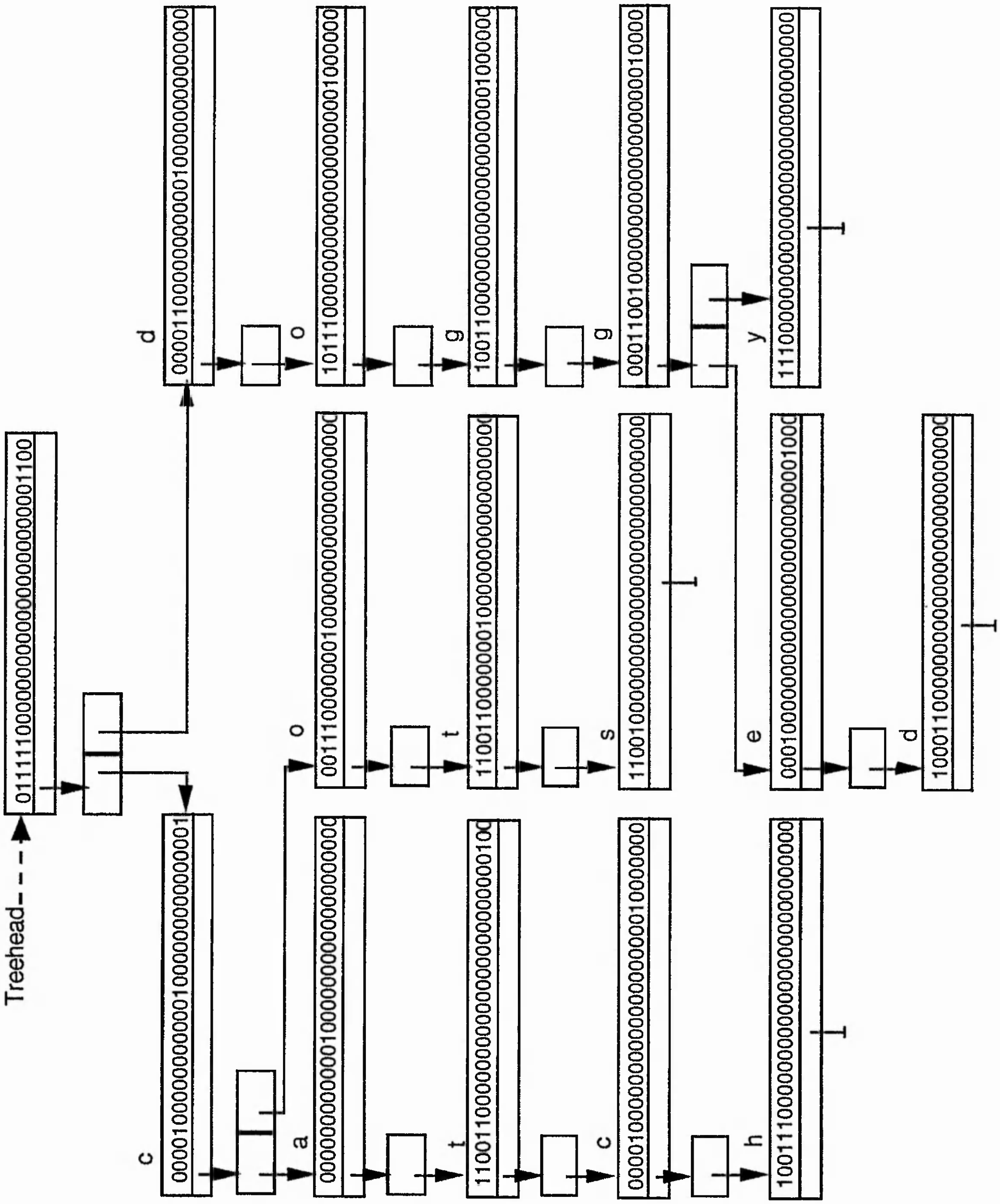


Figure 3.9 — A reduced memory tree

3.3.2 Mixed-method tree structures

There is an optimum point in the 26-way tree, below which it is more economical in memory to represent the remaining parts of the words in a linked list than in the standard tree nodes. This was suggested by Sussenguth (1963, see also Knuth, 1973). See figure 3.10.

Knuth (1973, p483) explains that we “can save memory space at the expense of running time if we use a linked list for each node vector, since most of the entries tend to be empty”, which certainly applies to our lists of words. So this can save considerably on memory, without substantially increasing search times, however the building of such a mixed-method tree would be both awkward and slow due to all the list manipulation necessary when adding new items.

For example, once a list goes beyond the optimal number, it must be un-linked, and the words built into the standard 26-way node method, with new items added in linked lists hanging from these new nodes, until they in turn reach the optimal value, and so on. If the data produces an exactly balanced tree, then this optimal point would be at the same level across all branches (Sussenguth, 1963), but this is not so with a lexicon, so the construction is much more difficult. This method was in fact implemented to establish its characteristics and for comparison with other methods, by building the tree as usual, and then pruning the branches to give the required structure as in figure 3.10. Comparisons of implementations are discussed below (§3.3.4).

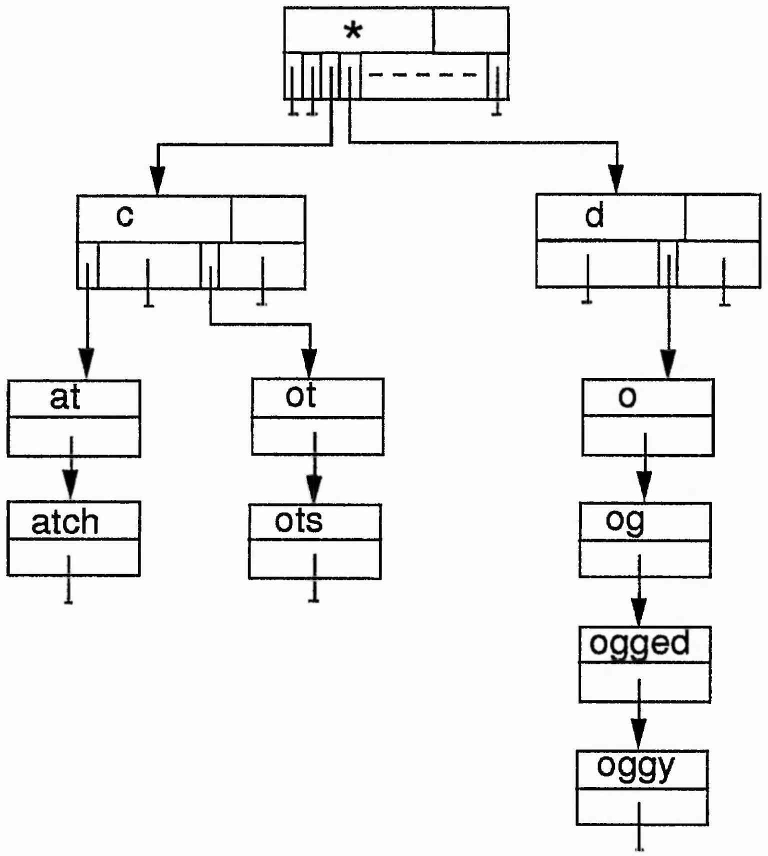


Figure 3.10 — A mixed method tree (26-way tree with linked list)

3.3.3 Dawg

An attempt was made to construct the dawg shown in figure 3.7 from an ASCII word list. Whilst the algorithm detailed in Appendix D (flowchart D3) was under development, it became clear that this representation was not the best for this particular structure. This is because multi-way nodes are represented via alternative (brother) pointers, and as new words are added, non-words are often introduced. Figure 3.11 shows a dawg for the words *stable, stabbed, stole, stolen, stable* and *dabbed*. However the non-words *dable, dole, dolen, doles, stoles, dablen, dables* and *stablen* have been introduced. This means that to use the node structure from the trie, few routes would be able to be re-used, resulting in a dawg which is little different from the original trie (figure 3.6).

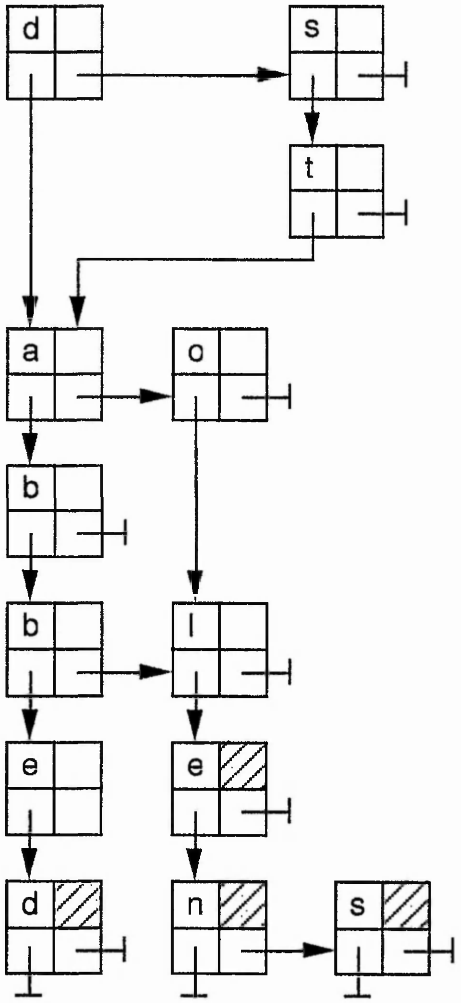


Figure 3.11 — Dawg showing introduction of non-words

An alternative structure was devised which is similar to the finite-state recogniser in figure 3.8, having labelled edges. The nodes are numbered for reference whilst the dawg is under construction (see figure 3.12). A first pass program keeps a record of available head and tail strings of words already stored in the dawg, and produces a list of transitions between the nodes, or states of the dawg. The available head and tail strings are stored in hash tables for speed of searching. As a new word is read from the

ASCII file, the head list is searched for the longest string available from the start of the word. When this is established (it may be the empty string) the tail list is searched to find the longest string available from the remaining part of the word (i.e. the word minus the head string). Again this may be an empty string. A flowchart of this algorithm is given in Appendix D (flowchart D3).

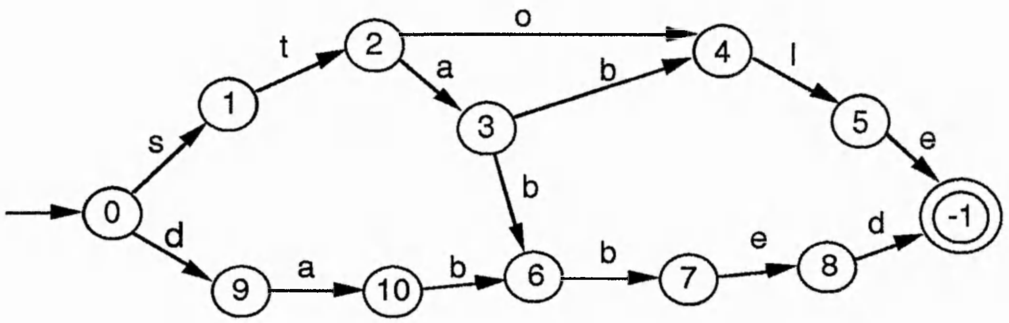


Figure 3.12 — Dawg with numbered nodes and labelled edges (first pass) for the words *stable, stole, stabbed* and *dabbed*

The dawg will diverge from the node at the end of the head string, and converge again at the first node of the tail string. The divergent and convergent nodes are joined by adding nodes for any letters from the middle part of the word which are not yet in the dawg. The transitions involved in these new nodes are printed out. Figure 3.12 produces the following transitions:

Node	Letter	Next node
0	s	1
1	t	2
2	a	3
3	b	4
4	l	5
5	e	-1
6	b	7
7	e	8
8	d	-1
0	d	9
9	a	10
10	b	6

Constructing new nodes in the dawg produces additional head and tail strings, so these are added to the lists. Finally any head string which uses the convergent node number must be removed as it is no longer available. Any tail string which uses the divergent node number is also removed for the same reason. This is done to ensure non-words are not introduced. Cycles can be introduced into the dawg if the divergent and convergent node numbers are equal. This must be checked for — if it is the case, a new head string with one fewer letter is used instead.

The list of transitions produced by the first program is sorted (numerically, by the first node number), and the node numbers are converted into edge numbers, by a simple *nawk* (Aho et al, 1988) program. A second pass program builds a dawg of labelled edges (as shown in figure 3.13) without reference to nodes, from the list of edge connections. The edges are stored in an array, so the edge number is its array index. Each element of the array stores the letter for the current edge, and an index to which edge follows (the equivalent of a child pointer). Multi-way nodes are represented by a "continued" flag stored for each edge. If this flag is set, then the current edge has alternatives (brothers). The list of transitions above yields the following set of edges:

Edge	Letter	Pointer to next edge	Continue flag
0	d	12	1
1	s	2	0
2	t	4	0
3	a	5	1
4	o	6	0
5	b	7	1
6	b	9	0
7	l	8	0
8	e	-1	0
9	b	10	0
10	e	11	0
11	d	-1	0
12	a	13	0
13	b	9	0

Memory requirements for the constuction and storage of a dawg for various word lists are given in § 3.3.5 (table 3.6).

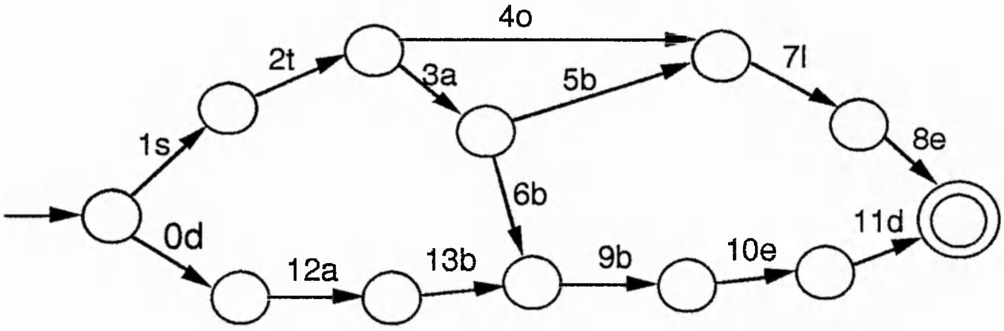


Figure 3.13 — Dawg with numbered and labelled edges (second pass)

The exact formation of the resulting dawg varies depending upon the order in which the words are presented. Figures 3.14 and 3.15 show alternative dawgs which represent the same three words but presented in different orders. For longer word lists, this can result in a different number of nodes (and therefore edges). This is because the sets of head and tail strings available at any point may be different.

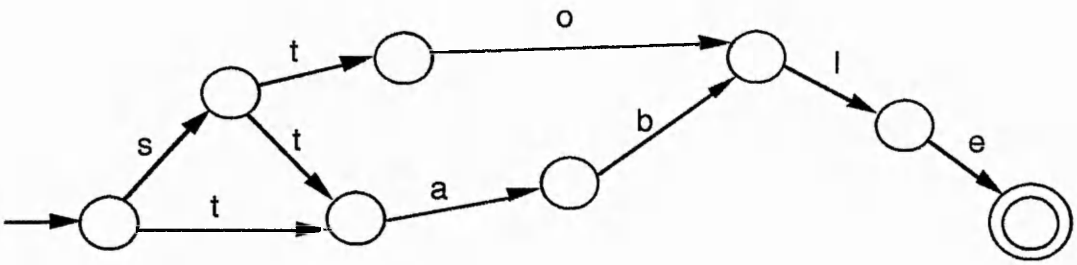


Figure 3.14 — Dawg for the words *stable, table* and *stole*

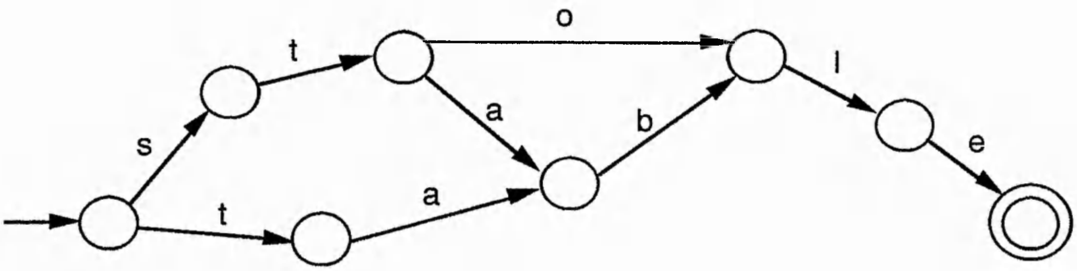


Figure 3.15 — Dawg for the words *stole, stable* and *table*

Consequently the amount of memory required for a dawg will also vary. Table 3.3 shows some alternative orderings for a word list of 2461 words, together with the number of edges needed to represent them in a dawg structure. Obviously the order in which the words are presented to the dawg building algorithm is important in order to keep the amount of memory used to a minimum. There will probably be an optimal ordering of the words which will give a minimal dawg, and further experiments should be performed to discover this ordering.

Table 3.3 — Various orders of words and number of edges required for dawg

Order of words [2]	Number of nodes	Number of edges
Alphabetic	1684	4144
Reversed	1816	4276
Backwards	1614	4074
Backwards & reversed	1597	4057

3.3.4 Initial comparisons

Four trees were constructed from the same word lists, and candidate strings from a test sentence checked against them. The results in table 3.4 were obtained for the test sentence:

> *mary had a little lamb its fleece was white as snow and everywhere that mary went the lamb was sure to go*

[2] Here, *reversed* is used to mean reverse alphabetic (e.g. 'zoo' comes before 'apple'); *backwards* means each word is considered back to front, for sorting purposes (e.g. 'walked' comes before 'talk').

Table 3.4 — Memory requirements and search times for various tree structures

Tree structure	Size of word list	Memory used (Mb - 2 D.P.)	Search time (mins:secs)
26-way	14769	4.93	0:06
26-way + linked list	14769	1.02	0:07
reduced memory	14769	0.54	0:13
reduced mem + linked list	14769	0.36	0:12
26-way	60144	15.11	0:33
reduced memory	60144	1.65	0:16
sequential file (22 words)	60144	0.7	2:00

Search times for the 26-way tree are fastest of the above methods, although it is very wasteful of memory. Especially for a large word list the memory requirements are prohibitive. Using the linked list method, the amount of memory used is much more acceptable, and there is virtually no change in speed for either tree structure. The reduced memory method has extremely reasonable memory requirements, although longer search times for the short word list compared with both 26-way tree methods. The most interesting result is that it gives quicker search times for a long word list over the 26-way methods. This is because the large 26-way tree (over 15Mb) had to page (swap) its memory, because it was running on a file server with only 12Mb of physical memory.

For comparison purposes it is interesting to note that searching a sequential ASCII file using a standard Unix command (grep) takes approximately two minutes to look for the 22 words of the test sentence, not including all the alternative candidate strings for each of these word positions. The word list takes 0.7 Mb to store in this ASCII form.

Experiments have shown that the shorter (14,769 item) word list contains insufficient items for practical purposes (i.e. the correct words were not found in the list), so search times for longer lists should be considered more important. There are disadvantages in using too long a list, in that more of the candidate strings will be allowable, and depending on the MRD that the list is taken from, can contain rare words that most native speakers would not recognise as "real" words. For example the input word *cake* gave allowable candidates including *roke, loke* and *boke*. It is considered better therefore to have a longer set of allowable candidate strings which includes the correct word although this may often include rarely occurring words. A shorter lexicon will discard the correct words on too many occasions. The exact size of the word list should depend on the recognition application.

Given all these factors, the simple reduced memory method would be most appropriate to the situation, being the optimum for memory size and search times for the 60,144 word list. It is also simpler to construct than a linked list mixed-method.

3.3.5 Further comparisons

After these initial comparisons had been completed, further experiments took place which included a trie structure. The results in table 3.5 were obtained for the same test sentence.

Table 3.5 — Memory requirements and search times for tree and trie structures

Tree structure	Size of word list	Memory used (Mb - 2 D.P.)	Search time (mins:secs)
26-way	15223	3.75	0:06
reduced memory	15223	0.55	0:12
trie	15223	0.81	0:09
26-way	43252	10.87	0:16
reduced memory	43252	1.58	0:15
trie	43252	2.34	0:13
26-way	60791	15.16	0:35
reduced memory	60791	2.2	0:20
trie	60791	3.3	0:16

It can be seen that the trie was in fact faster to search than the reduced memory tree in all three cases, however it uses more memory (approximately one and a half times as much as the reduced memory tree). It is also a particularly useful structure because it allows further information to be stored at the nodes.

Further improvements could be made to either a tree or a trie structure by using a method of tail end compression (§3.2.4). Common endings of words (for example *-ed, -ing, -tion*) which are at present duplicated in the tree for words like *walked, walking, shouted, shouting*, can be grouped together. This would not affect the speed of look-up, but would again decrease the memory requirements.

Ultimately a dawg structure (§3.2.5 and § 3.3.3) gives the best memory reduction (see table 3.6). The look-up speed should be similar to the trie (although this has not yet been established), but build times are much greater, especially for larger lexicons (for example 5,705 words takes approximately 10 minutes to build, but 68,856 words takes just over 11 hours [3]). Unfortunately, using the dawg means that any additional information about words (e.g. grammatical category) would have to be stored elsewhere (§5.4.3).

Table 3.6 — Memory requirements for dawgs for different length word lists

Number of words	Number of nodes	Number of edges	Memory used (bytes)	Reduced memory (thousand bytes)
2461	1684	4144	33.152	16.576
5705	3235	8939	71.512	35.756
13706	6567	20272	162.176	81.088
30201	12715	42915	343.320	171.660
68856	24934	93789	750.312	375.156

The current dawg structure requires eight bytes per edge. This comprises a char for the letter (1 byte), a boolean for the continued flag (1 byte) and a long integer for the next

[3] These timings are for all four stages of the dawg construction (first pass program producing transitions, sorting, awk conversion program producing edge connections, and second pass construction program). The timings are for test runs on a Sun Sparc 2 fileserver with sole usage.

edge index (4 bytes) — a total of 6 bytes which is rounded up to the nearest word boundary (on a Sun Sparc 2 file server), hence 8 bytes. This information for each edge could be packed into one 32-bit word (7 bits for the letter, 1 bit for the flag and 24 bits for the index), which would be only 4 bytes per edge. The last column in table 3.6 gives the memory requirement figures if this packing were carried out. The penultimate column gives the current memory requirements, which are twice what is necessary.

It was not possible to compare the 26-way tree, reduced memory tree and trie directly with the dawg because the word lists used in the earlier experiments were no longer available. The test runs were also performed on a different machine so it makes little sense to compare timings. There also seemed little point in rebuilding the earlier tree structures with new word lists on the latest machines.

3.3.6 Faster building of lexicon structure

Whichever tree structure is chosen, the time taken to build the structure from an ASCII word list is a few minutes for approximately 70,000 words. This time is taken up by reading each word from the ASCII file, working through it letter by letter, allocating any new nodes required and setting up the pointers correctly. It would be preferable to perform this process once only. The resulting structure can be saved in memory to be loaded each time the system starts up. This loading process is very fast (a few seconds for the same 70,000 words) because there is effectively only one piece of data to be read, instead of 70,000.

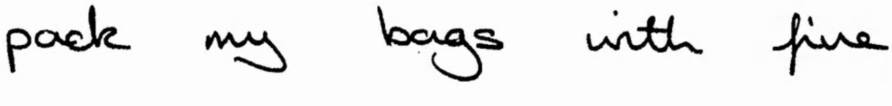
This can be achieved by separating the building process into two stages. Firstly the lexicon is pre-processed by using an array to hold the tree nodes, so that the memory required is allocated in one contiguous amount, rather than dynamically allocating each node. The pointers at each node become array indices rather than true memory addresses, but the structure is functionally unchanged. The nodes are built up per word, letter by letter, as the words are read from the ASCII file, just as before. When the structure is built, it is saved (in one chunk) to a binary file. This completes the pre-processing stage. The second stage occurs when the system is started up. Space for the tree structure array is allocated (again in one contiguous amount), and the binary file is loaded into the array. This second stage is very fast.

3.4 Look-up performance

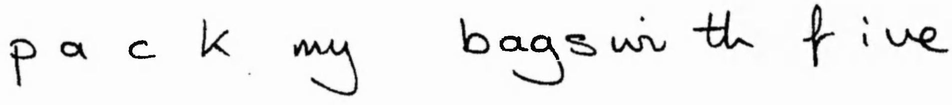
3.4.1 Failure of look-up — Wild cards

Once the tree structures had been implemented, it was found that some of the problems occurring from the recognition stage could very easily and neatly be solved from the nature of the tree. For example if the recogniser can not produce any character at a letter position within a word, we can use the tree of the lexicon to suggest what this missing character, or "wild card" might be (e.g. 'c a * e' the * could be 'f', 'k', 'r' or 's' etc.). The path through the tree is followed as normal, i.e. through the 'c' and 'a' nodes, then followed down all routes from the 'a' node to see if a path exists which ends in 'e', where the end of word flag is set. This method can be used even if the first letter position is unknown, or if there is more than one letter unknown within a word, whatever the word length. See also § 5.2.6 and § 5.3.3.3 for further discussion of this topic, with some possible implementations.

Another problem with the accuracy of the recogniser is the placing of word boundaries. When working with unconnected script, the samples of writing studied often had unnaturally spaced characters and words. Instead of simply:



using small spaces between letters and larger spaces between words which can easily be distinguished by the recogniser, the following was often found:



which obviously makes the contrast between letter and word spacing almost indistinguishable. Where two or more words have been incorrectly joined together, it is possible to utilise the structure of the word tree to note where ends of words may occur. For example, if there is no allowable string which spans the whole graph, then we can search in the same way as described above, but wherever the required path does not exist in the tree, check if that position in the tree is flagged for end-of-word. If it is, then take the next character in the graph, and start searching again from the head of the tree. The search continues to generate all allowable phrases (sequences of words) from the candidate letters in the graph. However this can give large numbers of phrases, most of which are nonsense.

If word boundaries are inaccurate, this kind of approach may be the only way to find them, except perhaps using syntactic information as well. Humans find it relatively easy to read sentences such as *packmybagswithfivedozenextraliquorjugs* and will generally use orthographic and syntactic information to do this, finding the word boundaries subconsciously. It may be possible to improve searching by noting sequences of letters commonly signalling the ends or beginnings of words, and/or by expectation of a new word by syntactic category. However the longer the sequence incorrectly joined, the more candidate strings/phrases are allowable. The sample input data *mybagswithfivedozen* (five words incorrectly joined together, with a number of alternative characters per letter position) generated 347,133 phrases of allowable strings, so there is a trade-off between finding the correct sequence, time, and the number of phrases found.

3.4.2 Ordering the list of allowable strings

Some way of ordering the list of allowable strings (or phrases) was needed, so that the most likely can be displayed as output from the recognition and post-processing system, with the other candidates available to be displayed if necessary. In this way, when the number of allowable strings or phrases is high, a few most likely can be stored — it was found that if all allowable phrases with long sequences were stored, there was insufficient memory on the computer! In practice we have found the combination (i.e. mean) of ranks to give the best results, but tends to produce tied combined ranks frequently. By "best" results here it is meant that the correct word occurs first or near the top of the ordered list of allowable strings. If the combined ranks give an equal result, using the combined probabilities to order those allowable candidate strings with the same ranks gives good results (see §2.3 for details of ranks and probabilities).

3.4.3 Results

Two example passages were written by a small number of subjects, and the script analysed by the current recognition system. The two passages are as follows:

Passage A (six samples of handwriting)

> *It has come to my attention that students have been copying software with staff encouragement. Some students are now openly broadcasting this fact. There are notices in departmental rooms regarding this matter and I would remind you that such activities as copying are illegal without a licence. It would be wise to desist from this activity forthwith.*

Passage B (four samples of handwriting)

> *Recently in the Department there has been an instance of a telephone order which has been brought to my attention by the County Council auditors. All orders must be placed through the Departmental secretary with my prior authority. Without this we are breaking our governing rules and it has been indicated that any future occurrence will make the person concerned liable to disciplinary action. Thus the procedure must cease.*

The results found by taking the first ten allowable strings in order of combined ranks and probabilities are shown in tables 3.7 and 3.8 below.

Table 3.7 — Passage A — Number of correct candidates (%)

Position	Sample 1	Sample 2	Sample 3	Sample 4	Sample 5	Sample 6
1	64.8	56.1	51.0	62.3	68.0	58.2
2	92.6	73.7	64.9	79.2	86.0	80.0
3	92.6	80.7	78.9	90.6	92.0	87.3
4	94.4	84.2	82.4	90.6	94.0	90.9
5	94.4	93.0	94.7	92.5	96.0	90.9
6	94.4	94.7	94.7	94.3	98.0	92.7
7	100.0	98.2	94.7	94.3	98.0	94.5
8		98.2	98.2	94.3	98.0	96.4
9		98.2	98.2	94.3	98.0	96.4
10		100.0	98.2	98.1	100.0	98.2

Table 3.8 — Passage B — Number of correct candidates (%)

Position	Sample 1	Sample 2	Sample 3	Sample 4
1	73.1	58.8	58.3	55.7
2	88.1	77.9	75.0	77.0
3	88.1	82.4	85.0	85.2
4	92.5	89.7	86.7	91.8
5	95.5	95.6	91.7	95.1
6	95.5	98.5	93.3	96.7
7	97.0	98.5	96.7	96.7
8	97.0	98.5	96.7	96.7
9	97.0	100.0	98.3	96.7
10	97.0		98.3	98.4

The most important result from these experiments is that if the first ten positions of the list of allowable candidate strings are considered, the correct words are present between 97% and 100% of the time, depending on the particular writing being analysed.

3.5 Conclusions

For effective implementation of a lexical look-up technique, an efficient data structure is needed for representation of the vocabulary. Such a data structure should be the best compromise with regard to processing time and memory requirements. From our first experiments with binary tree structures it became apparent that there were other structures available which would give much faster search times. The speed of negative searches is particularly important because most searches in the recognition system are unsuccessful.

The 26-way tree structure gave very fast search times, but memory requirements were almost unmanageable especially if the process was to be attempted on a small personal computer. Hence other methods were tried in order to reduce the memory overheads, and comparisons were made across a number of different methods.

Depending upon the particular system required, a balance must be found between the main variables of speed of searching and amount of memory used, especially for a larger vocabulary (approximately 60,000 to 70,000 words). The other major consideration is ease of expansion of the data structure if further information about the individual items of vocabulary is necessary. The facility to add more words to the lexicon should also be considered, especially for proper nouns and technical terms. This may require some kind of additional structure (at least temporarily) whilst the system is on-line, until the trie can be re-built with the new words at a later stage.

For our system the trie structure (§3.2.3.3) was most appropriate given the need for grammatical, morphological and semantic information in further stages of the script recognition process (see chapter 4 for further details), however the reduced-memory tree (§3.3.1) also gives fast search times and reasonable memory requirements, especially for experimental purposes with different word lists and test data on a limited memory computer.

The directed acyclic word graph (§3.2.5 and § 3.3.3) is the optimal structure for saving memory, but no additional information can be stored for each word. Such information is necessary for higher levels of analysis and will be discussed in chapter four. In contrast the trie can store other information and is therefore more appropriate at this stage. The dawg also takes much longer to construct, but would be ideal for a dedicated application on perhaps a PC where memory must be kept to a minimum and the structure would rarely need to be re-built.

Chapter Four

Integration

4.1 Introduction

The previous two chapters have discussed a system for on-line handwriting recognition. This system has a number of stages, beginning with a level of pattern recognition. Sequences of x,y coordinates are collected, their Freeman encodings are matched to a database, and a number of candidate characters are produced per character position. At the lexical level, candidate characters are combined into strings, which are checked against a tree structure of English words. Unacceptable strings are rejected, acceptable strings are ordered by their likelihood of being the correct word, and a few (usually less than twenty) top ranked words are stored for further processing.

The most successful handwriting recognition system to date is that of the human information processing system. Although it takes many years for a child to master the process of reading, once acquired, the skills are comprehensive and flexible enough to cope with a diversity of written material in a variety of fonts and formats including previously unknown ones such as unfamiliar handwriting.

The principal strengths of the human information processing system lie in its ability to make selective use of available visual cues and to utilise an understanding of the text to compensate for any degradation or ambiguity within the visual stimulus. This is possible because word images occur within a meaningful context, and we are able to exploit the syntactic and semantic constraints of the textual material (Rayner et al, 1983). Similarly computerised handwriting recognition would be facilitated by the use of such higher level knowledge.

Looking beyond the single word level, words combine into compounds and phrases, which act linguistically as a single lexical item, although being comprised of more than one orthographic word. The system should be able to make use of linguistic studies about the structure of words, phrases and sentences to improve the recognition results. This could be by establishing some criteria by which either more candidate words can be rejected or the scores of more likely candidates can be increased.

Using higher level knowledge in an automatic handwriting recognition system necessarily implies at least some integration of the different levels of processing and the information required by and produced by each level. For example the system will need to access information regarding the syntactic categories for all words in its vocabulary, as well as morphological details. It will also need to be able to process punctuation characters, upper case letters as well as lower case, digits and so on. Morphology can be approached in a number of ways, and has often proved to be problematic in computerised language processing systems. The following sections develop the idea of an indexing system for access to the dictionary which neatly includes morphological information.

Chapter three established that the current recognition system gives good results. However it deals with only a limited situation. The script is assumed to consist merely of lower case alphabetic characters, making up whole words, which are in turn assumed to exist in the lexicon being used for the look-up. Each of these assumptions will often not be the case, especially the first of these. Any text likely to be written will include uppercase letters, digits and other non-alphabetic characters. Other non-lexical items, such as abbreviations and acronyms, must also be allowed within the system. A method which accepts characters of varying case will be discussed in this chapter, along with other considerations for the integration of different levels of analysis within the recognition system.

4.2 Internal lexical structure

Words are composed of one or more morphemes (Crystal, 1987). A morpheme is the smallest unit of meaning in natural language. For example *walking* is composed of the root morpheme <walk> and the inflection (or affix) *-ing* which is the orthographic form of the morpheme <present participle>. Other affixes are *-ed*, *-er* and so on, hence *walk, walking* (= *walk* + *ing*), *walked* (= walk + *ed*), *walker* (= walk + *er*). Some affixes cannot simply be added to their stems, some other transformation of the word must also take place. For example *occur, occurring* (*occur,* with doubled final consonant + *ing*), *occurred* (*occur,* with doubled final consonant + *ed*), and many plural forms when *-s* is added, such as *cat, cats* (*cat* + *s*), *pony, ponies* (*pony*, *y* changes to *ie* + *s*).

Some recognition systems with lexicon-based look-ups have used a root morpheme + inflection (or stem + affix) approach, whereby the affix is stripped from the word, and the remaining base checked in a lexicon. This means that fewer lexical items have to be stored in memory, thereby reducing the size required by the lexicon and search times of the look-up. However there are many problems involved with affix-stripping systems (Paice, 1977; Taft, 1979; Lovins, 1968; Resnikoff and Dolby, 1965, 1966).

4.2.1 Problems with affix-stripping algorithms

If a word is stripped of potential affixes, problems can occur because the word was not actually affixed, for example consider *king,* which would be incorrectly stripped to *k* + *ing*, or *bother,* which would be incorrectly stripped to *both* + *er.* Affixes also combine together, so stripping rules must be applied more than once, looking for the longest affix possible to begin, and in a certain order. Irregularly affixed words such as *corpora, oxen,* and *leaves* must be stored separately in the dictionary, because rules will not cope correctly with them. It is not possible to say, given an input word, whether or not it is affixed. An approach such as "look the string up in the word list, and if the search is unsuccessful, apply affix-stripping algorithms" is inappropriate for a recognition system with so many candidate strings for every word position. Any string which is not allowable in the word list would be tested for potential affixes, when in fact the look-up which has already taken place, is sufficient for this level of analysis. In fact storing all separate words, including all affixed ones, gives a better solution because the look-up process is then less complex. The affixing information is important for syntactic purposes, and any syntax analysis would need such information. However to try to obtain this information at the lexical look-up stage would be complicating this stage unnecessarily. It is better left to the syntactic stage where the information is more directly relevant.

It is also interesting to note that due to the nature and structure of the look-up tree, the amount of extra space needed to store all derivations of root words is much less than at first glance. Including all derivations of words can increase the size of the list by up to three times, but the size of the tree structure representation of this list may only be slightly increased. This is because of the way the tree takes advantage of the number of words beginning with the same sequence of letters. So for example, *occur, occurred, occurring,* or *funny, funnier, funniest* only require a few extra nodes to store them in the tree, as they begin with the same few letters, which already have nodes in the tree as the root words.

For example, a word list of 32,920 items (without derivations) requires 1.55 Mb of memory, whereas the corresponding word list including derivations (60,269 words) requires 2.21 Mb of memory. In other words, a list of words twice as long requires a memory increase of only 1.5 times.

Table 4.1 — Memory increase for tree of words including derivations

Number of words	Number of nodes in tree	Memory used (Mb)
32920	96984	1.55
60269	137836	2.21

Methods of tree end-compression (§ 3.2.4) or a dawg structure (§ 3.2.5 and § 3.3.3) would also take advantage of the large numbers of words ending with the same sequences of letters, so the increase in memory requirement for a word list including derivations (compared to that without derivations) would be even smaller.

4.2.2 Approaches to morphology

In order for dictionaries to be accessed easily, to keep down storage costs, to aid syntactic and semantic processing and to reflect the morphological properties of user input, a morphologically based access system of some sort is required (e.g. Boguraev and Briscoe, 1989).

There have been two general methods of using morphology to access a lexicon and to produce a syntactic representation for an input string. One uses a word-grammar system (e.g. Ritchie et al, 1987) to decompose input strings into constituent morphemes and inflections. The other uses a limited rule-based affix-stripping routine to analyse an input string into its root and inflection (e.g. Holmes, 1988; Ramsay and Barrett, 1987). The latter is more limited in its range of coverage, but is computationally less complex. The word-grammar approach often over generates by producing morphologically legal but non-occurring words (e.g. *inter-possess*). The affix-stripping approach requires a lot of lexical checking and may produce an incorrect stem as discussed above (e.g. *k* + *ing*, *both* + *er*). Since these problems are all magnified for a recognition application where multiple input occurs, a different approach to analysing and representing morphology has been developed by Keenan (Keenan and Evett, 1989). This is only for the English language, although it appears that affix-stripping routines are more appropriate for languages such as Italian and Spanish (and possibly German) because of the high number of regular inflections.

4.2.3 An alternative approach to morphology

Keenan's approach is different from those described, since morphology for input is pre-processed into the system (Keenan 1989). The entries of a dictionary can be accessed via the headword addresses to obtain syntactic and semantic information. In order to access this information, an indexing system has been developed which incorporates morphology. This has been done in the following way. A list of non-inflected words was obtained from a machine-usable version of the Oxford Advance Learners Dictionary (OALD) (Hornby, 1988). Each of these was assigned a unique root-morpheme index. Codes from the OALD were used to generate the inflections for these basic words (e.g. comparative adjectives, plurals); the root index was then inherited by the derivations. So, for example, *funny, funnier* and *funniest* would all have the same index (see table 4.2).

Table 4.2 — Example words with their indices

Word	Index
funny	320
funnier	320
funniest	320
leaf	772
leaves	781
walk	1234
walking	1234
walked	1234
wood	1670

These index numbers are stored in the look-up tree at the ends of words at each node where the end of word flag is set, for each of the words represented in the tree. Note that one orthographic form may have more than one index, for example for homographs (e.g. *cone*), and for derivatives which could have different roots (e.g. *does*). In these cases, multiple entries will be represented as potential candidates. Since the recognition system already deals with multiple candidates as one of its basic functions, this does not present an additional problem.

When the candidate letter strings are checked against the tree of acceptable words, the indices for those strings which constitute acceptable words are obtained. Having obtained the root index for a word from the tree, more information for that word can be accessed if required. This might be additional information stored on disk, such as a word's definition. The entry stored in the dictionary is syntactically correct only for the morphological root. Some alteration to the syntactic information will be required for derivations. To determine the alterations required, the input string is compared with the root string (stored in the first field of the dictionary entry). The syntactic information for the input string is obtained by modifying the syntactic information for the root string depending on the differences between the two strings and the part of speech of the root string. Irregular derivations have their own entries.

Essentially, this method pre-processes affix-stripping into the system, avoiding the problem of production of incorrect stems. While this is not as comprehensive a system as a word-grammar, it is computationally simple, provides no look-up overhead, and avoids over-generation (Keenan and Evett, 1989).

4.3 External lexical structure

Words combine into larger units such as compounds, phrases, clauses and sentences. Compound nouns, verbs, adjectives and so on are sequences of two or more words that frequently occur together. They act linguistically and psycholinguistically (Wilson, 1984) as a single lexical unit. For example the meaning of *above board* has little to do with the separate meanings of *above* and *board*. Generally a compound is not simply a combination of the syntax and semantics of its separate words. The same can be said for commonly used phrases (e.g. *dear sir or madam*, *easy walking distance*). Some compounds and standard phrases like these are often highly domain dependent (Warren, 1978).

Historically certain types of compounds develop over time from separate words, usually become hyphenated, and finally lose the hyphen to produce one word, for example *tea cup, tea-cup, teacup*. Often all three will be in use at one time. Currently in English all three orthographic representations can be found, but they are linguistically identical, i.e. acting as one lexical unit. Other languages are different in their development of compounds — for example German misses the first two stages and combines words straight away into a longer word. (e.g. *Rathaus, Rat* + *Haus* or *Krankenwagen, Kranken* + *Wagen*). Dictionaries dedicated solely to common phrases and idioms are available.

4.4 Recognition of compounds

4.4.1 Introduction

Given the lexical look-up described in chapters two and three, the number of allowable candidate strings per word position is often large, especially for words of length 3 to 6 (see below and Appendix B for examples). Table 4.3 shows the number of allowable strings remaining after pattern recognition and lexical look-up for the test data:

> *any future occurrence will make the person concerned liable to disciplinary action thus the procedure must cease.*

Table 4.3 — Number of allowable words for a test sentence

Word	No. of allowable strings
any	88
future	6
occurrence	1
will	77
make	40
the	18
person	3
concerned	5
liable	5
to	3
disciplinary	2
action	26
thus	131
the	26
procedure	1
must	9
cease	4

Longer words (approximately 10 letters and over) are on the whole uniquely recognised. The list of allowable words is ordered, and only the top ten, or fewer are stored in a suitable structure. Given an allowable candidate word which is known to be able to start a compound or commonly used phrase, the list of allowable words for the following word position can be checked to see whether any of those words could be the second element of the compound or phrase, and so on through the following word positions until the end of the compound or phrase is reached, or until the next word of the compound is not found in the list of allowable words. For example, the allowable candidate words for the two phrases *easy walking distance* and *tennis courts* are shown below.

1)	easy	walking	distance
	cords	revoking	enhance
	colds	rocking	instance
	aids		
	odds		
2)	farms	courts	
	lawns	devils	
	firms	hairs	
	terms	covers	
	forms		
	tennis		
	tailors		
	terrors		
	blows		
	fulfils		

Having conducted a few preliminary experiments to check the validity and effectiveness of this approach, it was decided to construct another trie structure containing the compounds and common phrase information. This trie is separate from the existing look-up tree, but is accessed from the look-up if an allowable word is found which may be part of a compound. This access is via a flagging system (see later §4.4.3).

This process is an extremely effective way of reducing the lists of allowable strings. We can say that *easy walking distance* is more likely to be the correct sequence of words rather than *colds rocking enhance* or any other combination of allowable candidate strings. See Appendix C for some recognition results before and after the application of compounding information.

The MRD which provides the morphological information described earlier also contains information regarding compounds and some phrases. Others can be added manually to tailor the MRD to a particular topic or area of usage. For example estate agent's property details, business letters, newspaper articles, novels and so on.

4.4.2 The compound tree

Each word in the lexicon has an associated index (as explained in §4.2.3), and each compound or phrase also has an associated index, as shown in tables 4.4 and 4.5 below. The information about the compounds and common phrases needs to be structured so that each element (i.e. each separate word) is linked together and can access the compound's index. A trie was constructed for this purpose (see figure 4.1).

Table 4.4 — Example indices for words that are part of compounds

Word	Index for word
airing	6001
area	264
back	384
built	601
cupboard	1153
detached	1286
garden	2055
house	2324
in	2430
up	4994
wardrobe	6055

Table 4.5 — Example indices for compounds

Compound	Index for compound
airing cupboard	46
back garden	1
built in	135
built in cupboard	3
built in wardrobe	192
built up	204
built up area	333
detached house	6

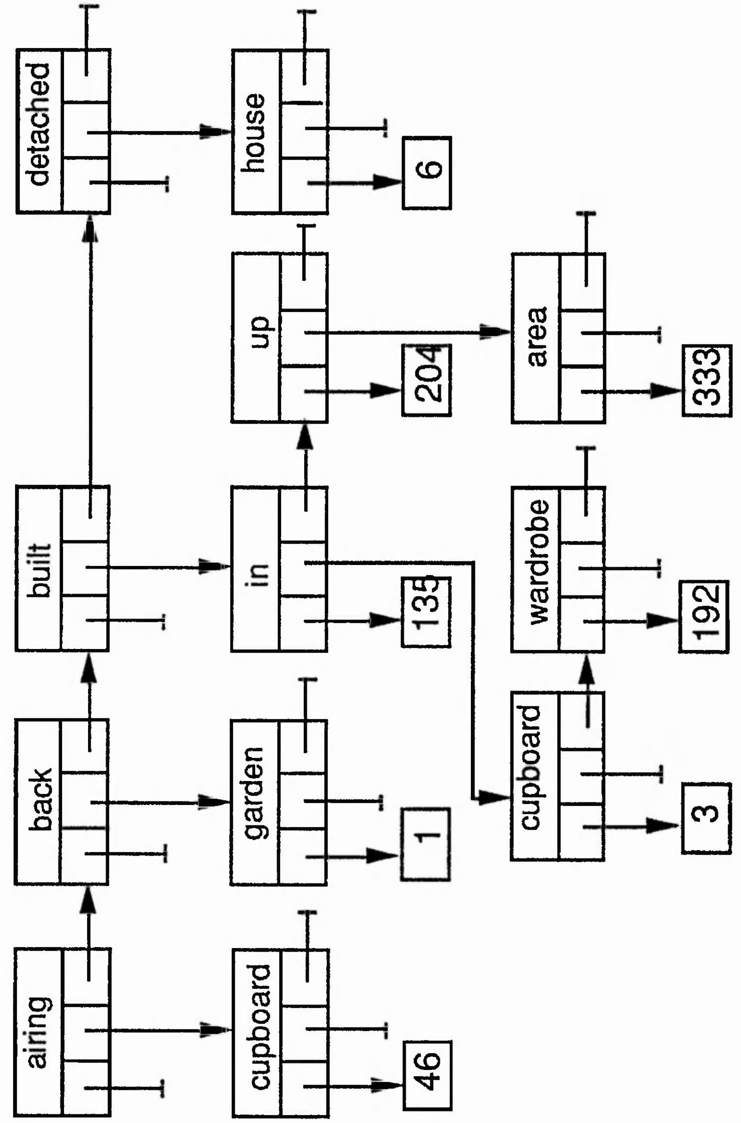


Figure 4.1 — Tree structure representation of compounds using one word per node

The trie structure shown in figure 4.1 is wasteful of memory because each word is stored at each node. This is not necessary because a word's index number could be used instead (see figure 4.2). Each node in the trie has the usual two pointers (one pointing to a child node, and one pointing to alternatives or siblings). There is also a third pointer which, if used, gives the index of the complete compound. When a path through the tree is followed and a possible end of compound or phrase is reached, there

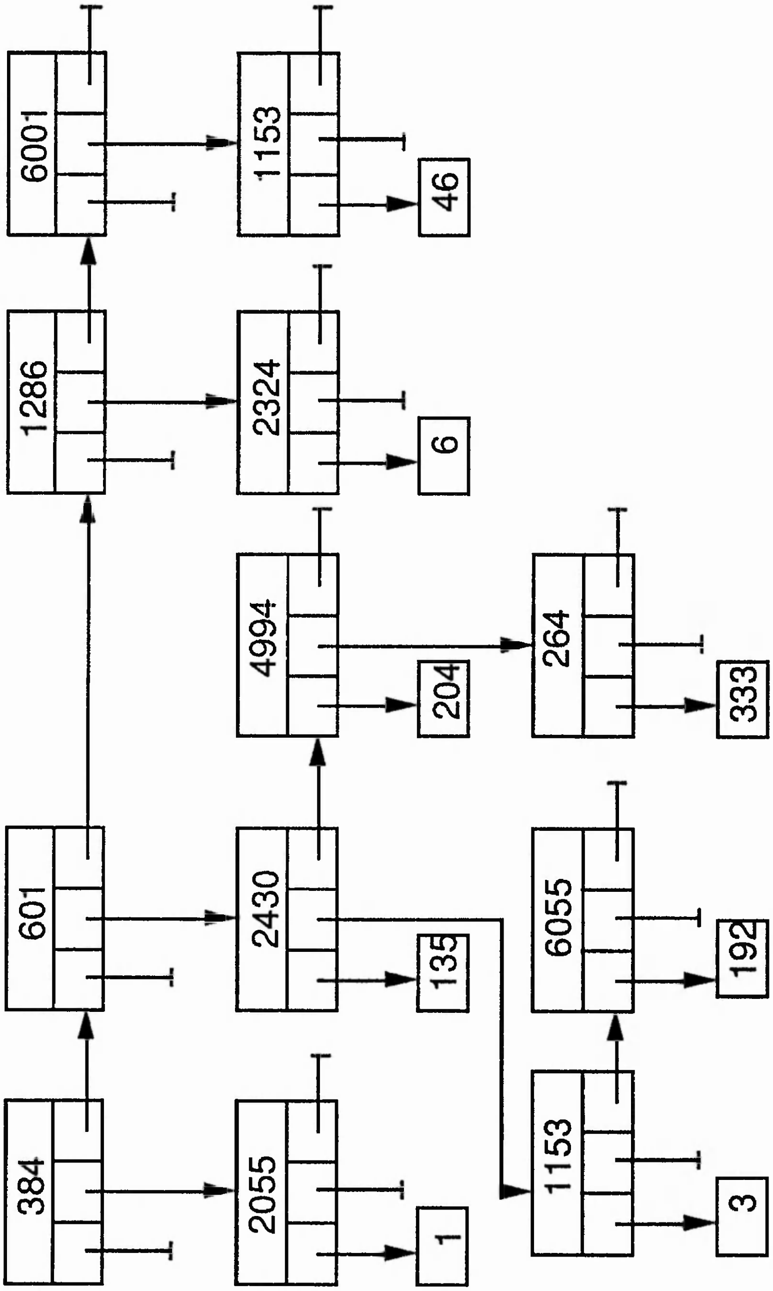


Figure 4.2 — Representation of compound tree structure using indices

is a pointer to the index of the whole compound or phrase so that information about its syntax and semantics can be obtained from the dictionary. The syntax and semantics of a compound or phrase is different from that of its constituent parts, but this required information can be obtained from a MRD.

4.4.3 Flagging system

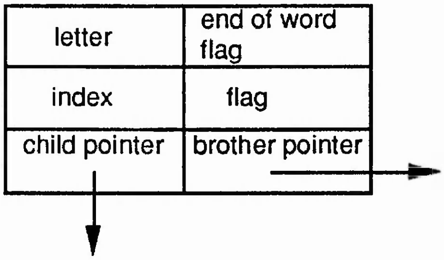
The tree representing the lexical look-up structure and the tree representing the compound information are completely separate. Each word that can start a compound or phrase is flagged as such. This flag is stored at the end of word node in the lexical look-up tree. In fact other information about words needs to be stored, and this can be achieved using the same flag, using a coding system described below.

Written language contains many cues beyond the word level which give information about higher levels of structure. These cues include the occurrence of capital letters and punctuation, so these would need to be acceptable to the recognition system, along with abbreviations, acronyms, digits and other characters (such as '?', '!' '%' and so on). Words which are names (proper nouns) which should always start with a capital letter can be flagged in the lexical look-up tree so that this is checked against the candidate character data at the look-up stage. A total of twelve codes are necessary for this flag to encode the relevant information (see table 4.6).

Table 4.6 — The twelve flags with their corresponding meanings

Code	Case status	Compound status	Example
0	starts with capital letter	not	Cindy
1	all same case	not	cat
2	any case (inc. mixed)	not	PhD
3	starts with capital letter	part of	Bridge
4	all same case	part of	cupboard
5	any case (inc. mixed)	part of	
6	starts with capital letter	start of	London
7	all same case	start of	airing
8	any case (inc. mixed)	start of	
9	misspelled word	not	recieved
10	misspelled word	part of	cuboard
11	misspelled word	start of	detatched

Each node in the lexical look-up tree is now of the following form:



As the trie is being constructed from an ASCII word list, there is a problem to be overcome. This is where there is more than one occurrence of the same orthographic word (ignoring case), with different flag codes. For example *frank* (code 1), and *Frank* (code 0). Only one code can be stored in the tree as that word's flag, and in fact only one is needed, because the codes form a hierarchy. A system was devised which expresses this hierarchy in order to calculate which codes should have priority when a duplicate word is encountered (see table 4.7).

Table 4.7 — Example words with multiple case codes showing which code has priority

Word	Code	Priority
airing	1	
airing	7	7
frank	1	
Frank	0	1
house	1	
house	4	4

A code of 7 implicitly allows a code of 1 (because they have the same case status, but 7 means the word can be the start of a compound), so 7 is given priority over 1, and 7 is the code stored as the flag at that node in the tree. Similarly to express both codes 1 and 0, code 1 should be used because it allows both cases, and so on. To allow for all possible combinations of the twelve codes, the data shown in table 4.8 was produced. The table is actually symmetric, but can be thought of as though the row code would be that of the code stored so far in the tree, and the column code would be that of the new code found in the word list.

Table 4.8 — Multiple case code priority

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	1	2	3	4	5	6	7	8	9	10	11
1	1	1	2	4	4	5	7	7	8	10	10	11
2	2	2	2	5	5	5	8	8	8	11	11	11
3	3	4	5	3	4	5	6	7	8	9	10	11
4	4	4	5	4	4	5	7	7	8	10	10	11
5	5	5	5	5	5	5	8	8	8	11	11	11
6	6	7	8	6	7	8	6	7	8	9	10	11
7	7	7	8	7	7	8	7	7	8	10	10	11
8	8	8	8	8	8	8	8	8	8	11	11	11
9	9	10	11	9	10	11	9	10	11	9	10	11
10	10	10	11	10	10	11	10	10	11	10	10	11
11	11	11	11	11	11	11	11	11	11	11	11	11

When a candidate string is checked in the look-up tree, its case priority is now also checked against the flag code found in the tree (see table 4.9).

Table 4.9 — Example candidate strings and their case priority

Candidate string	Case priority
Cat	0
cat	1
CAT	2

If the tree flag code is greater than eight (i.e. 9, 10 or 11) any candidate string (whatever its case priority) is said to be allowable - these are misspelled words. Otherwise if the case priority equals that of the word in the tree, then the candidate string is allowable. This is checked by dividing the flag code by three and comparing the remainder with the candidate string's case priority. If they are equal, the string is allowable. The only other acceptable conditions are when the flag code is 1, 4 or 7, and the candidate string begins with a capital letter. This deals with words other than proper nouns which have been written with a starting capital, i.e. at the beginning of a sentence. In this way both upper and lower case script (and mixed case) can be accepted by the recognition system.

4.5 Integration of higher levels

Acceptable candidate words from the lexical look-up stage of recognition can now be checked for sequences of words which belong to compounds or common phrases. Additional information about the candidate words is stored in the look-up tree (see previous § 4.4), and is therefore available with every allowable candidate to be passed onto further stages of analysis, as shown in the flowchart in figure 4.3 below.

The candidate words, their associated scores and dictionary indices are then passed on to the syntactic or semantic analysis stages for further processing. Each word has one or more grammatical category (e.g. noun, verb, adjective, preposition, conjunction) associated with it, and these are collected for processing. Sequences of grammatical category are scored for likelihood of occurrence in English. The scores of candidate words are increased in proportion with their syntactic likelihood of occurring in that word position (see § 4.5.1 for further details).

The candidate words, scores and indices are then passed onto the semantic analysis stage where combinations of words are measured for likelihood of occurring together by dictionary definition overlap and collocations (see § 4.5.2 for further details).

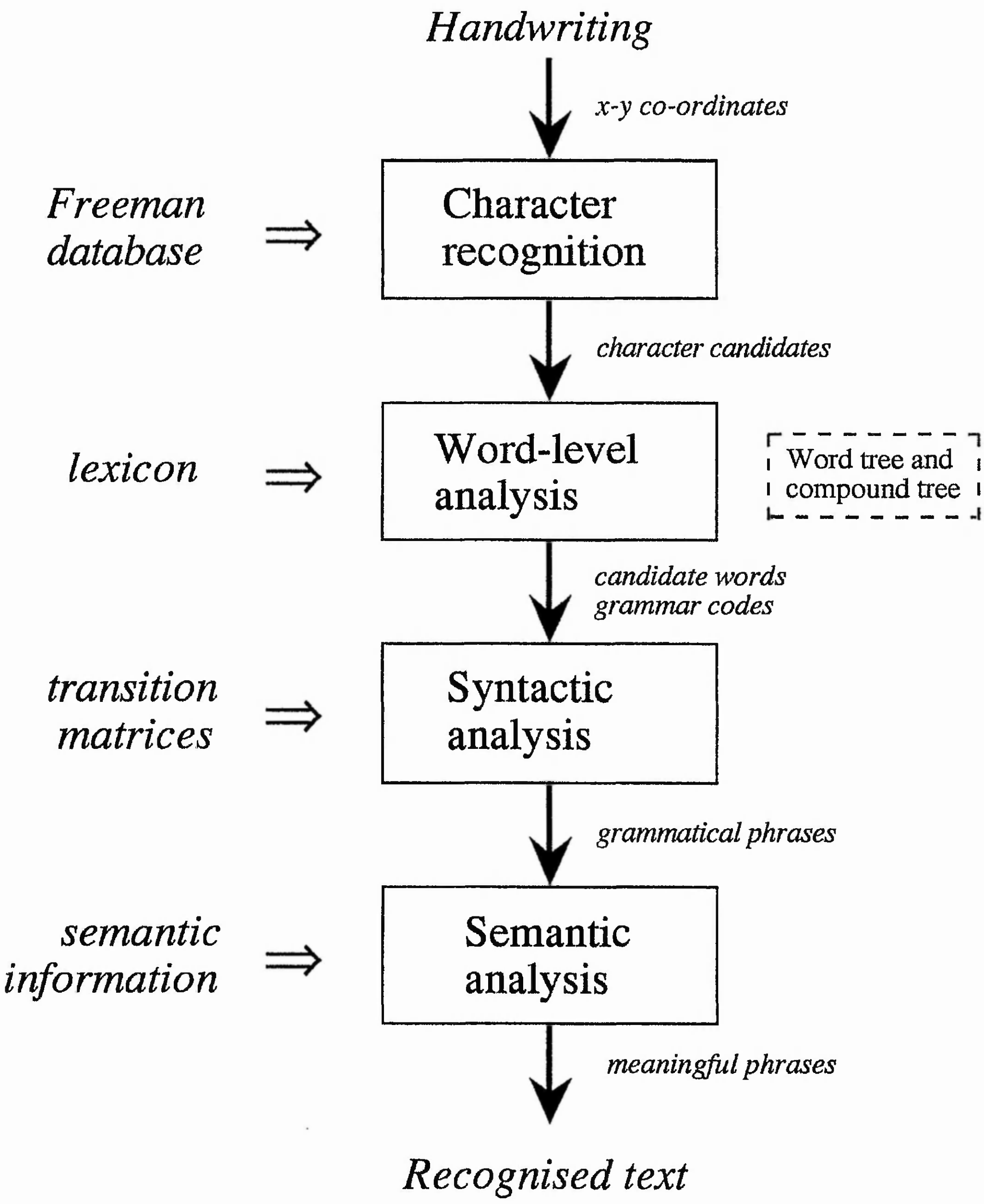


Figure 4.3 — Overview of handwriting recognition system

Currently the system is discrete and sequential, however this is not necessarily the best way of processing the candidate words. Each level has its own information to contribute, and the process is not necessarily serial (see also figure 6.2). Words can be processed by the syntactic and semantic analysers with or without compound information, and indeed could be in either order. The process could be parallelised.

Information about the candidate words could be passed on at a variety of stages. After each word is processed, after each sentence, or even at some point in between. There is not really enough information available for processing to be after each word position, because both syntax and semantics are concerned with linguistic units larger than a single lexical item. They are concerned with how words combine together. Consequently it would be more productive to start further processing after each sentence, because there would be much more information to consider. However, it may be that sentences cannot be detected reliably at the pattern recognition level. It would also mean a longer wait before any processing can begin, and more data would need to be held in memory. There may be some suitable compromise which will give any syntactic and semantic analysis sufficient information about a particular word and those surrounding it.

An effective structure which will hold the necessary information for all levels of analysis is also required. It may be passed between the different stages (especially if the processes follow serially) or may be accessed centrally (especially if the different processes are executed in parallel). In fact the word look-up stage is an ideal site for integrating different types of information because it provides the interface between the low level pattern recognition and the higher levels of linguistic processing.

4.5.1 Syntax analysis

Handwriting recognition could be improved by implementing information about the syntactic structure of text. This may be used to rule out phrases which are grammatically unacceptable. The syntax analysis stage of recognition described here is a statistical approach. Statistical methods for syntactic processing are more robust and better suited to the recognition task than rule-based approaches. This particular technique has been tested on text samples and an improvement in performance demonstrated (Keenan et al, 1991).

The best known method in which information about syntax is implemented computationally is the use of parsers based on competence grammars. Although such parsers are linguistically sophisticated, they produce a number (sometimes large) of parses for any sentence analysed, and require large amounts of computing resources. It is also not clear that the grammars they are based upon are capable of generating the range of constructions that occur in real life. For recognition purposes it is sufficient to know whether or not word strings are grammatically acceptable, rather than to know all possible parses. Statistical techniques achieve less deep analyses but are robust and easier and more efficient to implement. Since they have been developed with reference to real usage of language they are capable of dealing with language as it occurs.

4.5.1.1 Syntax implementation

In order for the syntactic analyser to process text, a number of resources are required. These consist of a corpus to determine the syntactic patterns of the language, a lexicon for storing the grammatical categories of the vocabulary, and a tag-set to encode the grammatical information.

The corpus used for the current project is the Lancaster Oslo Bergen (LOB; Johansson and Hofland, 1987). It comprises approximately one million words of text derived from a variety of British English (printed) texts. Each word is tagged with its grammatical category. The lexicon (stored as the lexical look-up tree) currently in use contains the grammatical information for more than 70,000 words and is derived from a combination of the Text710 version of the Oxford Advanced Learner's Dictionary of Current English (OALDCE; Mitton, 1986) and the LOB corpus. This combination also provides the tag-set of grammatical codes, resulting in a set of 109 tags. These are represented by the integers 1 to 109. Most words belong to a number of grammatical categories, although most of these are relatively rare and the most frequent category is almost always sufficient. Consequently the grammatical tags for each word in the lexicon are given a grammatical frequency factor (GFF) to distinguish the importance of the different tags. The GFF represents the frequency of a tag for a word, relative to its most frequent tag.

The application of syntactic information to handwriting recognition consists of two phases: firstly acquisition and secondly recognition. During the acquisition or development phase, transition matrices are created by analysing a corpus for frequencies of transitions between grammatical categories (e.g. how many times a determiner is followed by a noun). The frequencies are then normalised and stored in a transition matrix.

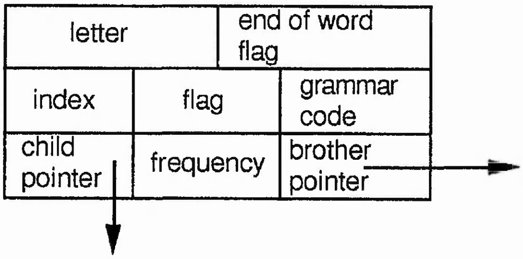
The recognition phase is the run-time application of the information derived during development. Given a graph or lattice of possible words, the syntactic processor ranks the correct words according to their ability to combine grammatically with neighbouring words. This decision is made by first finding the possible grammatical categories of the words in the lattice. The categories are found from the lexicon. Then the probability of each of the possible transitions between the categories is retrieved from the transition matrix. The words are assigned a rating based on the probabilities of the transitions in which they participate.

Experiments have shown that recognition is most improved when the total transition probability rating is combined with the GFF and the score previously calculated from the pattern and lexical recognition levels (Keenan et al, 1991). Appendix C gives details of recognition results before and after syntax analysis.

This statistical syntax analysis technique can also predict the most likely category for a particular word position, given the grammatical categories of surrounding word positions. This information can be held in the central structure and may be made use of in cases where a word is unknown (not recognised) for some reason. This will be discussed in more detail in chapter five (§5.3.4.1).

4.5.1.2 Access to syntactic information

For ease of access, the grammatical categories and frequency numbers for each word in the lexicon, are stored in the look-up tree. Each node in the tree is now of the following form:



This is overloading the tree unnecessarily, as it is now fulfilling two different functions — firstly to establish whether or not a word exists, and secondly to provide a convenient access method for other information. In chapter five, a further situation provides an opportunity to separate these two functions (§5.4.3 and § 5.4.4).

4.5.2 Semantic analysis

Just as syntactic knowledge can be applied to computerised handwriting recognition, the use of semantic information should also improve its performance. Phrases which are most likely to be correct can be selected from the candidate phrases. Attempts to develop theories of natural language semantics have tended to be for small and concrete domains (e.g. Katz and Fodor, 1963). Extending such theories for large real world vocabulary is difficult. Firstly, the hand-crafting of features for a large vocabulary would be an almost impossible task. Secondly, while some theories may work well for concrete subjects such as “table” and “chair”, they may well not be applicable to abstract concepts such as “justice”, “insurance” or “business”.

An alternative approach is to use existing sources of information about meaning. Dictionaries and text corpora now exist in reasonable numbers in electronic form, and they provide information about large vocabularies. The current system has developed a number of techniques to apply semantic information from such sources (Rose et al, 1991). These are using a “definition overlap” technique (Lesk, 1988) from dictionary definitions and from linguistic collocation information. The processing of text using subject domains has also been investigated.

The “definition overlap” technique measures the likelihood that two words would occur in the same sentence by comparing their definitions. If the first word is found in the definition of the second, or vice versa, then the pair are said to have a strong overlap. If they do not have a strong overlap, but a third word occurs in both the definitions, then they are said to have a weak overlap. Any overlap established is scored accordingly, strong scoring more than weak. Scores are calculated for all pairs of content words (i.e. function words such as *the, but, and* are ignored) within four word positions of each other. Experiments have shown that this method distinguishes between related and unrelated pairs, although not with 100% accuracy. It also selects the correct word from recognition candidates at a greater than chance level, with good results for definitions taken from the OALDCE.

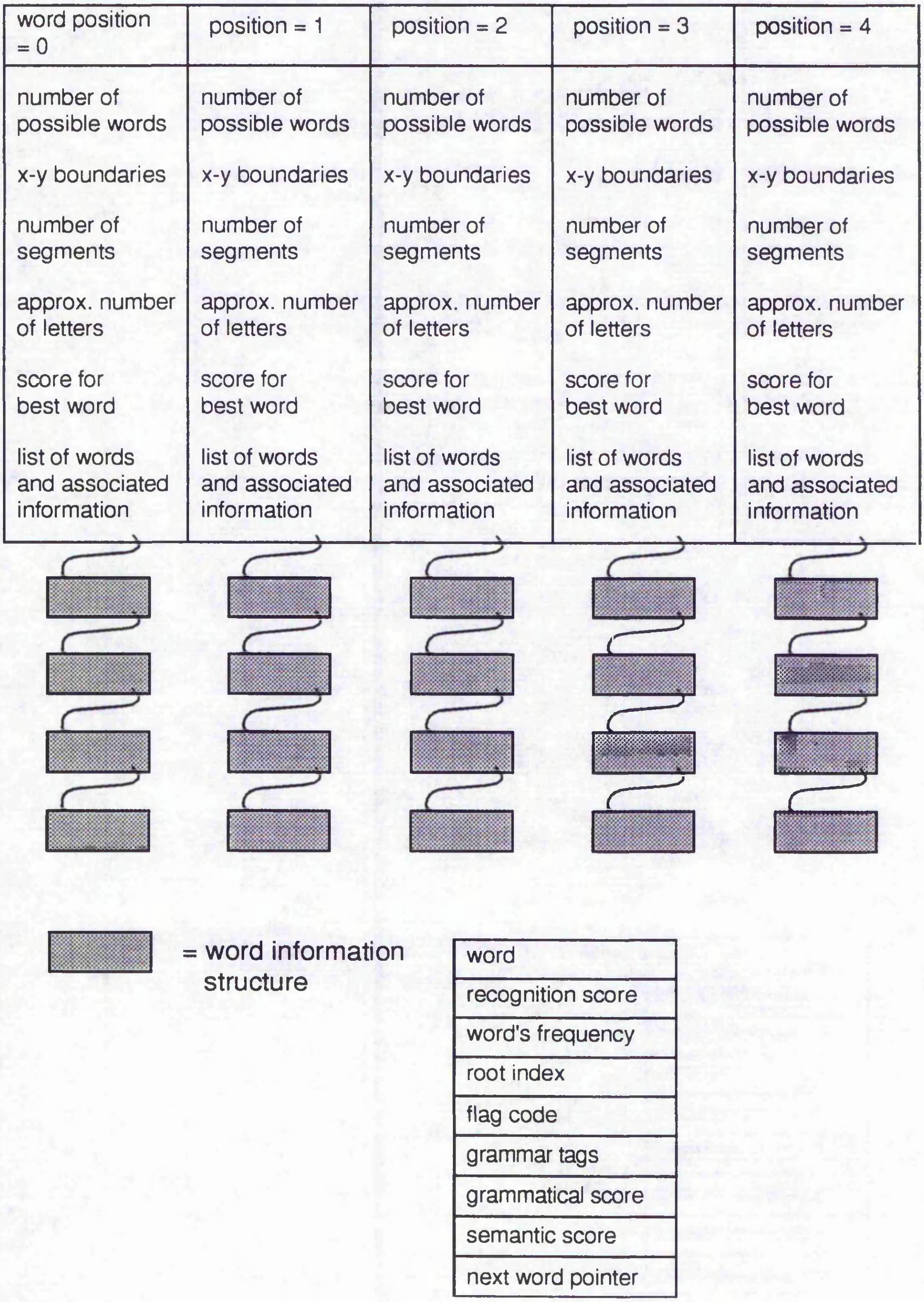
The same experiment using linguistic collocations instead of dictionary definitions reliably chooses the correct candidate. Certain words form distributional patterns in text, that is they frequently occur in combination with other words, these are the collocates of that word. Collocational information is derived from the LOB corpus. However the LOB corpus can only reasonably provide a collocation dictionary for some 7130 entires. A corpus much larger than the LOB would be necessary to gain sufficient information for a larger vocabulary.

Appendix C gives recognition results for a test sentence before and after semantic analysis.

4.5.3 Interface structure

The different levels of analysis described above produce further information about the remaining candidate words. The words themselves, and each piece of information associated with them, need to be stored in some suitable structure which is available to all levels of recognition. As a new piece of information is found, for example a word's grammar code is retrieved from the lexicon, it can be stored in this structure. Once some processing has taken place and perhaps a score has been calculated for a word, it can also be stored in this structure.

The current system holds a window of information about five word positions at any one time. This seems to be sufficient at present as it holds more than one word, so a number of positions can be compared, for example grammatical tri-grams can be checked, or semantic overlaps (across a maximum of four word positions) compared. More than five positions does not really provide any additional information, and of course requires more storage. The structure implemented consists of two parts: a header section and a list section. Figure 4.4 illustrates the structures.



word position = 0	position = 1	position = 2	position = 3	position = 4
number of possible words x-y boundaries number of segments approx. number of letters score for best word list of words and associated information	number of possible words x-y boundaries number of segments approx. number of letters score for best word list of words and associated information	number of possible words x-y boundaries number of segments approx. number of letters score for best word list of words and associated information	number of possible words x-y boundaries number of segments approx. number of letters score for best word list of words and associated information	number of possible words x-y boundaries number of segments approx. number of letters score for best word list of words and associated information

▨ = word information structure

word
recognition score
word's frequency
root index
flag code
grammar tags
grammatical score
semantic score
next word pointer

Figure 4.4 — Interface storage structure for candidate words and associated information

The header section contains information about a word position in general (e.g. numbered word position in the recognised text so far, total number of candidates found to be allowable at that position, co-ordinates of x,y bounding box and so on). The list section contains a list of the ten (or less) top ranked candidate words for that word position, along with grammar codes, scores from all levels, index number and so on.

4.6 Special characters

In addition to allowing both upper and lower case letters of the words from a chosen lexicon, the pattern recognition stage should also be able to recognise non-alphabetic characters. For example digits, punctuation and other special characters such as Greek letters and so on.

Marks of punctuation (full stop, comma, apostrophe, exclamation mark, question mark, hyphen) and other special characters (all kinds of bracketing, £,$,%,&,*,/,\ could be passed from the recogniser as "words" (i.e. separated between two spaces) in their own right, or as part of a word (i.e. between other letters, as the starting character of a string, or as the finishing character of a string). For example:

house,
done.
there!
(oops)

and/or
b*****d

In these situations a lexical look-up is obviously inappropriate. Separate characters are easily accounted for, as are special characters at the beginning or end of words, they are simply ignored for look-up purposes. It is more difficult if they occur between letters (e.g. *and/or, b*****d*). It may be possible to check for the existence of a word before the special character, and another afterwards. For hyphenated words and apostrophes, certain rules can be applied. For example *tea-cup*, this could be allowed if the string preceding the hyphen is a word (found by the look-up) and also the string following the hyphen. So *tca-cup* would not be allowable. If an apostrophe is found, only a few strings could follow it, so these can be checked for (e.g. *n't, 's, 're, 'll, s'* and so on). In fact the kind of strings apostrophes occur with are also from a relatively small set. Strings such as *won't, can't, they're* and so on could actually be included in the look-up tree and flagged as requiring an apostrophe between two letter positions.

For *'s, s'* etc, (i.e. possessives, both singular and plural, and *it's*) this would be a simple check to see if the string preceding the apostrophe could have such a possessive (i.e. it is a noun or pronoun — e.g. *the dog's bone, my sister's house, the boys' sweets*).

Similarly there are a small number of cases where digits are found mixed with letters (e.g. 1st, 2nd, 3rd, 4th, 126th and so on). These could be checked. Strings such as *hou5e* would have to be ruled out (as *5* often confuses with *s*). A string consisting of all digits (i.e. a sequence of digits between two spaces) should also be allowed, for telephone numbers, addresses etc). Any other sequence should not be allowable.

Other strings which should be checked for in the system could be Greek characters (again these are acceptable if found on their own, or if the sequence is a formula for example, but should not be allowable if they are mixed with alphabetic characters). Similarly for Roman numerals e.g. MCMLXXXIX, the problem here is of course that Roman numerals look exactly like any sequence of upper case letters, and would therefore be passed for lexical checking. Sequences such as post codes could be checked against their own particular syntax. For example NG9 1PR is acceptable, but NG67 is not because 67 is outside the range of acceptable area digits for Nottingham.

Some algorithms have been implemented as part of the current recognition system. These accept some special characters and some sequences of "mixed" characters. For example, the following would now all be allowed by the system:

word 1234 (word) [1234] < > 63% % word-word

4.7 Conclusions

This chapter has shown that recognition can be improved by using additional linguistic information. This includes compound words, common phrases and idioms, syntax and semantics. A method which deals neatly with the potential problem of morphology has been presented. This used existing and available information from a machine-readable dictionary source to develop an indexing system. These indices are stored in the lexical look-up tree, along with other information (such as grammatical category) required by higher levels of analysis. The look-up tree is an ideal site for integrating different types of information because it provides the interface between the low level pattern recognition process and higher level linguistic processes.

The machine-readable dictionary also contains details of compounds and commonly used phrases. The use of such structures to aid recognition was investigated and an initial method has been implemented which employs another trie data structure to hold compounding information. A system of coded flags has also been discussed which provides access to compound and case information. The inclusion of other characters such as punctuation and digits has been considered and some simple cases included in the system.

Methods for implementing techniques which make effective use of higher level information have been described. A suitable structure which aids the integration process by storing the required and produced information has been established. Attempts have been made to combine all such information in some meaningful way for this particular application of handwriting recognition, and results have been presented which show a significant improvement in recognition.

Chapter Five

Combining Sources of Information

5.1 Introduction

The previous chapter discussed the integration and interface between the different levels of analysis of the handwriting recognition system. This included details of the information needed by each level of analysis, and the additional information produced by that level. Such information needs to be considered together when selecting the most likely word from all candidates, which implies that the information must be combined in some way. So far it has been assumed that levels of processing within the system operate serially, from the pattern recognition, to the lexical look-up, then onto syntactic and semantic analysis. In other words information is collected at the pattern recognition stage, and all further levels select from this. This need not be the case, since each level has information of its own to contribute to the overall picture, as well as selecting from the existing information. The current chapter will discuss ways in which all levels of analysis can suggest alternative or additional candidate letters, words (or perhaps phrases) in order to supplement information from the pattern level. The discussion will include areas where the pattern recogniser could be improved, and will describe methods used to implement some of these ideas. In particular these ideas will be applied to existing errors within the system. This will necessarily involve some interaction between the different levels of analysis.

Psychological models of reading propose interaction of levels of information in the human language processing system. In particular, Morton's "Logogen Theory" (Morton, 1969), Marslen-Wilson's "Cohort Theory" of word recognition (Marslen-Wilson and Welsh, 1978), and Rumelhart and McClelland's "Interactive Activation Model" (McClelland and Rumelhart, 1981; Rumelhart and McClelland, 1982) are relevant here.

5.1.1 Psychological theories of word recognition

According to Logogen Theory, each word in the mental lexicon is assumed to have a "logogen". This is a theoretical entity that contains a specification of the word's defining characteristics. Word recognition occurs when the activation of a single lexical entry (i.e. a logogen) crosses some critical threshold value. Logogens accept input from both bottom-up sensory analysers, and top-down contextual mechanisms. Both sensory and contextual information interact in such a way that there is a trade-off between them; the more contextual information input to a logogen from its top-down sources, the less sensory information is needed to bring the logogen above threshold for activation.

Cohort Theory views word recognition (for speech input) as a bottom-up process of eliminating possible candidates by de-activation. This is in contrast with its predecessor (Logogen Theory) which assumes activation of only a single lexical item. According to Marslen-Wilson and Welsh, a set of potential word candidates (the "cohort") is activated during the earliest phases of the word recognition process solely on the basis of bottom-up sensory information. That is, all words sharing the same initial sound characteristics become activated in the system. As the system detects mismatches between initial bottom-up sensory information and the top-down information about the expected sound representation of words generated by context, inappropriate candidates within the initial cohort are de-activated. A word is said to be recognised at the point when a particular word can be uniquely distinguished from all other words in the cohort. As in Logogen Theory, word recognition and the subsequent lexical access are viewed as a result of a balance between the available sensory and contextual information about a word at any given time. In particular, when de-activation occurs on the basis of contextual mismatches, less sensory information is therefore needed for a single word candidate to emerge.

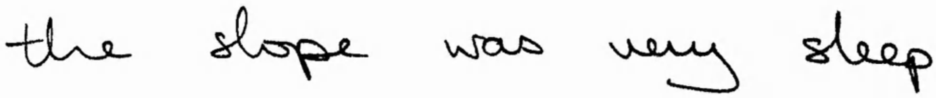
In Rumelhart and McClelland's Interactive Activation Model, perception results from excitatory and inhibitory interactions of detectors for visual features, letters, and words. The central feature of this model is that the processing of information in reading is assumed to consist of a series of levels. Each level is concerned with forming a representation of the input at a different level of abstraction. Information flows in both directions at once – from lower to higher levels and from higher to lower levels. A visual input excites detectors for visual features in the display. These excite detectors for letters consistent with the active features. The letter detectors in turn excite detectors for consistent words. Active word detectors mutually inhibit each other and send feedback to the letter level, strengthening activation and hence perceptibility of their constituent letters. Context also aids perception. Hence perception is fundamentally an interactive process. So top-down (or "conceptually driven") processing works simultaneously and in conjunction with bottom-up (or "data driven") processing.

The current script recognition system processes information in a similar fashion to that of Rumelhart and McClelland's model. The pattern recognition stage takes input of a visual kind and extracts features, which are passed on to letter and word levels of processing. However there is no feedback in our system at present. A study of the above model suggests strongly that such a feedback mechanism would facilitate recognition, if some suitable implementation could be devised. Taking this further, interaction with higher contextual knowledge is also possible. Higher levels of syntactic and semantic processing can provide their own information to add to existing word level information. This information may confirm existing candidates (excitatory connections), or may conflict with existing candidates (inhibitory connections). Our word information would have to be stored in some appropriate fashion to facilitate "activation" of certain words from both directions, i.e. from both the contextual and the feature/letter levels.

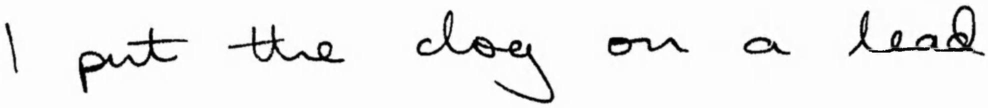
5.1.2 Examples of the use of higher level knowledge

As we have seen so far, the results from pattern recognition are far from perfect, and no matter what improvements are made to recognition, it is never going to reach 100% accuracy. People can write words so that they look like other words, or don't look like any word at all, so the correct word can only be found from the surrounding words in context. For example:

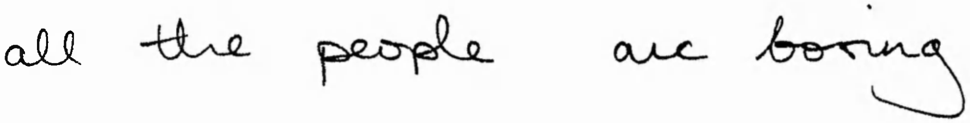
1. "The slope was very sleep" (uncrossed *t*)



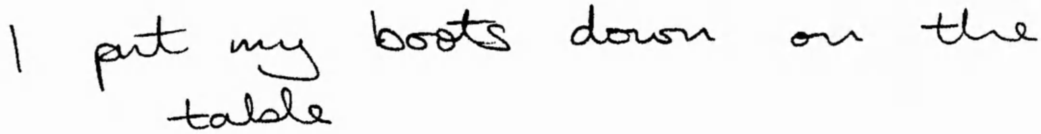
2. "I put the clog on a lead" (badly written *d*)



3. “All the people arc boring” (badly written *e*)



4. “I put my boots down on the table” (meant *books*)



In example 1 above, the *t* of *steep* has been left uncrossed, making the word look like *sleep* instead, and in example 2 the word *dog* looks like *clog*. Even a reliable pattern recogniser could only find a word which best fits the script as it appears, having no knowledge to suggest that that word is incorrect. A good system needs to be able to detect when errors occur, and hopefully also be able to employ some means of correcting them, or at least attempting to correct them. Higher levels of analysis can provide extra knowledge, both to aid detection of errors, and to make alternative suggestions for possible correction of a detected error. However, using information from more levels of analysis means finding some suitable way of combining that information.

Example 1 above should be detected as an error by analysis of the grammatical categories of the words. Syntax suggests that an adjective should follow the word *very*, but *sleep* is either a noun or a verb. The sequence *very steep* is (syntactically) 1,500 times more likely to occur than *very sleep* (from transition frequency counts of the LOB corpus). Similarly for example 3, the words *people* and *arc* are less likely to occur together than *people* and *are*. However for examples 2 and 4, syntactic analysis would not identify any problems, because they are grammatically acceptable. Example

2 does not really make sense (although it is possible to think of situations where this might occur), but example 4 does. Semantic analysis of these examples might be able to identify a problem with 2, but example 4 would probably pass through all processing and be accepted. There is no way of knowing that the writer intended to write the word *books*, but by a lexical substitution error wrote *boots* instead.

Given the current recognition system, it is most unlikely that such examples will occur in this form. As we have seen in previous chapters, it is infrequent that allowable candidate words are unique. From a sample of 106 words, approximately 13% were unique (12% gave no allowable words, 27% had between 2 and 5 allowable candidate words, 33% between 6 and 20 candidates, and 15% more than 20 candidates). Consequently there will be other candidate words at most word positions, and all combinations of words must be considered. Given a choice between *sleep* and *steep* for example 1, both syntax and semantics would choose *steep*. Similarly for example 3, syntax would choose *are* over *arc*, and for example 2, semantics would choose *dog* rather than *clog*. Example 4 still remains a problem, but it may be that *books* could be suggested as more likely than *boots* from semantic co-occurrence information, or from frequency of use from a corpus (*books* is five times more likely to occur than *boots*), especially if weighted from a count of occurrence in the script recognised so far. However these choices will only occur if the correct words are contained in the list of alternative candidate words, which may not be the case. It must also be noted that preceding word positions may have been recognised incorrectly, given that most positions have alternative candidates, and often the correct word is not first in the ordered list. This means that the situation is more ambiguous than as just described. Whether the alternative candidates confuse the situation more often than they aid it can only be found from adequate testing of the system. Table 5.1 shows sample recognition results for examples 1 to 4 discussed above.

The intended words for examples 1 and 4 (*steep* and *books*, respectively) did not occur as alternatives. In example 2 the intended word *dog* was the second candidate word, and in example 3 the intended word *are* was the top candidate, actually rated higher than the "correct" word *arc*. The intended words might be able to be suggested for examples 1 and 4 if a method of whole word recognition were employed, because *sleep* and *steep*, *boots* and *books* have the same overall shape.

Traditional spelling error detection and correction data is for typed input. For handwriting there are problems not only of spelling errors, but also illegible words. For recognition, a third problem is idiosyncrasies of the recogniser. Errors in the current script recognition system must be studied in detail in order to establish whether they can

Table 5.1 — Recognition results for deliberate errors

Word	Rank of correct word	Total no. allowable cands.	Top candidate	Next few candidates
the	1	14	the	for too few ten tea tore
slope	2	4	hope	scope dope
was	1	28	was	race voice has iris overs oars
very	2	44	way	ray vary ivory ivy only wiry
sleep	5	36	deep	shop keep ship slap sloop sheep
I	1	2	I	l
put	2	6	but	port pat fall tall
the	1	9	the	too till fill toil
clog	1	7	clog	dog slog clay day slay
on	1	4	on	oh or oil
a	1	2	a	n
lead	1	12	lead	load bad local
all	1	8	all	ale ace cill cool oil
the	1	27	the	for few too ten tea tell
people	1	2	people	pebble
arc	2	22	are	air our own core corn cove owe
boring	1	4	boring	boiling bowling poring pouring
I	1	2	I	l
put	2	46	pat	port but bat foot roof woof out
my	1	5	my	ivy wig rug wry
boots	1	1	boots	
down	1	6	down	clown colour odour clover dover
on	1	21	on	or
the	1	10	the	tell top thy fog she
table	1	5	table	fable feeble foible tousle

always be identified. If they can, then effective error correction methods can be considered. These may be standard algorithms, or alternative techniques may be suggested by the specific types of error produced in the system. For example there may be further information available at the pattern level, or provided by higher levels of analysis, which will be able to suggest alternative candidate words for the position in error. If an effective method can be found, then it may be possible to utilise the same information to aid the choice of candidate words in positions where no error has been found.

In summary, the areas to be addressed in this chapter are:

- the identification, or detection of errors in the system;
- the correction of errors if any have been detected;
- finding additional information which can aid both of the above;
- combining this information in some meaningful way.

5.2 Errors

5.2.1 Introduction

Given the lexical look-up of the script recognition system, as described in earlier chapters, there are cases where the system will not be able to suggest any allowable candidate string. There will also be cases when a stray candidate from the lexical look-up is rejected by the later stages of analysis. There are various possible reasons for these failures, including errors of recognition and errors of spelling. Such errors could also produce an incorrect but acceptable candidate. This could only be detected and corrected by the user. The system can only attempt to deal with possible errors of recognition or spelling. Mis-recognitions and errors of spelling are impossible to distinguish from each other, given the nature of the data output from the pattern recogniser, however they should be detected and corrected if possible. The following sections discuss accepted literature methods of error detection and correction.

5.2.2 Traditional methods for detecting errors

Humans, when reading, often miss spelling errors, because their expectations of what the text should say, influence the visual system, so that they read the intended word, rather than what is actually in front of them. This is especially so for handwriting, when not only spelling errors, but also illegible words can be deduced from the context.

With the increase in word and text processing computer systems, programs which check and correct spelling have become more and more common (see Peterson, 1980 for a review). A standard method of spelling error detection (also known as checking or verifying) is a deterministic approach by look-up in a table or word list (Bledsoe and Browning, 1959; Shinghal and Toussaint, 1979; Srihari, Hull and Choudhari, 1983; Bozinovic and Srihari, 1982; Berghel, 1987). If the word is not found then it is said to be misspelled, so either correction is attempted, or it is returned to the user for verification.

Alternative techniques are probabilistic, using constituent analysis (n-grams or string segments), which are faster than lexicon search routines, but less precise. For example the unix facility TYPO checks sequences of di-grams and tri-grams and computes an index of peculiarity for each word — if a word contains several very rare di- or tri-grams, it is potentially misspelled (Morris and Cherry, 1975). Many other systems (Riseman and Hanson, 1974; Ullman, 1977; Hull and Srihari, 1982) use similar methods with grams, but it should be noted that the majority of these are designed to check the spelling of typed input. Any studies of handwritten spelling errors (e.g. Ellis, 1979; Wing and Baddeley, 1979; Mitton, 1987) were manually checked.

There are also hybrid techniques which use some of both look-up and constituent analysis. These include affix stripping routines. Constituent analysis is used to identify and remove legitimate affixes from word tokens, and a table look-up procedure is employed to determine whether or not the root of the word is correctly spelled.

Given an input file of text, the task of a spelling checker is to identify those words which are incorrect, but first it must perform some document normalisation. This includes the standardisation of words with regard to case (so "DOG" matches "dog") or to alternative spelling (so "judgment" matches with "judgement"), the removal of any formatting symbols, and sensible handling of digits, apostrophes, hyphens and punctuation symbols. For example hyphens functioning as delimiters are essential, whereas those signifying word breaks at the ends of lines are extraneous, and the two parts of the word should possibly be joined together.

A good spell checker has to minimise errors both of type 1 (a correct word is marked as incorrectly spelled), and of type 2 (an incorrectly spelled word is not marked). A checker that detects errors simply by lexical look-up will obviously fail to spot "real world" errors, such as *wether* for *whether*, i.e. where a misspelling has transformed one English word into another. The problem becomes worse as the dictionary gets larger. A checker that did not have *wether* in its lexicon would flag *wether* as an error (correctly or not), but one with a comprehensive dictionary would fail to do so. Interestingly, it appears that if a word is misspelled, any errors will usually be later in the word, the first letter is usually correct (Mitton, 1987). The few first letter errors which do occur are in words with silent consonants, for example *know* and *write*. Errors are often phonetically based, so *f* may be written for *ph* and so on. Another study of typing errors showed that lower frequency letters are more likely to be replaced by higher frequency letters (Grudin, 1983).

5.2.3 Classification of spelling errors

Damerau (1964; see also Peterson, 1980; Ullman, 1977) states that 80% of errors are the result of the following four types of error (for typescript). Presumably the remaining 20% are formed from combinations of these four classifications:

1) transposition of two letters;
2) one extra letter, or insertion;
3) one missing letter, or omission;
4) one wrong letter, or substitution.

The following table shows some examples of these errors. Resulting error type A means that the intended word is transformed into another English word (a “real world” error, whereas type B is where it has been transformed into a non-English string.

Table 5.2 — Examples of errors

Error type	Intended word	Resulting error A	Resulting error B
1	cast	cats	
1	received		recieved
2	breath	breathe	
2	hopeful		hopefull
3	breathe	breath	
3	change		chage
4	care	case	
4	student		sludent

Salmina and Khodashinskii (1986) give the same four classes of errors (their examples being from Russian typescript) and include approximate percentages of occurrence of the four types:

1) transpositions	10 - 15 %
2) insertions	25 - 35 %
3) omissions	30 - 40 %
4) substitutions	15 - 20 %

These are the most frequently occurring errors, and infrequent errors such as combinations of the above, account for approximately 4 - 9 % of the total.

Wing and Baddeley (1979) used the same four classifications for handwritten errors:

1) transpositions	3 %
2) insertions	13 %
3) omissions	49 %
4) substitutions	36 %

5.2.4 Methods for error correction

As we have seen, generally some kind of n-gram or lexical check is used for error detection. Spelling correction algorithms usually suggest a few alternative words which are in some sense similar to the detected misspelled word. A mathematical function grades how different these suggestions are from the misspelling, and the nearest few are suggested.

Given Damerau's four main error types, it is possible to approach these situations in order to find potential correct words. For each type of error, an algorithm can be found which will attempt to find the intended word. For a string of length *m* characters, there will be an additional *4m* searches of the lexicon.

Table 5.3 — Algorithms for error correction

Error type	Correction algorithm	No. searches
transpositions	transpose each adjacent pair of letters	m - 1
insertions	remove each letter in turn	m
omissions	add a "wild card" between existing letters, also at the start and finish	m + 1
substitutions	substitute a wild card for each letter position	m

Peterson (1980) explains that most misspellings can be generated from their correct counterparts by using these four rules, and in fact form the basis of the DEC-10 spelling corrector. The resulting strings produced by applying the rules are searched for in the lexicon in the normal way. Thus a candidate list of possible words is formed by multiple searches of the lexicon. The search techniques can be improved, for example by using a lexicon which is indexed by length. For each of the additional searches of the lexicon, the length of the required string is always known, so searching the whole lexicon is wasteful. However this also depends on the chosen memory structure for the lexicon. These tend to be standard methods such as hashing (§5.4.3), trees or tries (see chapter 3). Peterson (1980) comments that spelling correction is not cheap, but then neither is it prohibitively expensive, and it is not normally needed. It is only employed for the word tokens from the input text which have not been found in the lexicon.

Common literature methods (Tappert, 1982; Hall and Dowling, 1980; Berghel, 1987) have used string matching techniques, (e.g. nearest match methods, approximate string matching) to find candidates for intended words once an error has been identified. However most of these assume that misspelled words have been identified, and concentrate on methods of comparison of the word in error to a number of candidates.

Factors such as the number of letters different, and word length are often involved. The algorithms suggest one candidate to be more likely than the others.

Spelling correcting programs can be interactive. When a misspelling is identified, it is highlighted, and a number of options are available to the user. The program can suggest a list of alternative words and allow the user to choose a substitute, the user can edit the file to correct the word, or the user can confirm that this is in fact a correctly spelled word, and should be added to the program's dictionary. Thus correction involves substituting the correct spelling of the intended word for its misspelled counterpart, but controlled by the user. It would be undesirable for this process to be fully automated.

A spelling corrector must of necessity use a lexicon. Typically several lexical lists are used, especially when primary storage is at a premium. The lists are arranged in a hierarchy (Peterson, 1980), for example there may be a small, static lexicon of very common words (perhaps 100-200), and a dynamic small to moderately sized document specific lexicon (perhaps 1000-2000). In secondary storage there will be a large, static lexicon of anything between 10,000 to 100,000 words.

Having found a set of words from the lexicon which may be the correct spelling of an identified misspelling, common correction techniques compute an index of matching for each candidate word. This can be thought of as a probability measure that the word token in question resulted from a misspelling of a dictionary word. Berghel (1987) distinguishes between three types of orthographic similarity. They are firstly, positional similarity, a relation referring to the degree to which matching characters in two strings are in the same position. Secondly, ordinal similarity, a relation referring to the degree in which characters in two strings occur in the same order. Thirdly, material similarity, a relation referring to the degree to which two strings consist of the same characters.

These three classes fit neatly with Damerau's four types of errors. Substitution errors are positional, transpositions are material, and both insertions and omissions are ordinal. Berghel goes on to explain that positional similarity is too narrow for spelling correction, whereas material similarity is too broad. Ordinal similarity is the one that many algorithms have employed, for example Soundex (Odell and Russell, 1918 and 1922).

Additional information can often be used to increase correction accuracy and speed, for example by studying the sources of errors. For typed input this can mean knowing the layout of the keyboard, because keys close to each other are more likely to have been substituted, inserted or transposed. For true spelling errors (rather than typographical mistakes), a corrector which knows something about pronunciation will do better than one without, because a writer who is unsure of a spelling will probably attempt to spell a word as it sounds (Mitton, 1987). The following sections analyse the types of errors found from test data from the script recognition system, and discuss the feasibility and effectiveness of applying standard spelling correction techniques.

5.2.5 Analysis of errors from a recognition system

The following table of classifications of errors from test recognition data shows that it is impossible to tell what type the original error was. A word can be misspelled, mis-recognised or mis-written, but all three cases appear the same when looking at the pattern recogniser output. These examples are from ten samples of two test passages (6 samples of 57 words, and 4 samples of 79 words, i.e. a total of 658 words). 36 words were incorrectly recognised. The strings in the recogniser output column are the calculated top ranked candidate string for each word. No words were found to be allowable in these positions, so instead the top-ranked string is stored.

Table 5.4 — Classification of errors from recogniser

Intended word	Recogniser output	Type of error
students	sludents	substitution
copying	copylng	substitution
forthwith	forthw-ith	
in	ln	substitution
departmental	deparl-mental	
that	l\al-	
desist	desisl-	
authority	authoritvj	
action	actiin	substitution
occurrence	occurence	omission
must	musl	substitution
in the	inle	
county	cou\nty	
breaking	brec\king	
has	ha?	unknown char
occurrence	occu\rence	
liable	lia\sle	
thus	lhus	substitution
the	fhe	substitution
software	soltw\are	
this	thi?	unknown char
are	\are	
departmental	deportmental	substitution
would	w-ould	
are	lre	substitution
forthwith	forthwi\h	
remind	r\emi?d	
are	ore	substitution ?
brought	bro?ght	unknown char
council	councill	insertion
orders	ords	
has	hus	substitution
been	beeh	substitution
future	futule	substitution
will	vvill	
disciplinary	disciiplinary	insertion

Of these 36 errors, all but one were detected because the initial lexical look-up found no other candidate string to be allowable at each of these word positions. The one case where another string was allowable is where the intended word *are* was recognised as *ore*. However because *ore* is a noun and *are* is a verb, this error would most likely be spotted by the syntactic analysis, because *ore* would not fit into the same word position as *are* in the sentence.

It can be seen from the table above that the most frequently occurring errors are of the substitution type. Of the 36 errors, there were 13 occurrences of the substitution error, which is a third of the total. There are also two errors of insertion of an extra letter, and one of omission of a letter. This gives a total of 16 out of 36 which could be solved using standard error correction algorithms. Including the four cases of an unknown character, this gives 20 out of 36 cases which could be solved. The remaining 16 occurrences are mostly combinations of at least two of the described problems. It is also interesting to note that for 33 out of the 36 errors, the top rated candidate strings have the same overall word shape as the intended words (see §5.3.3.2 for further discussion of word shape).

It should be noted here that the recognition output such as "\" and "-" are strokes of the pen which the recogniser has been unable to join to any other stroke to give a possible letter. In fact they are produced by ligatures which were missed during pattern recognition when ligatures are removed from the sequences of Freeman vectors. The character "?" is the recogniser's unknown character, a stroke which does not match with any vector encoding in the database.

5.2.6 Unmatched characters

The character "?", or "wild card" character (mentioned previously in §3.4.1) is sometimes given as output from the pattern recogniser for letter positions where the recognition could not match with any known character encoding. A fairly simple algorithm has been implemented to search for possible alternatives for these letter positions, given the surrounding candidate letters (as detailed in §3.4.1). The initial results from this implementation show it to be effective in many cases, but rather counter-productive in others. In the example passages of test data discussed in the previous section, four out of the 36 words in error contained an unknown character.

Three of these four cases can be solved by the implemented algorithm, as shown in the following table.

Table 5.5 — Cases of unknown characters

Intended word	Recogniser output	Solved ?	No. of candidates found
has	ha?	yes	9
this	thi?	yes	2
remind	r\emi?d	no	0
brought	bro?ght	yes	1

It was found that by removing a backslash "\" from a candidate string after the letters *r*, *v* and *w* gave an allowable word. This changes two of the above 36 errors. One of these (*software*, recognised as "soltw\are") would now become a simple substitution error, and the other (*remind*, recognised as "r\emi?d") would become a simple unknown character problem, with only one word in the lexicon which matches the pattern. This means that all four cases can now be solved. A remaining problem with substitutions for unknown characters is that the resulting candidates cannot be ordered by likelihood of being correct, because the word could equally well be any of them. For example for "ha?", if *had, hag, ham, has, hat, haw* and *hay* were found to be candidates, it would be left to further stages of analysis of the system to determine which of these is more likely to be the intended word, although they can be suggested in order of their frequency of use calculated from a corpus of English text.

5.2.7 Detection and types of errors in our system

Looking at an example from recognition test data, an original handwritten word was *students*, but was recognised as *sludents*. It is possible to imagine that the person writing it didn't cross the *t*, which is a common occurrence in handwriting. Should that be classified as a misspelling, or was it just "wrongly" written? There is certainly a theoretical distinction, but from the practical view of attempting correction, there probably is not.

As far as the script recognition system is concerned, the cases which suggest that a word is an error, are where the lexical look-up gives no allowable candidate strings, or all candidate strings are rejected by further stages of analysis. Thus for detection of errors, there are two situations:

1) no allowable candidate strings;

2) candidates rejected by higher levels of analysis (syntax and semantics).

The reasons for these two situations arising could be because the correct word is not in the word list, but this in fact accounts for very few cases. Or it may be because the recogniser has for whatever reason, not suggested the correct letters. This could be because the word was misspelled in the original script (user dependent error), or due to mis-recognition of at least one character (recogniser dependent error). Thus there are three reasons for errors occurring:

1) misspelling

2) mis-recognition

3) correct word is not in the lexicon

There is a theoretical distinction between (1) a misspelling and (2) a mis-recognition, but the results from recognition will not enable us to distinguish between them, and indeed they can be treated similarly for attempted correction. It is feasible that there will be situations where words that are misspelled or mis-recognised are actually transformed by the error into other words, (i.e. resulting in error type A — §5.2.3) which could fit into the sentence. It is uncertain how likely this is to occur, but the possibility seems remote. Of the 36 errors discussed above, only one gave a "real world" error. In such cases the system will be unable to spot any error as they will pass successfully through all stages of analysis. These could only be corrected by the user. It is also not clear whether there is a direct relationship between the two cases where errors can be detected, and the three reasons for errors. This is another interesting theoretical point, but probably of little consequence as far as error correction is concerned.

5.2.8 Application of traditional methods to the recognition system

It is not certain whether it would be worthwhile to apply common methods of coping with misspellings to our recognition system. Initial experiments have suggested that most traditional methods would not be immediately applicable, because of the nature of the original data from the pattern recogniser. However by applying combinations of other algorithms, it may be possible to take some relatively simple steps towards improving existing recognition rates by some kind of error correction system. For example the letters *l* and *t* are often confused with each other, so *t* could be substituted for all occurrences of *l*, and *l* could be substituted for all occurrences of *t*.

If the algorithms for correction of the above four types of spelling errors were to be applied to our system, it is unlikely that applying them to the highest-rated allowable string (if there are any allowable strings) is going to find the intended word. However it is not clear whether they should be applied to the complete list of allowable words, or to the list of candidate strings, or indeed to some subset of either. The major problem in this situation is the number of candidates involved. If correction techniques are employed upon the list of allowable strings, the intended word may still not be found because it is sufficiently different from the entries in the existing list of allowable strings. In many cases this list already numbers over 100, so correction techniques applied to such a long list would probably take too long to be worthwhile implementing, and the results from such techniques would produce many more candidates which would all have to be processed through the remaining stages of the recognition system.

Alternatively these techniques could be applied to every one of the candidate strings for each word position identified as a potential error (again it should be noted that this assumes that errors can be identified, which may be doubtful for our system). However we have already seen (see Appendix B) that the total number of candidate strings can number in the thousands, so any correction techniques applied to these would reach explosive proportions in terms of time taken and number of additional candidates produced. Some experiments must be tried on words of varying length to establish how explosive the problem really is. To apply such techniques to a relatively small subset of candidate strings is a preferable solution, but some grounds for deciding on a subset would have to be established.

This could be done by using the ranks of the candidate letters to direct correction. Using the ranking (or confidence) information for the list of candidate strings, we can reduce the problem somewhat, by trying the standard four error correction approaches on just the top 10 candidate strings. It may also be feasible to look at individual letter confidences to determine an ordering of letter positions in which to apply the substitution algorithm. Whether this would help to solve the problem is uncertain, and some initial experiments would have to be undertaken to establish its effectiveness. Just because the recogniser has little confidence in a particular character need bear no resemblance to whether or not that is the incorrect character in a misspelled word. However it may help with badly written (and therefore mis-recognised) words.

Given the problems explained above, it may be difficult, and in some cases counter-productive to attempt any correction of errors within our system, however initial investigations suggest there may be particular situations where correction can be attempted and is in fact useful to improve recognition rates.

Looking at the remaining 16 error cases from our system (see table 5.4) in more detail, it is clear that some of these can in fact be solved by applying some simple substitutions where the recognition has failed. For example *will* was recognised as "vvill" (top candidate), so substituting *w* for "vv" would provide the correct result. Similarly *departmental* was recognised as "deparl-mental", so looking for sequences such as "l-" and substituting *t* would give the correct result (also *y* for "vj", *a* for "c\", and there may well be others). In fact simple substitutions of characters such as *l, i, t, f,* and "\" for each other is an effective first attempt to find correct words, because these characters frequently confuse with each other.

Furthermore, if the characters "\" and "-" are ignored completely, another four examples could be solved (i.e. forthw-ith, cou\nty, \are, w-ould). This gives a grand total of 30 out of 36 errors which could be corrected, by using simple algorithms as a first attempt before going into more complicated algorithms which may be impractical. Such impracticality could perhaps be measured as a function of word length. If the word is short (i.e. less than some lower bound) then attempt some correction. If the word is too long (i.e. longer than some upper bound) then do not attempt correction, in which case the word might be sent back to the user to be re-written.

The important point to note as a summary here, is that the algorithms suggested for solving 30 out of 36 errors, would find the correct (i.e. intended) word by looking only at the top ranked candidate string, which neatly solves the problem described earlier of what to choose as a starting point for possible error correction, and avoids the potentially explosive situations.

5.2.9 *Alternative methods for correction of errors*

As explained above, attempting correction on all candidate strings would reach explosive proportions in most cases, but correction techniques can be directed, in order to reduce the number of strings on which the algorithms are tried.

A completely different approach for solving user dependent errors (misspellings) involves inserting common misspellings into the tree of the lexicon (Peterson, 1980). Such entries would have to be flagged in the tree structure to show that they are misspellings to distinguish them from correct words (as mentioned previously in §4.4.3 during the discussion of the flagging system and the 12 codes necessary to represent proper nouns, compounds and phrases). Thus *greatfully* would be flagged as a misspelling of *gratefully* and *prehaps* as a misspelling of *perhaps*. A misspelled word included in the tree would have the index of its corresponding correct word. For example *recieve* would have the same index as *receive* and so on.

As Peterson (1980) explains, “this approach has not been included in any current spellers, probably because of the lack of an obvious source of known misspellings and the low frequency of even common misspellings”. Any list of common misspellings would most likely have to be collected by hand. Some could perhaps be collected from the input to currently available spelling checkers, as long as the words are verified as worthwhile including as common misspellings. How much this would increase the size of the tree structure and the speed of look-up, is not clear because it depends upon the number of words stored.

For example, the following misspellings of *accommodation* may occur: *accomodation, acommodation, acomodation, accomadation, acommadation* and *acomadation*. This makes six extra words, at least, but they are not equally as common. Some criteria for deciding what to include and what not to include would have to be established. It is also very difficult to say how successful this would be within the current recognition system.

5.2.10 Conclusions

The above discussions have noted that in general, error detection and correction are difficult techniques to implement, especially given the ambiguous nature of script recognition data. However, by studying individual errors, it has been demonstrated that in fact simple algorithms can be implemented which significantly improve recognition rates. For example from the sample data of 658 words, there were 36 errors (i.e.recognition rate following lexical look-up of 94.5%, taking up to ten allowable strings per word position). If 30 of the 36 errors can be corrected, this gives a recognition rate of 99%.

Without knowledge of the particular type of error, any attempted correction is bound to give the incorrect solution in some cases, but more detailed investigation is necessary to establish whether this proportion is significant.

Heuristics are obviously needed, and in fact looking at the types of errors actually found in recognition data, some suggested ones would appear to be quite successful. Algorithms such as reversing sequences of *ie* to *ei* following a letter *c* are effective first attempts before going on to more complicated techniques, or even instead of such techniques which may turn out to be counter-productive in terms of time taken and the number of candidates produced. The errors from the recognition system can usually be detected, and are often solved by simple methods of correction due to prior knowledge of the types of recognition errors found. More traditional error correcting techniques can be used as a last resort, and if they are applied to just the top-rated candidate string this should avoid the potentially explosive problem.

A combination of inserting common spelling errors into the tree structure of the lexicon, and using heuristic methods, should be particularly effective.

5.3 Interaction between levels of analysis

5.3.1 Introduction

The preceding section discussed traditional error detection and correction methods, mostly applied to typographical spelling errors. It is uncertain whether they are directly applicable to our system, where the majority of errors are specific to the pattern recogniser. Only a few of the errors are of the same kind as spelling or typing errors. With knowledge of the particular error, it is possible to implement some correction algorithms. However this knowledge is not available in our recognition system. The indication of a possible error is if the lexical look-up produces no allowable candidate strings, or if all candidates at a word position are rejected by further stages of analysis. Thus we can detect at least some of the errors — others may slip through un-detected if one of the candidate words (although incorrect) fits into the sentence.

The following discussion investigates improvements which could be made to augment the pattern recognition information. Taken together with higher level knowledge, these could provide us with an alternative technique for correcting errors within the system. It may also be possible to use higher level knowledge to contribute additional information rather than merely selecting from existing candidates.

Currently, the recogniser does not fully exploit information about the physical properties of the input. That is, it does not directly make use of information about the length of words, or information about the overall word shape. The recogniser codes the input as letter strokes which are then combined to produce possible characters. Physical size and position of strokes are not incorporated into the coding scheme. Thus information about the presence or absence of ascenders and descenders is only implicit and not directly derivable from the coded version of the input. Information about shape and size will of course be to some extent writer dependent. However, parameters for them for individual writers could be extracted from an initial training phase for a script recognition system. Additionally, length and shape need not be dependent upon absolute physical size but could be coded in a way that represented information about them relative to any individual input letter string. For example, the approximate number of characters per string could be used in later stages of processing, and information about the extension of characters relative to a middle zone could be calculated. Knowing word length and word shape is effective in reducing the number of possible words (Sinha, 1990); we need to know how useful information about them is when it is only approximate and uncertain.

Such information would be useful in a number of instances. Firstly, when the letter strings have been looked-up in the word-list, a number of word candidates remain. If it were the case that some of the remaining words were radically different from the approximate values for shape and length they could be removed from the list of candidates. Secondly, there are a number of different types of situation where there is missing information: either the recogniser produces no candidates for a letter string, or it produces some candidates. Where nothing is forthcoming for a word position, shape and length could be used along with higher level information to select potential words; where some characters are suggested these could be used in addition to this information; if only one or two letter positions in a word have no candidates, shape information could be used to select letter candidates, along with the restrictions provided by the adjacent letters. Finally, it may be the case that a number of word candidates are produced, but they appear to be incorrect. Again, knowledge about physical properties and higher level attributes could be used to select alternative words.

The syntax analyser currently operates using statistical information about the combination of sequences of grammatical categories. Thus for any word position it can produce predictions about the expected grammatical category for that position. Such information could be usefully combined with lower level information to help improve performance.

The following sections investigate the utility of these different sources of information, given that they will of necessity be uncertain. If they do appear to be effective for recognition, this will have consequences for the design of pattern recognisers.

5.3.2 Initial investigations

From the above discussion, two cases where improvement is needed have been identified:

Case 1 — to reduce the list of candidate words;

Case 2 — to suggest some candidates where none was found from the original data.

For Case 2, there are actually two sub-cases, although they can be treated similarly:

Case 2a — where no candidate words are found from the word look-up at all;

Case 2b — where none of the candidate words seem to fit with the syntactic processing of the sentence.

From initial investigations, it appeared that some measure of the number of letters in a word, and the word shape, would be quite restrictive for the list of possible candidates (for Case 1), as discussed below.

5.3.3 Case 1 — Reducing the list of candidate words

5.3.3.1 Word length

The initial investigations involved analysing samples of handwriting and calculating the effectiveness of the word length and shape information, assuming that such information were available. Table 5.6 shows a sample passage of handwritten text together with the recognition results for it, and the reductions in the number of candidates which would have been allowed, had a measure of word length been used.

This data is for a trained writer (ie. the Freeman vector database contains details of the writer's handwriting), and as can be seen, the number of candidate words can be high, and in such cases the spread of lengths of the candidates is quite wide. It would be desirable to be able to discard those candidates which are too short or too long to be the correct word — this would reduce the number of candidate words to be considered for further processing.

To achieve this aim a measure of approximate number of letters in a word is needed. As a first attempt we have tried to calculate this from the mean letter width for a particular writer, which can be obtained from the raw x-coordinate data for the script at training time. The mean letter width value is obtained by summing the x differences (x max - x min) for each word in the training set, and dividing by the number of letters written.

Table 5.6— Length distribution of candidate words

Word	Position	Total no.	Length distribution of candidates											No. candidates	
	in list	cands	2	3	4	5	6	7	8	9	10	11	12	Same length	±1
any	1	88		15	44	28	1							15	59
future	1	6					3	3						3	6
occurrence	1	1									1			1	1
will	1	77	1	30	44	2								44	76
make	1	40			10	18	12							10	28
the	1	18		7	11									7	18
person	1	3				1	2							2	3
concerned	1	5							2	3				3	5
liable	1	5					5							5	5
to	1	3	3											3	3
disciplinary	1	2											2	2	2
action	2	26			6	14	5	1						5	20
thus	2	131		3	24	49	18	4						24	76
the	1	26		7	19									7	26
procedure	1	1								1				1	1
must	1	9			3	6								3	9
cease	2	4				4								4	4

During look-up, the number of letters in a word of script is calculated by the x difference of the word divided by the mean letter width figure. Allowing for the actual word being within ±1 of this figure, we have a measure of approximate word length. The following table shows experimental results summed for four writers using this calculated mean letter width figure, allowing only those candidate words which are within ±1 of the calculated word length.

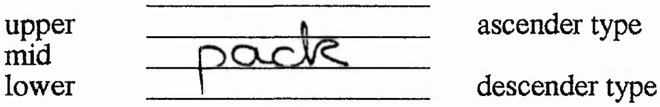
Table 5.7 — Reduction of number of candidates using length restriction

No. of words tested	No. of word candidates generated	Percentage of candidates discarded	Percentage of candidates incorrectly discarded
319	4259	18	0.25

The number of candidate words being discarded as too short or too long is quite small, and in fact the number of correct candidates being incorrectly discarded is slightly discouraging. The effectiveness of a measure of word length is very writer dependent, and alternative methods of calculating this figure more accurately and consistently need to be evaluated.

5.3.3.2 *Word shape*

It has been shown that overall word shape by some coding of ascending and descending letters relative to a mid zone (upper, middle and lower zone) is restrictive across a lexicon (Sinha, 1990). See also Appendix E for lexicon frequency counts of word shape.



Each letter of the alphabet is given a code (‘m’, ‘u’ or ‘l’ — corresponding to 1, 2, 3 after Sinha), so a word gets a complete code string. This code string can also be reduced, so only the changes are noted, sequences of the same code are reduced to a single code.

code ‘u’	⇒	upper zone	⇒	b d f h i k l t
code ‘m’	⇒	mid zone	⇒	a c e i m n o r s u v w x z
code ‘l’	⇒	lower zone	⇒	f g j p q y z

The coding of letters into the three categories should be a little flexible for letters such as *f, i,* and *z* which can be written in different ways, so a coding of ‘a’ was included to represent any zone. The following table shows some example words with their corresponding zone codes and reduced zone codes.

As can be seen, words such as *dog, frog* and *happy*, have different zone codes, but these all reduce to the same reduced zone code. If shape information were available accurately from the pattern recogniser it should enable more candidate words to be discarded due to incorrect shape. For example if we know that a section of script has an ascender close to the beginning of a word, and there are candidate words without one, then we can reject those candidates.

Table 5.8 — Example zone codings for words

Word	Zone code	Reduced zone code
cat	mmu	mu
dog	uml	uml
frog	umml	uml
happy	umlll	uml
sad	mmu	mu
window	mamumm	mamum
parrot	lmmmmu	lmu

Initial experiments showed that shape information is potentially very useful for reducing the number of candidate words (see Table 5.9).

Table 5.9 — Analysis of recognition results including shape information

Word	Position in list	Total no. of candidates	No. candidates matching exact code	No. candidates matching reduced code
any	1	88	1	3
future	1	6	1	4
occurrence	1	1	1	1
will	1	77	2	7
make	1	40	1	6
the	1	18	1	11
person	1	3	1	1
concerned	1	5	1	1
liable	1	5	1	1
to	1	3	2	2
disciplinary	1	2	1	1
action	2	26	2	6
thus	2	131	4	16
the	1	26	3	10
procedure	1	1	1	1
must	1	9	2	4
cease	2	4	1	1

The above table shows the reduction is very effective if candidates can be rejected by exact zonal coding of the letters (eg. candidates *will* and *hill* would be kept for code "muuu", but *roll* and *wool* would be rejected). However this by definition means only accepting candidates exactly the same length, which we have already established is most likely not going to be possible, so allowing candidates with the same reduced zonal coding is more realistic (eg. for code “muuu” all candidates *will, rill, roll, wool, awl, oval* and *oral* would be kept). However this still gives considerable reduction, much better than that already seen above for word length, but taken together with some approximate word length measure would be even more effective. This is of course with the proviso that these measures can be calculated accurately. Methods for achieving these measurements accurately require further investigation.

At present, the recogniser gives an indication of zone (‘u’, ‘m’ or ‘l’) for each candidate letter, (which may in fact not match the usual zone for that letter). No checking is done for this, because the letter was suggested as possible purely through its Freeman-encoded match with an encoding in the database. However this existing code can actually be made use of, as discussed below.

5.3.3.3 Wild Cards

Sometimes the recogniser gives no candidate letter when nothing matches in the Freeman vector database (§3.4.1 and §5.2.6), and in such situations a “wild card” algorithm is implemented to attempt to fill such blank letter positions (shown as * below) by searching in the word look-up tree.

eg.   ca*e   *ope   dea*   p**t

This can be done for any letter position, including the first and the last letters of a word, and for more than one letter position, although it is not a good policy to allow more than two per word if the accuracy of the recogniser is to be relied on at all.

Searches in the lexicon may give:

cafe
cake
came
cape
care
case

dope
hope
rope

dead
deaf
deal
dear

part
peat
pest
poet
punt

However as with other characters, the recogniser gives a zone code for a wild card position as well. It also tells us whether the unknown letter is a single segment letter, or made up from a combination of two segments. Taking these two pieces of information together, we can categorise all 26 letters:

Table 5.10 — Letters categorised by segment source and zone

Letter source	Upper zone	Mid zone	Lower zone
Single segment	f i l t	c e i o r s v	f j z
Combination of segments	b d h k	a n o r u v w x	g p q y

So instead of trying all 26 letters at a wild card position, we only need to try the letters in one of these 6 subsets. This cuts down both the necessary search and the number of candidates found to be allowable.

Again the usefulness of the zonal information tends to very writer dependent, as some writers are very "mid zone". Making full use of the shape information may also mean coding the lexicon by shape for ease of search.

As an experiment, using the currently available zone codes from the recogniser, if the recogniser's zone code of a candidate word does not match the shape code of the word, the confidence in the candidate word can be reduced, thus it will be further down in the rank ordered list of candidate words. The following table shows results for three untrained writers (U).

Table 5.11 — Performance of the recogniser's zone codes

Writer	No. correct candidates incorrectly moved down the list	Total no. words tested
U1	10	106
U2	7	105
U3	14	106

For these untrained writers, the results are rather discouraging. It is hoped that this stems from the fact that the zonal code given per letter by the recogniser is not as accurate as some overall shape information would be, if it were obtained from the x,y coordinates of the script.

An alternative coding for shape that will give better results is required. A less restrictive coding would be more useful for case 2, but not for case 1. Any alternative coding technique suggested would have to provide an acceptable trade-off between the reduction of the number of candidates, and selecting the correct word.

5.3.3.4 First letters of words

Generally, writers form letters more clearly at the start of words, so the effectiveness of the recogniser at the beginning of words was investigated. The following table shows results for three untrained writers (U), and one trained writer (T).

Table 5.12 — Performance of recogniser for first letters of words

Writer	No. words where first letter was incorrectly identified	Total no. words tested	Percentage first letters correct
U1	5	106	95
U2	12	105	89
U3	22	106	79
T1	1	22	95

5.3.4 Case 2 — Suggesting candidates

5.3.4.1 Interaction with syntactic processing

The preceding sections investigated a number of potentially useful techniques to aid Case 1, namely reducing the number of candidate words found by the word look-up. For Case 2a, we could employ some search of the lexicon on our partial information, such as by first letter of the word, approximate word length and word shape.

For Case 2b (and indeed also for Case 2a), if some predictive feedback from the syntactic processing stage were also available, (as detailed in chapter 4) the lexicon could be searched on the partial information, and also by probable grammatical category. The lexicon is coded into 109 separate grammar codes. Tables 5.13 i-iv show sample distributions of words in a lexicon of just over 60,000 items, by length, first letter and grammatical category, for some of the letters in the alphabet, and a selection of grammar codes.

Obviously the effectiveness of the lexicon search on a combination of the partial information depends greatly on exactly what is being searched for — a short adjective beginning with *z* would be almost uniquely identified, whereas a mid length noun beginning with *s* would be virtually impossible to find. However this data does not take word shape into account. Taken together with shape information the searches should be vastly reduced. This requires much further evaluation and testing, for example to obtain the frequency distribution of the lexicon by word shape (probably using the reduced zone codes), and cross-referencing with this data.

Table 5.13 i — Numbers of words in lexicon by length and first letter

First letter of word	All lengths	3-5 letters	4-6 letters	5-7 letters	6-8 letters	7-9 letters	8-10 letters
a	3393	366	676	1015	1311	1506	1514
d	4050	413	776	1207	1582	1785	1716
j	527	126	210	285	297	244	150
n	1023	161	272	388	457	472	408
p	5128	517	997	1600	2042	2286	2171
s	7102	899	1773	2696	3406	3539	3137
z	87	30	42	44	42	31	17

Table 5.13 ii — Numbers of common singular nouns by length and firstletter

First letter of word	All lengths	3-5 letters	4-6 letters	5-7 letters	6-8 letters	7-9 letters	8-10 letters
a	946	142	223	302	376	450	439
d	989	154	255	301	381	399	391
j	160	58	78	75	71	49	43
n	315	60	96	108	137	135	132
p	1517	239	408	552	628	660	606
s	1900	365	590	743	797	820	742
z	27	10	17	11	14	8	7

Table 5.13 iii — Numbers of prepositions in lexicon by length and first letter

First letter of word	All lengths	3-5 letters	4-6 letters	5-7 letters	6-8 letters	7-9 letters	8-10 letters
a	20	12	15	13	5	4	2
d	2	0	1	2	2	1	0
j	0	0	0	0	0	0	0
n	3	2	2	1	0	0	0
p	2	1	1	1	1	1	0
s	0	0	0	0	0	0	0
z	0	0	0	0	0	0	0

Table 5.13 iv — Numbers of adjectives in lexicon by length and first letter

First letter of word	All lengths	3-5 letters	4-6 letters	5-7 letters	6-8 letters	7-9 letters	8-10 letters
a	384	37	79	124	164	188	197
d	404	45	74	92	133	166	191
j	39	9	20	28	24	16	9
n	144	26	35	51	54	67	62
p	604	44	92	148	198	231	267
s	507	91	211	297	359	341	315
z	6	2	2	3	3	3	2

Appendix E contains some frequency counts of word shape information using the zone codes introduced in §5.3.3.2, for a number of different word lengths. The tables show that this information is not restrictive across a large lexicon, especially for the words of commonly occurring lengths. However for a much smaller vocabulary, the zonal information seems to be more restrictive, especially when applied in combination with word-initial characters.

5.3.5 Conclusions

The above experiments and discussions have identified some potentially useful information for improving current handwriting recognition systems. We have seen that this information is really only useful if it can be obtained accurately and consistently across writers. However even currently available estimates already show encouraging results in the case 2 scenario described above. In this situation, any information, even vague or partial, is better than nothing, and is in fact useful to restrict any lexicon search. We have also seen that results are much better for trained writers, and it may be that a "general purpose" system for any writer will not prove to be an effective system for cursive handwriting.

It appears that the word length information is not very useful unless it can be calculated accurately. However the word shape information seems much more promising, especially when combined with some predictive feedback from the syntactic analyser and some partial character information from the pattern recogniser.

The results provide a strong argument for using physical information in a pattern recogniser. The current recogniser should be improved so that better advantage can be taken of these factors. They must be available accurately and consistently if they are to be relied upon. Similarly any predictions from syntactic analysis must contain the correct code. It may be found from testing that this is impossible to achieve due to the ambiguity in previous word positions. Semantic analysis may also be able to provide some feedback mechanism concerning the domain of the sentence so far. This could also suggest candidate words in conjunction with other information, if the unknown word is a content word.

5.4 Structure for storage and search of information

5.4.1 Introduction

The preceding discussions have established that in a number of cases, some kind of lexicon search by general information about words would be useful to correct any detected errors within the script recognition system. Given certain pieces of data, it is necessary to calculate how many words in the lexicon match this search criteria. It is not worthwhile retrieving the matched word strings themselves if the number of matches is too large, but it is if this number is within a reasonable limit. A suitable structure for storing the required information in memory must be established.

5.4.2 Search methods

A standard method for searching any kind of database (and a lexicon with various pieces of additional information can certainly be thought of as a database) on a key other than the primary one (the primary key for a lexicon is the word itself) is by using inverted list structures (Claybrook, 1983; Date, 1986). An inverted list is simply a list of indexes, it is known as "inverted" because the accessing is in some sense "backwards". Relations in databases are designed to be searched by the primary keys. Each different piece of information in the structure being accessed has its own list of indexes. Hence all words with grammar code 4 could be accessed together from the relevant list, for example, or all words six letters long, and so on.

However inversion would not appear to be particularly appropriate given the actual data involved in this case. A better method can be found, given the two important considerations of speed of search and memory requirements for storage. The chosen method which has been implemented is described below, and became known as the "backwards", or "inverted look-up".

5.4.3 Inverted look-up structure

The required information is at present stored in the trie (§4.5.1.2). It can alternatively be stored in an array where the number of array elements is the number of words in the lexicon. The main search criteria are most likely to be length, grammar code and first letter. This should give sufficient cut down of search so that word shape need only be checked on a relatively small number of words. Hence word shape information need not be stored in the inverted look-up structure, it can be generated from the matched words at run-time and compared with the shape of the unknown word being searched for. The information needs to be stored in such a form that will facilitate fast testing of whether it matches what is being searched for. The fastest comparison is bit-wise, so the data is stored in two 32 bit integers as follows.

The first letter of the word is stored simply as one of the rightmost 26 bits out of the 32 available in the first long integer. The length and grammatical code are both stored in the second long integer. The rightmost 10 bits are reserved for the length information. Length is stored as exact length ±1, so for example if a word has 5 letters, then bits 4, 5 and 6 will be set. For words with only one letter, bits 1 and 2 are set. Ten bit positions are sufficient for most words (83% of words in a lexicon of 70,000 words are up to 10 letters long), so for words 10 letters long, bits 9 and 10 are set, and for words more than 10 letters long, only bit 10 is set. If the length being searched for is 10 or more letters, a further test checks for a ±1 match with the required length.

If the bit-wise comparisons match (and the length is checked more accurately if it is more than 10) then the word shape is checked. If this matches, then the array subscript is saved. If the number of successful matches is less than some specified number (the system has been tested with this limit set to 100), then the actual words for the saved array subscripts are retrieved from the structure. The ten most frequently occurring (from a corpus frequency count) are ordered and can either be presented to the user, or passed on to further stages of analysis, depending upon the implementation.

The word strings themselves are stored in a simple array structure. This is because it would be wasteful to store a character array as part of the main lexicon structure. Either a maximum word length array would have to be part of the structure, which wastes a lot of memory for the words (almost all of the lexicon) which are shorter than that maximum. A dynamic allocation of exactly the right amount of memory for the word cannot be implemented because the structure needs to be contiguous in memory for the fast saving and reading in from a file (as discussed in §3.3.6). So the main lexicon structure contains a pointer (actually another array subscript) to the words array in order to access the word string itself.

Given the preceding description of the lexicon structure, there is one major drawback. The lexicon structure contains exactly the same information as the look-up tree. This is a problem not only conceptually (it is pointless duplicating the information), but also from a practical point of view. The information may become inconsistent, and even if this is carefully watched, the two structures are using more memory than is really necessary, due to the duplication. However it is possible to find a method which allows the additional information to be removed from the tree (as previously mentioned in §4.5.1.2). It is needed when the ordinary (forwards) look-up finds a word, but can just as easily be accessed from the new lexicon structure by using a hashing algorithm on the word string.

A hashing algorithm computes the location of a particular array element (Knuth, 1973; Cooper and Clancy, 1985; Claybrook, 1983). The algorithm is used both for originally arranging the array, and to check whether a particular value is present. A good hash function will distribute values uniformly throughout a waiting array (used for access) called the hash table. If the returned hash value is greater than the defined table size, the modulus operator wraps around back to the start of the table. If a hash function were perfect, it would automatically put every incoming value into a different slot in the hash table. Unfortunately, hash functions tend to be imperfect. Unless the table is made excessively large, two or more different values will eventually be sent to the same slot. This is called a collision.

To avoid clustering of collisions, chaining was chosen as a collision resolution technique. Instead of storing the values themselves in the hash table, each table entry becomes the head of a linked list. Incoming values are stored by adding them to the appropriate linked list. Collisions add new elements to the linked list associated with that particular hash value. Some lists end up longer than others, but no values interfere. When searching a chained hash table, the appropriate chain must be traversed to locate the required element.

There are no “best” methods with hashing, there are always trade-offs to be considered between computer storage space and search times. A large size hash table means using a large amount of memory, but a minimally-sized hash table means slower searching and resolving collisions. The chosen lexicon of approximately 70,000 words, with an average chain length of 10, gives a table size of 7,000. Choosing a prime number as the table size helps to distribute elements evenly throughout the table (e.g. 7001).

5.4.4 Conclusions

This new method described above is not noticeably any slower than the old one, nor is it noticeably faster. However it does mean that the additional information can be removed from the look-up tree, which in turn means that a compression method can now be applied to the tree, such as utilising a directed acyclic word graph (§ 3.2.5 and § 3.3.3). Consequently the system is left with a more useful structure for the storage of the lexicon, which can be searched on a number of factors to attempt correction of recognition errors. This structure should be generally more useful in the future because it can easily be expanded to store extra information, should this become necessary. Table 5.14 shows that the combination of hash table and dawg (last column) has very similar memory requirements to the uncompressed trie including all extra word information (second column).

Table 5.14 — Memory requirements (thousand bytes) for trie, dawg and hash table for various word lists

Number of words	Trie including word info.	Trie without info.	Hash table for word info.	Dawg	Dawg + hash table
2461	140.364	110.832	85.027	33.152	118.179
13706	748.136	583.664	580.224	162.176	742.400
30201	1649.900	1287.488	1257.093	343.320	1600.413
68856	3568.168	2628.928	2860.509	750.312	3610.821

5.5 Experimental results

A passage of test data (written by one person) was used to test the inverted look-up technique on word positions where no candidate words suggested from the combination of recognised alternative characters were found in the lexicon. Tables 5.15 i-iii give recognition results for three different lexicons. The passage was as follows:

> *Professor Sloman has brought spelling up to date except where this would involve changes in pronunciation, accentuation and capitalization. In the introduction he has covered every aspect of the play under the headings of date, sources, structure and theme, language and metres, staging and texts.*

Tables 5.15 i-iii give the number of candidate words found, the ranked position of the correct word in the list of candidates, and four columns detailing the additional information being searched for in the cases where no candidates were found. The letter "y" here means the information was correct, "n" means it was incorrect. The figure "0" in the grammar code column means that the syntax analysis could not suggest a code. This usually occurs when one of the previous word positions had no candidate words. The final column gives a reason if the correct word was not identified by any method. A number in this column signifies more than one reason; "lex" means the correct word was not in the lexicon; "recog" means the recogniser did not provide all the characters; "shape" means the word shape zone information was incorrect; "gram" means that the predicted grammar code was incorrect; and ">100" means that more than 100 words were suggested, so they were not retrieved from the hash table.

It can be seen that as the lexicon size increases, more candidate words are allowable, so sometimes the inverted look-up is not used when it might have found the correct word. However some correct words are not in the smaller lexicons. Larger lexicons also result in more than 100 words matching the partial information more often. Of the four features of additional information, it is the grammar code that is most often incorrect. This is because the prediction algorithm relies on the previous word positions being identified correctly, and this is often not the case.

Table 5.15 i — Recognition results from inverted look-up (4506 word lexicon)

Word	No. of cands	Pos of correct	Word length	Gram. code	First letter	Shape	Reason
professor	46	11	y	0	y	y	
sloman	0		n	0	y	y	lex
has	6	1					
brought	8		y	0	y	n	2
spelling	1						lex
up	1	1					
to	1	1					
date	1	1					
except	2	1					
where	2		y	n	y	y	gram
this	6	1					
would	1		y	n	n	n	3
involve	10		y	n	y	y	gram
changes	2	1					
in	6	1					
pronunciation	0		y	n	y	y	gram
accentuation	0		y	0	y	y	lex
and	2	1					
capitalization	0		n	0	y	n	2 + lex
in	7	1					
the	4	1					
introduction	0		y	n	y	n	2
he	3	1					
has	9	1					
covered	5	2					
every	0		y	n	y	y	gram
aspect	2	2					
of	1						recog
the	10						recog
play	1	1					
under	4	1					
the	6	1					
headings	0		y	n	y	y	lex
of	4	1					
date	1	1					
sources	0		y	n	n	n	3
structure	1	1					
and	3	1					
theme	1	1					
language	0		y	n	n	n	3
and	4						recog
metres	1						lex
staging	0		y	n	y	y	lex
and	3	1					
texts	126		y	0	y	n	> 100

Table 5.15 ii — Recognition results from inverted look-up (21011 word lexicon)

Word	No. of cands	Pos of correct	Word length	Gram. code	First letter	Shape	Reason
professor	239		y	0	y	y	> 100 lex
sloman	1						
has	8	1					
brought	2		y	n	y	n	2
spelling	3						
up	6	1					
to	1	1					
date	2	1					
except	2	1					
where	5		y	n	y	y	gram
this	14	1					
would	8		y	n	n	n	3
involve	472		y	n	y	y	2
changes	2	1					
in	10	1					
pronunciation	1	1					
accentuation	0		y	n	y	y	1+ lex
and	11	1					
capitalization	1		n	0	y	n	2+ lex
in	11	1					
the	9	1					
introduction	0		y	n	y	n	2
he	7	1					
has	16	1					
covered	8	2					
every	7						recog
aspect	2	2					
of	6						recog
the	23						recog
play	3	1					
under	8	1					
the	14	1					
headings	3		y	n	y	y	1+ lex
of	4	1					
date	1	1					
sources	0		y	n	n	n	3
structure	1	1					
and	11	1					
theme	1	1					
language	0		y	n	n	n	3
and	14						recog
metres	5	4					
staging	1	1					
and	12	1					
texts	3						recog

Table 5.15 iii — Recognition results from inverted look-up (68856 word lexicon)

Word	No. of cands	Pos of correct	Word length	Gram. code	First letter	Shape	Reason
professor	761						> 100
sloman	3						lex
has	16	1					
brought	3		y	n	y	n	2
spelling	6						recog
up	13	1					
to	1	1					
date	5	1					
except	2	1					
where	3		y	n	y	y	gram
this	28	1					
would	1		y	n	n	n	3
involve	0		y	n	y	y	gram
changes	4	1					
in	11	1					
pronunciation	1	1					
accentuation	0		y	n	y	y	gram
and	19	1					
capitalization	19		n	0	y	n	3
in	14	1					
the	14	1					
introduction	0		y	n	y	n	2
he	12	1					
has	29	1					
covered	23	2					
every	23						recog
aspect	2	2					
of	8						recog
the	28						recog
play	8	1					
under	14	1					
the	18	1					
headings	6	2					
of	10	1					
date	2	1					
sources	1						recog
structure	2	1					
and	15	1					
theme	3	1					
language	0		y	n	n	n	3
and	23						recog
metres	6	4					recog
staging	2	1					
and	16	1					
texts	8						recog

5.6 Conclusions

This chapter has identified the kinds of problem areas leading to errors in the handwriting recognition system. These include misspellings and mis-recognitions either due to badly written words or to idiosyncrasies of the pattern recogniser. It appears that most errors will be detected because none of the candidate strings is found to be allowable by the lexical look-up.

A number of alternative techniques for error correction were introduced and evaluated. Traditional methods seem unlikely to be useful, mainly due to the ambiguity of an original character string to compare with, and because algorithms may reach explosive and counter-productive proportions if many original candidate strings are used. In practice, a number of intermediate "tweaks" of recognition data seem particularly effective for the types of errors found from the test data collected so far. Indeed they have suggested a few areas where the pattern recogniser seems quite weak.

In addition, a number of potentially useful pieces of information were identified. These include extra physical information from the pattern level — namely some measure of word length and word shape. This might be a count of the number of ascending and descending letters in a word, and their approximate position, i.e. near the beginning, middle or end of a word. First letters of words need to be more accurately recognised, and the number of letter candidates could be reduced, especially as writers form the beginnings of words more clearly. There are also no ligatures to confuse the start of the letter as there are in other letter positions. Other information from higher levels of analysis, for example the grammar code, may also be useful, especially if the possible codes can be identified more accurately. Positions where the identified code is a large category (e.g. nouns), even when subdivided (e.g. singular countable nouns) need much better information from other levels. In these cases it may be that searches may have to rely on matches with length, shape and first letter. Semantic analysis may also help by identifying a domain code for content words. The exact method for effectively applying all available information needs more evaluation, but initial experimentation is encouraging.

Given this information as used in the treatment of errors, there are indications that it may also be effective to apply it to aid the reduction of the list of candidate strings in word positions where no error has been detected. Again this needs further evaluation, but a useful lexical database structure has been established. A truly interactive system (after Rumelhart and McClelland's parallel distributed processing model of word perception) could use higher level information to reject unsuitable candidate words. Other candidates could also be suggested which may be better than those found simply from the pattern recognition and lexical check.

In conclusion, it seems that it is in fact possible to get higher levels of analysis to contribute to recognition. They can help to identify errors, and to solve some of these errors with additional help from extra physical information from pattern level. Improved recognition rates can be achieved, the process is no slower, and has obvious leanings to parallelisation of at least some stages of the recognition process.

Chapter Six

Summary and Discussion

6.1 Summary

The preceding chapters have described a script recognition system which attempts to overcome the inherent problem of ambiguity present in handwriting. A functional system has been demonstrated through experimental results. Using a number of sources of information, including orthography and higher level linguistic constraints, the system shows improved results, and word recognition rates can reach 98%. Figure 6.1 shows the various stages of the current system.

6.1.1 Introduction and review

The automatic recognition of handwriting is necessary as a natural mode of communication with computers, and appears to be appropriate for a number of applications. Interest in this field has expanded in recent years, along with interest in speech recognition and optical character recognition (OCR), especially with the advances in technology. However there is insufficient information present in script for unambiguous identification of characters and words. Human readers can understand many badly formed letters and seemingly illegible words due to information gained from the surrounding context. A number of past and current approaches to the area of handwriting recognition were reviewed. These various approaches have a number of problems, and it was established that contextual information is necessary in addition to a pattern recogniser. Some systems have employed letter level and word level information, in the form of n-grams or a limited word look-up. Machine-readable dictionaries can be used as a source of linguistic information.

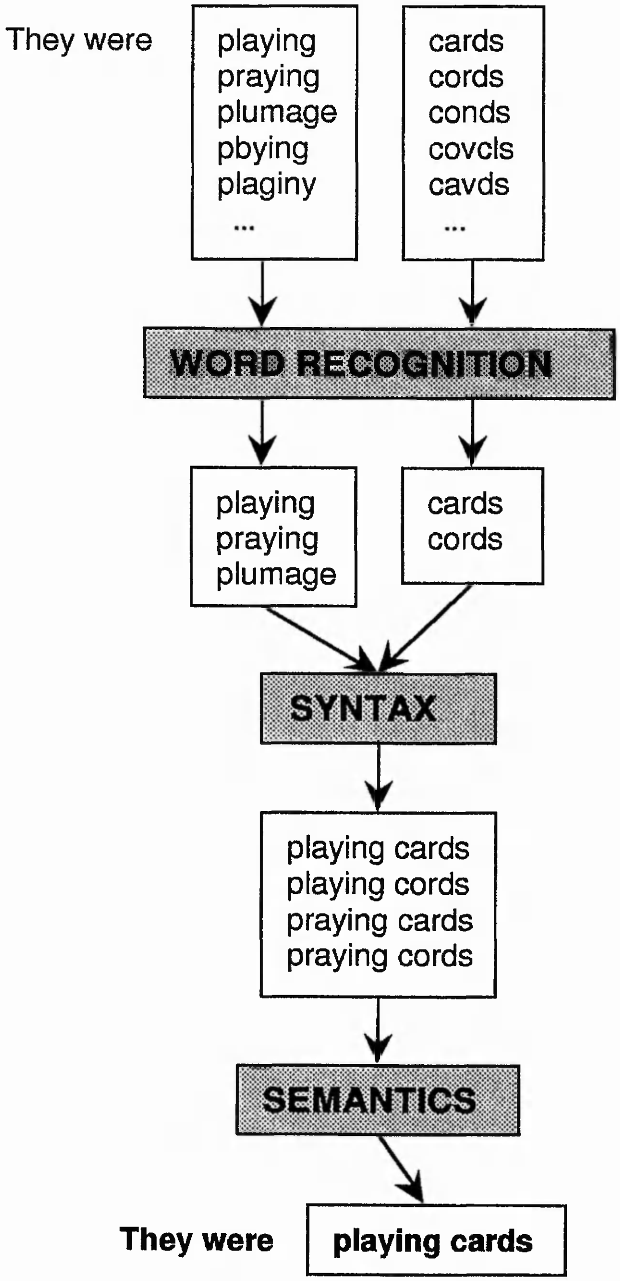


Figure 6.1 — System overview with example recognition

6.1.2 Pattern recognition

Pattern recognition techniques were introduced, specifically those used for handwriting. Details of the particular on-line cursive script recogniser and the interface to further levels of processing were given. Briefly, sequences of x,y coordinates are collected, their Freeman vector chain codes are matched to a database, and a number of candidate characters are produced per character position. The output from the pattern recogniser is poor, and requires further processing to improve. Methods commonly used for such processing involve using transitional probabilities (for example the Viterbi algorithm or Markov modelling), using information about how letters combine (for example n-grams), using lexical look-up, or combinations of these.

Statistical methods involve selecting one "correct" answer and thus have a built-in margin of error. Experimental results showed that a lexical look-up is more effective than n-grams in terms of reduction of candidate strings. It also gives a more useful reduction because it guarantees lexical output. The limitation of this method is that an input word may not be included in the look-up vocabulary, however this is unavoidable (see discussion §6.2). This particular problem also exists for statistical methods since they sample from the language and assume a reliable distribution.

6.1.3 Word recognition

For effective implementation of a lexical look-up technique, an efficient data structure is needed for representation of the vocabulary. Such a data structure should be the best compromise with regard to processing time and memory requirements. A number of alternative structures (lists, trees, hash tables and graphs) were described, illustrated and compared. The speed of negative searches is particularly important because most searches in the recognition system are unsuccessful. Details and results of comparisons between some implemented data structures were presented, and some methods of memory reduction such as tail-end compression and the use of a directed acyclic word graph were discussed.

For our system the trie structure (§3.2.3.3) was most appropriate given the need for grammatical, morphological and semantic information in further stages of the script recognition process, however the reduced-memory tree (§3.3.1) also gives fast search times and reasonable memory requirements, especially for experimental purposes with different word lists and test data on a limited memory computer. The dawg structure (§3.2.5 and § 3.3.3) is optimal for memory requirements (§3.3.5), but does not allow additional information about words to be stored. The trie structure does allow such information to be stored at the end of word nodes.

Acceptable word candidates remaining after lexical look-up are stored for further analysis.

6.1.4 Integration

Recognition can be improved by using additional linguistic information. Alternative word candidates are combined to form candidate phrases, many of which may be ungrammatical or meaningless. Techniques for the integration of further levels of processing were discussed along with information needed by and produced by each level. These include for example syntax and semantics, and the use of information about compounds, commonly used phrases and idioms. A suitable structure for the transfer and sharing of information between all levels of processing was illustrated. Results showing improved recognition rates after further analysis are presented. The use of compounding information has been implemented, and tested on small samples of test data taken from an Estate Agent's document. The results can be seen in Appendix C which shows the improved recognition when using information about compounds and phrases.

A morphological indexing system was developed whereby each word in the lexicon is associated with its root. These indices are stored in the lexical look-up tree. The tree is an ideal site for integrating different types of information because it provides the interface between the low level pattern recognition process and higher level linguistic processes. Additional information such as grammatical category and word frequency can also be accessed via the tree structure, and a set of coded flags was developed to provide details about compounds and case. The recognition system was also extended to allow punctuation marks, digits and other non-alphabetic characters in certain situations.

6.1.5 Combining sources of information

The different levels of analysis in the handwriting recognition system are similar to those used by the human processing system. Psychological studies of word recognition propose models of interaction between the different levels of processing which combine to give recognition. Such models receive information from both top-down and bottom-up sources, and feedback exists between all levels. This principle could be applied to the current script recognition system to make best use of all available information. It could be especially useful to solve recognition errors.

Such errors were studied, along with traditional error correction techniques. Alternative sources of information were investigated, with the aim of discovering additional information which could aid error detection, correction and even recognition. These include physical measures of word length and overall word shape (e.g. details of ascending and descending characters), the accuracy of recognition of the first letters of words, and possible feedback from syntactic analysis. Alternative candidate letters and words may be suggested where recognition has failed to provide any. An "inverted" search method based on partial information about words is discussed. This includes a large hash table structure to store all additional information about the words in the lexicon. Consequently this data can be removed from the lexical look-up tree, which means that the dawg structure mentioned in chapter three becomes viable. Initial experimental results were presented.

6.2 Discussion

The preceding chapters have detailed a script recognition system, which has been summarised above. The results from the current system are insufficient for a really practical system. Improvements are necessary in a number of areas. For example the pattern recogniser could make much better use of partial information, and more accurate information is needed from higher levels of analysis. The exact implementation of different areas of the system will also depend on the particular application and will therefore need to be tailored.

The methods described and implemented are not only applicable to on-line cursive script recognition. The techniques are more generally applicable to all areas of text recognition. This includes both on and off-line cursive and unconnected handwriting and optical character recognition (OCR). The data structures developed for the representation of lexicons are useful in any situation where vocabularies are needed, for example in word processors, spelling checkers, or for the classification of electronic documents (such as e-mail). Semantic domain codes can be used for identifying the subject areas of text (Walker, 1986; Rose, 1991). The current system has been developed for English, but the methods would be applicable to other languages which use the same or similar alphabets.

The advent of notepad computers brings many new opportunities for applications suitable to this new form of computer. They are lightweight (a few pounds), approximately A4 in size, and easily portable, yet still include powerful processors. The equivalent of a 386 PC is already available, and a 486 is planned. Millions of people work away from their desks, so the situations where such a computer may be appropriate are widespread. These include note-taking almost anywhere, form-filling, taking orders, stock control and so on in warehouses. Notepad computers are already on trial in a hospital accident and emergency unit, and doctors and dentists surgeries are other potential markets. This kind of technology would be useful wherever diagrams need to be drawn with notes, and for almost any type of salesman. Many of these applications may need some form of handwriting recognition. It should be noted however, that many computer applications may not require recognition of pen input. Sometimes it is necessary and desirable to leave hand-drawn and handwritten input as it is. Recognition and associated techniques are also applicable to standard computers of all forms, mainframes, workstations and personal computers, especially in office systems.

There may also be educational applications, perhaps for teaching children to write. The system will only recognise standard letter formations so the characters must be written properly by the children. Important work is also progressing in the recognition of engineering drawings for both the lines of the drawings and for text found on the diagrams (Waite, 1989; Dori, 1991; Lysak and Kasturi, 1991), and of musical notation (Fahmy and Blostein, 1991).

The current state of handwriting recognition can reach high recognition figures, but only on consistently reasonably neat handwriting, even for a user-dependent system. A pattern recogniser working on some form of segmentation relies on the fact that the input script contains all the necessary characters correctly formed. More often than not these constraints are not met, especially in note-taking situations when users are by necessity writing speedily. Script becomes untidy, can often become illegible, and only comprehensible given the surrounding context, and even then still occasionally impossible to interpret. Script will also contain many abbreviations, often unique to the individual writer.

A truly usable recognition system would have to learn from whole word recognition techniques and combine them in some fashion with existing segmentation techniques (Ho et al, 1991; Hull et al, 1991). This may also include some "fuzzy-matching" (nearest matches) of the lexicon on partial information. For example writers usually form the beginnings of words reasonably well, but often this tails off towards the ends of words. Partial matches with the lexicon may have a few characters from the beginning of a word, together with approximate word shape information (see also §6.3). Assuming the lexicon includes all words necessary, the recognition process can be lexically driven, in other words the pattern recogniser need not pursue segmentations which lead to characters that cannot follow the preceding characters because that sequence does not occur in any of the words in the lexicon. The lexicon would have to include the usual forms of abbreviations used by a particular writer.

The above considerations make a general purpose handwriting recognition system a virtual impossibility. The particular application must be tailored (in terms of lexicon) and trained for individual users. Any training should be user-friendly to a naïve user — a doctor or warehouse clerk does not want to be concerned with the segmentation required for handwriting recognition.

Different recognisers have different areas of strength and weakness, so it may be possible to combine them into one system. The final decision about a word's identity would be made by combining the results of all recognisers in use, assuming lexical output from each. The most meaningful and effective way in which such output can be combined is a topic of on-going research (Hull et al, 1991). Hull has found that a reasonable measure appears to be the "borda" count, which is a sum of the distance of a particular word from the bottom of each ranking in which it occurs. The word with the maximum borda count is chosen as the best candidate.

There will always be cases where the look-up either fails to find any allowable strings, or the correct word (i.e. the input word of script) is not in the list of allowable strings. Chapter five dealt with misspellings and mis-recognitions, but there are always going to be cases when the input word is not in the look-up word list. Obviously the choice of which words are included in the look-up word list is paramount to the efficiency of the system. Too short a list means that the input words will quite often be missed, and too long a list can mean that the list of allowable candidate strings is vast, and will often contain words that most people would not recognise as English words. In fact, however complete a word list you may think you have, it will never give full coverage (Sampson, 1989). It is therefore better to reach some compromise, and perhaps use a word list tailored to the particular domain. It may be appropriate to have a basic core vocabulary in use all the time, with additional lexicons available depending on the particular domain of application. There will also be the need for a user-specific lexicon with the facility to add and delete items, especially for proper nouns, individual abbreviations and misspellings, as well as words which may have been missed by the other dictionaries.

For situations when all other attempts to suggest a word have failed, it would be preferable to provide a string of characters which is in some way a best guess, even if that string is not a word known to the lexicon. A closest match algorithm would be of use here, or some combination of the highest rated character candidates which contains frequently occurring sequences. The orthographic information stored in the lexical look-up tree could be used in a similar way to n-gram look-up discussed earlier, whereby non-occurring letter sequences are ruled out, and alternative character candidates are tried instead. General word shape information could be used in addition.

To provide a user-friendly environment based upon the latest Human Computer Interface techniques, the script recognition system could incorporate an interactive gesture-based front-end (Welbourn and Whitrow, 1989; Wilson and Whitrow, forthcoming). This may be as part of a pen-driven word processor where already existing text can be edited as with pen and paper at present. Gesture-based editing symbols for insert, delete, move text and so on would be included, and any inserted or altered text could be handwritten and recognised. Such systems would also be useful for the recognition and editing of diagrams.

Other kinds of information give cues in written language, for example the layout of the writing (or typescript). This includes the spacing between sections of text, the paragraphs, headings, subheadings and so on. This information is useful for syntax, and semantics (for example, a heading gives clues to the domain or content of following text). Form filling applications can give this kind of information accurately, for example where an address or a telephone number is expected, so the restrictions put on allowable strings are even greater.

Both syntactic and semantic information could be used more effectively than at present. Some kind of feedback process could be implemented whereby all levels of analysis can learn from the identification of the correct words as chosen by the user (see figure 6.2). The system should assume that the top-rated candidates are correct unless the user chooses another of the existing candidates or enters an alternative word. Given this new information, both the syntax and semantic processors can update their information accordingly. This should improve any predictions made for unknown or mis-recognised words. Not only can the pattern recognition stage be lexically-driven, but suggestions can also be made from higher level knowledge which will affect the recognition process. As words are confirmed, all levels should be able to improve their future results. The large numbers of incorrect candidates contribute to the deterioration in performance of the higher levels of analysis. If these numbers can be reduced by a system that learns, the over all recognition performance should be increased.

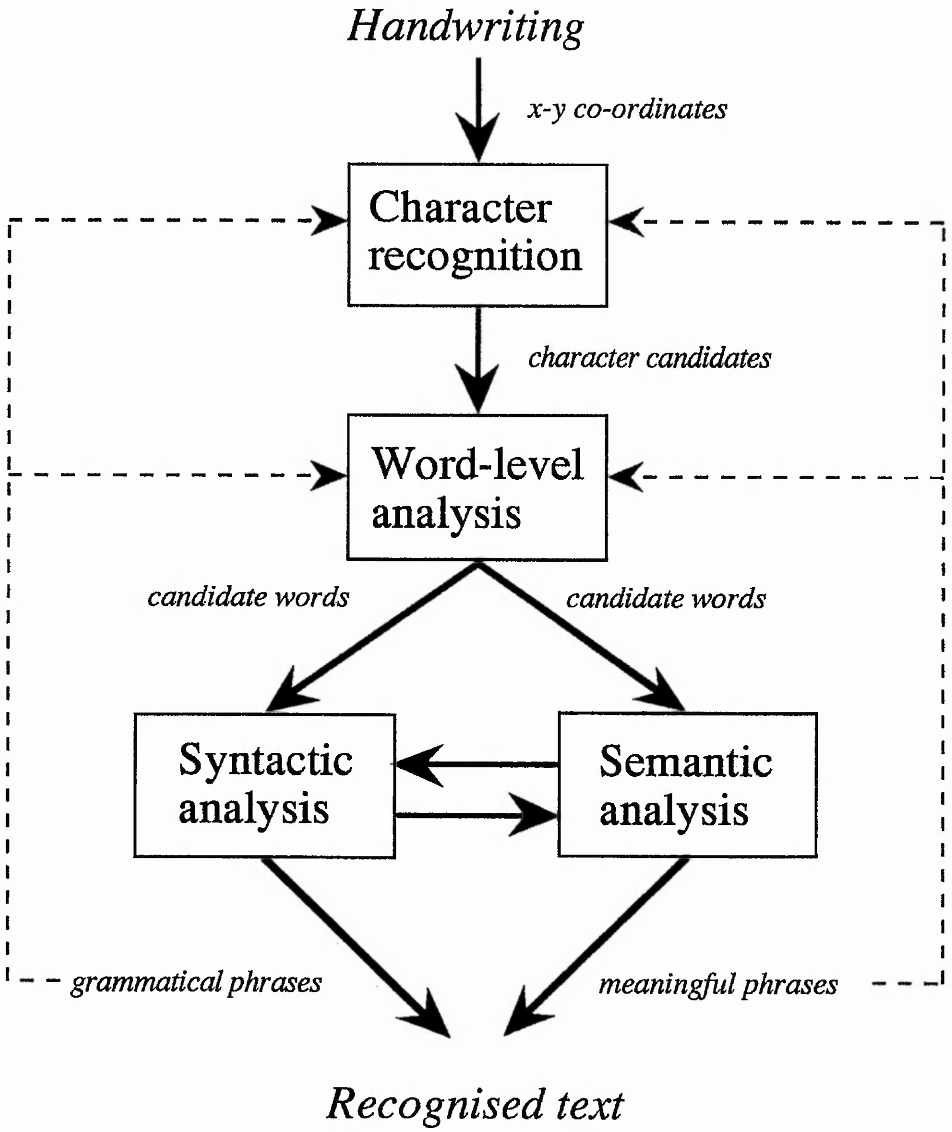


Figure 6.2 — Contextual recognition system with feedback of information

6.3 Future work

There are a number of areas where further work would be most important. Some of these have been identified in the above discussion. The following ideas are more immediately applicable to the current system. The pattern recogniser should be improved, perhaps by the use of some interactive techniques whereby the recogniser and the lexical look-up work together so that the look-up may be able to predict which characters could be next within a word. Not only may the recogniser be able to be lexically-driven, but other levels of information (e.g. syntax and semantics) should also be able to direct the pattern recognition. In fact all levels should be able to interact and feed information back to each other. The current architecture will not easily allow this, because the processes are separate and serial. Alternative architectures including some parallelisation would appear to be very useful for this situation (see figure 6.2).

The pattern recognition should also be able to identify the beginnings of words much more accurately than at present, and more investigation of the efficacy of some measure of word length would be useful. Whole word recognition would appear to have its place especially when used in conjunction with other recognition techniques, but the current recogniser does not supply this type of information. It may be that a separate recogniser could be constructed that would concentrate on these sort of features, i.e. the shape of a word found from its ascending and descending characters.

In a lexicon of only 4,000 words, the most frequently occurring tri-gram at the start of a word only occurs 52 times (it is the tri-gram *pro*). A measure of word shape information such as the reduced zone code discussed in §5.3.3.2 is then restrictive across those 52 words. More details of the frequency distribution of word-initial tri-grams are given in Appendix E.

The system should be tailored to a particular domain, as accuracy can be much greater in a restricted situation. The layout of documents can provide additional information which also places constraints on the recognition process. For example in the layout of a letter, different dictionaries should perhaps be accessed in different parts of the letter. A form filling application would also restrict recognition to digits or capital letters in certain places. An order form for spare parts for motor cars would probably only have to recognise digits and part names, which would be from a small domain-specific lexicon.

6.4 Conclusions

To conclude, a basic script recognition system has been demonstrated. It is functional, and gives very encouraging results. At present the system allows a large vocabulary of English words which can be represented in memory with practical size and processing requirements, and is searchable in real time. Recognition rates are as yet insufficient for a practical system, and need considerable improvement. The preceding chapters have indicated some areas where improvements need to be made, and have suggested some techniques for this. This system could be useful in a number of situations (as discussed above), and the lexical look-up and use of additional linguistic information is not restricted to on-line handwriting recognition.

As an area for on-going research, the system could be both extended to allow input from other recognisers for alternative applications, and also restricted to particular domains.

Bibliography

TE Ahlswede (1985) 'A toolkit for lexicon building', *Proc. 23rd Ann. meeting of Assoc. Comput. Ling.*, Chicago, pp. 268-276

AV Aho, JE Hopcroft and JD Ullman (1983) *'Data Structures and Algorithms'*, (Addison-Wesley)

AV Aho, BW Kernighan and PJ Weinberger (1988) *'The AWK Programming Language'*, (Addison-Wesley)

W Amsbury (1985) *'Data Structures: from arrays to priority queues'*, (Wadsworth)

RA Amsler (1982) 'Computational lexicology: a research program', *Proc. AFIPS Nat. Comp. Conf.,* Houston, **51**, pp. 657-663

RA Amsler (1984) 'Machine-readable dictionaries', *Annual review of information science and technology* (ARIST), **19**

AW Appel and GJ Jacobson (1988) 'The world's fastest scrabble program', *Communications of the ACM,* **31**(5), pp. 572-578 + 585

R Bayer and E McCreight (1972) 'Organization and maintenance of large ordered indexes', *Acta Informatica,* **1** (3), pp. 173-189

HL Bergel (1987) 'A logical framework for the correction of spelling errors in electronic documents', *Information Processing and Management,* **23** (5), pp. 477-494

WW Bledsoe and I Browning (1959) 'Pattern recognition and reading by machine', *Proc. Eastern Joint Computer Conf.*

A Blumer, J Blumer, D Haussler, A Ehrenfeucht, MT Chen and J Seiferas (1985) 'The smallest automaton recognising the subwords of a text', *Theor. Comput. Sci.*, **40**, pp. 31-55

B Boguraev and T Briscoe (1989) (Eds) *'Computational Lexicography for Natural Language Processing'*, (Longman, London)

B Boguraev, D Carter and T Briscoe (1987) 'A multi-purpose interface to an on-line dictionary',*Third conference of the European Chapter of the Association for Computational Linguistics*, Copenhagen, Denmark

B Boguraev, T Briscoe, J Carroll, D Carter and C Grover (1987) 'The derivation of a grammatically indexed lexicon from the Longman Dictionary of Contemporary English', *Association for Computational Linguistics*

B Boguraev and T Briscoe (1987) 'Large Lexicons for Natural Language Processing: Utilising the Grammar Coding System of LDOCE', *Computational Linguistics,* **13**

RS Boyer and JS Moore (1977) 'A fast string searching algorithm', *Communications of the ACM,* **20** (10), pp. 762-772

RM Bozinovic (1985) 'Recognition of off-line cursive handwriting — a case of multi-level machine perception', *Unpublished PhD thesis,* University at Buffalo (SUNY), New York

RM Bozinovic and SN Srihari (1982) 'A string correction algorithm for cursive script recognition', *IEEE Trans. on Patt. Anal. and Mach. Intell.,* PAMI-**4** (6), pp. 655-663

RM Bozinovic and SN Srihari (1989) 'Off-line cursive script word recognition', *IEEE Trans. on Patt. Anal. and Mach. Intell.,* PAMI-**11**, pp. 68-83

D Bradley (1980) 'Lexical Representation of derivational relation', in: M Aronoff and M-L Kean (eds) *'Juncture'*, Anma Libri, pp.37-55

R De la Briandais (1959) 'File Searching Using Variable Length Keys', *Proceedings of the Western Joint Computer Conference,* **15**, pp. 295-298

MK Brown and S Ganapathy (1980) 'Cursive script recognition', *Proc. Int. Conf. on Cybernetics and Society'*, pp. 47-51

RM Brown, TH Fay and CL Walker (1988) 'Handprinted symbol recognition system', *Pattern Recognition,* **21** (2), pp. 91-118

J Brustkern (1985) 'Structure of a word database for the German Language', *Proc. Intl. Conf. on Databases in Humanities and Soc. Sc.,* pp. 31-36

DJ Burr (1983) 'Designing a handwriting reader', *IEEE Trans. Patt. Anal. and Mach. Intell.,* PAMI-**5**, pp.554-559

N Calzolari (1984) 'Detecting patterns in a lexical database', *Proc. COLING 84,* pp. 170-173

JM Carroll (1979) 'Complex compounds: phrasal embedding in lexical structures', *Linguistics* **17**, pp. 863-877

DM Carter (1987) 'An Information-Theoretic Analysis of Phonetic Dictionary Access', *Computer Speech and Language,* **2**, pp. 1-11

JM Cattell (1885) 'The inertia of the eye and brain', *Brain,* **8**, pp. 295-312

BG Claybrook (1983) 'File Management Techniques' (Wiley)

R Coates (1987) 'Lexical Morphology', in: J Lyons et al (eds) *'New horizons in linguistics 2'*, pp. 103-121 (Penguin Books)

M Coltheart (1980) 'The semantic error: types and theories', in: M Coltheart and JC Marshall (eds) *'Deep Dyslexia'* (RKP)

D Cooper and M Clancy (1985) *'Oh! Pascal!'* (Norton)

JR Cowie (1987) 'A direct access technique for sequential files with variable length records', *Software - Practical Experience (UK),* **17** (10), pp. 719-728

D Crystal (1987) *'The Cambridge Encyclopedia of Language'* (CUP)

FJ Damerau (1964) 'A technique for computer detection and correction of spelling errors', *Comm. ACM,* **7** (3), pp. 171-176

CJ Date (1986) *'An Introduction to Database Systems — Vol. I'* (Addison-Wesley)

JL Dawson (1974) 'Suffix removal and word conflation', *ALLC Bulletin,* **2** (3), pp. 33-46

AS Dolgopolov (1986) 'A program of automatic text correction', *Auto. Doc. and Math. Linguist. (USA),* **20** (4), pp. 116-121

D Dori (1991) 'Symbolic representation of dimensioning in engineering drawings' *Proc. ICDAR-91 1st Intl Conf. on Document Analysis and Recognition,* pp. 1000-1010

P Downing (1977) 'On the creation and use of English compound nouns', *Language,* **53**, pp.810-842

LD Earnest (1962) 'Machine recognition of cursive writing', *Information processing 1962 (Proc. IFIP Congr.),* pp. 462-466

RW Ehrich and KJ Koehler (1975) 'Experiments in the contextual recognition of cursive script', *IEEE Trans. Computers,* **24**, pp. 182-194

MA Eldridge, I Nimmo-Smith and AM Wing (1984) 'The variability of selected features in cursive handwriting: Categorical measures', *Journal of the Forensic Science Society,* **24**, pp.179-219

DG Elliman and IT Lancaster (1990) 'A review of segmentation and contextual analysis techniques for text recognition', *Pattern Recognition,* **23** (3/4), pp. 337-346

AW Ellis (1979) 'Slips of the pen', *Visible Language,* **13** (3), pp. 265-282

LJ Evett and GW Humphreys (1981) 'The use of abstract graphemic information in lexical access', *Quarterly Journal of Experimental Psychology,* **33A**, pp. 325-350

LJ Evett, CJ Wells, FG Keenan, TG Rose and RJ Whitrow (1989) 'How words combine: the effects of consraints on word combinations on word recognition', *Esprit research report.*

LJ Evett, CJ Wells, FG Keenan, TG Rose and RJ Whitrow (1991) 'Using liguistic information to aid handwriting recognition', *Proc. 2nd Intl. workshop on Frontiers in Handwriting Recognition,* pp. 303-311

H Fahmy and D Blostein (1991) 'A graph grammar for high-level recognition of music notation' *Proc. ICDAR-91 1st Intl Conf. on Document Analysis and Recognition,* pp. 70-78

F Fallside and WA Woods (1985) *'Computer Speech Processing'* (Prentice-Hall)

RF Farag (1979) 'Word-level recognition of cursive script', *IEEE Trans. Computers,* **28** (2), pp. 172-175

DM Ford and CA Higgins (1990) 'A tree-based dictionary search technique and comparison with n-gram letter graph reduction', in: R Plamondon and CG Leedham (eds)*'Computer Processing of Handwriting'* (World Scientific) pp. 291-312

DM Ford (1991) 'On-line recognition of connected handwriting', *Unpublished PhD thesis,* University of Nottingham, England

E Fredkin (1960) 'Trie Memory', *Communications of the ACM* , **3** (9), pp. 490-499

H Freeman (1961) 'On the encoding of arbitrary geometric configurations', *IEE Trans. on Electronic Computers,* pp. 260-268

CC Fries (1952) *'The structure of English'* (Longman) pp. 87-141

LS Frishkopf and LD Harmon (1961) 'Machine reading of cursive script' in: C Cherry (ed), *'Information Theory'* (Butterworth) pp. 300-316

EJ Galli and H Yamada (1967) 'An Automatic Dictionary and the Verification of Machine-Readable Text', *IBM Systems Journal,* **6** (3), pp. 192-207

EJ Galli and H Yamada (1968) 'Experimental studies in computer assisted correction of unorthographic text', *IEEE Trans. Eng. Writing and Speech,* EWS-**11** (2), pp. 75-84

JJ Giangardella, JF Hudson and RS Roper (1967) 'Spelling correction by vector representation using a digital computer', *IEEE Trans. Eng. Writing and Speech,* EWS-**10** (2), pp. 57-62

CD Gibler and DS Childress (1984) 'Adaptive dictionary for computer-based communication aids', *Proc. Sixth Ann. Conf. on rehabilitation engineering promise of technology,* pp. 165-167

A Goshtasby and RW Ehrich (1988) 'Contextual word recognition using probabilistic relaxation labelling', *Pattern Recognition,* **21** (5), pp. 455-462

J Grudin (1983) 'Non-hierarchic specification of components in transcription typewriting', *Acta Psychologica,* **54**, pp. 249-262

PAV Hall and GR Dowling (1980) 'Approximate string matching', *ACM Computing Surveys,* **12** (4), pp. 381-401

P Hanks (ed) (1978) *'The Collins Dictionary of English Language'* (Collins)

AR Hanson, EM Riseman and E Fisher (1976) 'Context in word recognition', *Pattern Recognition,* **8**, pp. 35-45

LD Harmon (1962a) 'Automatic reading of cursive script', in: GL Fisher Jr. et al (eds) *'Optical Character Recognition'*, (Spartan, Washington DC) pp. 151-152(A)

LD Harmon (1962b) 'Handwriting reader recognizes whole words', *Electronics*, **35**, pp. 29-31

LD Harmon (1972) 'Automatic recognition of print and script', *Proc. IEEE*, **60**, pp. 1165-1176

L Henderson (1982) *'Orthography and word recognition in reading'* (Academic Press)

CA Higgins and RJ Whitrow (1984) 'On-line cursive script recognition', *Proc. Interact 84, 1st IFIP Conf. HCI*, pp. 140-144

CA Higgins and DM Ford (1991a) 'A new segmentation method for cursive script recognition', *Proc. 2nd Intl. workshop on Frontiers in Handwriting Recognition*, pp. 241-252

CA Higgins and DM Ford (1991b) 'Stylus driven interfaces — the electronic paper concept', *Proc. ICDAR-91 1st Intl Conf. on Document Analysis and Recognition*, pp. 853-862

TK Ho, JJ Hull and SN Srihari (1991) 'Word recognition with multi-level contextual knowledge', *Proc. ICDAR-91 1st Intl Conf. on Document Analysis and Recognition*, pp. 905-915

JN Holmes (1988) *'Speech synthesis and recognition'* (Van Nostrand Reinhold)

AS Hornby (1988) *'Oxford Advanced Learners Dictionary'* (Oxford University Press)

Y Huizhong (1986) 'A new technique for identifying scientific/technical terms and describing scientific texts', *Literary and Linguistic computing*, **1** (2), pp. 93-103

JJ Hull (1986) 'Hypothesis generation in a computational model for visual word recognition', *IEEE Expert*, pp. 63-70

JJ Hull (1987) 'A computational theory and algorithm for fluent reading', *Proc. 3rd Conf. on AI aplications*, pp. 176-181

JJ Hull, TK Ho, J Favata, V Gorindaraju and SN Srihari (1991) 'Combination of segmentation-based and wholistic handwritten word recognition algorithms' *Proc. 2nd Intl. workshop on Frontiers in Handwriting Recognition*, pp. 229-240

JJ Hull and SN Srihari (1982) 'Experiments in text recognition with binary n-gram and viterbi algorithms', *IEEE Trans. on Patt. Anal. and Mach. Intell.*, pp. 520-530

JJ Hull, SN Srihari and R Choudhari (1983) 'An integrated algorithm for text recognition: comparison with a cascaded algorithm', *IEEE Trans. Patt. Anal. and Mach. Intell.*, PAMI-**5**, pp. 384-395

GW Humphreys, LJ Evett and PT Quinlan (1990) 'The orthographic description in visual word processing', *Cognitive Psychology*, **22**, pp. 517-560

P Isabelle (1984) 'Another look at complex nominals', *Proceedings of COLING 84. Association for computational linguistics*, pp. 509-516

M Isoda, H Aiso, N Kamibayashi and Y Matsunaga (1986) 'A model for a lexical knowledge base', *Proc. COLING 86,* pp. 451-453

JK Jankovic (1986) 'The N-V dichotomy in the structure of English noun compounds?', *Literary and Linguistic Computing,* **1** (2), pp. 143-155

S Johansson and K Hofland (1987) 'The Tagged Lob Corpus: Description and Analyses' in: W Meijs (ed) *'Corpus Linguistics and Beyond'* , Proc. 7th Intl. Conf. on Computerised Corpora, Rodopi, Amsterdamk

PN Johnson-Laird (1987) 'The mental representation of the meaning of words', *Cognition,* **25**, pp. 189-211

M Kadirkamanathan and PJW Rayner (1990) 'A scale-space filtering approach to stroke segmentation of cursive script' in: R Plamondon and CG Leedham (eds)'*Computer Processing of Handwriting*' (World Scientific) pp. 133-166

JJ Katz and JA Fodor (1963) 'The structure of a semantic theory', *Language,* **39**, pp. 170-210

WJ Kashyap and BJ Oommen (1984) 'Spelling correction using probabilistic methods', *Pattern Recognition letters,* **2** (4), pp. 147-154

FG Keenan (1989) 'Overview of a post-processing system for script recognition', *Esprit Working Group Meeting,* Ulm, W. Germany

FG Keenan, LJ Evett and RJ Whitrow (1991) 'A large vocabulary stochastic syntax analyser for handwriting recognition' *Proc. ICDAR-91 ist Intl. conf. Document Analysis and Recognition,* pp. 794-802

FG Keenan and LJ Evett (1989) 'Lexical structure for natural language processing', *Proc. First Int. Workshop on Language Acquisition IJCAI-89,* Detroit, USA

DE Knuth (1973) *'Art of Computer Programming vol 3: Sorting and Searching'* (Addison-Wesley)

JF Korsh (1980) *'Data Structures, Algorithms and Program Style'* (Wadsworth)

H Kucera and WN Francis (1967) *'Computational Analysis of Present-Day American English'*, (Brown University Press)

SM Lamb and WH Jr Jacobsen (1961) 'A High-Speed Large-Capacity Dictionary System', *Mechanical Translation,* **6**, pp. 76-107

G Leedham (1990) 'Automatic recognition and transcription of Pitman's handwritten shorthand', in: R Plamondon and CG Leedham (eds)'*Computer Processing of Handwriting*' (World Scientific) pp. 235-269

WG Lehnert and MH Ringle (eds) (1982)*'Strategies for natural language processing'*, (Lawrence Erlbaum Associates)

M Lesk (1986) 'Why I want the OED on my computer and when I'm likely to have it', *SIGCUE Outlook (USA)*, **19** (1-2), pp. 62-66

JN Levi (1978) *'The syntax and semantics of complex nominals'* (Academic Press)

JB Lovins (1968) 'Development of a stemming algorithm', *Mechanical Translation*, **11**, pp.22-31

HP Luhn (1959) 'Potentialities of auto-encoding of scientific literature', *Technical Report RC-101*, IBM Corp., Research Center, Yorktown Heights, New York

DB Lysak Jr. and R Kasturi (1991) 'Interpretation of engineering drawings of polyhedral and non-polyhedral objects' *Proc. ICDAR-91 1st Intl Conf. on Document Analysis and Recognition*, pp. 79-87

K Maly (1976) 'Compressed Tries', *Communications of the ACM*, **19** (7), pp. 409-415

E Marsh (1984) 'A computational analysis of complex noun phrases in Navy messages', *Proceedings of COLING 84. Association for computational linguistics*, pp. 505-508

WD Marslen-Wilson and A Welsh (1978) 'Processing interactions and lexical access during word recognition in continuous speech', *Cognitive Psychology*, **10**, pp. 29-63

JL McLelland and DE Rumelhart (1981) 'An interactive activation model of context effects in letter perception: part 1, An account of basic findings', *Psychological Review*, **88** (5), pp. 375-407

W Meijs (1985) 'Lexical organisation from three different angles', *ALLC Journal*, **6**, pp. 1-10

G and C Merriam Co. (1963) *'Webster's New Collegiate Dictionary (7th Edition)'*, (G. and C. Merriam Co., Springfield, Massachusetts)

WJ Meys (1975) *'Compound adjectives in English and the ideal speaker listener'*, (North Holland)

GA Miller, EB Newman and EA Friedman (1958) 'Length-frequency statistics for written English', *Information and Control*, **1**, pp. 370-389

DC Mitchell (1982) *'The Process of Reading — A Cognitive Analysis of Fluent Reading and Learning to Read'* (Wiley)

R Mitton (1986) *'The Machine Usable Form of the Oxford Advanced Learners Dictionary'*, Oxford Text Archive

R Mitton (1987) 'Spelling checkers, spelling correctors and the misspellings of poor spellers', *Info. Proc. and Mgmt.*, **23** (5), pp. 495-505

P Morasso and S Pagliano (1991) 'Neural models for handwriting recognition', *Proc. 2nd Intl. workshop on Frontiers in Handwriting Recognition*, pp. 327-340

J Morton (1964) 'The effects of context on the visual duration threshold for words', *British Journal of Psychology*, **55**, pp. 165-180

J Morton (1969) 'Interaction of information in word recognition', *Psychological Review*, **76** (2), pp. 165-178

J Morton (1970) 'A functional model for memory', in: DA Norman (ed)*'Models of Human Memory'* (Academinc Press), pp. 203-254

JH Munson (1968) 'Experiments in the recognition of hand-printed text: Part I - Character recognition', *Proc. AFIPS Fall Joint Computer Conference*, pp. 1125-1138

MS Neff, RJ Byrd and OA Rizk (1988) 'Creating and querying lexical databases', *Proc. Conf. on Appl. Nat. Lang. Processing*, pp. 84-92

DL Neuhoff (1975) 'The Viterbi algorithm as an aid in text recognition', *IEEE Trans. Inform. Theory*, **21**, pp. 222-226

J Nievergelt (1974) 'Binary search trees and file organisation', *Computing surveys*, **6**(3)

MK Odell and RC Russell (1918) US Patent no. 1,261,167 and (1922) US Patent no. 1,435,663

MA O'Hair and M Kabrisky (1991) 'Recognizing whole words as single symbols', *Proc. ICDAR-91 1st Intl. Conf. on Document Analysis and Recognition,* pp. 350-358

Oxford (1989)*'The New Oxford English Dictionary'*, (Oxford University Press)

CD Paice (1977) *'Information retrieval and the computer'*, (MacDonald and Jane's Computer Monographs)

G Pavlovic-Lazetic and E Wong (1986) 'Managing text as data', *Proc. VLDB Twelvth Intl. Conf.,* pp. 111-116

JL Peterson (1980) 'Computer programs for detecting and correcting spelling errors', *Comm. of ACM,* **23** (12), pp. 676-687

JL Peterson (1982) 'Use of W7 New Collegiate Dictionary to construct a master hyphenation list', *Proc. AFIPS Nat. Comp. Conf.,* **51**, pp. 665-670

E Picchi and N Calzolari (1986) 'Textual perspectives through an automized lexicon', *Proc. of XII Intl. ALLC Conf*

DB Pisoni, HC Nusbaum, PA Luce and LM Slowiaczek (1985) 'Speech perception, word recognition and the structure of the lexicon', *Speech Communication 4,* pp. 75-95

P Procter (ed.) (1978) *'Longman's Dictionary of Contemporary English'*, (Longman Group Ltd.)

A Ramsay and R Barrett (1987) *'AI in Practice: Examples in POP11'*, (Ellis Horwood)

J Raviv (1967) 'Decision making in Markov chains applied to the problem of pattern recognition', *IEEE Trans. Inform. Theory,* IT-**3**, pp. 536-551

K Rayner, M Carlson and L Frazier (1983) 'The interaction of syntax and semantics during sentence processing: Eye movements in the analysis of semantically biased sentences', *Journal of Verbal Learning and Verbal Bahaviour,* **22** (3), pp. 358-374

HL Resnikoff and JL Dolby (1965) 'The nature of affixing in written English: Part I', *Mechanical Translation,* **8** (3), pp. 84-89

HL Resnikoff and JL Dolby (1966) 'The nature of affixing in written English: Part II', *Mechanical Translation*, **9** (2), pp. 23-33

EM Riseman and RW Ehrich (1971) 'Contextual word recognition using binary digrams', *IEEE Trans. Comput.*, **20** (4), pp.397-403

EM Riseman and AR Hanson (1974) 'A contextual post processing system for error correction using binary n-grams', *IEEE Trans. Comput.*, C-**23**, pp.480-493

G Ritchie, A Black, S Pulman and G Russell (1987) 'The Edinburgh Morphological Analyser and Dictionary System: System Description, Version 3', *Software Paper 11,* Dept. of AI, University of Edinburgh

T Roeper and MEA Siegel (1978) 'A lexical transformation for verbal compounds', *Linguistic Inquiry,* **9** (2), pp. 199-260

TG Rose LJ Evett and RJ Whitrow (1991) 'The use of semantic information as an aid to handwriting recognition', *Proc. ICDAR-91 1st Intl Conf. on Document Analysis and Recognition,* pp. 629-637

A Rudnicky and L Baumeister (1987) 'The lexical access component of the CMU continuous speech recognition system', *Proc. Intl. Conf. Acoustics and Signal Processing,* **1**, pp. 376-379

DE Rumelhart and JL McLelland (1982) 'An interactive activation model of context effects in letter perception part 2. The contextual enhancement effect and some tests and extensions of the model', *Psychological Review,* **89** (1), pp. 60-94

GJ Russell, SG Pulman, GD Ritchie and AW Black (1986) 'A Dictionary and Morphological Analyser for English', *Proc. 11th Intl. Conf. Computational Linguistics,* pp. 277-279

NY Salmina and IA Khodashinskii (1986) 'Methods and tools of automatic spelling correction', *Auto. Doc. and Math. Linguist. (USA),* **20** (5), pp. 105-112

G Sampson (1989) 'How fully does a machine-readable dictionary cover English text?', *Literary and Linguistic Computing,* **4** (1), pp. 29-35

KM Sayre (1973) 'Machine recognition of handwritten words: A project report', *Pattern Recgnition,* **5**, pp. 213-228

LRB Schomaker and HL Teulings (1990) 'A handwriting recognition system based on properties of the human motor system', *Proc. Intl. Workshop on Frontiers in Handwriting Recognition,* Montreal

LRB Schomaker and HL Teulings (1991) 'Stroke- versus character-based recognition of on-line, connected cursive script', *Proc. 2nd Intl. Workshop on Frontiers in Handwriting Recognition,* pp 265-277

R Schreuder (1986) 'Using lexical databases in psycholinguistic research', *Interfaculty research unit for language and speech*, University of Nijmegen, Netherlands

EJ Schuegraf (1976) 'A survey of data compression methods for non-numeric records', *Cam. J. Info. Sci.,* **2** (1), pp. 93-105

E Schwartz (1963) 'A dictionary for minimum redundancy encoding', *J. of ACM,* **10**, pp. 413-439

BA Sheil (1978) 'Median Split Trees: A Fast Lookup Technique for Frequently Occurring Keys', *Communications of the ACM*, **28**, pp. 947-958

D Sherman (1974) 'A new computer format for W7 Collegiate Dictionary', *Comput. and Hum. (USA)*, **8** (1), pp. 21-26

R Shinghal, D Rosenberg and GT Toussaint (1978) 'A simplified heuristic version of a recursive Bayes algorithm for using context in text recognition', *IEEE Trans. Syst. Man. and Cybern.*, SMC-**8** (5), pp. 412-414

R Shinghal and GT Toussaint (1979a) 'Experiments in text recognition with the modified viterbi algorithm', *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-**1** (2) pp. 184-193

R Shinghal and GT Toussaint (1979b) 'A bottom-up and top-down approach to using context in text recognition', *Intl. Journal of Man-machine studies*, pp. 201-212

JM Sinclair (1966) 'Beginning the study of lexis', in: C Bazell et al (eds)'*In memory of J R Firth*' (Longman) pp. 410-30

RMK Sinha (1987) 'Some characteristic curves for dictionary organization with digital search', *IEEE Trans. Syst. Man. and Cybern.*, SMC-**17** (3)

RMK Sinha and B Prasada (1988) 'Visual text recognition through contextual processing', *Pattern Recognition*, **21** (5), pp. 463-479

RMK Sinha (1990) 'On partitioning a dictionary for visual text recognition', *Pattern Recognition*, **23** (5), pp. 497-500

J Skrzypek, E Tisdale and K Frankel (1991) 'Neural architectures for recognition of cursive handwriting: comparative analysis', *Proc. 2nd Intl. workshop on Frontiers in Handwriting Recognition*, pp. 341-351

SN Srihari, JJ Hull and R Choudhari (1983) 'Integrating diverse knowledge sources in text recognition', *ACM Trans. on Office Systems*, **1** (1), pp. 68-87

SN Srihari and RM Bozinovic (1987) 'A multi-level perception approach to reading cursive script', *Artificial Intelligence,* **33**, pp. 217-255

TA Standish (1980) *'Data structure techniques'* (Addison-Wesley)

DF Stubbs and NW Webre (1985) *'Data structures with absract data types and pascal'* (Brooks/Cole)

EH Sussenguth Jr (1963) 'Use of Tree Structures for Processing Files', *Communications of the ACM,* **6** (5), pp. 272-279

MN Swamy and K Thulasiraman (1981) *'Graphs, Networks and Algorithms'* (Wiley)

M Taft (1979) 'Lexical access via an orthographic code: the Basic Orthographic Syllabic Structure (BOSS)', *Journal of Verbal Learning and Verbal Behaviour,* **18**, pp. 21-39

M Taft (1983) 'The decoding of words in lexical access: a review of the morphographic approach', in: Besner, MacKinnon and Waller (eds) *'Reading research'*, **5**

M Taft (1987) 'Morphological Processing: The BOSS re-emerges', in: M Coltheart (ed) *Proceedings of the 12th international symposium on Attention and Performance* (Lawrence Erlbaum Associates) pp. 265-279

CC Tappert (1982) 'Cursive script recognition by elastic matching', *IBM Journal of Research and Development,* pp. 765-771

CC Tappert (1988) 'A divide-and-conquer cursive script recognizer', *IBM Res. Report,* RC14070

CC Tappert, CY Suen and T Wakahara (1990) 'The state of the art in on-line handwriting recognition', *IEEE Trans. on Pattern Analysis and Machine Intelligence,* **12** (8), pp. 787-808

W Teubert (1984) 'Applications of a lexical database for German', *Proc. COLING 84*, pp. 34-37

HL Teulings, LRB Schomaker, J Gerritsen, H Drexler and M Albers (1990) 'An on-line handwriting-recognition system based on unreliable modules' in: R Plamondon and CG Leedham (eds)'*Computer Processing of Handwriting*' (World Scientific) pp. 167-185

HL Teulings and LRB Schomaker (1991) 'Unsupervised learning of prototype allographs in cursive-script recognition using invariant handwriting features', *Proc. 2nd Intl. workshop on Frontiers in Handwriting Recognition*, pp. 45-55

P Trigano, P Morizet-Mahoudeaux and P le Beux (1988) 'Dialphil - a man machine interface in natural language', *Proc. 2nd Intl. Exp. Sys. Conf.*, pp. 391-398

H Trost and E Buchberger (1986) 'Towards the automatic acquisition of lexical data', *COLING 86*, pp. 387-389

JR Ullman (1977) 'A binary n-gram technique for automatic correction of substitution, deletion, insertion and reversal errors in words', *The Computer Journal*, **20**, pp. 141-147

RL Venezky (1970) 'The structure of English Orthography', *Janua Linguarum 82*, (Mouton Press)

AJ Viterbi (1967) 'Error bounds for convolutional codes and an asymptotically optimal decoding algorithm', *IEEE Trans. Inform. Theory*, **IT-13**, pp. 260-269

M Waite (1989) 'Data structures for the reconstruction of engineering drawings', *Unpublished PhD thesis*, Nottingham Polytechnic

DE Walker (1986) 'Knowledge resource tools for information access', *Future Generation Computer Systems*, **2** (3), pp. 161-171

DE Walker (1989) 'Developing lexical resources', *Proceedings of the Fifth annual conference of the UW centre for the New Oxford English Dictionary,* pp. 1-22

B Warren (1978) *'Semantic patterns of Noun-Noun compounds'*, Gothenburg studies in English, no. 41

E Wehrli (1985) 'Design and implementation of a lexical database', *Proc. 2nd Conf. Europ. Chap. Assoc. for Comput. Ling.,* pp. 146-153

ESC Weiner (1985) 'The new OED: problems in computerization of a dictionary', *Univ. Comput. GB (OUP),* **7** (2), pp. 66-71

LK Welbourn and RJ Whitrow (1990) 'A gesture based text and diagram editor', in: R Plamondon and CG Leedham (eds)'*Computer Processing of Handwriting*' (World Scientific) pp. 221-234

CJ Wells, LJ Evett, PE Whitby and RJ Whitrow, (1989) The use of letter patterns for script recognition, *Proc. IEE Colloquium,* Oct. 1989

CJ Wells, LJ Evett, PE Whitby and RJ Whitrow (1990a), 'The use of orthographic information for script recognition', in: R Plamondon and CG Leedham (eds)'*Computer Processing of Handwriting*' (World Scientific) pp. 273-289

CJ Wells, LJ Evett, PE Whitby and RJ Whitrow, (1990b) 'Fast dictionary look-up for contextual word recognition', *Pattern Recognition,* **23** (5), pp. 501-508

CJ Wells, LJ Evett, and RJ Whitrow, (1991) 'Word look-up for script recognition — Choosing a candidate', *Proc. ICDAR-91 1st Intl Conf. on Document Analysis and Recognition,* pp. 620-628

RJ Whitrow and CA Higgins (1987) 'The application of n-grams for script recognition', *Proc. 3rd Int. Symp. on Handwriting and Computer Applications,* pp. 92-94

M Wilson (1984) 'The composition of the mental lexicon', *Unpublished PhD thesis*, Cambridge University

M Wilson (1987) 'MRC Psycholinguistic Database: Machine Usable Dictionary. Version 2.00', *RAL-87-054*, Rutherford Appleton Laboratory

MS Wilson and RJ Whitrow (forthcoming) 'The development of a gesture based freehand editor', paper to be presented at *HCI 92, People and Computers*

AM Wing (1979) 'Variability in handwriting characters', *Visible Language*, **13** (3), pp. 283-298

AM Wing and AD Baddeley (1979) 'Spelling errors in handwriting', in: U Frith (ed) *'Cognitive Processes in Handwriting'* (Academic Press)

PH Winston (1984) *'Artificial Intelligence'* (Addison-Wesley)

N Wirth (1976) *'Algorithms + Data Structures = Programs'* (Prentice-Hall)

KH Wong and F Fallside (1985) 'Dynamic programming in the recognition of connected handwriting script', *Proc. 2nd Intl. Conf. on Artificial Intelligence Applications*, (IEEE Comput. Soc.), pp. 666-670

PT Wright (1989) 'Algorithms for the recognition of handwriting in real-time', *Unpublished PhD thesis*, Nottingham Polytechnic

Appendix A

Table A1 — Example pattern recogniser results

Word	Recognised as
a	oi
superb	siyroib
detached	defadiod
house	howse
situated	atiiafed
in	cii
a	oe
most	iiioid
sought	sovwfiie
after	affci
residential	iendaitoh
area	aioi
on	oiii
the	fhc
carters	xerkrs
estate	chcife
within	mhwii
easy	covsij
walking	wahaiioj
distance	dihaiia
of	oif
western	iisfciii
grove	cfiivie
with	iiith / iiihi
its	its
multiple	iuiihysb
shopping	dwippwg
facilities	faochfies

N.B. "Recognised as" means the highest rated candidate string that the pattern recogniser would have chosen, given no further processing, i.e. no lexical look-up.

Appendix B

Passage of test data:

> *a superb detached house situated in a most sought after residential area on the carters estate within easy walking distance of western grove with its multiple shopping facilities. the cliff top with its access to sandy bathing beaches is also within easy walking distance. the area is well served with excellent local sporting facilities including numerous local golf courses, tennis courts and local sports centres which are all within easy reach. the area is also well served with excellent local schools for children of all ages. the property is in need of modernisation and redecoration but offers good sized accommodation with three reception rooms, four good bedrooms and large secluded well stocked rear garden.*

Table B1 (overleaf) shows results from look-up in a lexicon of 15,223 words.

Table B1— Numbers of candidate strings produced for a passage of test data

Word	No. of candidate strings	No. allowable
a	16	1
superb	222196	0
detached	588362328	3
house	305305	2
situated	41927138	3
in	96	5
a	27	1
most	59814	10
sought	8242458	1
after	15744	11
residential	12335900000	1
area	8202	4
on	386	4
the	611	10
carters	203586	7
estate	91560	2
within	4013362	5
easy	6036	8
walking	257735766	4
distance	36896304	1
of	136	4
western	1655245	2
grove	64451	1
with	13957	8
its	39	2
multiple	395809874	1
shopping	64433076	0
facilities	50511162	9
the	611	10
cliff	4824	4
top	372	4
with	20783	11
its	66	3
access	48576	39
sandy	649464	4
bathing	292947684	4
beaches	4270144	11
is	13	1
also	5484	4
within	4125537	4
easy	5204	5
walking	345349488	3
distance	38645296	3

Table B1 (continued)

Word	No. of candidate strings	No. allowable
the	496	8
area	11169	6
is	15	1
well	1913	9
served	43530	5
with	16909	1
excellent	15211278	1
local	21378	1
sporting	4627560	1
facilities	91260610	8
including	11607300000	2
numerous	15334700000	2
local	15511	71
golf	2760	1
courses	483330	1
tennis	562347	51
courts	18137	3
and	17602	6
local	32094	1
sports	19170	2
centres	308338	31
which	966712	7
are	672	8
all	652	5
within	4410160	7
easy	13364	4
reach	844620	17

Table B1 (continued)

Word	No. of candidate strings	No. allowable
the	833	13
area	17265	11
is	13	2
also	3114	1
well	2725	13
served	102489	7
with	27310	12
excellent	43207298	2
local	18594	2
schools	593082	1
for	207	4
children	38863960	3
of	167	8
all	364	6
ages	6188	7
the	475	7
property	25083010	2
is	13	1
in	91	5
need	11103	29
of	167	7
modernisation	149629000000000	1
and	14566	7
redecoration	600077000000	1
but	3270	5
offers	8446	3
good	5663	4
sized	1108	1
accommodation	4850070000000000	1
with	33938	5
three	15913	3
reception	389160864	1
rooms	884128	2
four	29499	62
good	14098	4
bedrooms	313429991	1
and	21160	4
large	63151	1
secluded	7392332	1
well	1770	6
stocked	444636	2
rear	23610	9
garden	7378812	1

Appendix C

Recognition results from lexical look-up, with no use of compounding information:

a(1/1) —(0) detached(1/3) house(1/2) situated(1/3) in(1/5) a(1/1) most(1/10) sought(1/1) after(1/10) residential(1/1) area(1/4) on(1/4) flu(3/10) carters(1/7) estate(1/2) within(1/5) easy(1/8) walking(1/4) distance(1/1) of(1/4) western(1/2) grove(1/1) with(1/8) its(1/2) multiple(1/1) —(0) families(2/9). the(1/10) cliff(1/4) top(1/4) with(1/10) its(1/3) access(1/10) to(1/1) sandy(1/4) framing(2/4) beaches(1/10) is(1/1) dine(2/4) whim(2/4) easy(1/5) walking(1/3) distance(1/3). the(1/8) area(1/6) is(1/1) coal(9; correct word not recognized) sewed(4/5) mill(1; correct word not recognized) excellent(1/1) local(1/1) spoiling(1; correct word not recognized) facilities(1/8) including(1/2) numerous(1/2) local(1/10) golf(1/1) courses(1/1), tennis(1/10) comb(1; correct word not recognized) duel(4/6) local(1/1) sports(1/2) cams(2/10) which(1/7) arc(3/8) cry(4/5) whim(2/7) easy(1/4) reach(1/10). he(2/10) and(2/10) us(2/2) also(1/1) wool(2/10) saved(3/7) with(1/10) excellent(1/2) local(1/2) schools(1/1) for(1/4) children(1/3) of(1/8) all(1/6) ages(1/7). the(1/7) property(1/2) is(1/1) in(1/5) weed(2/4) off(2/6) modernisation(1/1) anal(2/7) redecoration(1/1) but(1/5) offers(1/3) god(3/4) sized(1/1) accommodation(1/1) with(1/5) three(1/3) reception(1/1) rooms(1/2), four(1/10) grid(2/4) bedrooms(1/1) and(1/4) large(1/1) secluded(1/1) well stocked(1/2) wear(9; correct word not recognized) garden(1/1).

Recognition results after look-up in compound tree:

a(1/1) —(0) detached house(1/1) situated(1/3) in(1/5) a(1/1) most(1/10) sought after(1/1) residential area(1/1) on(1/4) flu(3/10) carters(1/7) estate(1/2) within(1/5) easy walking distance(1/1) of(1/4) western(1/2) grove(1/1) with(1/8) its(1/2) multiple(1/1) —(0) families(2/9). the(1/10) cliff top(1/1) with(1/10) its(1/3) access(1/10) to(1/1) sandy(1/4) framing(2/4) beaches(1/10) is(1/1) dine(2/4) whim(2/4) easy walking distance(1/1). the(1/8) area(1/6) is(1/1) coal(9; correct word not recognized) sewed(4/5) mill(1; correct word not recognized) excellent(1/1) local(1/1) spoiling(1; correct word not recognized) facilities(1/8) including(1/2) numerous(1/2) local(1/10) golf courses(1/1), tennis(1/10) comb(1; correct word not recognized) duel(4/6) local(1/1) sports centres(1/1) which(1/7) arc(3/8) cry(4/5) within easy reach(1/1). he(2/10) and(2/10) us(2/2) also(1/1) wool(2/10) saved(3/7) with(1/10) excellent(1/2) local schools(1/1) for(1/4) children(1/3) of(1/8) all(1/6) ages(1/7). the(1/7) property(1/2) is(1/1) in(1/5) weed(2/4) off(2/6) modernisation(1/1) anal(2/7) redecoration(1/1) but(1/5) offers(1/3) good sized accommodation(1/1) with(1/5) three(1/3) reception rooms(1/1), four(1/10) grid(2/4) bedrooms(1/1) and(1/4) large(1/1) secluded(1/1) vial(4/6) stocked(1/2) wear(9; correct word not recognized) garden(1/1).

The above results are from look-up in a lexicon of 15223 words.

Recognition results with top five candidate words for the test sentence:

> *the cliff top with its access to sandy bathing beaches is also within easy walking distance*

Table C1 — Recognition results after lexical check

Word	Candidates and scores				
the	the (100)	me (100)	flu (93)	file (85)	five (80)
cliff	cliff (76)	dolt (60)	cook (50)	colic (40)	
top	top (80)	his (80)	tip (67)	lip (53)	
with	with (100)	him (87)	oath (85)	mill (80)	will (80)
its	its (100)	us (90)	ox (70)		
access	access (97)	does (95)	cubes (84)	dices (84)	codes (80)
to	to (100)				
sandy	sandy (100)	sanely (93)	sorely (87)	solely (80)	
bathing	framing (100)	bathing (94)	training (70)	paining (69)	
beaches	beaches (94)	barons (83)	baring (73)	having (70)	havens (70)
is	is (90)				
also	dine (85)	dire (75)	also (75)	cure (70)	
within	whim (100)	within (100)	whom (80)	follow (60)	
easy	easy (95)	cords (88)	colds (76)	odds (75)	aids (75)
walking	walking (83)	revoking (78)	rocking (66)		
distance	distance (93)	enhance (80)	instance (78)		

Table C2 — Recognition results after compounds checked

Word	Candidates and scores				
the	the (100)	me (100)	flu (93)	file (85)	five (80)
cliff	cliff (100)				
top	top (100)				
with	with (100)	him (87)	oath (85)	mill (80)	will (80)
its	its (100)	us (90)	ox (70)		
access	access (97)	does (95)	cubes (84)	dices (84)	codes (80)
to	to (100)				
sandy	sandy (100)	sanely (93)	sorely (87)	solely (80)	
bathing	framing (100)	bathing (94)	training (70)	paining (69)	
beaches	beaches (94)	barons (83)	baring (73)	having (70)	havens (70)
is	is (90)				
also	dine (85)	dire (75)	also (75)	cure (70)	
within	whim (100)	within (100)	whom (80)	follow (60)	
easy	easy (100)				
walking	walking (100)				
distance	distance (100)				

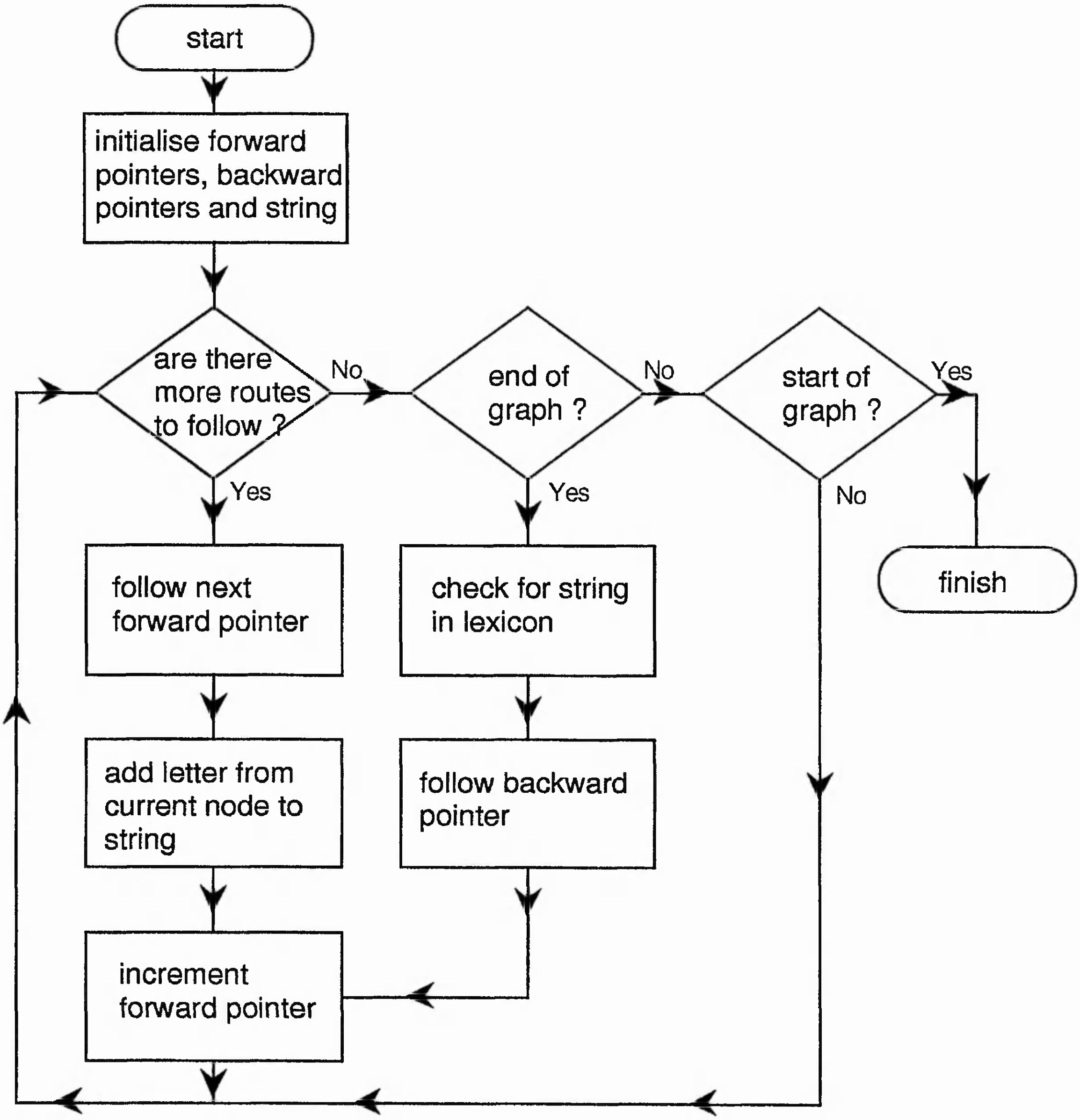
Table C3 — Recognition results after syntactic analysis

Word	Candidates and scores				
the	the (150)	me (145)	file (126)	five (122)	tire (121)
cliff	cliff (150)				
top	top (150)				
with	with (150)	him (134)	will (128)	oath (128)	mill (124)
its	its (148)	us (140)	ox (115)		
access	access (146)	does (142)	dices (132)	cubes (131)	codes (129)
to	to (150)				
sandy	sandy (150)	sanely (132)	sorely (126)	solely (119)	
bathing	framing (150)	bathing (144)	training (120)	paining (90)	
beaches	beaches (138)	barons (127)	havens (114)	hiding (107)	failing (92)
is	is (140)				
also	dine (132)	also (125)	dire (124)	cure (117)	
within	within (150)	whim (133)	whom (129)	follow (103)	
easy	easy (150)				
walking	walking (150)				
distance	distance (150)				

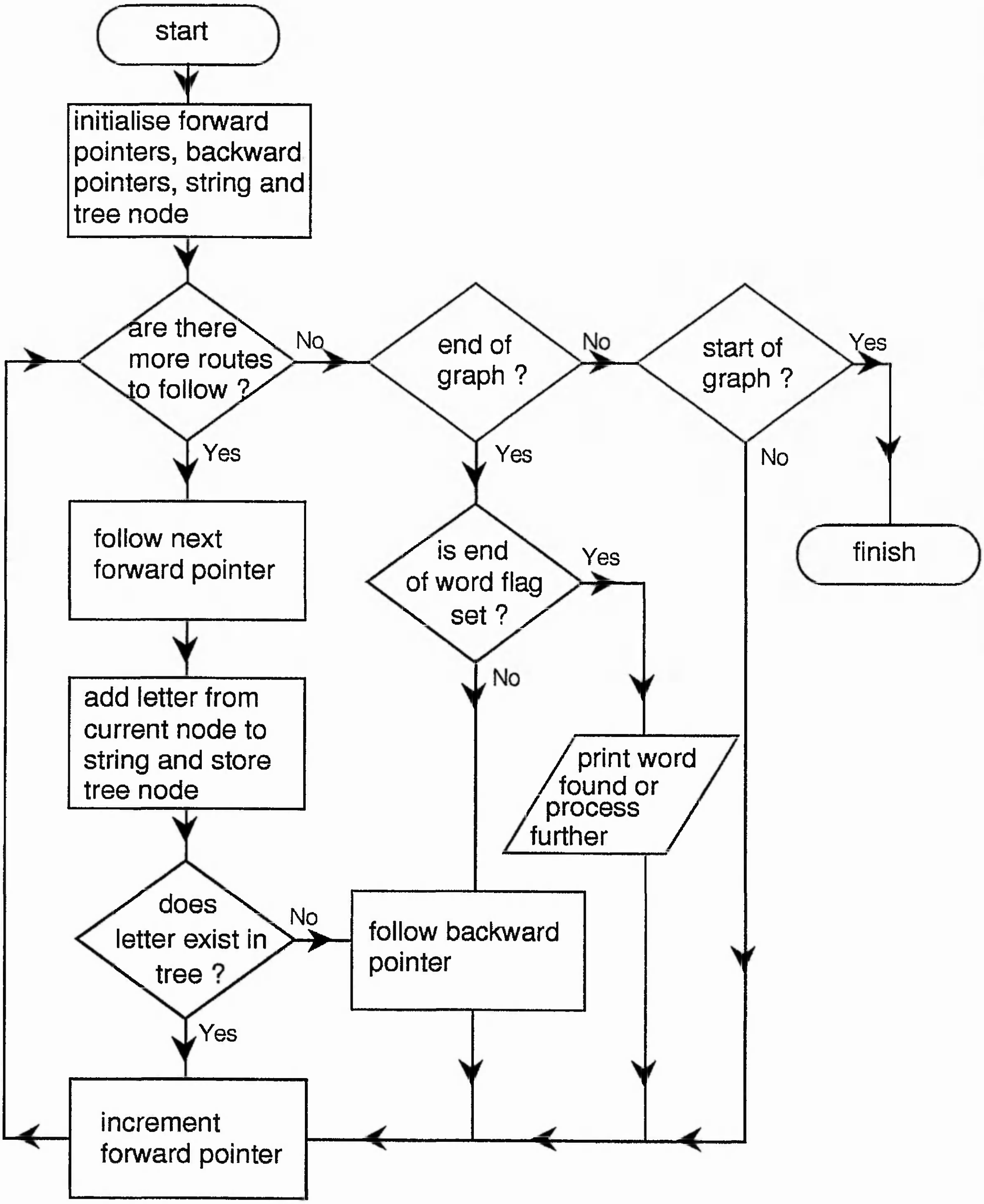
Table C4 — Recognition results after semantic analysis

Word	Candidates and scores				
the	the (175)	me (170)	file (140)		
cliff	cliff (175)				
top	top (175)				
with	with (175)	him (159)	will (141)	oath (136)	
its	its (173)	us (165)			
access	access (168)	does (167)	dices (147)	dicey (142)	odes (142)
to	to (175)				
sandy	sandy (175)	sanely (136)	sorely (130)		
bathing	bathing (169)	framing (159)			
beaches	beaches (163)	barons (137)	havens (115)		
is	is (165)				
also	also (150)	dine (133)	dire (126)	cure (121)	
within	within (175)	whom (154)	whim (142)		
easy	easy (175)				
walking	walking (175)				
distance	distance (175)				

Appendix D

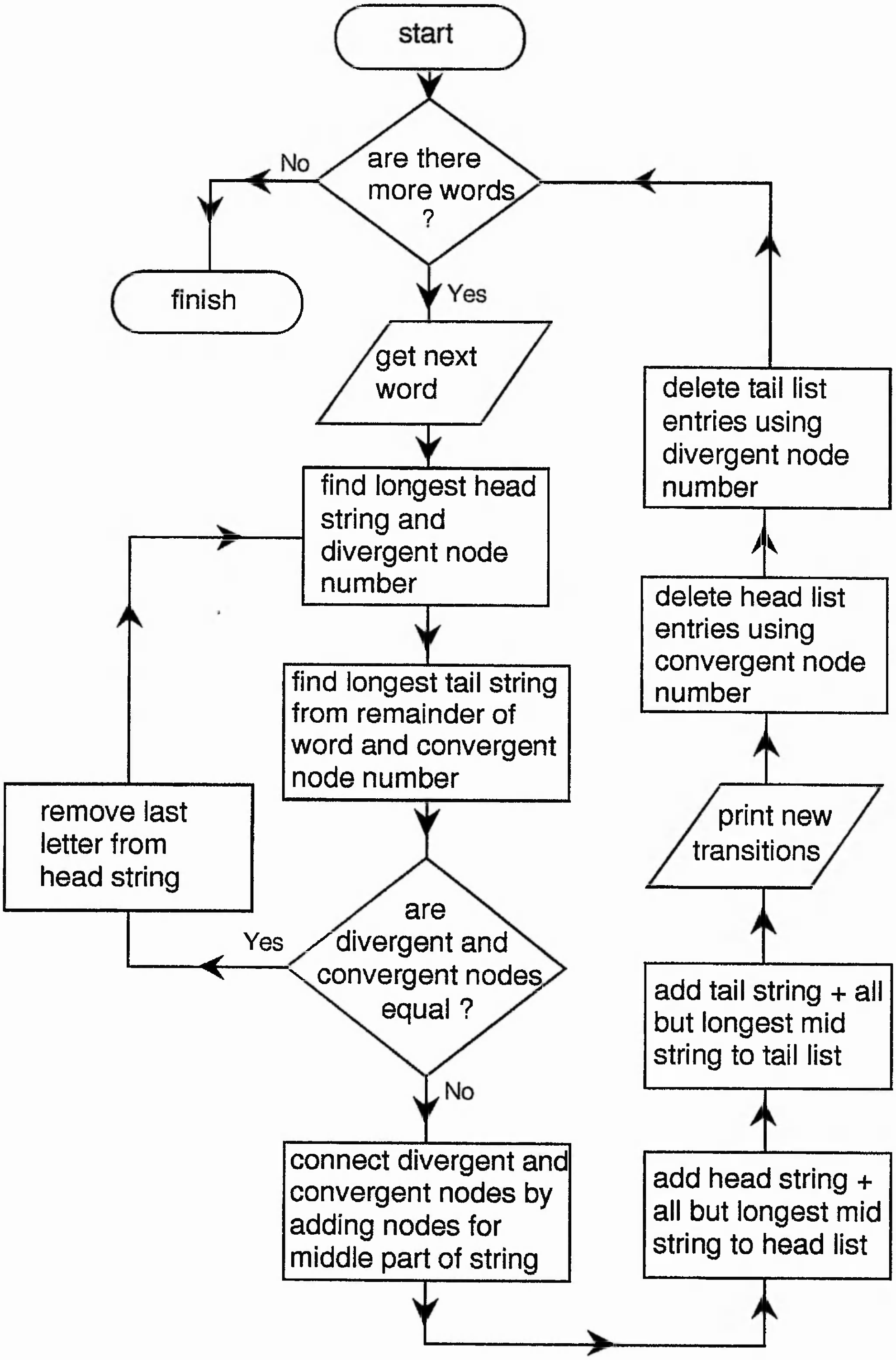

start
initialise forward pointers, backward pointers and string
are there more routes to follow ?
No
Yes
end of graph ?
No
Yes
start of graph ?
Yes
No
finish
follow next forward pointer
add letter from current node to string
increment forward pointer
check for string in lexicon
follow backward pointer

Flowchart D1 — Graph traversal algorithm (§2.5)


start
initialise forward pointers, backward pointers, string and tree node
are there more routes to follow ?
No
Yes
end of graph ?
No
Yes
start of graph ?
Yes
No
finish
follow next forward pointer
is end of word flag set ?
Yes
No
add letter from current node to string and store tree node
print word found or process further
does letter exist in tree ?
No
Yes
follow backward pointer
increment forward pointer

Flowchart D2 — Algorithm for checking letter strings against tree structure of lexicon

(see §3.3)

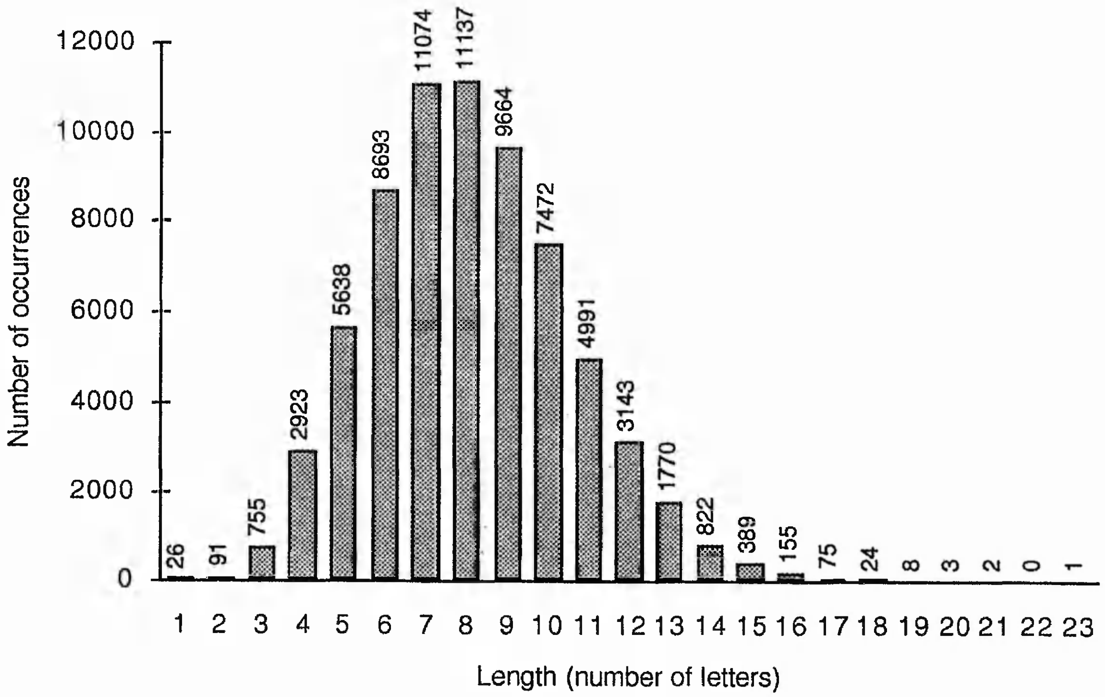

start
are there more words ?
No
finish
Yes
get next word
find longest head string and divergent node number
find longest tail string from remainder of word and convergent node number
are divergent and convergent nodes equal ?
Yes
remove last letter from head string
No
connect divergent and convergent nodes by adding nodes for middle part of string
add head string + all but longest mid string to head list
add tail string + all but longest mid string to tail list
print new transitions
delete head list entries using convergent node number
delete tail list entries using divergent node number

Flowchart D3 — Dawg construction algorithm (§ 3.3.3)

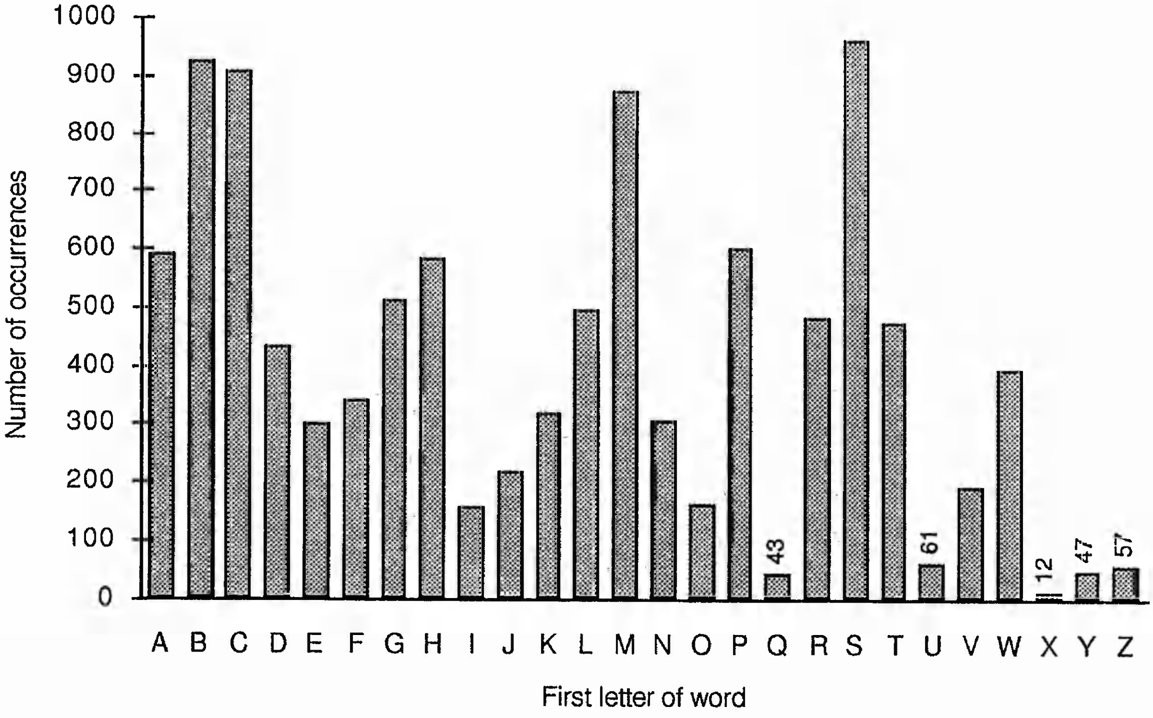
Appendix E

N.B. The data for figures E1 to E7 was collected from a lexicon of 68856 words. Tables E3 to E9 are for words of lower case only (i.e. no proper nouns or acronyms).

Histogram E1 — Lexicon frequency distribution by length



Histogram E2 i — Lexicon frequency distribution by first letter



Histogram E2 ii — Lexicon frequency distribution by first letter

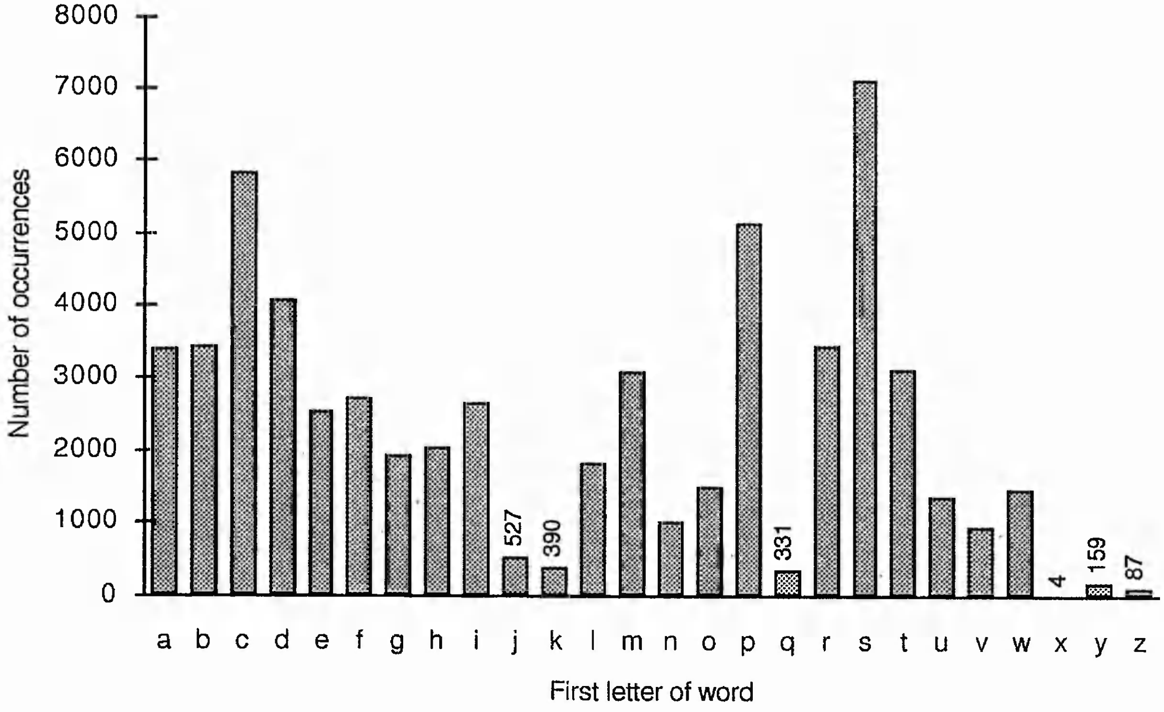


Table E3 — Zone code frequency distribution of two letter words

[8 out of 9 possible codes occur]

Zone code	Frequency
ll	0
ul	1
lu	1
ml	4
lm	5
uu	6
mu	13
um	15
mm	30
Total	75

Table E4 — Zone code frequency distribution of three letter words

[22 out of 27 possible codes occur]

Zone code	Frequency	Example word
mlu	2	apt
mll	2	egg
ulm	2	dye
luu	2	jib
lum	4	gin
lul	4	ply
mlm	8	ape
uul	9	big
mul	10	alp
lml	11	peg
uuu	14	hid
uum	17	the
muu	18	wit
lmu	20	yet
mml	26	rag
lmm	26	yes
mum	28	elm
umu	31	but
uml	35	fog
mmu	63	and
umm	63	has
mmm	75	men
Total	470	

Table E5 — Zone code frequency distribution of four letter words

[59 out of 81 possible codes occur]

Zone code	Frequency
mlml	1
mlul	1
mllm	1
umlu	1
umll	1
uulu	1
luul	1
llmm	1
mmlu	2
mmll	2
mulu	2
mluu	2
ulmu	2
ullm	2
mmul	3
mlum	3
uuul	3
luml	4
luum	4
luuu	4
lulm	4
mulm	5
uulm	5
lmul	5
mlmu	6
ulmm	7
uuml	9
lumu	9
muul	11
umul	11
uuuu	11
lmlm	11
mlmm	12
lumm	14
muuu	18
lmml	18

(continued overleaf)

Table E5 (continued)

Zone code	Frequency
lmuu	20
uuum	21
mmlm	23
umlm	25
muml	26
uumu	31
lmum	35
mumu	38
muum	39
umml	39
lmmm	40
mmml	41
lmmu	48
mmuu	51
umum	52
uumm	53
umuu	56
mumm	57
mmum	87
mmmu	132
ummu	143
mmmm	144
ummm	147
Total	1545

Table E6 — Zone code frequency distribution of five letter words

[147 out of 243 possible codes occur]

Zone code	Frequency
32 codes	1
11 codes	2
12 codes	3
10 codes	4
8 codes	5
3 codes	6
4 codes	7
3 codes	8
4 codes	9
7 codes	10
1 code	11
5 codes	12
2 codes	13
1 code	15
3 codes	16
2 codes	19
2 codes	20
2 codes	21
1 code	23
2 codes	24
3 codes	27
1 code	28
2 codes	29
1 code	30
mmmuu	33
lmmum	34
lmmmu	35
umumu	37
mmmml	38
umuum	42
uummu	42
mmumu	44
mumum	46
umumm	51
uummm	51
ummuu	56
lmmmm	58
mmuum	66
mummu	72
mmmum	89
mummm	101
mmumm	104
ummmu	110
ummum	110
mmmmu	118
mmmmm	143
ummmm	148
Total	2596

Table E7 — Zone code frequency distribution of six letter words

[307 out of 729 possible codes occur]

Nmber of codes	Frequency
69 codes	1
41 codes	2
25 codes	3
27 codes	4
12 codes	5
9 codes	6
10 codes	7
9 codes	8
7 codes	9
10 codes	10
5 codes	11
8 codes	12
4 codes	13
3 codes	14
4 codes	16
7 codes	17
5 codes	18
2 codes	19
1 code	20
3 codes	21
3 codes	22
4 codes	23
1 code	24
1 code	25
1 code	26
5 codes	27
2 codes	28
3 codes	29

(continued overleaf)

Table E7 (continued)

Zone code	Frequency
mmmuml	32
lmmmmm	32
lmmumu	35
uummmm	37
ummuum	44
mumumm	45
mmummu	46
mummum	46
umuumu	47
mmmuum	51
mmuumu	54
umuumm	58
mummmu	59
mmmmum	65
ummmum	65
mummmm	70
ummumm	73
mmmumu	80
ummumu	80
mmummm	87
mmuumm	98
ummmmu	98
mmmumm	106
mmmmmu	113
ummmmm	119
mmmmmm	143
Total	3622

Table E8 — Word-initial tri-gram frequency distribution

[taken from a lexicon of 3,994 words — 1021 grams occur]

Occurrences	Number of tri-grams	Tri-gram
0	16555	
1	310	
2	223	
3	112	
4	93	
5	81	
6	48	
7	37	
8	28	
9	18	
10	9	
11	12	
12	9	
13	5	
14	6	
15	7	
16	1	
17	4	
18	1	
19	3	
20	0	
21	3	
22	2	
23	0	
24	1	
25	1	
26	1	
27	0	
28	1	
31	1	sta
32	1	for
35	1	com
47	1	con
52	1	pro

Table E9 — Reduced zone code frequency distribution of words beginning *pro*

[taken from a lexicon of 3,994 words]

Zone code	Frequency
lmlum	1
lmlu	1
lmlul	1
lmumul	1
lmlmlmum	1
lmlmul	2
lmuml	2
lmlmum	3
lmu	4
lmlmu	4
lmumu	4
lmumum	4
lm	5
lmlm	7
lmum	13
Total	53