1-1-1996

OCRspell: An interactive spelling correction system for OCR errors in text

Eric Stofsky
University of Nevada, Las Vegas

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

OCRSpell: An Interactive Spelling Correction System for OCR Errors in Text

by

Eric Stofsky

A thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
in
Computer Science

Department of Computer Science
University of Nevada, Las Vegas
August 1996

The thesis of Eric Stofsky for the degree of Master of Science in Computer Science is approved.

Chairperson, Kazem Taghva, Ph.D.

Examining Committee Member, Thomas A. Nartker, Ph.D.

Examining Committee Member, John T. Minor, Ph.D.

Graduate Faculty Representative, Shahram Latifi, Ph.D.

Graduate Dean, Ronald W. Smith, Ph.D.

University of Nevada, Las Vegas
August, 1996

Abstract

In this thesis we describe a spelling correction system designed specifically for OCR (Optical Character Recognition) generated text that selects candidate words through the use of information gathered from multiple knowledge sources. This system for text correction is based on static and dynamic device mappings, approximate string matching, and n-gram analysis. Our statistically based, Bayesian system incorporates a learning feature that collects confusion information at the collection and document levels. An evaluation of the new system is presented as well.

Table of Contents

Acknowledgments

I would like to thank Dr. Kazem Taghva and the rest of the Text Retrieval group at ISRI. I would also like to thank Julie Borsack for her help in the development of the OCRSpell system. Her suggestions for potential OCRSpell options and subsequent testing of the system was invaluable. Also, Jeff Gilbreth aided in the implementation of the confusion generator, and Andrew Bagdanov proofread the original draft of this thesis. I am also very grateful to Dr. Thomas Nartker, Dr. John Minor, and Dr. Shahram Latifi for serving on my graduate committee.

Chapter 1

Introduction

Research into algorithmic techniques for detecting and correcting spelling errors in text has a long, robust history in computer science. As an amalgamation of the traditional fields of artificial intelligence, pattern recognition, string matching, computational linguistics, and others, this fundamental problem in information science has been studied from the early 1960's to the present [12]. As other technologies matured, this major area of research has become more important than ever. Everything from text retrieval to speech recognition relies on efficient and reliable text correction and approximation.

While research in the area of correcting words in text encompasses a wide array of fields, in this thesis we report on OCRSpell, a system which integrates many techniques for correcting errors induced by an OCR (optical character recognition) device. This system is fundamentally different from many of the common spelling correction applications which are prevalent today. Traditional text correction is performed by isolating a word boundary, checking the word against a collection of commonly misspelled words, and performing a simple four step procedure: insertion, deletion, substitution, and transposition of all the characters in the string [14]. While the "corrective engine" in this approach may seem overly simplistic, it works quite well for standard applications. In fact, Damerau [6] reported that 80% of all misspellings can be

corrected by the above approach. However, this sample contained errors that were typographical in nature. For OCR text, the above procedure can not be relied upon to deliver corrected text for many reasons:

- **In OCR text, word isolation is much more difficult since errors can include the substitution and insertion of numbers, punctuation, and other nonalphabetic characters.**

- **Device mappings are not guaranteed to be one-to-one. For example, the substitution of *iii* for *m* is quite common. Also, contrary to Pollock and Zamora's [16] statement that OCR errors are typically substitution based, such errors commonly occur in the form of deletion, insertion, and substitution of a string of characters [19].**

- **Unlike typographically induced errors, words are often broken. For example, the word *program* might be recognized as *pr~ gram*.**

- **In contrast to typographical errors caused by common confusions and transpositions produced as artifacts of the keyboard layout, particular OCR errors can vary from device to device, document to document, and even from font to font. This indicates some sort of dynamic confusion construction will be necessary in any OCR-based spell checker.**

Many other differences also demonstrate the need for OCR-based spell checking systems. Our system borrows heavily from research aimed at OCR post-processing systems [11, 18, 19, 22] and is statistical in nature. It is our belief that the ability to

interactively train OCRSpell for errors occurring in any particular document set results in the subsequent automatic production of text of higher quality. It is also important to note that it is also our belief that for some applications, fully automatic correction techniques are currently infeasible. Therefore, our system was designed to be as automatic as possible and to gain knowledge about the document set whenever user interaction becomes necessary.

Chapter 2

Preliminaries

2.1 Background

When designing any automated correction system, we must ask the all important question, "*What sort of errors can occur and why?*" Since most of the errors produced in the document conversion process are artifacts of the procedure used, we can trace most of the problems associated with OCR generated text to the basic steps involved in the conversion process itself. Figure 1 shows the typical process. The procedure involves four standard steps:

1. **scanning the paper documents to produce an electronic image**
2. **zoning the document page to identify and order the various regions of text**
3. **the segmentation process breaks the various zones into their respective components (zones are decomposed into words and words are decomposed into characters)**
4. **the classification of characters into their respective ASCII characters**

Each of the preceding steps can produce the following errors as artifacts of the process used:

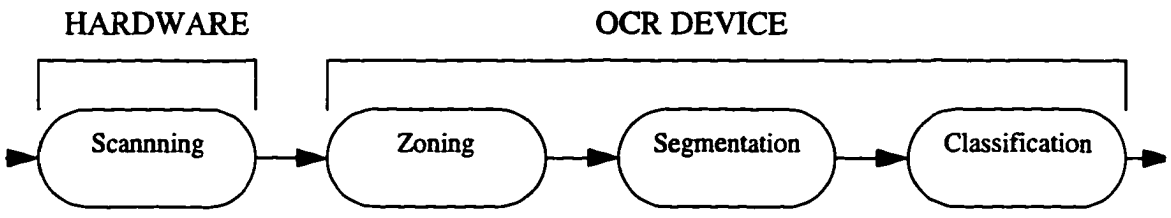

Figure 1: The Standard OCR Procedure

- **Scanning**

Problems can be caused by poor paper/print quality of the original document, poor scanning equipment, etc. The results of such errors can lead to errors in every other stage of the conversion process.

- **Zoning**

Automatic zoning errors are generally caused by incorrect decolumnization. This can greatly affect the word order of the scanned material and produce an incoherent document.

- **Segmentation**

Segmentation errors can be caused by an original document containing broken characters, overlapping characters, and nonstandard fonts. Segmentation errors can be divided into three categories. Table 1 contains a list of the segmentation error types and the respective effects of such errors.

TYPE	PROBLEMS	EXAMPLES
TYPE I	Single characters recognized as multiple characters	m -> rn n -> ii
TYPE II	Multiple characters recognized as one character	cl -> d iii -> m
TYPE III	Division and concatenation of words	cat -> c at the cat -> thecat

Table 1: Types and Results of Segmentation Errors

- **Classification**

Classification errors are usually caused by the same problems as segmentation errors. Typically they result in single character replacement errors where the correct character is replaced by a misrecognized character, but other effects can be seen as well.

OCRSpell was designed to remedy classification errors, all the classes of segmentation errors, and to help reduce the number of scanning errors remaining in the resulting documents. Zoning errors are not handled by the system due to their very nature. Manual or semi-automatic zoning usually resolves such errors in document collections prone to this effect.

2.2 Influences

There has been considerable work done in the areas of OCR generated text correction and spell checking in general over the years [6, 7, 11, 12, 14, 15, 18, 19, 22, 23, 24, 25, 26].

The seminal work in the field is perhaps [6]. String matching and correction are classical computer science problems which also have a very long history [1, 2, 4, 9, 24]. Our system can be viewed as an amalgamation of many of these fields and relies on many of their concepts, heuristics, and algorithms. The exact nature of the origin of many of the components of the system will be discussed as they are presented.

The motivation for this work is obvious. Recent advances in optical character recognition and computer technologies in general have lead to OCR's widespread use in preparing large scale collections of documents for both presentation and for text retrieval purposes. The inherent limitations of OCR technologies present the need for software which allows for the semi-automatic correction of device generated text.

2.3 Effects of OCR Generated Text on IR Systems

It is easy to see how OCR generated errors can affect the overall appearance of the text in question. The effects of such errors on information retrieval systems is less obvious. After all, if the image of the original document is saved by the retrieval system for later display and the performance of the query engine applied to the OCR generated text is not affected by the confusions in the document's text, correction systems such as ours would not be necessary for IR systems. Here we begin by introducing some basic IR terminology then proceed to explain why a system like OCRSpell may significantly increase the performance of text retrieval systems that rely on OCR output for their input.

The goal of information retrieval (IR) technology is to search large textual databases and return documents that the system considers relevant to the user's query. Many distinct models exists for this purpose and considerable research has been conducted on

all of them [20, 21]. Most of the commercial IR applications are based on these models.

In order to establish the effectiveness of any IR system, two notions are generally used:

$$Recall = \frac{\text{numberof documents retrieved that are relevant}}{\text{total number of relevant documents}}$$

$$Precision = \frac{\text{number of documents retrieved that are relevant}}{\text{total number of retrieved documents}}$$

From [20], we know that, in general, average precision and recall are not significantly affected by OCR errors in text. We also know, however, that other elements of retrieval systems such as document ranking, handling of special terms, and relevance feedback may be affected considerably. Another consideration is the increase in storage space needed to store index terms created from OCR generated text.

Other problems may result if non-stopwords are misrecognized as stopwords, traditionally ignored in information retrieval systems. Also, systems based on nonprobabilistic models typically have no means of factoring the probabilities of words occurring in the collection being misrecognized. Moreover, words with low frequency in the collection are weighted high in the ranking scheme. If such words are also rare in the document(s) they occur in and are misrecognized, obviously, the IR system will be affected negatively. This may not occur enough to change the system's overall recall and precision but can have catastrophic effects if queries typically take on this sort of

flavor.

Thus, depending on the collection to be processed and the purpose and needs of the users, some sort of correction system may be needed prior to a documents insertion into a text retrieval system. Furthermore, if confidence in such a system is to be maximized, a semi-automatic system such as ours may prove to be the best option in many instances.

2.4 Implementation

OCRSpell was designed to be a tool for preparing large sets of documents for either text retrieval or for presentation. It was also developed to be used in conjunction with the MANICURE Document Processing System [22]. The Hypertext Markup Language (HTML) feature makes OCRSpell an excellent tool for correcting documents for display on the World-Wide Web [3]. The system is designed around common knowledge about typical OCR errors and dynamic knowledge which is gathered as the user interactively spell checks a document. Approximate string matching techniques [24, 23] are used to determine confusions. Consider the following misspelling:

rnouiitain

It is easy to see that the confusions *rn->m* and *ii->n* have occurred. We refer to the above confusions as device mappings. Whenever OCRSpell fails to provide an adequate choice for a misspelled word, the system isolates the new confusions that have occurred and adds them to the device mapping list. This ensures that future misspellings containing the same confusions will have corrections offered by the spelling engine.

OCRSpell allows a user to set default statistics or to develop statistics for a particular document set. This ensures that the statistics used by the spelling engine will be adequate to find corrections for most of the errors in the document with minimal user interaction. Segmentation errors (resulting in splitting words) can also be handled interactively through the use of the *join next* and *join previous* options.

Conceptually, the system can be seen as being composed of 5 modules:

1. **a parser designed specifically for OCR generated text**
2. **a virtual set of domain specific lexicons**
3. **the candidate word generator**
4. **the global/local training routines (confusion generators)**
5. **the graphical user interface**

The actual implementation of the system closely follows this model. Each of these components will be discussed in the following chapters. Issues affecting the creation of domain specific lexicons will be addressed in Chapter 4.

At the heart of the system is a statistically-based string matching algorithm that uses device mapping frequencies along with n-gram statistics pertaining to the current document set to establish a Bayesian ranking of the possibilities, or suggestions, for each misspelled word. This ensures that the statistically most probable suggestions will occur at the beginning of the choices list and allows the user to limit the number of suggestions without sacrificing the best word alternatives. The algorithms and heuristics used in this system are presented in detail in Chapter 5.

Chapter 3

Parsing OCR Generated Text

Just as the method for candidate word generation is important in any spelling correction system, an effective scheme for parsing the text is essential to the success of the system. For our system, we chose to implement the OCR generated text parser in Emacs LISP [13] due to its robust set of high level functions for text searching and manipulation. Rather than designing many parsing algorithms for different types of working text, we chose to make the parser as general as possible and provide the user with a robust set of filtering and handling functions.

The unique attributes of OCR generated text necessitate a unique parser. The distictness of the parser can be seen in its word boundary code. It is reported in [12] that for essentially all spelling correction techniques, word boundaries are defined by whitespace. The inherent characteristics of the text output from OCR prevent such a simplistic approach and demand fundamentally different approaches to many of the standard techniques for dealing with text in a spell checker. Everything from the treatment of whitespace and punctuation characters, to the treatment of hyphens and other combining symbols used in the creation of compound words has to be handled in a manner that is quite distinct to OCR generated text.

At the highest level, the file to be spell checked is loaded into an Emacs buffer and

processed one line at a time. Before the line is sent to the word generation module (a self contained executable), markup, non-document character sequences, and other strings which the user does not wish to be spell checked are filtered out. Since text in general varies so much between specific subject domains and styles we allowed for user controlled parser configuration. This can easily be seen in the dichotomy that exists between a mathematical paper and a short story.

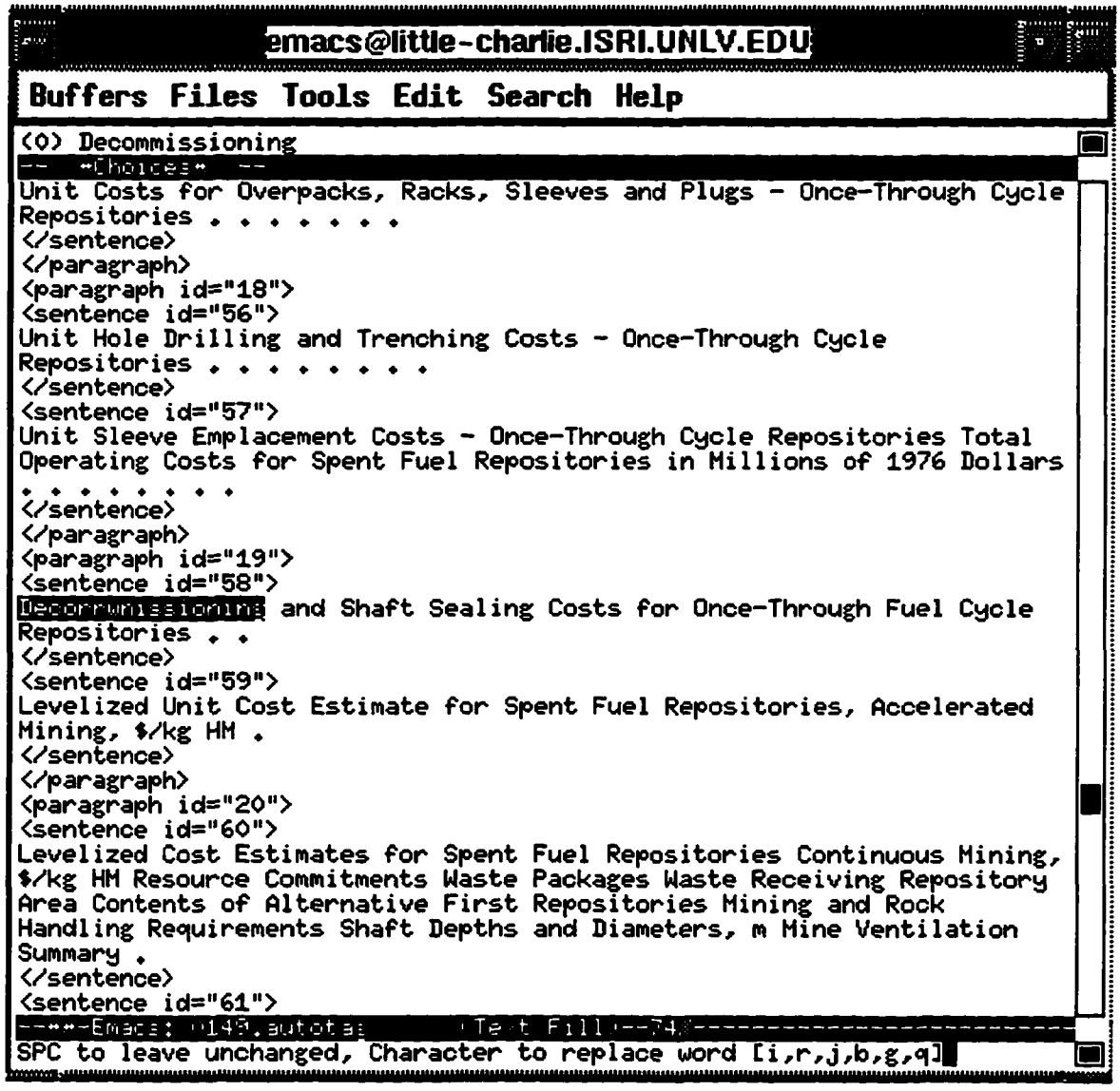

Figure 2: The OCRSpell User Interface

We probably would not want to query the generation module on every occurrence of a numeric sequence containing no alphabet characters in the math paper, while such an effect may be desired in the short story. Included in the implementation are filter mechanisms allowing for skipping number words (words containing only numbers), filtering HTML mark-up, and general regular expressions.

The EMACS text parser also aides in word boundary determination. Our approach is fundamentally different from the standard approach. Rather than using the traditional methods of defining word boundaries via whitespace or non-alphabetic characters, we use a set of heuristics for isolating words within a document. In our system, if the heuristic word boundary toggle switch is on, the parser tries to determine the word boundary for each misspelling which makes the most sense in the context of the current static device mappings.

If the switch is off, a boundary which starts and ends with either an alphabetic or a tilde ("~") character is established. Essentially the parser tries to find the largest possible word boundary and passes this to the word generator. The word generator then determines the most likely word boundary from the interface's delivered text. The generator delivers the new candidate words formed from static mappings of spelling errors to the parser in the form:

& <misspelled-word> <number-of-candidates> <offset> :

<candidate-list>

The **<misspelled-word>** field contains the entire misspelled word. This is used by

the parser to determine what part of the text to delete when inserting any candidate selection or manual replacement.

The **<number-of-candidates>** field contains a non-negative integer indicating the number of words generated by the static device mappings of the word generator.

The **<offset>** field contains the starting position of the misspelled word (the lower word boundary).

The **<candidate-list>** is the list of words generated by static mappings. Words in the **<candidate-list>** are delimited by commas and contain probabilistic information if that is desired.

The parser then receives this information and uses the **<offset>** as the left starting point of the boundary of the misspelled word. Futhermore, the parser invokes the dynamic confusion generator and the unrecognized character heuristic, if required. The above process is much different from many of the general spell checkers which determine word boundary through the use of a set of standard non-word forming characters in the text itself. In our system, non-alphabet characters can be considered as part of the misspelling and as part of the corrections offered. Also, if the word boundary is statistically uncertain, then the parser will send the various probable word boundaries to the word generator and affix external punctuation, as necessary, to the words of the candidate list so that the text to be replaced will be clearly defined and highlighted by the user interface. The internals of OCRSpell's word generation will be discussed in

Chapter 5.

To further illustrate the differences between our system and traditional spell checkers, consider the following misspellings:

(A) 1egal

(B) (iiiount@in)

(C) ~fast"

(D) D~ffer~ces

(E) In trcduc tion

In example **(A)**, typical spell checkers would query for a correction corresponding to the word "ega." Our system, however, determines that the character "1" is on the left hand side of several of the static device mappings and appropriately queries the word generator with "1egal" which generates a singleton list containing the highly ranked word "legal". Furthermore, since the left offset contains the index of either the leftmost alphabet character or the leftmost character used in a device mapping, the **<offset>** returned for this instance is 0. Also, since the entire string was used in the generation of the candidate, the string "1egal" will occur in the **<misspelled-word>** field in the list returned by the word generator. This means that the string "legal" will replace the string "1egal" in the buffer. This is important because even if the correct word could have been generated from "ega," after insertion, the resulting string in the buffer would have been "1legal1" which is incorrect in this context.

Example **(B)** demonstrates that confusions can be a result of a sundry of mapping types. The parser's role in determining the word boundary of "(iiiount@in)" is as follows. The parser grabs the largest possible word boundary, which in this case is the entire string and passes it to the word generator. The word generator produces the singleton list containing the word "mountain". The **<offset>** field is set to 1 since the first alphabet character and the first character used in the transformation occurs at character position 1 in the string. Subsequently, since the first and the last character are not used in any applied device mapping, the **<misspelled-word>** is "iiiount@in." Hence, the final correction applied to the buffer would be "(mountain)." Since the beginning and trailing punctuation were not involved in the generation process they are left intact in the original document.

In example **(C)**, we see how the tilde character takes precedence in our procedure. Since the string "~fast"" contains a correct spelling, "fast" surrounded by punctuation, in the typical context the parser would simply skip the substring. Since the tilde character has special meaning (unrecognized character) in OCR generated text, whenever we parse a string containing this character we automatically attempt to establish a word boundary. The parser sends the entire constructed string to the word generator. Assume that the candidate list is null due to the current configuration of static mapping statistics. This may or may not be true, depending only on the preprocessing training. The word generator would return a null list. Next the parser would evoke the dynamic device mapping generator. If we assume that this error (i.e. ~ -> ") has occurred in the current document set before then, the locally created confusions will be inserted into the

misspelling and offered as candidates. Also, the unrecognized character heuristic (discussed in Chapter 5) will be invoked. The most probable results of the above procedure would be the word list:

(1) "fast (2) fast

Also note that if no mappings for the quote character exists, the above list will be offered as replacements for the string "~fast"." Here the heuristic word boundary procedure has indicated that the trailing quote is not part of the word.

The fourth example, **(D)**, demonstrates how the parser deals with a series of unrecognized characters in a stream of text. Once again we will assume that the word generator returns a null list. Also we will assume this time that no dynamic mappings for the character "~" will produce a word in the current lexicon. Now the unrecognized character heuristic is called with the string "D~ff~er~ces." The heuristic, discussed in Chapter 5, is applied. After candidate word pluralization and capitalization, the parser replaces the misspelling with "Differences."

(E), the last example, demonstrates the semi-automatic nature of the parser. It consists of the text stream, "In trcduc tion." When the parser firsts attempts to process this stream it determines that the word "In" is correct. Next, the subsequent string "trcduc" is isolated as a distinct word boundary. At this point the normal procedure is followed. If the user uses the **<b>** (backward join) feature the string "In trcduc" is

replaced with "Intrcduc" and that string is passed to the word generator. Since that string does not occur in the lexicon, a word list consisting of "Introduce" is offered by the word generator. If the user selects this choice it will be inserted into the buffer. However, if the user uses the <j> (forward join) feature, the entire original text substream is sent to the generator with no whitespace and replacement "Introduction" is offered. This string is once again passed to the word generator, but since the word occurs in the lexicon, the parser continues along the text stream. Other similar situations rely on the semi-automatic nature of the parser as well.

The parser also handles hyphenated text. In the context of OCR generated text, hyphens and other word combining symbols such as "/" present many unique problems. Once again, by examining large samples of text of various styles from various domains we came to the conclusion that no one parsing technique would be generally adequate. Obviously, in text rich in hyphenation, such as scientific scholarly text, querying the user at each occurrence of such symbols would become tedious. On the other hand, in collections with light hyphenation such a practice may be very desirable. The problem lies in the fact that the hyphens and other word combining symbols can be the result of recognition errors and, hence, be the left hand side of static or dynamic device mappings. The situation is further complicated by the fact that most standard electronic dictionaries do not include words containing such combining symbols. If we make any sequence of correctly spelled words combined with such symbols correct by convention, in many circumstances the system would perform erroneously. For these reasons we designed the parser with toggle switches that control how hyphenation is handled.

In its default setting OCRSpell treats hyphens as standard characters. This means

that hyphenated words are treated as single words, and the hyphens themselves may be included in mappings. Candidate words are generated for the entire sequence with dynamic mappings being established in the same manner as well. This mode is intended for OCR generated text where light hyphenation is expected.

For text that is hyphen intensive, a second mode that essentially treats hyphens as whitespace is included in the parser as well. This mode has the advantage that each word in a hyphenated sequence of words is spell checked individually. Also, in the previous setting if a misspelling occurred in a combined sequence of words, the entire sequence is queried as a misspelling. In this schema only the term which does not occur in the lexicon is queried. The parser filters out the hyphens prior to sending the current line of text to the static word generator to prevent the hyphens from affecting either static device mappings or word boundary determinations. Dynamic device mappings on hyphen symbols are still generated and applied by the parser when confusions are known to have occurred. Choosing between the two parsing methods involves the classical dilemma of efficiency versus quality. The best results will always be achieved by using the parser in its default setting, but sometimes the frequency of necessary, naturally occurring hyphens in the collection makes this method too time consuming.

The OCRSpell parser was designed to be efficient, expandable, and robust enough to handle most styles of document sets effectively. The system's treatment of word boundaries, word combining symbols, and other text characteristics is essential to the overall success of the system. The other components of the system rely heavily on the parser to make heuristically correct determinations concerning the nature of the current text being processed.

Chapter 4

Organization of the Lexicon

Another important issue to address prior to the development of any candidate word selection method is the organization of the lexicon, or dictionary, to be used. Our system allows for the importation of Ispell [26] hashed dictionaries along with standard ASCII word lists. Since several domain specific lexicons of this nature exist, the user can prevent the system from generating erroneous words that are used primarily in specific or technical unrelated domains. Stemming is applied to the word list so only non-standard derivatives need to be included in any gathered lexicon. OCRSpell also allows the user to add words at any time to the currently selected lexicon.

It is important for any spelling correction system to have an organized, domain specific, dictionary. If the dictionary is too small, not only will the candidate list for misspellings be severely limited, but the user will also be frustrated by too many false rejections of words that are correct. On the other hand, a lexicon that is too large may not detect misspellings when they occur due to the dense "word space." Besides over acceptance, an overly large lexicon can contaminate the candidate list of misspellings with words that are not used in the current document's domain. According to [15], about half of a percent of all single character insertions, deletions, substitutions, and transpositions in a 350,000 word lexicon produced words in the lexicon. In a device

mapping system like ours, an overly large dictionary could prove catastrophic.

Other studies indicate that, contrary to popular opinion, there is no need for vast electronic dictionaries. For example Walker and Amsler [25] determined that 61% of the terms in the *Merriam-Webster Seventh Collegiate Dictionary* do not occur at all in an 8 million word sample of the *New York Times* newspaper. They also determined that 64% of the words in the newspaper sample were not in the dictionary.

Our system does not solve the lexicon problem; however, it does provide an infrastructure that is extremely conducive to lexicon management. Since the system allows for the importation of dictionaries, they can be kept separate. Optimally, each collection type (i.e. newspaper samples, sociology papers, etc.) would have its own distinct dictionary that would continue to grow and adapt to new terminolgy as the user interactively spell checks documents from that collection. The only problem to this approach is the vast disk space that would be required since most of the various dictionaries would contain identical terms. So once again a careful balance must be reached. It is clear that automatic dictionary management is a problem that deserves considerable research.

Chapter 5

Design

5.1 System Design

The OCRSpell system consists of three parts:

1. **A two-level statistical device mapping word generator which is used to generate possibilities for misrecognized words (implemented in the C programming language).**
2. **The confusion generator which is used to determine the longest common subsequence and the subsequent confusions for words that have been manually replaced (implemented in the C programming language).**
3. **The user interface which combines (1) and (2), and adds many options and features to insure an easy to use, robust system. This interface was written in Emacs LISP and was designed to run under Emacs Version 19.**

The interface can be controlled by a series of meta commands and special characters. Figure 2 shows the overall design of the OCRSpell interface. Many of the commonly used interface options can be selected directly from the menu. The user can join the current word with the previous or next word, insert the highlighted word or character

sequence into the lexicon, select a generated choice, or locally/globally replace the highlighted text by a specified string. If the user chooses to replace the text, the confusion generator is invoked and the subsequent confusions are added to the device mapping list. This means that any errors occurring later on in the document with the same confusions (e.g. *rn* -> *m*) will have automatically generated choices in the interface's selection window. Of course, this means the effectiveness of OCRSpell improves as it gains more information about the nature of the errors in any particular document set. Table 2 contains a list of all of the interactive features of the system.

Key	OCRSpell Feature
[i]	insert highlighted word into lexicon
[r]	replace word, find confusions
[b]	backward join (merge previous word)
[j]	forward join (merge next word)
[g]	global replacement
[<space>]	skip current word or highlighted region
[<character>]	replace highlighted word with generated selection
[q]	quit the OCRSpell session

Table 2: OCRSpell's Interactive Features

5.2 Algorithms and Heuristics Used

The OCRSpell system integrates a wide array of algorithms and heuristics. We start our

description of the overall algorithmic design of the system by introducing some key terms, algorithms, and heuristics. The overall integration of these concepts can be seen in Figures 5 and 6 which visually demonstrate how these various components fit together.

- A simple level saturation technique is used to generate new words from static confusions. This technique relies heavily on a Bayesian ranking system that is applied to the subsequent candidate words. The mapping process and Bayesian ordering are as follows:

A successful **word mapping generation** is defined as:

$$A^+ \rightarrow B^+ \rightarrow C^+$$

where A^+, B^+, and C^+ are strings of 1 or more characters, A^+ doesn't occur in the lexicon, and B^+ or C^+ occurs in the current lexicon. String B^+ is generated by applying one mapping to A^+. String C^+ is generated by applying one mapping to B^+.

Character or **device mappings** are of the form:

$$M_0 \rightarrow M_1$$

where M_0 and M_1 consists of a sequence of 0 or more characters, and $M_0 \neq M_1$.

The **Bayesian candidate function** is defined as:

$$P\left(Y_i \middle| X\right) = \frac{P(Y_i)\,P(Y_i \rightarrow X)}{\sum_{j=1}^{q} P(Y_j)\,P(Y_j \rightarrow X)}$$

where the probability $P(Y_i \rightarrow X)$ is the statistical probability that the string X was mapped from string Y_i, given the current state of the static device mapping list of confusion probabilities. Y_i can be thought of as being bounded by trigram space.

The **Bayesian ranking function** is defined as:

$$P(Y|X) = Max\left(\prod\left(\frac{P(Y_j)P(X_i \rightarrow Y_j)}{P(X_i)}\right)\right)$$

where the product is taken over every device mapping $(X_i \rightarrow Y_j)$ used in the transformation, and $P(Y_j)$ and $P(X_i)$ are determined through an n-gram analysis of the character frequencies in the current document set. Y may be generated from X by intermediate transformations $X_1, X_2,...,X_n$, where n is greater than 0. The maximum of the product is taken so that if multiple distinct mappings produce the same result, the statistically most probable will be selected. In our implementation n is bound to be no greater than 2.

The **collocation frequency** between any word pair is measured as [10]:

$$F(X \wedge Y) = log_2 \frac{P(X,Y)}{P(X)P(Y)}$$

where P(X) and P(Y) are the statistical frequencies of words X and Y in the current document set and P(X,Y) is the frequency of word X and Y occurring as consecutive words in the current document set. The words need not be in the current lexicon.

The **n-gram (character) analysis** of the document set is performed as follows:

For each string X of length L,

$$\omega(L, X) = \frac{\text{number of occurances of string X}}{\text{total number of strings of length L}}$$

where L, the string length, currently takes on the values 1, 2, and 3 (i.e. unigram, bigram, and trigram) for all $\omega(L,X)$, $L = |X|$.

The device mapping statistics are **normalized** upon success with the following n-gram function:

$$N(A \rightarrow B) = \frac{\omega(|B|,B)}{\left(\sum_{A \rightarrow X_i} \omega(|X_i|,X_i)\right)\omega(|A|,A)}$$

where A, B, and X_i are strings of characters of length between 0 and 3. This function is used in conjunction with the Bayesian functions above to produce normalized statistics pertaining to any particular mapping instance. The numerator of the above function determines the statistical likelihood that the string B occurs in the current document set (i.e. its frequency of occurrence in the current document set). The denominator is the product of all other current static device mapping instances from A multiplied by the probability that the correct string is in fact A.

In our approach, static device mappings are implemented as an ordered list of three dimensional vectors of type (string, string, real) that contain (generated-string, correct-string, mapping frequency) of the associated device mapping. We limit the number of mappings in any transformation to two for two reasons. First, empirical evidence

suggest that the words generated after two applications of mappings are usually erroneous. Secondly, if this transformation process is left unbounded, the procedure becomes quite time consuming.

- The ranking of word suggestions is achieved using the above statistical equations. After the probabilities of each of the word suggestions is computed, the list is sorted so that the words are offered in decreasing order of likelihood. The process is as follows:

Misspelling	Suggestions	Ranking
thc	the	0.336984
	th-c	0.002057
	rho	0.000150
	tic	0.000001
	thy	0.000001
	th	0.000001
rnount@in	mountain	0.000010
MineraI	Mineral	0.013608
i1legal	illegal	0.000460
iiieii	men	0.000491

Table 3: Example of Static Mapping Word Generation Rankings

First, all the suggestions using the static device mappings are generated with their statistical ranking calculated as above. These words are then ordered from most probable to least. Next, the same procedure is performed on the word with the dynamic device mappings. This list is then ordered and inserted at the beginning of the candidate

list generated in step 1. Next, words are generated using the unrecognized character heuristic, if at least one unrecognized character is present in the word. If no words are generated using this heuristic, the word is iteratively stemmed, and the selected root is processed using the same heuristic. Any candidate words subsequently generated from this stemming process are concatenated with the suffix obtained from the original misspelling. These words are sorted alphabetically and appended to the end of the candidate list. Throughout the process capitalization and pluralization is performed as necessary. After this word list generation process is complete, duplicates are removed by merging high. Table 3 contains a few examples of misspellings with the corresponding ranking of each member of the candidate list generated by the static device mappings of a sample document set.

One of the more interesting effects of the above procedure is that often the candidate list consists of a single high probability suggestion. Also, treating the words generated through each distinct process separately increases the performance of the system. It weighs dynamically gathered information higher than static information. Furthermore, since the words generated by the unrecognized character heuristic cannot be ranked statistically, appending them to the end of the list preserves that statistical integrity of the rest of the candidate list. An evaluation of the OCRSpell system can be found is Chapter 8.

- The confusion generator was developed to use the dynamic programming longest common subsequence algorithm. This algorithm was chosen so that heuristically optimal subsequences would be selected.

The method used here is from [4]. If we let X[1...i] represent the prefix in the string X[1...m] of length i and c[i, j] be the length of an LCS for sequences X[1...i] and Y[1...j] for two strings X[1...m] and Y[1...n]. Then we can define c[i, j] recursively as:

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ c[i-1,j-1]+1 & \text{if } i,j > 0 \text{ and } X[i] = Y[j] \\ max(c[i,j-1],c[i-1,j]) & \text{if } i,j > 0 \text{ and } X[i] \neq Y[j] \end{cases}$$

After the longest common subsequence has been calculated, the character sequences not in the LCS are correlated and saved as dynamic device mappings. The time required to compute dynamic confusions can be improved by using a more efficient LCS algorithm such as [1] or [2]. Also, confusions can be computed by other means entirely as demonstrated by [9]. The creation of dynamic device mappings from the LCS of two distinct strings of text can be seen in Figure 3.

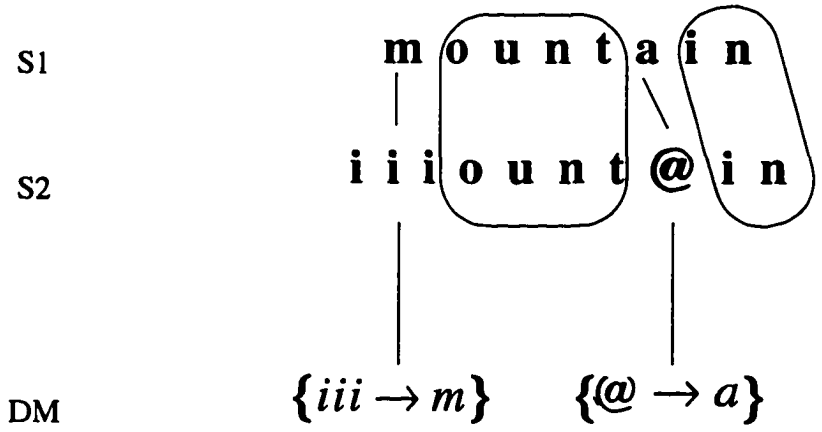

Figure 3: Example of dynamic device mapping construction from LCS. The word occurring in the document (S2) is iiiount@in. The user manual replacement (S1) is mountain. The new device mappings created are {iii -> m}, {@ -> a}.

- Dynamic device mappings are created and applied by the user interface in much the same way that static device mappings are applied in the level saturation generation process. A longest common subsequence process is invoked whenever the user manually inserts a replacement to a misspelling.

- We implemented an intelligent number handling feature as an extension of our device mapping generator. After detecting the word boundary of a given misspelling we parse the left and right hand side of the isolated word. If we encounter characters with static device mappings associated with them, we include them in the word generation process as well. Hence, the same n-gram and device mapping analysis takes place.

As an example of how this process works consider the following scenario. Assume a static device mapping for the character "1" exists. If the word "1ega1" occurs in the document, then, using the above approach, the word boundary which is isolated will include the entire string. Hence all candidate words will be generated from the string "1ega1." The likely result of this process will be a candidate word list including the word "legal."

- Stemming on misspellings and words occurring in the lexicon is performed in a heuristic manner. If there is an apparent common suffix in a misspelling where OCRSpell offers no suggestions, the suffix is removed, and the root is reconstructed. The suggestions, if any, subsequently offered by OCRSpell are then

unstemmed.

The stemming procedure used here can be described as a very nonaggressive "Porter-like" stemmer [5]. Since it is not important that the words generated in conflation are in fact related to the root, the process of stemming is significantly relaxed. Furthermore, since all nonstandard forms are assumed to occur in the lexicon, the only problems associated with this process are:

1. **Legitimate words that are not recovered in the stemming process**
2. **Illegitimate words that are generated in the stemming process**

Problem 1 is eased by allowing for the importation of a wide variety of lexicons. Since these lexicons differ in the various word forms they contain, the odds of the lexicons not containing either the word or a stem-able root of the word is reduced by using domain specific dictionaries. As the user processes any coherent collection of documents and inserts new terms into the working lexicon, occurrences of the first problem should drastically decrease. Problem 2 is less easy to deal with. Since it is impossible to determine what is a legitimate word that is not in the lexicon set and what is the result of excessive conflation, we do not attempt to deal with this problem here. Empirical evidence suggest that often times human beings perform excessive conflation as well, necessitating the offering of words generated in this class to be offered as suggestions by the OCRSpell system.

- A new heuristic was developed to handle unrecognized characters. Essentially,

whenever a word with at least one tilde ("~") is spell checked, not only is the typical device mapping analysis performed but a heuristic lookup function is called as well. Figure 4 contains the algorithm that generates the candidate words using this heuristic.

The overall organization of this collection of algorithms and heuristics can be seen in Figures 5 and 6. Figure 5 pictorially demonstrates the overall OCRSpell word generation process. Here static and dynamic device mappings are applied to the word boundary using the current user selected lexicon(s). The use of the unrecognized character heuristic in this procedure is also demonstrated along with its required auxiliary stemming functions. Figure 6 diagrams the user verification process, or front-end, of the system. The interactive LCS construction of dynamic confusions can be seen within the larger picture of the user verification process. These two figures comprise the whole of the system we have developed at a very high level. Chapter 7 is devoted to the training of the system which has not been covered in detail here.

5.3 Performance Issues

All of the algorithms used in this system are polynomial in nature. The most time expensive procedure used in the system is the level saturation word generation. This technique basically takes n device mappings and applies them to some particular string of length m. Since only two mappings can be applied to any particular string, this procedure is still polynomial in nature. Although this mapping list can grow quite large, it typically contains sparse mappings when applied to any particular word. As stated before improve-

ments can, however, be made by substituting the quadratic confusion generation routines for a more optimal linear time approach. Possible algorithms for improving the confusion generator can be seen in [2] and [1].

Other improvements in speed and efficiency can be made in the area of the lexicon access and organization. This will be addressed in Chapter 9. Many of these improvements can be used in the future to help compensate for the expensive overhead of running the application under Emacs.

```
Algorithm Generate-Words-From-Unrecognized (string original-word)

string lookup-word;                    {for grep regular expression}
int max-length;                        {heuristic word reject length}
array of string word-list;             {structure to store new candidates}

max-length = length (original-word) +
        no-of-unrecognized-chars (original-word);
lookup-word = original-word;
replace all ~'s in lookup-word with *'s;
word-list = grep of lookup-word in lexicon;
if word-list = nil then
        lookup-word = stem of lookup-word;
        lookup-stem = suffix of lookup-word;
        word-list = grep of lookup-word in lexicon;
        word-list = unstem (word-list, lookup-stem);
fi
if first char of lookup-word is uppercase then
        word-list = upper (word-list)
fi
if lookup-word appears plural then     {i.e. ends in "s", "es", etc.}
        word-list = plural (word-list)
fi
remove all words w from word-list where length (w) > max-length
sort word-list lexicographically
end
```

where the functions **stem** and **suffix** return the root and the rest of the string respectively, function **unstem** heuristically removes the stem it is passed as the second argument from all the words it is passed in the first parameter word-list, the function **upper** simply capitalizes each of the words in the word-list, and the function **plural** heuristically pluralizes each of the words in word-list and returns the list constructed. **No-of-unrecognized-chars** returns the number of tildes in the string.

The call to **grep** simply looks up the new term in the lexicon, returning all terms that match the regular expression where each "*" can match zero or more of any alphabet character.

Example:

Generate-Words-From-Unrecognized("D~ff~rences") produces a singleton word list containing only the word "Differences."

Figure 4: Unrecognized character heuristic

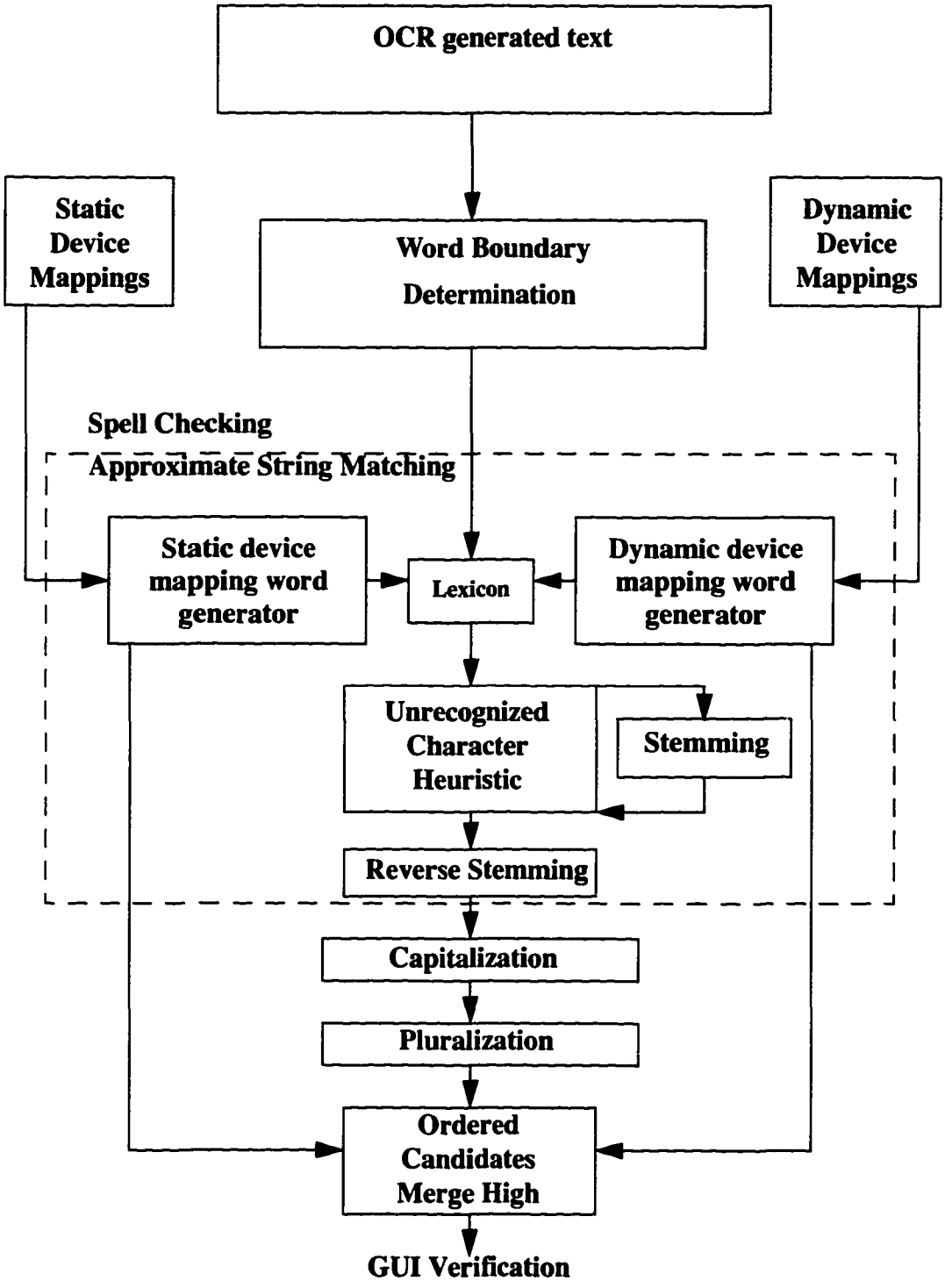

Figure 5: Overall OCRSpell Generation Process

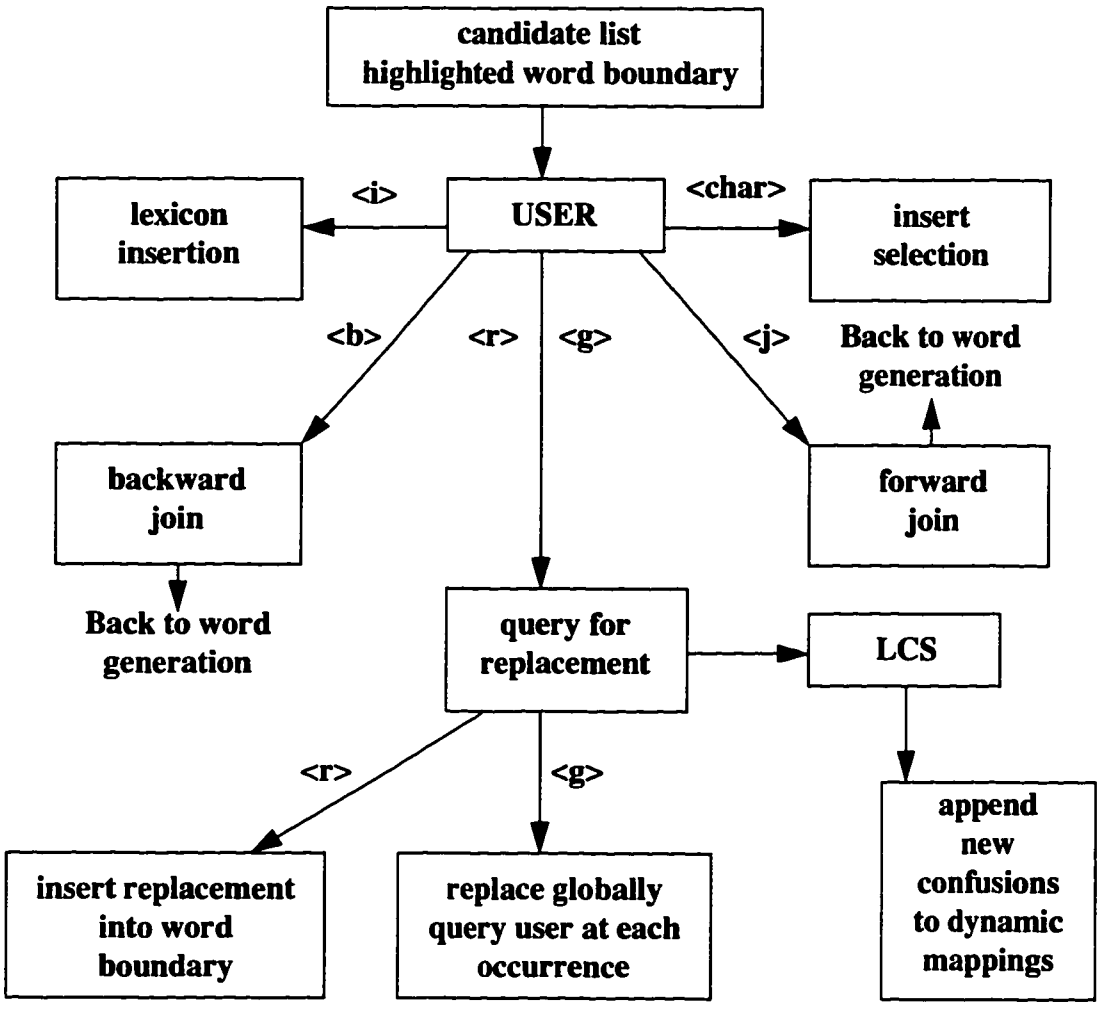

Figure 6: OCRSpell User Verification

Chapter 6

Features

6.1 Simplicity

The OCRSpell Emacs interface was designed with ease of use in mind. All operations can be performed by a single keystroke. The interface invokes all of the other aspects of the system, so they are transparent to the user. Some of the options included are the ability to:

- **Create a file.choices buffer, which records all changes to a document in a buffer in the form *original -> replacement.***
- **Skip non-document markup in tagged text. Currently only the Hypertext Markup Language (HTML) (derived from SGML [8]) is supported.**
- **Load and save dynamic confusion/session files. This allows the user to apply the information gathered in a current OCRSpell session at some future time.**
- **Specify the use of alternate dictionaries and static confusions files.**
- **Process various types and styles of document sets.**

6.2 Extendibility

The design of the system leads itself to easy expandability. In fact there are plans to implement clustering [7, 19] in the system. Also, the nature of the system's design allows new code to be written in either the C programming language or in Emacs LISP.

6.3 Flexibility

OCRSpell gives the user the ability to control most of the higher elements of how any particular document is spell checked right from the interface. The maximum number of choices for any misspelled word can be set with the statistically most probable selections being delivered. Also, the user can specify how numbers, hyphens, and punctuation should be handled in the spelling process.

In addition, the modularity of the Emacs LISP code allows for the easy addition of new features. Processing modes for any current or future markup language can easily be written. Futhermore, the statistical model that the system follows is easily modifiable. Also, due to the manner in which the program allows for the importation of new dictionaries, the system can be easily modified to allow for spell checking languages other than English. It is the authors' hope that this system will be viewed as a prototype to be expanded by others.

Chapter 7

OCRSpell Trainer

A primitive statistical trainer was developed for the OCRSpell system. It is different from that of the interface in that it is fully automatic with new device mappings becoming static in nature. The trainer currently works by accepting extracted word tuples in the form of ground truth and recognized words and adjusting the global static statistics accordingly. A future version of the system will allow for more advanced statistical training at the document level.

The current statistical trainer allows the initial dynamic confusions construction for a document to be less dependent on the user since all of the common non-standard confusions would have been added to the list in the training phase. Figures 7 and 8 show the two distinct methods of training the system. Figure 9 demonstrates how these two distinct learning steps can be used together. So the entire system can be viewed as an adaptive process where the knowledge base of the system is refined as the user processes documents from any particular collection.

All of information gathered from either training method can be saved to and loaded from a file. This allows users to develop statistics for more than one collection type.

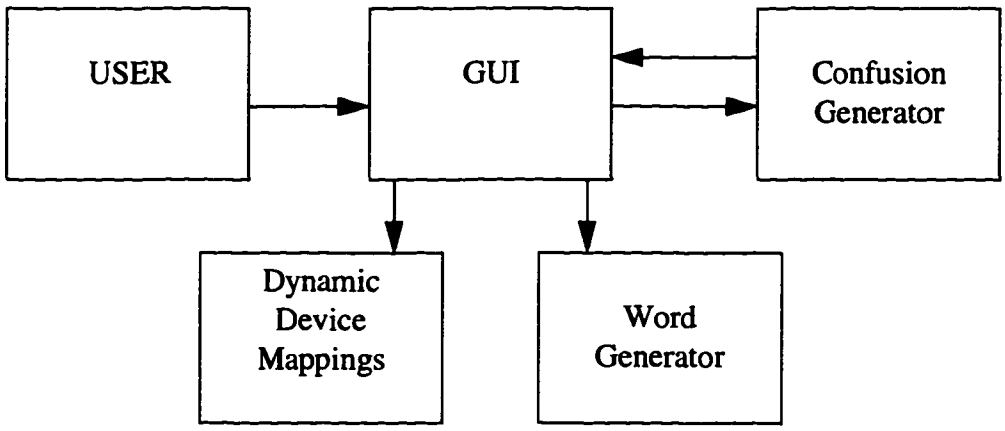

Figure 7: User Interface Training. This figure demonstrates the typical construction of dynamic confusions at run time. Confusions are collected as the user interactively uses the graphical user interface (GUI).

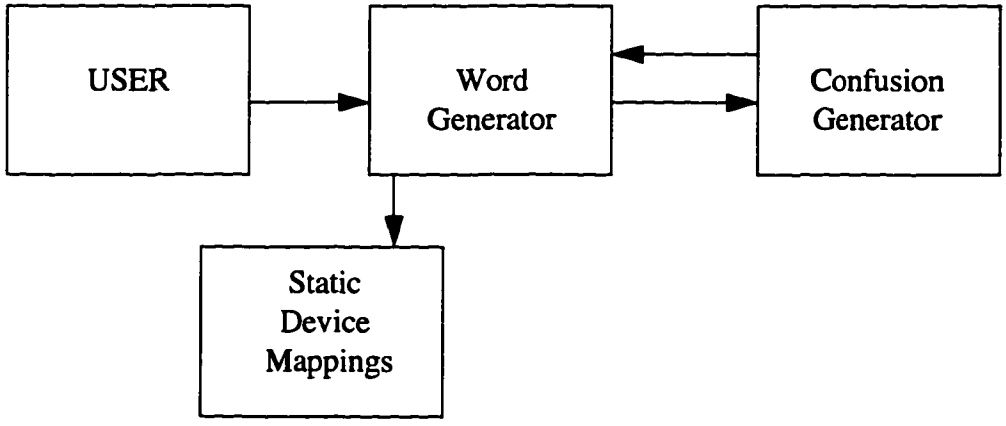

Figure 8: Static Confusion Training. This figure pictorially represents how new static confusions are formed in the training phase. Word tuples are sent to the confusion generator via the word generator, and static device mappings are statistically adjusted or created.

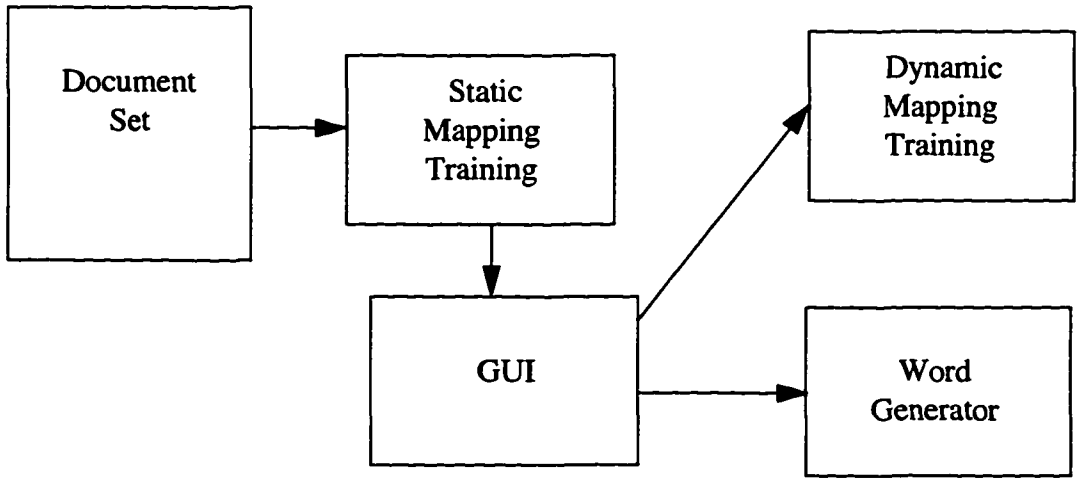

Figure 9: Use of Static and Dynamic Training. In the typical OCRSpell training process, dynamic and static device mappings are collected by using both methods in conjunction.

Chapter 8

Evaluation

OCRSpell was evaluated in two distinct tests. The first test consisted of selecting, at random, word tuples from the ISRI DOE text sample.

The tuples were of the form (*incorrect word, correct word).* A retired sample from 1994 was selected for the test, and the incorrect words were selected from the collection of generated text produced by the Calera WordScan and Recognita Plus DTK. These two devices were chosen due to the fact that they had the highest and lowest, respectively, word accuracy rates of the 1994 ISRI test [17]. The second test consisted of selecting two SIGIR Proceedings papers and interactively OCRSpelling them and calculating the increase in word accuracy and character accuracy.

8.1 OCRSpell Test 1

As stated above, the first test of the OCRSpell system consisted of extracting words from the ground truth of the ISRI DOE sample and the corresponding device generated text. These words were assembled into a large collection and the following steps were applied as a precursor to the test.

- **All tuples where the generated words occurred in the lexicon were excluded.**

- **All tuples where the correct word consisted of a character sequence that was not in the current lexicon were excluded.**

- **All tuples where the generated or correct words consisted of entirely nonalphabetic characters were excluded.**

- **All tuples where the correct or incorrect word was split were excluded in this test.**

After these steps were followed, 600 word tuples were selected at random from both the Calera WordScan and the Recognita Plus DTK. Tables 4 and 5 contain the results of these automated tests. Here we use the term *hit* to indicate that the correct word was offered as the first suggestion by OCRSpell. *Near miss* is used to indicate that the correct word was in fact offered by OCRSpell (but not the first word offered). Finally, a *complete miss* indicates that OCRSpell failed to generate the correct word. Each of these classes were defined to be case insensitive. An automated front end was constructed for OCRSpell to ease the process of conducting this test. Since these tests were fully automated, the dynamic confusion generated was invoked at each complete miss. This means that word was calculated as a complete miss and any new device mappings were appended afterward.

The *hit ratio* is defined as:

$$\frac{\text{number of hits}}{\text{total number of words}}$$

The *near miss ratio* is define as:

$$\frac{\text{number of near misses}}{\text{total number of words}}$$

The *complete miss ratio* is defined as:

$$\frac{\text{number of complete misses}}{\text{total number of words}}$$

All of these ratios are rounded to the nearest one-hundredth in Tables 4 and 5.

Statistics	Subsample A	Subsample B	Subsample C	Subsample D
# of Attempted	150 words	150 words	150 words	150 words
# of Hits	99 words	85 words	117 words	71 words
# of Near Misses	21 words	49 words	23 words	39 words
# of Complete Misses	30 words	16 words	10 words	40 words
Hit Ratio	0.66	0.57	0.78	0.47
Near Miss Ratio	0.14	0.33	0.15	0.26
Complete Miss Ratio	0.20	0.11	0.07	0.27

Table 4: Recognita Plus DTK (1994)

Statistics	Subsample A	Subsample B	Subsample C	Subsample D
# of Attempted	125 words	125 words	125 words	125 words
# of Hits	70 words	62 words	74 words	57 words
# of Near Misses	40 words	37 words	46 words	28 words
# of Complete Misses	15 words	26 words	5 words	40 words
Hit Ratio	0.56	0.50	0.59	0.46
Near Miss Ratio	0.32	0.30	0.37	0.22
Complete Miss Ratio	0.12	0.21	0.04	0.32

Table 5: Calera WordScan (1994)

8.2 OCRSpell Test 2

To test the performance of OCRSpell on entire documents we chose two papers at random from the current ISRI Text Retrieval Group's SIGIR electronic conversion project. This project involves converting the proceedings of various ACM-SIGIR conferences into electronic form (HTML) using the MANICURE Document Processing System [22]. Two documents that had been automatically processed, manually corrected and proofread were chosen at random. The following steps were then applied to ensure a fair test.

- **The text in the OCR generated file that was replaced in the ground truth file by images was removed.**
- **The OCR generated file was then loaded into Emacs and spell checked by a single user using the OCRSpell system.**
- **The changes in *word accuracy* and *character accuracy* were recorded.**

Word accuracy and *character accuracy* was determined as defined by [17].

Word accuracy is defined as:

$$\frac{\text{number of words recognized correctly}}{\text{total number of words}}$$

where words are defined to be a sequence of one or more letters.

Character accuracy is defined as:

$$\frac{n - |errors|}{n}$$

where n is the number of correct characters and |errors| indicates the number of character insertions, deletions, and substitutions needed to correct the document.

The results of these tests can be seen in Table 6. All of the percentages are rounded to the nearest one-hundredth in this table. As can be seen, the OCRSpelled documents demonstrate a substantial improvement in both character accuracy and word accuracy.

Document Name	Original Word Accuracy	Original Character Accuracy	New Word Accuracy	New Character Accuracy
Miller	98.18	99.30	99.70	99.79
Wiersba	98.46	97.57	99.87	99.85

Table 6: SIGIR Test Results

8.3 Test Results Overview

While the two tests performed on OCRSpell do demonstrate a lower baseline of performance, they do not demonstrate typical usage of OCRSpell. The system was designed to be used on large homogeneous collections of text. Such a test was not feasible for this thesis. We can, however, see from the above tests the improvement of OCRSpell over typical spell checkers when dealing with OCR generated text. The main problem with testing a semi-automatic system like OCRSpell is that the user is central to the whole process. For

our system in particular, the user's responses are responsible for the creation and the maintenance of the dynamic device mappings. The artificial front end we constructed for the first test is not comparable to typical human interaction. Regardless of these issues, the above two tests do indicate some level of the performance improvement for our system on OCR generated text.

Further testing of the system is necessary. Future tests could include an evaluation of other conventional spell checkers on the same samples. Also, a larger sample taken from many subject domains could provide interesting results.

Chapter 9

Conclusion and Future Work

Although OCRSpell was first intended to be a component of the ISRI Post Processing System [22], it has evolved into a project of its own. An evaluation is currently underway to establish the overall performance of this system on OCR text.

Word clustering, and perhaps the introduction of a stochastic grammatical parser to provide some pseudo contextual information are currently being considered as potential additions to the OCRSpell system. Also, new routines are being added to allow the system to be less dependent on an external spell checker.

Other improvements, mentioned in the previous chapters, can be made to improve the efficiency of the system. Also, the trainer can be improved to allow for the processing of full documents or even sets of documents.

Bibliography

[1] A. Apostolico, S. Browne, and C. Guerra. Fast Linear-Space Computations of Longest Common Subsequences. *Theoretical Computer Science*, 92(1992), 3-17.

[2] Richardo A. Baeza-Yates. Searching Subsequences. *Theoretical Computer Science*, 78(1991), 363-376.

[3] Tim Berners-Lee et al. The World-Wide Web. *Communications of the ACM*, 37(8):76-82, August 1994.

[4] Thomas H. Corman, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, tenth edition, 1993.

[5] W. Bruce Croft and Jinxi Xu. Corpus-Specific Stemming using Word Form Co-occurrence. In *Proceedings of the Fourth Annual Symposium on Document Analysis and Information Retrieval*, 147-159, 1995.

[6] F. J. Damerau. A Technique for Computer Detection and Correction of Spelling Errors. *Communications of the ACM*, (3): 171-176, March 1964.

[7] Yaacov Choueka. Looking for Needles in a Haystack. In *Proceedings of RAIO*, 609-613, 1988.

[8] C. F. Goldfarb. *The SGML Handbook*. Oxford University Press, 1990.

[9] Patrick A. V. Hall and Geoff R. Dowling. Approximate String Matching. *ACM Computing Surveys*. 12(4):382-402, December 1980.

[10] Tao Hong and Jonathan J. Hull. Degraded Text Recognition Using Word Collocation. *Document Recognition*. SPIE Vol. 2181, 1994.

[11] Mark A. Jones, Guy A. Story, and Bruce W. Ballard. Integrating Multiple Knowledge Sources in a Bayesian OCR Post-Processor. In *Proceedings*

of IDCAR-91 (St. Malo, France), 925-933.

[12] Karen Kukich. Techniques for Automatically Correcting Words in Text. *ACM Computing Surveys*, 24(4):377-439, December 1992.

[13] Bil Lewis, Dan Laliberte, and the GNU Manual Group. *The GNU Emacs Lisp Reference Manual*, Edition 1.05, Free Software Foundation, 1992.

[14] James L. Peterson. Computer Programs for Detecting and Correcting Spelling Errors. *Communications of the ACM*, 23(12):676-687, December 1980.

[15] James L. Peterson. A Note on Undetected Typing Errors. *Communications of the ACM*, 27(7), July, 1986, 633-637.

[16] Joseph J. Pollock and Antonio Zamora. Automatic Spelling Correction in Scientific and Scholarly Text. *Communications of the ACM*, 27(4):358-368, April 1984.

[17] Stephen V. Rice, Junichi Kanai, and Thomas A. Nartker. An Evaluation of OCR Accuracy, Technical Report, Information Science Research Institute, University of Nevada, Las Vegas, April 94.

[18] Kazem Taghva, Julie Borsack, and Allen Condit. An Expert System for Automatically Correcting OCR Output. Document Recognition, In *Proceedings of the International Society for Optical Engineering*, 2181:270-278, 1994.

[19] Kazem Taghva, Julie Borsack, Bryan Bullard, and Allen Condit. Post-editing through Approximation and Global Correction. *International Journal of Pattern Recognition and Artificial Intelligence*, Vol. 9, No. 6 (1995) 911-923.

[20] Kazem Taghva, Julie Borsack, and Allen Condit. Results of Applying IR to OCR Text. In *Proceedings of the Seventeenth Annual International ACM/ SIGIR Conference on Research and Development in Information Retrieval*, pages 202-211, Dublin, Ireland, July 1994.

[21] Kazem Taghva, Julie Borsack, and Allen Condit. Evaluation of Model-Based Retrieval Effectiveness with OCR Text, *ACM Transactions on Information Systems*, 14(1), January 1996, 64-93.

[22] Kazem Taghva, Allen Condit, Julie Borsack, John Kilburg, Changshi Wu, and Jeff Gilbreth. The MANICURE Document Processing System,

Technical Report, Information Science Research Institute, University of Nevada, Las Vegas, April 1995.

[23] H. Takahashi, N. Itoh, T. Amano and A. Yamashita. A Spelling Correction Method and its Application to an OCR System. *Pattern Recognition* 23(3/4):363-377, 1990.

[24] Robert A. Wagner and Micheal J. Fischer. The Sting-to-String Correction Problem. *Journal of the Association for Computing Machinery* 21(1): 168-173, January 1974.

[25] D. E. Walker and R.A. Amsler. The Use of Machine-Readable Dictionaries in Sublanguage Analysis. In *Analyzing Language in Restricted Domains: Sublanguage Description and Processing*. Lawrence Erlbaum, Hillsdale, N.J., 69-83.

[26] Pace Willisson, R.E. Gorin, Walt Beuhring, Geoff Keunning, et al. Ispell, a free software package for spell checking files. The UNIX community, 1971-present.