Feature recognition in Ocr text

Jeffrey Todd Gilbreth

University of Nevada, Las Vegas

Follow this and additional works at: https://digitalscholarship.unlv.edu/rtds

Repository Citation
https://digitalscholarship.unlv.edu/rtds/587

This Thesis is brought to you for free and open access by Digital Scholarship@UNLV. It has been accepted for inclusion in UNLV Retrospective Theses & Dissertations by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.
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
Feature Recognition in OCR Text

by

Jeffrey T. Gilbreth

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
May 1996
The thesis of Jeffrey T. Gilbreth for the degree of Master of Science in Computer Science is approved.

Chairperson, Kazem Taghva, Ph.D.

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

Examiner Committee Member, Laxmi P. Gewali, Ph.D.

Graduate Faculty Representative, Stephen H. Lepp, Ph.D.

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

University of Nevada, Las Vegas
May, 1996
Abstract

This thesis investigates the recognition and extraction of special word sequences, representing concepts, from OCR text. Unlike general index terms, concepts can consist of one or more terms that combined, have higher retrieval value than the terms alone (i.e. acronyms, proper nouns, phrases). An algorithm to recognize acronyms and their definitions will be presented. An evaluation of the algorithm will also be presented.
Contents

Abstract ................................................................. iii
List of Figures ......................................................... v
Acknowledgments .................................................. vi

1 Introduction 1
1.1 Origins ............................................................... 2
1.2 Influences ............................................................ 3

2 Background 5
2.1 Information Retrieval ............................................. 5
   2.1.1 Parsing Clean Text .......................................... 6
2.2 Characteristics of OCR Text ................................... 7
   2.2.1 Determining Accuracy of OCR Text .................. 8
   2.2.2 Parsing OCR Text .......................................... 10

3 Acronym Recognizer 11
3.1 Definition of an Acronym ....................................... 11
3.2 Outline of the Acronym Definition Recognizer .......... 11
   3.2.1 Initialization .................................................. 12
   3.2.2 Filtering the input .......................................... 12
   3.2.3 Word parsing ............................................... 12
   3.2.4 Applying the algorithm .................................. 15
3.3 More Examples .................................................... 25

4 Performance Analysis 27
4.1 Training and Test Sets .......................................... 27
4.2 Evaluation and Results ......................................... 28

5 Conclusions and Future Work 30
Bibliography .......................................................... 31
Acknowledgments

I would like to thank Dr. Kazem Taghva, for I doubt I would have pursued this degree without his help and encouragement. I am also very grateful to Julie Borsack for her advice, suggestions, and also for the evaluation of the program’s performance. Many thanks to the all the members of the Text Retrieval group here at ISRI for helping with various aspects of this project. I would also like to thank Dr. Tom Nartker, Dr. Laxmi Gewali, and Dr. Stephen Lepp for serving on my graduate committee.
Chapter 1

Introduction

Current trends in Information Retrieval (IR) research consist of moderate extensions to traditional systems that achieve only incremental improvements in precision and recall. At their very best, traditional systems can only crudely approximate the understanding that is desired and required from next generation information systems. If the industry is to make any significant steps toward increased information retrieval performance it will most likely be achieved by document understanding models. Natural language processing has shown promise but has also proven to be very difficult.

Behind this lack of significant progress of traditional systems seems to be the total dependence on individual index terms to convey document meaning. Whether the system is boolean, probabilistic, or vector space, the presence or absence of index terms is the most important piece of information. It should be clear that document meaning based on the sum of the words in a document is inferior to document meaning as a set of relationships between words in a document. The work by Fagan on phrases [7] is a step in this direction but has not been very fruitful.

Some well known systems (SMART [14] and INQUERY [2]) have provisions for special recognizers that can contribute to both the quantity and quality of information available to the retrieval system but demonstrations of their practical use have not been forthcoming. More advanced and less traditional approaches to information retrieval have surfaced that make use of limited document understanding via specific special recognizers and parsers built around them. Several examples of such systems are Mauldin’s FERRET [10], a document skimming parser; the Associations System by Conrad and Utt [3]; Rau’s automatic indexing system used at GE [11]; and a
text post-processing system (PPSYS [16]), developed here at the Information Science Research Institute (ISRI).

These new systems are based on the same approach; the use of special recognizers. The heart of Rau’s system is a company name recognizer. The Associations System uses both company and person name recognizers. PPSYS has, in its repertoire of tools, an acronym definition recognizer. Mauldin’s system is more expansive than the other systems; it is based on a parser that identifies dates, times, numbers, and quantities.

It should be clear that while true natural language processing is exceptionally difficult, a parser built on top of a set of special purpose recognizers can provide richer results than traditional retrieval systems and may be an acceptable stepping stone until natural language processing systems mature.

1.1 Origins

Our interest in acronyms started with the development of a post-processing system (PPSYS) for the improvement of text output from optical character recognition (OCR) devices [16]. Originally, acronyms were a nuisance—words that almost never appeared in dictionaries—but of course, were known to be valid strings. The most fundamental part of PPSYS involved finding and correcting misrecognized words. So our first acronym finder removed these words from the text to alleviate erroneous clustering and correction by PPSYS.

Recently, as a part of our research on the issues associated with retrieval from OCR text[18, 17, 19], we observed that OCR devices generally have lower accuracy rates for certain groups of words, for example proper nouns. This lower accuracy is due to the fact that these devices rely heavily on the use of lexicons and statistical distribution of character n-grams. Unfortunately, these groups of special words are also identified as having higher retrieval value[11, 10, 5].

There are many automated procedures to extract features from documents in order to populate databases[11, 3, 1]. Since acronyms are found in documents with their definitions, the probability that they are correct is quite high;

a database can be built and used to identify further instances in the current
document or document set. They can also be used to enhance retrieval and/or identify associations and relationships to be used for a hypertext browsing system.

In a hypertext system, acronyms can be used to link documents which are related to each other. These links can be used to identify all documents written on a specific project or about a particular government agency. Furthermore, since government documents contain a large number of acronyms, a useful tool for the reader would be a routine that can provide acronym definitions immediately. With this routine, one could click on an acronym and find its definition.

The program that recognizes acronyms and acronym definitions does not rely on a lexicon for validation of words (except for the short list of stopwords). This means that the spelling of a word is of little concern to the acronym finder. Most modern OCR devices are especially good at correctly recognizing common words[13], so misspelled stopwords are not a major concern.

1.2 Influences

When the project started, we were designing a system to filter out garbage from error-prone OCR output. Our system for identifying acronyms in a set of terms was fairly primitive. It was at this time that we decided to look at FERRET, a text skimming system by Mauldin[10]. The FERRET system used complex lexical analysis to tokenize special words, quantities, dates, and other textual objects. This system influenced the building of a simple parser to identify acronyms in free text. While FERRET uses Lex[9] for its implementation, our acronym finding program, AFP, was designed specifically for finding acronyms.

Our next influence was the company name recognition work by Rau[11]. Upon seeing the various methods and approaches applied to the recognition of company names, we tried some proper name and acronym parsing using some of these methods. The company name variation scheme involved the generation of acronyms based upon a previously extracted company name in order to find alternative name forms. Considering this process in reverse, we surmised that one could use a candidate acronym to find a plausible definition. If found, we could be more certain that the candidate was indeed an acronym. As a side effect of this process, we would have a definition
associated with each acronym.

It was also the work of Rau that inspired us to deal with stopwords in an intelligent way. Stopwords are words that have high frequency in documents, but have low retrieval value (e.g., "the," "a," "and," "or," "in"). Stopwords are normally ignored in retrieval applications but we found they could not be ignored by AFP. By examining the approaches taken by Rau for recognizing company names, we developed a solution for handling stopwords in section 3.2.3.
Chapter 2

Background

2.1 Information Retrieval

We will first discuss the basics of IR systems, and the relationships between index terms, stopwords, and features.

An IR system provides a method of extracting information from a database of objects; in this case, we will only concern ourselves with databases of text documents. One can request information from an IR system using queries. These requests are compared to the database of documents to determine similarities. The comparison method between query and documents differs among the three main information retrieval models, but the underlying mechanisms are similar.

The most common structure for document storage in IR models is the inverted index. An inverted index transposes the document-term relationship to a term-document relationship. For each term in the collection, the documents in which that term occurs are assigned to that term.

Another common practice in most IR implementations is the removal of stopwords. Stopwords can be defined as those words in the text that do not add to a document’s substance or meaning. An example stopword list might include: the, and, to, a, in, that, through, but, or.

Special terms or features are concepts which can be expressed in one or several individual terms. For example, in the date format “February 10, 1978” there are 3 individual terms. These terms, as evaluated by a retrieval system have very little individual value, but processed by a specialized date recognizer, this string has
meaning.

A list of potential document features is shown below:

- person names
- other proper nouns
- acronyms
- dates
- times
- numbers
- quantities
- chemical formulas

### 2.1.1 Parsing Clean Text

Extensive work has been done in the field of natural language processing. A few examples of feature recognizers are Fagan’s work with phrase indexing[7], Deerwester’s sentence parser[6], Rau's company name extractor[11], and Mauldin’s FERRET[10]. Other systems that use feature recognizers include Conrad and Utt’s Associations System[3], the SMART retrieval system[14], and INQUERY[2]. Regardless of the goal, the techniques employed in these systems all depend on the fundamental assumption of parsing “clean” text.

Some assumptions necessary for text parsing are:

- words are correctly spelled
- capitalization is correct
- punctuation is accurate
- words are in their proper order

Since OCR devices are error-prone, any one of these assumptions, or possibly all of them, may no longer hold. Methods of processing clean text can be used as a starting point for parsing OCR output but OCR characteristics must be analyzed and in turn, compensations made.
2.2 Characteristics of OCR Text

OCR devices have come a long way in the past 2 to 3 years, with some devices obtaining 97% character accuracy[13]. Although average character accuracy is fairly high, we have considerable difficulty parsing most OCR output. The reasons for this difficulty go beyond the basic misspelling of words in text.

Most devices follow the same basic paradigm as shown in figure 2.1. The process involves four basic steps:

1. scanning the hard copy to produce an image,
2. automatic zoning to identify and order regions of text,
3. segmentation determination: breaking zones into words and words into characters,
4. character classification.

At each stage of the process, errors can be introduced. For example:

scanning

problems: (1) words scanned from adjacent pages, (2) clipped edges, (3) artifacts, (4) excessive skew.

cause: poor quality originals, careless scanning, damage during handling.

effects: all stages of the OCR process.

automatic zoning

problems: continuity errors.

cause: incorrect/aggressive decolumnization.

effects: word order.

segmentation
problems: (1) single characters in original recognized as two, (2) multiple characters recognized as one, (3) insertion or deletion of spaces.

cause: overlapping characters, artifacts, broken characters, unusual fonts, skew.

effects: (1) upper case characters mostly, some lower case (e.g. m's and n's → iii and ii). (2) punctuation and some lower case sequences (e.g. rn → m). (3) division and concatenation of words.

classification

problems: all other character recognition failures.

cause: same as segmentation.

effects: most of the single character errors.

These problems all affect the parsing of OCR text. Obviously, any problems with segmentation and character classification will produce misrecognized words. Punctuation is often useful in recognizing features, and that can be distorted or omitted from the output. Even capitalization is affected, although changes in case are not as common as other character errors.

Of all the errors that occur in OCR processing, the most difficult to deal with are those caused by inaccurate automatic zoning. Most features extracted for processing are made up of multiple terms and are found by characteristics adjacent to the feature itself. Errors in word order will render these feature recognizers ineffective. Most evaluations of OCR systems measure performance using character accuracy [12, 13], but use manual zoning to prevent automatic zoning from skewing results.

Manual zoning input is accepted for most devices, but since human intervention is required, the process is expensive and tedious. For IR applications in which term indexing is applied, zoning errors have almost no effect. Systems that perform statistical or semantic processing however, may be affected, but by how much is not known.

2.2.1 Determining Accuracy of OCR Text

class character accuracy

Character accuracy is frequently used to measure the quality of OCR systems and is the total characters minus the number of errors divided by the total number of characters (c) in the correct text:
Character Accuracy = \frac{c - \text{errors}}{c}.

The number of errors is calculated from the minimum number of insertions (i), substitutions (s), and deletions (d) required to correct the OCR output to agree with the correct text:

\text{errors} = i + s + d.

word accuracy

OCR performance should not be quantified by character accuracy alone. Character accuracy only gives a general idea of the accuracy of the device. In dealing with OCR text, it would be good to know how well the device performs on different types of words. Word accuracy is simply the number of correctly recognized words in the output divided by the number of words in the original text. By knowing what kinds of words are recognized best by the device, algorithms can be designed to exploit this information. The ISRI annual tests have reported word, stopword, non-stopword, and phrase accuracy. However, for the purpose of feature extraction we would like to know how well the desired feature(s) are recognized by the device.

stopword accuracy

Because stopwords are the most common words in the English language, OCR devices should do very well in recognizing these words in order to attain high character accuracy rates. Statistical methods are often used in OCR devices, and stopwords by definition, have the highest statistical probabilities in text. It should be no surprise that stopword accuracy rates are usually quite high[13].

feature accuracy

From previous studies[19], we have found that feature accuracy is lower than non-stopword accuracy. We found that devices tended to do 3–4% poorer on recognizing features than non-stopwords. This poor performance affects more than just the recognition of features. Since OCR devices use lexicons to aid in recognition and features
(proper nouns and acronyms in particular) are rarely in the dictionary, the device will often misrecognize the feature in favor of a closely spelled dictionary term. Secondly, if a feature happens to be correctly recognized, it can be inadvertently "corrected" to a more common term by an OCR post-processing system.

2.2.2 Parsing OCR Text

Work has been done by DeSilva and Hull[15] to recognize proper nouns in digitized images before character recognition. It was apparent that lexical techniques would not be effective in recognizing these words and that it was important to locate them for special processing. First, potential candidates were identified by capitalization alone. All non-candidates were processed normally, making parts-of-speech information available for proper noun classification. Lastly, seven characteristics were used to classify the candidates. These seven characteristics are: (1) word length, (2) length of previous word, (3) length of following word, (4) part-of-speech of previous word, (5) part-of-speech of following word, (6) capitalization of the previous word, (7) and capitalization of the following word.

For AFP, several text characteristics are used: capitalization, word length, spelling, and punctuation. Capitalization and word length are used to identify candidate acronyms; i.e. the point where a definition search is initiated. Limited use of spell checking is used to reject words that meet the candidate requirements of capitalization and length, but are known not to be acronyms. Spelling is also used to identify stopwords during the recognition process. Punctuation is ignored in the recognition process except for word hyphenation. When searching for a likely definition, it is often useful to look at all parts of hyphenated words to find correlations.

Since very little lexical information is used by the algorithm, problems are minimized. When dealing with OCR text, spelling is always suspect. Therefore it is only used to identify the most common words, which have a high probability of being correctly recognized. Character accuracy is still important, but the recognizer is more forgiving of errors.

The next chapter discusses in detail the approach and methods devised to recognize acronyms and definitions in OCR text.
Chapter 3

Acronym Recognizer

This chapter describes the implementation of the acronym finder, AFP. While the goal is to find acronyms, the program actually looks for acronym definitions. In this way, we can be more sure of the authenticity of acronym candidates.

3.1 Definition of an Acronym

Webster’s 7th Dictionary defines “acronym” as:

a word (such as radar or snafu) formed from the initial letter or letters of each of the successive parts or major parts of a compound term.

Our working definition of an acronym candidate, however, is simply an upper-case word from 3 to 10 characters in length. This is straightforward except for the length restriction. The lower bound is a compromise between recall (acronyms of 2 characters do exist) and precision (approximate matching on anything less than 3 characters is very error prone). The upper bound is an arbitrary but reasonable assumption. Acronyms longer than 10 characters are quite rare.

3.2 Outline of the Acronym Definition Recognizer

The program consists of four phases: initialization, input filtering, parsing the remaining input into words, and the application of the acronym algorithm.
3.2.1 Initialization

The input for the algorithm is composed of several lists of words, with the text of the document as the final input stream. These inputs are:

1. A list of stopwords—commonplace words that are often insignificant parts of an acronym (e.g., “the,” “and,” “of”). It is important to distinguish these stopwords from regular words for the algorithm to make good matches with the definitions. This list is required.

2. A list of reject words—words that are frequent in the document, or in general, but are known not to be acronyms (e.g., “TABLE,” “FIGURE,” Roman Numerals). The fewer acronym candidates there are, the more efficient the program, and in turn, the fewer coincidental matches. This list is optional.

3. A database of acronyms and their accompanying definitions. This information can be used to either override the program’s searching routine or as a fall-back mechanism when a search is fruitless. This database is optional.

4. The text of the document (or collection) to be searched.

3.2.2 Filtering the input

The input is pre-processed to disregard lines of text that are all uppercase (e.g., titles and headings). Upon identifying an acronym candidate, the reject word list is consulted before subsequent processing. If the candidate does not appear in the reject list, then an appropriate text window[3] surrounding the acronym is searched for its definition. The text window is divided into two subwindows, the pre-window and the post-window. Each subwindow’s length in words is set to twice the number of characters in the acronym.

3.2.3 Word parsing

In order for this algorithm to find a reasonable number of acronym definitions, a precedence has to be assigned to different types of words. Currently, these types are limited to (1) stopwords, (2) hyphenated words, (3) acronyms themselves, and (4)
ordinary words that do not fall into any of the above categories. The following gives
the philosophy behind categorizing the words into types.

**Stopwords** — Normally ignored in traditional text retrieval applications, stopwords
cannot be eliminated from the definition search process. If the algorithm ignores
stopwords completely, many acronyms are not found. Similarly, if stopwords are
not ignored, many acronyms will not be correctly identified. Precedence of non-
stopwords over stopwords in the matching process helps resolve these problems.
For example:

- stopwords must be counted  Department of Energy (DOE)
- as low as reasonably achievable (ALARA)

- stopwords must be ignored  Office of Nuclear Waste Isolation (ONWI)

**Hyphenated Words** — Hyphenated words are treated as a special case. Acronym
definitions often contain hyphenated words in which either the first, or all of
the word parts of the hyphenated word correspond to letters of the acronym.
Both cases must be checked to find the best match. For example,

- first word part matches  X-ray photoelectron spectroscopy (XPS)
  high temperature gas-cooled reactor (HTGR)

- all word parts match  non-high-level solid waste (NHLSW)
  June–July–August (JJA)

**Acronyms** — Acronyms sometimes occur within short word distances of each other.
Since acronyms sometimes include other acronyms in their definitions, we don’t
want to abort processing if this situation occurs. What we can do is to abort
processing if the acronym encountered is the same as the one we are trying to
define. For example,

what we want to find:
ARINC Communications and Reporting System (ACARS)

what we don’t want to find:

with SIMS. In most cases, separate SIMS profiles were

Normal Words — Words that don’t fall into any of the above categories are considered normal words. These words make up the majority of the words in acronym definitions and require no special handling.

When a subwindow is parsed, we generate two symbolic arrays for that window: the leader array, consisting of the first letter of each word, and the type array, consisting of the type of each word in the subwindow. For simplicity, we use the characters s, h, h, a, and w to denote stopwords, the initial part of hyphenated words, following parts of hyphenated words, acronyms, and normal words, respectively. These abstractions simplify the main engine since it becomes unnecessary to scan the text strings. We can systematically search through the text windows for matches of the first letters of words and the acronym letters.

Example 1

Given the text:

spent fuel and recycling the recovered uranium and plutonium results in the generation of transuranic (TRU) non-high-level solid waste (NHLSW). Volumes and characteristics of these wastes, and methods for

the pre-window for the acronym NHLSW is:

[results in the generation of transuranic (TRU) non-high-level solid waste]

The leader and type arrays are:

[r i t g o t t n h l s w] leaders
[w s s w s w a H h h w w] types
3.2.4 Applying the algorithm

The algorithm identifies a common subsequence of the letters of the acronym and the leader array to find a probable definition. Following [4], a subsequence of a given sequence is just the given sequence with some elements removed. For two sequences $X$ and $Y$, we say that a sequence $Z$ is a common subsequence of $X$ and $Y$ if $Z$ is a subsequence of both $X$ and $Y$. For example, if $X = acbceac$ and $Y = cebaca$, then $cba$ is a common subsequence of $X$ and $Y$ of length 3. Observe that $ceac$ and $cbca$ are also common subsequences of $X$ and $Y$ (length 4), and there are no common subsequences of length greater than 4 (i.e., $ceac$ is a common subsequence of maximum length).

The longest common subsequence (LCS) of any two strings $X$ and $Y$ is a common subsequence with the maximum length among all common subsequences. We also want to point out that LCS $ceac$ can be generated from $X$ by indices $[2, 5, 6, 7]$ or indices $[4, 5, 6, 7]$. The need for this distinction will be apparent shortly.

There are well known and efficient algorithms to find an LCS of two sequences [4][8]. Most of these algorithms only find one LCS. To fully explain AFP, we first introduce the LCS algorithm as described in [4], then we present an algorithm to generate all possible LCS’s. Finally, we give our algorithm to locate the acronym definition.

We use the notation $X[1 \ldots i]$ to denote the prefix of length $i$ in the string $X[1 \ldots m]$. Now, for two strings $X[1 \ldots m]$ and $Y[1 \ldots n]$, let $c[i, j]$ be the length of an LCS of the sequences $X[1 \ldots i]$ and $Y[1 \ldots j]$. We observe that when either $X$ or $Y$ are empty sequences, then the LCS is an empty string and $c[i, j] = 0$. We also know that $c[i, j]$ can be obtained from the following recursive formula:

$$c[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } X_i = Y_j \\ \max(c[i-1, j], c[i, j-1]) & \text{if } i, j > 0 \text{ and } X_i \neq Y_j \end{cases} \quad (3.1)$$

This recursive equation states that in order to compute the LCS of $X$ and $Y$ (in notation $LCS(X, Y)$), we should test to see if $X[m] = Y[n]$. In case the equality holds, calculate $LCS(X[1 \ldots m - 1], Y[1 \ldots n - 1])$, otherwise choose the larger of $LCS(X[1 \ldots m], Y[1 \ldots n - 1])$ and $LCS(X[1 \ldots m - 1], Y[1 \ldots n])$.

Figure 3.1 shows a dynamic programming algorithm [4] of the recursive equation 3.1. The algorithm computes the length of an LCS for strings $X$ and $Y$ and
build-LCS-matrix(X, Y)

1  m ← length[X];
2  n ← length[Y];
3  for i ← 1 to m do
4     c[i, 0] ← 0;
5  for j ← 1 to n do
6     c[0, j] ← 0;
7  for i ← 1 to m do
8      for j ← 1 to n do
9         if X[i] = Y[j] then
10            c[i, j] ← c[i-1, j-1] + 1;
11            b[i, j] ← \"\";
12         else if c[i-1, j] ≥ c[i, j-1] then
13            c[i, j] ← c[i-1, j];
14            b[i, j] ← \"\";
15         else
16            c[i, j] ← c[i, j-1];
17                b[i, j] ← \"\";
18  return c and b;

Figure 3.1: The build-LCS-matrix routine.
stores this value in \( c[m, n] \). If this LCS length falls below the confidence level threshold, no further processing for this acronym will be done. The calculation of this confidence level will be explained in more detail in section 3.2.4. The LCS construction method in [4] utilizes the matrix \( b \) to show the path from which an LCS can be constructed.

A "\" entry in \( b[i, j] \) asserts that \( X[i] = Y[j] \), and \( c[i-1, j-1] + 1 \) is the selected value in equation 3.1. A "\†" or "←" in \( b[i, j] \) asserts that \( X[i] \neq Y[j] \), and \( c[i-1, j] \) or \( c[i, j-1] \) is the selected value in equation 3.1, respectively.

**Example 2**

Consider the following text:

This work was conducted as part of the Department of Energy's (DOE) National Waste Terminal Storage program under the management of the Office of Nuclear Waste Isolation (ONWI). A primary objective of the program is to develop and demonstrate the technology for safe disposal of nuclear waste including spent commercial reactor fuel.

the pre-window for the acronym ONWI is:

[management of the Office of Nuclear Waste Isolation]

the leader and type arrays are:

[ motoonwi ] leaders
[wsswswww] types

Then build-LCS-matrix("onwi", "motoonwi") will produce the \( b \) and \( c \) matrices in Figure 3.2. Matrix \( b \) is superimposed over \( c \) to show their relationship. The length of \( LCS(\"onwi\", \"motoonwi\") \) is 4.

The matrix \( b \) is used to construct an LCS by starting from the lower right-hand corner; each "\" corresponds to an entry where \( X[i] = Y[j] \). The LCS construction method used in [4] only finds one LCS. For AFP, we are interested in all ordered arrangements of indices leading to an LCS. We developed the procedures parse-LCS-matrix and build-vector in Figure 3.3 to accomplish this goal. Let
Figure 3.2: The c and b matrices computed by build-LCS-matrix on \( X = \text{onwi} \) and \( Y = \text{motoonwi} \).

<table>
<thead>
<tr>
<th></th>
<th>c</th>
<th>m</th>
<th>o</th>
<th>t</th>
<th>o</th>
<th>n</th>
<th>w</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

\( b[i,j] \) be an entry in the matrix \( b \) with value "\( \wedge \)", then the procedure limits its search to the sub-matrix \( b[i+1...m, j+1...n] \) to build the rest of the LCS. The procedure uses a stack to store the partial sequences leading to the LCS. Finally, the procedure uses the indices of the LCS to construct a vector representation of a possible definition for the acronym.

Earlier we showed that LCS(“onwi”, “motoonwi”) was found to be of length 4. The parse-LCS-matrix routine will produce the following ordered lists of indices (or equivalently, the stacks built by this routine):

- \((1,2), (2,6), (3,7), (4,8)\)
- \((1,4), (2,6), (3,7), (4,8)\)
- \((1,5), (2,6), (3,7), (4,8)\)

The notation \((i,j)\) indicates that the \(j\)th leader entry matches the \(i\)th letter of the acronym. The build-vector routine creates the vectors by setting the \(j\)th entry to the value \(i\) for all \((i,j)\) entries in the stack, with the remaining entries set to 0. For this example, the corresponding vectors are:

\[
\begin{align*}
[0 & 1 0 0 0 2 3 4] \\
[0 & 0 0 1 0 2 3 4] \\
[0 & 0 0 0 1 2 3 4] \\
\end{align*}
\]
parse-LCS-matrix(b, start.i, start.j, m, n, lcs.length, Stack, Vectorlist)

1 for i ← start.i to m do
2   for j ← start.j to n do
3       if b[i, j] = "\" then
4           s ← build-stack(i, j);
5           push(Stack, s);
6       if lcs.length = 1 then
7           vector ← build-vector(Stack, n);
8           add(Vectorlist, vector);
9       else
10          parse-LCS-matrix(b, i+1, j+1, m, n, lcs.length—1, Stack, Vectorlist);
11         pop(Stack);
12     return;

build-vector(Stack, n)

1 v ← allocate-vector(n);
2 for j ← 1 to n do
3   v[j] ← 0;
4 s ← Stack;
5 while s ≠ NIL do
6   v[s[j]] ← s[j];
7   s ← next[s];
8 return v;

Figure 3.3: The parse-LCS-matrix and build-vector routines.
Figure 3.4: The parsing of the $c$ and $b$ matrices by parse-LCS-matrix. To reconstruct the elements of all LCS's, start at the upper left-hand corner; searching for the first $(i,j)$ such that $X[i] = Y[j]$, indicated by an "\" entry in the matrix. When a matching $(i,j)$ is found, recursion is used to parse the sub-matrix $b[i+1 \ldots m, j+1 \ldots n]$ (shaded). Every matching is processed in this way; increased shading is used to illustrate the recursive processing of sub-matrices.

Referring back to our leader array in example 2, the second vector indicates that for acronym ONWI, the letters o, n, w, and i occur as the leaders of the 4th, 6th, 7th and 8th words in the pre-window.

The last part of the algorithm deals with selecting the appropriate definition for the acronym from the vectors generated by parse-LCS-matrix. The procedure vector-values($V$) in Figure 3.5 calculates the following four values for each vector:

1. $\text{misses}[V]$:
   The number of zero entries in the vector; disregarding leading zeros, trailing zeros, and those zero entries corresponding to words of types s or h. Gives the number of words in the definition that do not match a letter of the acronym.

2. $\text{stopcount}[V]$:
   The number of stopwords that will be used in the acronym definition if the vector is selected.

3. $\text{distance}[V]$:
The index of the last non-zero entry. This value measures the proximity of the definition to the actual acronym.

4. \textit{size}[^V]:
   The number of entries in the vector after removing leading and trailing zeros.
   This value represents the length of the definition in words.

Finally, the procedure \texttt{compare-vectors}(A,B) in Figure 3.6 will choose one of two input vectors by comparing the vector values of \textit{A} with the vector values of \textit{B}. The procedure chooses a vector by priority processing. If all conditions fail to resolve the comparison, the procedure will return vector \textit{A}. In practice, this situation is rare (we have not seen one). The following type array and vectors are constructed artificially to illustrate that this last case \textit{can} occur:

\begin{verbatim}
[w H h w H h w w s] types

[0 1 2 0 3 0 4 5 0] vector A
[0 1 0 2 3 4 0 5 0] vector B
\end{verbatim}

vector values of \textit{A} and \textit{B}:

\begin{tabular}{|c|c|}
\hline

<table>
<thead>
<tr>
<th></th>
<th>\textit{A}</th>
<th>\textit{B}</th>
</tr>
</thead>
<tbody>
<tr>
<td>misses</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>stopcount</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>distance</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>size</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
\hline
\end{tabular}
vector-values(V)

1  i ← 1;
2  while i < length[V] and V[i] = 0 do
3       i ← i + 1;
4  first ← i;
5  i ← length[V];
6  while i > 0 and V[i] = 0 do
7       i ← i - 1;
8  last ← i;
9  size[V] ← last - first + 1;
10  distance[V] ← length[V] - last;
11  for i ← first to last do
12     if V[i] > 0 and types[i] = 's' then
13       stopcount[V] ← stopcount[V] + 1;
14     else if V[i] = 0 and types[i] ≠ 's' and types[i] ≠ 'h' then
15       misses[V] ← misses[V] + 1;

Figure 3.5: The vector-values routines.

Example 3

Recall that in example 2, the parse-LCS-matrix routine generated the following vectors:

\[
\begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 2 & 3 & 4 \\
0 & 0 & 0 & 1 & 0 & 2 & 3 & 4 \\
0 & 0 & 0 & 0 & 1 & 2 & 3 & 4
\end{bmatrix}
\]

vector A

vector B

vector C

The values calculated by the vector-values routine are as follows:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>misses</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>stopcount</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>distance</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>size</td>
<td>7</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

The call compare-vectors(A, B) will return B, since misses[A] > misses[B]. The call compare-vectors(B, C) will return B since stopcount[B] < stopcount[C]. Therefore, vector B is chosen, producing the definition:

"Office of Nuclear Waste Isolation".
compare-vectors(A, B)

1     vector-values(A);
2     vector-values(B);
3      if misses[A] > misses[B] then
4          return (B);
5      else if misses[A] < misses[B] then
6          return (A);
7      if stopcount[A] > stopcount[B] then
8          return (B);
9      else if stopcount[A] < stopcount[B] then
10         return (A);
11      if distance[A] > distance[B] then
12         return (B);
13      else if distance[A] < distance[B] then
14         return (A);
15      if size[A] > size[B] then
16         return (B);
17      else if size[A] < size[B] then
18         return (A);
19      return (A);

Figure 3.6: The compare-vectors routines.
Confidence Level

Once the length of the LCS(acronym, leaders) is known (found in build-LCS-matrix), the next step in the definition searching process is to compute the confidence level of the current acronym candidate. The confidence level is simply:

$$\frac{\text{length of LCS}}{\# \text{ of acronym letters}} + \text{(error percentage)}$$

where the error percentage is configurable at runtime (20% by default). If the confidence level is greater than or equal to one, the algorithm continues with parse-LCS-matrix. If the confidence level is less than one, the search is abandoned since there is not an adequate correlation between the text window and the letters in the acronym (i.e. there probably isn't a definition to be found).

An exact matching algorithm is less error-prone, but allowing limited misses in definitions compensates for some of the more creative and unusual acronym definitions:

Northeast Utilities Service Company (NUSCO)
Intergranular stress-corrosion cracking (IGSCC)
Superconduction Quantum Interference Device (SQUID)
independent interim plutonium oxide storage facility (IIPSF)
3.3 More Examples

Example 4

Given the following text:

These costs also include the effect of additions to utility supplies such as electrical substation; heating, ventilating, and air conditioning (HVAC); compressed air; and similar auxiliaries at the FRP; as well as the cable, piping, and other bulk materials incorporated directly into the FRVSF.

the pre-window for the acronym HVAC is:

[as electrical substation; heating, ventilating, and air conditioning]

the leader and type arrays are:

[a e s h v a a c] leaders
[s w w w w w w] types

producing two LCS's with the following vector representations:

[0 0 0 1 2 3 0 4] vector A
[0 0 0 1 2 0 3 4] vector B

Calculating the vector values, we get:

\[
\begin{array}{cc}
A & B \\
\text{misses} & 1 & 0 \\
\text{stopcount} & 1 & 0 \\
\text{distance} & 0 & 0 \\
\text{size} & 5 & 5 \\
\end{array}
\]

Vector B will be chosen, since misses[A] > misses[B].
Example 5

Given the following text:

Threat scores produced by NMC’s operational regional model (the Limited area Fine-mesh Model, or LFM) for 0.25 mm of precipitation in the 12-24h forecast period are considerably higher (averaging ~0.40) and have shown a slight increase since 1976 (Fig. 4).

the pre-window for the acronym LFM is:

[(the Limited area Fine-mesh Model, or]

the leader and type arrays are:

[t l a f m m 0] leaders
[s w w H h w s] types

LCS vectors:

[0 1 0 2 3 0 0] vector A
[0 1 0 2 0 3 0] vector B

Calculating the vector values, we get:

\[
\begin{array}{cc}
A & B \\
misses & 1 & 1 \\
stopcount & 0 & 0 \\
distance & 2 & 1 \\
size & 4 & 5 \\
\end{array}
\]

Vector B will be chosen since distance[A] > distance[A], producing

“Limited area Fine-mesh Model”

as the definition, rather than

“Limited area Fine-mesh”. 
Chapter 4

Performance Analysis

4.1 Training and Test Sets

AFP was tested on a collection of documents provided to ISRI by the Department of Energy (DOE). This collection is almost entirely made up of government studies relevant to the Yucca Mountain Waste Disposal Project. The ASCII text of the collection is considered to be 99.8% correct. This collection consists of 1328 documents in a variety of formats. The documents have a wide content range and represent the work of many different organizations and authors. Since government agencies seem to be the source of most acronyms, we felt this collection was appropriate for our testing.

The training and test sets, while mutually exclusive, involved only a fraction of the documents in the collection. To select these sets, the full collection was automatically analyzed and sequenced according to the approximate ratio of acronyms to document length. A small set of these were selected for training and 17 documents were randomly selected from the remaining top 10% of the sequenced list for the test set. The training set was used to tune the acronym finding algorithm, develop new strategies, eliminate bugs, and adjust parameters. For example, the appropriate window size, word categories (e.g., stopwords, hyphenated words), and the default error percentage were tuned using the training set. No changes were made to the algorithm at evaluation time; and except for the information about the high incidence of acronyms in the test documents, no other information about their content was known prior to our evaluation.
4.2 Evaluation and Results

Our evaluation method for AFP mirrors the standard methods applied in most text retrieval experiments. We use:

\[
\text{recall} = \frac{\text{# of correct acronym definitions found by AFP}}{\text{total # of acronym definitions in the document}}
\]

\[
\text{precision} = \frac{\text{# of correct acronym definitions found by AFP}}{\text{total # of acronym definitions found by AFP}}
\]

An independent evaluator tallied the number of acronym definitions in the text, as well as manually examined the algorithms performance on the test set. The results did not include what the evaluator classified as abbreviations. Abbreviations encompass acronyms, so the evaluator distinguished between them by applying the following rules:

- Abbreviations shorten single words, acronyms do not.
- Abbreviations can include break characters, acronyms do not (e.g. ".").
- Abbreviations are used for unit measures, acronyms are not.
- All other shortened word forms were counted as acronyms.

Excluded words:

- DOP  dioctyphthalate
- MFBM thousand board feet measure
- TRU  transuranic
- MW-hr megawatt-hour

Included words:

- EDBH  Engineered design borehole
- D&E  Development and evaluation
- CHEMTREC Chemical Transportation Emergency Center

Following this definition, there were 463 acronym definitions in the 17 documents used for the evaluation. Of these, 398 were correctly identified by AFP, yielding:
\textit{recall} = 86%  \\
\textit{precision} = 98%

We made a conscious decision to exclude acronyms of two or fewer characters. If we exclude these from our evaluation, the recall results improve:

\textit{recall} = 93%  \\
\textit{precision} = 98%

Acronyms missed by AFP, and the reasons they were missed include:

\textbf{MSRE: molten salt reactor}— Falls below the default 80% threshold.

\textbf{R&D: research and development}— Was not considered an acronym candidate due to the ' & ' symbol.

\textbf{GBL: grain boundary sliding controlled by lattice diffusion}— Filtered out due to too many misses.

\textbf{TWCA: Teledyne Wahchang Albany}— Falls below the default 80% threshold.

\textbf{USGS: U.S. Geological Survey}— "U.S." was considered a single word when parsed, and therefore falls below the default 80% threshold.
Chapter 5

Conclusions and Future Work

AFP did quite well on a difficult document collection. Of course, with hindsight, it is easy to see how the program could be improved; most notably, the inclusion of ‘&’ as an acronym character would increase recall. Some adjustments like special acronym characters or acronym length could be provided as options to AFP so the program could be tailored to a document’s or collection’s content. But in its current form, the program’s framework is quite solid for its dedicated task.

Previous work involving the automatic recognition of features [3, 5, 6, 7, 11] implicitly assumes “clean” text, not the error-prone output of OCR devices. As a result of allowing misses in AFP, this algorithm is naturally suited for use with OCR data without any further modifications except possibly tuning the allowable error percentage. Further analysis is needed to determine the algorithm’s precision and recall on OCR text.

Since acronyms pose problems with applications that require comprehensive dictionaries, the idea of building a database of acronyms and definitions has been proposed. A project to marry the acronym recognizer algorithm and a World-Wide-Web (WWW) indexing robot is under consideration; the Web is an immense resource for potential IR research.
Bibliography


