Hairetes: A Search Engine for OCR Documents

Kazem Taghva and Jeffrey Coombs

Information Science Research Institute
University of Nevada, Las Vegas
taghva@isri.unlv.edu

Abstract. In this paper, we report on the architecture and preliminary implementation of our search engine, Hairetes. This engine is based on an extended concept of *Retrieval by General Logical Imaging* (RbGLI). In this extension, word similarity measures are computed by EMIM and Bayes' theorem.

1 Introduction

During the 1990's the Information Science Research Institute (ISRI) at the University of Nevada, Las Vegas conducted a series of large scale OCR tests to better understand OCR accuracy with respect to retrieval effectiveness [13,14,15,16]. These tests generally imply that average precision and recall are not affected by OCR errors. They also imply that certain ranking algorithms can produce marked variability in document ranking. These ranking problems are essentially due to normalization factors such as document length which one can overcome by using length normalization as defined by Singhal [12].

One of the more interesting experiments we conducted was the role of OCR errors with respect to feedback [14]. This experiment showed that in the manually corrected collection, average precision keeps improving as more terms are added to queries. But in the OCR collection, precision values level off after a certain number of term expansions. Further analysis showed that this complication was a result of a few "difficult to retrieve" documents within the OCR collection. Consequently, one can assume for documents with low OCR accuracy (such as handwritten texts, faxes, or nth-generation photocopies) that the retrieval may require more effort.

Putting OCR issues aside for a moment and concentrating on retrieval concepts based on statistical models, one observes that *term mismatch* has played an important role in hindering the user from finding relevant documents [2]. Typically, a document is not retrieved unless the query and document have some terms in common. Many approaches such as *latent semantic indexing*, use of thesauri, and query expansions were developed to address the term mismatch problem [1,5,6]. One of the more recent and promising approaches is *Retrieval by General Logical Imaging* (RbGLI) proposed by Crestani and Van Rijsbergen [3, 4,11]. This technique is heavily dependent on the assumption that a measure of similarity on the term space can be evaluated. Hairetes is an implementation of this idea extended by OCR word similarities adopted from OCRSpell [18]. We

D. Lopresti, J. Hu, and R. Kashi (Eds.): DAS 2002, LNCS 2423, pp. 412–422, 2002.

believe by augmenting RbGLI with similarities based on OCR errors, we can provide an environment to study retrieval effectiveness from poorly recognized document collections.

This paper is organized into six sections. Section one is this Introduction. Section two covers some basic material on index construction. Section three shows how word similarity calculations are performed. Sections four and five are short discussions on query processing and compression. Finally, section six is the conclusion and a prospectus of future work.

2 Document Information

In building a search engine for OCR, one needs to take into consideration the type of problems that OCR errors cause and an alternative way of displaying information. In this section, we will give a modified version of a typical construction following [19].

The information associated with the collection is kept in three structures. The first structure keeps the document information. Each record in this structure looks like:

doc no.	doc weight	summary	doc-type	words	image bit	image	categories

`Summary` is a pointer to the document summary. `Doc-type` can be ASCII, OCR-ASCII, OCR with word bounding boxes, or some other possibility. `words` are pointers to the words in the document. The fields `image` and `categories` contain pointers to images and categories.

The `doc weight` in search engines represents the length of the document. This length is typically the number of distinct terms in a document or the frequency of the term with the highest occurrence [8,9]. It is shown in [13,14] that these sorts of length estimation cause problems when the collection is predominantly OCR text. Singhal [12] defined the length of the document as byte size and showed that this notion of length works well with OCR collections. In our engine, we use this length measure.

The second structure is the lexicon with records of the form:

word t	word no.	f_t	max sim	similar	max OCR-related	OCR-related	Excp

f_t is the number of documents in which `word t` occurs. `Max sim` is the number of non-empty pointers to similar terms. `Similar` contains pointers to similar words. `Max OCR-related` is the number of non-empty pointers to OCR related terms. `OCR-related` contains pointers to OCR related terms. `Excp` is an exception bit indicating that the term is an acronym, proper noun, garbage string, etc.

The third structure contains the posting for each word. These postings contain document word occurrence plus other useful information such as word position in the document and bounding box coordinates.

2.1 Creating an Inverted File

The inverted file is built following [19].

1. /*initialization*/
   a) create an empty lexicon S.
   b) create an empty temporary file.
2. /* process text and write temporary file */
   For each document D_d in the collection $1 \leq d \leq N$
   a) Read D_d, parsing it into index terms.
   b) for each index term $t \in D_d$
      i. let $f_{d,t}$ be the frequency in D_d of term t.
      ii. search S for t.
      iii. if t is not in S, insert it.
      iv. write a record (t,d,$f_{d,t}$) to the temporary file, where t is represented by the word number.
      v. call acronym(t).
      call graphic-text-recognizer(t) if t is OCR text.
      call proper-noun(t).
      If any return "yes", set `Excp`, the exception-bit, to 1 in the lexicon entry for t.
   c) /* Populate Document-info record */
      i. enter doc-num and doc-type.
      ii. enter document weight as $bytesize^{0.375}$ [12].
      iii. generate summary.
      iv. compress and store this document posting and make the pointers to the word in the document point to this location in the document posting file.
3. /* Internal sorting to make runs */
   Let k be the number of records that can be held in memory.
   a) Read k records from the temporary file.
   b) sort into nondecreasing t order, and for equal values of t, nondecreasing d order.
   c) write the sorted run back to the temporary file.
   d) repeat until there are no more runs to be sorted.
4. /* Merging */
   Pairwise search runs in the temporary file until it is one sorted run.

3 Word Similarity Calculation

Word similarity has been studied in the field of information retrieval for a long time. Thesauri construction is an example of word similarity efforts. Most of the automatic word similarity procedures depend on word frequency and co-occurrence. In the case of OCR errors, word similarity represents the closeness of a misrecognized word to other correctly recognized words in the collection. We typically need to divide the indexed terms into two groups of correctly recognized and incorrectly recognized words. This can be done with the help of a dictionary or, as we recently discovered, with dimensionality reduction techniques [17].

3.1 Apply a Dimensionality Reduction

Hence we will define a constant `Doc-Dim-Reduction` which can be changed and experimented with. For example, if `Doc-Dim-Reduction` = 3, then we will only look at the words occurring in 3 or more of the documents. We can then divide the lexicon into 2 sets of words: The set `Correct` will contain words with $f_t \geq 3$ or `Excp` = 1, and the set `Suspect` will be the complement of `Correct`.

3.2 Apply EMIM

We will then apply EMIM (*Expected Mutual Information Measure*) on the `Correct` list to generate `similar`, the similar-words' field in the lexicon:

word t	word no.	f_t	max sim	similar	max OCR-related	OCR-related	Excp
t_1				$x_1 \ldots x_n$			
t_2				$y_1 \ldots y_m$			
$\ldots$				$\ldots$			

where x_1 is the most similar to term t_1 and x_n is least similar, and y_1 is most similar to t_2 while y_m is the least similar to t_2.

EMIM is defined as:

$$EMIM(t_1, t_2) = \sum_{i=0}^{1} \sum_{j=0}^{1} r(t_1^* = i, t_2^* = j) \log_2 \left(\frac{r(t_1^* = i, t_2^* = j)}{p(t_1^* = i) p(t_2^* = j)} \right).$$

Here the expressions t_1^* and t_2^* represent functions from terms t_1 and t_2 to the set {0,1} where 0 indicates the absence and 1 the presence of a term in a document. Thus $p(t_1^* = i)$ is the probability that t_1^* returns the value i, and $r(t_1^* = i, t_2^* = j)$ is the probability that t_1^* and t_2^* return the values i and j respectively. If i and j take the value 1, then $r(t_1^* = 1, t_2^* = 1)$ is the probability that t_1 and t_2 will co-occur in a given document.

Van Rijsbergen in [10] shows that EMIM can be estimated in the following way. First define the contingency table:

	$t_l \in doc_k$	$t_l \notin doc_k$	
$t_m \in doc_k$	n_{11}	n_{01}	$n_{.1} = tf_m$
$t_m \notin doc_k$	n_{10}	n_{00}	$n_{.0}$
	$n_{1.} = tf_l$	$n_{0.}$	N

The value n_{11} is the number of documents in which both terms t_l and t_m occur (that is, $t_l^* = 1$ and $t_m^* = 1$), n_{01} the number of documents in which t_m occurs but t_l does not, n_{10} the number of documents containing t_l but not t_m, and n_{00} is the number of documents in which neither term occurs. The value $n_{1.}$ is the total number of documents in which t_l appears, that is, $n_{1.}$ is the *term frequency* tf_l of t_l. The expression $n_{0.}$ stands for the number of documents in which t_l does *not* occur, which is the same as $N - tf_l$ where N is the total number of documents in the collection.

Any item in the contingency table can be calculated *given* that n_{11} and the values of tf_m, tf_l, and N are known. The marginal values are easy to calculate since tf_m and tf_l are the term frequencies of t_m and t_l, which are available from the inverted file, and N is the total number of documents in the collection. Getting n_{11}, however, may prove to be computationally expensive and may require sophisticated estimation techniques to calculate effeciently.

Van Rijsbergen uses the contingency table definitions to state an estimate for EMIM which is strictly monotone to that measure:

$$\widehat{EMIM} = n_{11}\log_2\frac{n_{11}}{tf_l tf_m} + n_{10}\log_2\frac{n_{10}}{tf_l n_{.0}} + n_{01}\log_2\frac{n_{01}}{n_{0.}tf_m} + n_{00}\log_2\frac{n_{00}}{n_{0.}n_{.0}}.$$

According to Van Rijsbergen, the first term indicates the similarity of the two terms, the second and third measure dissimilarity, and the last term is likely to be constant in large samples. Also, we must define $0\ log\ 0 = 0$.

Consider as an example the following document collection from [19]:

1. Pease porridge hot. Pease porridge cold.
2. Pease porridge in the pot.
3. Nine days old.
4. Some like it hot. Some like it cold.
5. Some like it in the pot.
6. Nine days old.

The contingency table for term 1 (*pease*) and term 2 (*porridge*) is:

	pease		
porridge	2	0	2
	0	4	4
	2	4	6

and the $\widehat{EMIM}$ values for other terms with respect to *pease* are:

	$\widehat{EMIM}$		$\widehat{EMIM}$
pease,porridge	-10.00	pease,nine	-10.00
pease,hot	-14.32	pease,days	-10.00
pease,cold	-14.32	pease,old	-10.00
pease,in	-14.32	pease,some	-10.00
pease,the	-14.32	pease,like	-10.00
pease,pot	-14.32	pease,it	-10.00

where, for example, $\widehat{EMIM}$ for *pease* and *porridge* is calculated as

$$\widehat{EMIM}(pease, porridge) = 2\log\frac{2}{4} + 0\log\frac{0}{8} + 0\log\frac{0}{8} + 4\log\frac{4}{16} = -10$$

So, the "closest" terms to *pease* are *porridge*, *nine*, *days*, *old*, *some*, *like*, and *it*.

3.3 OCR Similarity

Our next goal is to create the OCR-related terms entries for the lexicon, such that:

		OCR-related words:	
C_j	...	$S_1 \ldots S_m$	...

where each S_i is a `Suspect` term and C_j is a `Correct` term in the lexicon.

To construct these entries, a spelling correction system developed by ISRI especially for OCR text errors called *OCRSpell* can be used. OCRSpell is similar to a spell-checking program such as ISpell [7] but with the ability to help correct errors typically generated by OCR software.

In particular OCRSpell attempts to identify and suggest corrections for segmentation and classification errors resulting from the OCR process. Segmentation errors encompass such errors as recognizing single letters as multiple characters, for example, reading 'rn' for 'm', reading multiple characters as single letters, e.g., 'ci' for 'd', and incorrect concatenation and division of terms, such as recognizing 'c at' for 'cat'. Classification errors are errors such as replacing 'o' with '9' in 'J9hn'.

OCRSpell uses a specially designed parser, domain specific dictionaries, and a statistical device mapping word generator to create a list of word candidates as replacements for incorrect terms. Also provided with each replacement candidate is a probability that the corrected term is in fact the correction for that particular incorrect term. For example, if OCRSpell is given an expression such as "iiien", it produces the output:

```
@(#) Ispell Output Emulator Version 1.0.00 08/17/95
iiien
 original word: iiien
**********************************
amen     0.000359
man      0.000530
mien     0.000190
men      0.012714
iii-en   0.002057
**********************************
```

This output indicates that OCRSpell believes that there is a probability of 0.000359 that "amen" is the correct replacement for "iiien", a 0.000530 probability that "man" is the correct replacement, and so on. Further details of the OCRSpell system are available in [18].

To use OCRSpell to generate the OCR-related entries, however, we must note that the output of OCRSpell gives us $P(C_j|S_i)$, the conditional probability that C_j is the `Correct` term given we started with the `Suspect` term S_i. For the OCR-related terms, however, we want a measure like $P(S_i|C_j)$, the conditional probability S_i is the term we want given we have C_j.

We can use Bayes' Theorem to get this from the OCRSpell results:

$$P(S_i|C_j) = \frac{P(S_i) \cdot P(C_j|S_i)}{P(S_1) \cdot P(C_j|S_1) + \cdots + P(S_k) \cdot P(C_j|S_k)}$$

but the difficulty here is getting all the information needed to make the calculation. There are two problems here: (1) how to get $P(C_j|S_k)$ for all k, and (2) how to get all the $P(S_k)$'s.

With regard to the first problem, consider the output from OCRSpell for the `Suspect` term *iiien* listed earlier. What this output gives us is:

$P(C_1 = amen|S = iiien) = 0.000359$
$P(C_2 = man|S = iiien) = 0.000530$
$P(C_3 = mien|S = iiien) = 0.000190$
$P(C_4 = men|S = iiien) = 0.012714$
$P(C_5 = iii - en|S = iiien) = 0.002057$

However, what we really need in order to use Bayes' rule is a list for all k of the probabilities $P(C_j|S_k)$ for a specific C_j. For example, we want to know for all k $P(C_i = men|S_k)$, that is, the probability that *men* is `Correct` for each given `Suspect` term that OCRSpell claims is similar to *men* in the document collection.

One way to get this information is to maintain a Term Probability B^+-tree (or some other appropriate data structure) during the lexicon building phase. When the first occurrence of a `Suspect` term is discovered, OCRSpell should be run on the term, and for each `Correct` term having non-zero probability, that term should be added/updated to the B^+-tree with an entry consisting of the `Suspect` word and associated probability. Hence the entry for *men* in the Term Probability B^+-Tree would look like:

men	iiien 0.012714	mqn 0.007673	...

This entry is updated when *iiien* is first discovered in the document collection and again when *mqn* and subsequent `Suspect` terms are found.

In general, the procedure is as follows: Suppose the lexicon L of `Correct` terms exists, and there exists a list of `Suspect` terms SW. Create a useful data structure, like a B^+-tree, called B. Add items from SW to B in this way:

```
for each term s in SW
    for each Correct term c returned by OCRSpell(s)
        if c is in L
            add c to B if not there
            add s and its probability to c's entry in B
```

For the second problem of defining $P(S_k)$, we may use the *inverse document frequency (idf)*

$$P(S_k) = idf(S_k) = -\log \frac{f_{S_k}}{N}$$

since the lexicon entry of `Suspect` words will contain the frequency they occur in the collection. Now, to calculate the probabilities of the OCR-related terms, let c be a `Correct` term in B (our Term Probability B^+-Tree) and s_i one of c's `Suspect` terms and p_i its probability. That is, going back to the previous example:

c	s_1 p_1	s_2 p_2	
men	iiien 0.012714	mqn 0.007673	...

Let E be the number of OCR-related words entered in c's lexicon so far. A procedure (not necessarily the most efficient) to select the OCR-related word entries for the lexicon would be:

for each `Correct` term c in B
$E = 0$
  for each `Suspect` term entry s_i in c's entry in B
    calculate $P(s_i|c) := \frac{idf(s_i)\cdot p_i}{\sum_{k=0} idf(s_k)\cdot p_k}$
    if $E < m$,
      add a pointer to s_i and
      add $P(s_i|c)$ to c's entry in the lexicon.
      $E = E + 1$
    else if $P(s_i|c) > min(P(s_k|c))$
    (where the $P(s_k|c)$'s are values already entered under c),
        replace the minimal s_k and $P(s_k|c)$ with s_i and $P(s_i|c)$

4 Query Processing

We plan orginally to construct Hairetes using *Retrieval by General Logical Imaging (RbGLI)*. Later we will implement a *vector space* version. For more about RbGLI see [2,3,4].

4.1 Opinionated Function

First, we define the *opinionated-function* as follows:

opinionated-function $(f_t, total\_no, POS)$
  return $((-\log \frac{f_t}{N}) \cdot (2^{total\_no - POS}))$

where f_t is the number of documents containing the term t, *total_no* is the total number of terms similar to t listed in t's lexicon entry, and *POS* is the position of a term t'_i in t's lexicon entry:

term		no.	similar words:	
t	...	N	$t'_1 \ldots t'_m$	...

where t'_1 through t'_m are pointers to terms similar to t. N is a binary number indicating the number of non-empty pointers. For example 1111000000 means there are four such pointers.

4.2 Query Process

We want to retrieve r documents in response to a query. We will keep an array called the *accumulator (A)* to keep track of the similarity between the documents and the query. So, if this array is indexed with the document number, it is more likely that most of the similarities will be zero. Among the non-zero similarities we want to select the top r documents following the model of a *heap*.

Let's assume we have query Q:

1. Set $A \leftarrow \phi$, where A is the accumulator array.
2. For each query term $t \in Q$,
   a) search the lexicon
   b) record f_t and the address I_t, the inverted file entry for t.
   c) set $P(t) = -\log \frac{f_t}{N}$
   d) read the inverted file entry I_t.
      i. For each pair $(d, f_{d,t})$ in I_t
         If $A_d \notin A$ then
            set $A_d \leftarrow 0$
            set $A \leftarrow A + A_d$
         set $A_d \leftarrow A_d + P(t)$
      ii. look at the list of terms similar to t, call them $t_1, \ldots, t_k$
         A. sort this list on `word no.`
         B. look at the list of the words in d and identify among $t_1, \ldots, t_k$ those terms which are *not* in d and call these $t'_1, \ldots, t'_l$.
         C. for each t'_i look in the lexicon for the word t in the t'_i list of words.
         D. if found, set
            `opinionated-value` $\leftarrow$*opinionated-function*$(f_{t'_i}, TOT, POS)$
            where TOT is the total number of words in the similar-word list for t'_i and POS is the position of t in the similar-word list for t'_i.
         E. set $A_d \leftarrow A_d +$ `opinionated-value`
      iii. Assume $t'_1, \ldots, t'_l$ are the list of OCR-related terms of term t with probabilities $p_1, \ldots, p_l$. For each t_i in this list
         A. read I_t, the inverted file entry. For each $(d, f_{d,t})$ in this list
            if $A_d \notin A$ then
               set $A_d \leftarrow 0$
               set $A \leftarrow A + A_d$
            set $A_d \leftarrow A_d + \left(-\log \frac{f_{t_i}}{N} \times p_i\right)$
3. For each $A_d \in A$, set $A_d \leftarrow \frac{A_d}{weight-of-d}$ from the document info file
4. select the r documents with the highest A_d value.

5 Compression of Files

As it was pointed out in Section 2, OCR collections require certain considerations to compensate for errors. Index size and image display are two prominent factors. The compression of the postings and bounding boxes has to be taken

into consideration for a manageable and efficient system. We mainly rely on unary and γ-code [19] to compress these entries. For the sake of completeness, we define these two codes here.

Example: The unary code for a number x is defined as $(x-1)$ 1-bits followed by a 0-bit. So if x is 9, the unary code is 111111110. The γ-code for x is defined as follows for $x = 9$:

1. Take $\lfloor \log x \rfloor$.
   $\lfloor \log 9 \rfloor = 3$.
2. represent $1 + \lfloor \log x \rfloor$ as a unary code.
   $1 + \lfloor \log x \rfloor = 1 + 3 = 4$ has code 1110.
3. calculate $x - 2^{\lfloor \log x \rfloor}$.
   $9 - 2^3 = 9 - 8 = 1$.
4. Now represent $x - 2^{\lfloor \log x \rfloor}$ as $\lfloor \log x \rfloor$ binary code.
   In our example, $9 - 2^3 = 1$ is represented as 001.
5. So the code for 9 is 1110 001.

To decode example 1110 001, extract the unary code, in this case 1110, which is 4. Treat the next 3 $(4 - 1 = 3)$ bits as a binary code, in this case 001, which is the binary code for 1. The original number is $2^{4-1} + 1 = 8 + 1 = 9$.

We can use γ-code to compress the posting and bounding boxes for each term. As an example, suppose document number 10 has 5 words as shown below:

doc.	no. words	word no.	occurrences of word 50 in doc	x_1	y_1	x_2	y_2	x_3	y_3
10	5	50	3	10	15	302	47	614	312
		75	2	12	18	300	320		
		...	...						

We compress the information for these 5 words as follows:

(γ-code 5)(γ-code 50)(γ-code 3)(unary x_1)(unary y_1)(unary x_2)(unary y_2) (unary x_3)(unary y_3)(γ-code 75)(γ-code 2)(unary x_1)(unary y_1)...

6 Conclusion and Future Work

Statistically-based information retrieval engines are robust enough to deal with typical OCR errors in text. If the collection is poor in quality, then certain documents may be hard to locate and retrieve. Hairetes is designed to address this problem and avoid the extreme variability in the the ranked result set.

Hairetes is currently being implemented and we hope to report on its performance in the near future.

References

[1] Jean Aitchison, Alan Gilchrist, and David Bawden. *Thesaurus Construction and Use : A Practical Manual*. Fitzroy Dearborn, 4th edition, 2000.

[2] Fabio Crestani. Exploiting the similarity of non-matching terms at retrieval time. *Journal of Information Retrieval*, pages 25–45, 2000.
[3] Fabio Crestani and C.J. Van Rijsbergen. A study of kinematics in information retrieval. *ACM Transactions on Information Systems*, 16:225–255, 1998.
[4] Fabio Crestani, Ian Ruthven, M. Sanderson, and C.J. van Rijsbergen. The troubles with using a logical model of ir on a large collection of documents. experimenting retrieval by logical imaging on trec. In *Proceedings of the Fourth Text Retrieval Conference (TREC-4)*, 1995.
[5] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.
[6] William B. Frakes. Stemming algorithms. In William B. Frakes and Ricardo Baeza-Yates, editors, *Information Retrieval: Data Structures and Algorithms*, pages 131–160. Prentice Hall, 1992.
[7] R. E. Gorin, Pace Willisson, Walt Buehring, Geoff Kuenning, et al. Ispell, a free software package for spell checking files. The UNIX community, 1971. version 2.0.02.
[8] Donna K. Harman. Ranking algorithms. In William B. Frakes and Ricardo Baeza-Yates, editors, *Information Retrieval: Data Structures and Algorithms*, pages 363–392. Prentice Hall, 1992.
[9] Donna K. Harman. Relevance feedback and other query modification techniques. In William B. Frakes and Ricardo Baeza-Yates, editors, *Information Retrieval: Data Structures and Algorithms*, pages 241–263. Prentice Hall, 1992.
[10] C. J. Van Rijsbergen. A theoretical basis for the use of co-occurrence data in information retrieval. *Journal of Documentation*, 33(2):106–109, June 1977.
[11] C. J. Van Rijsbergen. A non-classical logic for information retrieval. *The Computer Journal*, 29:481–485, 1986.
[12] Amit Singhal, Gerard Salton, and Chris Buckley. Length normalization in degraded text collections. In *Proc. of SDAIR-96 5th Annual Symposium on Document Analysis and Information Retrieval*, pages 149–162, Las Vegas, NV, 1996.
[13] Kazem Taghva, Julie Borsack, and Allen Condit. Results of applying probabilistic IR to OCR text. In *Proc. 17th Intl. ACM/SIGIR Conf. on Research and Development in Information Retrieval*, pages 202–211, Dublin, Ireland, July 1994.
[14] Kazem Taghva, Julie Borsack, and Allen Condit. Effects of OCR errors on ranking and feedback using the vector space model. *Inf. Proc. and Management*, 32(3):317–327, 1996.
[15] Kazem Taghva, Julie Borsack, and Allen Condit. Evaluation of model-based retrieval effectiveness with OCR text. *ACM Transactions on Information Systems*, 14(1):64–93, January 1996.
[16] Kazem Taghva, Julie Borsack, Allen Condit, and Srinivas Erva. The effects of noisy data on text retrieval. *J. American Soc. for Inf. Sci.*, 45(1):50–58, January 1994.
[17] Kazem Taghva, Thomas A. Nartker, and Julie Borsack. Recognize, categorize, and retrieve. In *Proc. of the Symposium on Document Image Understanding Technology*, pages 227–232, Columbia, MD, April 2001. Laboratory for Language and Media Processing, University of Maryland.
[18] Kazem Taghva and Eric Stofsky. Ocrspell: An interactive spelling correction system for OCR errors in text. *Intl. Journal on Document Analysis and Recognition*, 3(3):125–137, March 2001.
[19] I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes: Compressing and indexing documents and images*. Morgan Kaufmann, 2nd edition, 1999.