Document Image Matching and Retrieval With Multiple Distortion-Invariant Descriptors

Jonathan J. Hull

Ricoh California Research Center
2882 Sand Hill Road
Menlo Park, California
hull@crc.ricoh.com

ABSTRACT

A method for organizing a database of document images is discussed that is based on image data extracted from the text portion of the documents. This allows for the location of matching documents that may have been reformatted or re-imaged so that they appear significantly different. Each document image is represented by a number of descriptors that are invariant to the geometric distortions of translation, rotation, and scaling. Individual descriptors capture information about local features and the overall set of descriptors for a document image provide a redundant description of its content. Experimental results are presented that demonstrate the approach.

1. Introduction

Document image databases contain digital representations of documents that are captured on scanners or digital photocopiers. Queries about the contents of the database are answered by searching for a match between the query and an image in the database. One form of query that is of interest for a document database is given the image of a passage of text. The objective is to determine whether that passage of text exists elsewhere in the database. The constraints are that the database entry may be formatted differently from the query and the appearance of the two images may be significantly different from one another.

Text image matching provides an important capability for a document database. This is useful in a legal office where locating previous revisions of a given document is an important part of tracking its history. Another example of the usefulness of text image matching is in an application where classified materials are copied and digital images of the copies retained. Given a classified document, it is sometimes necessary to determine when the document was copied as well as which other documents were copied at approximately the same time. The context in which a document was copied is also pertinent in an application where only a single page is available from a multi-page document and a user would like to recover the complete original.

An obvious solution to text image matching is to apply optical character recognition (OCR) to a document and use the recognized text to search for matching passages. However, this has the disadvantage that an OCR process must be applied to all the text in the database

documents and the resulting ASCII data stored for subsequent searches. The OCR must also be tolerant to the range of image distortions that occur in practice.

An alternative solution is to perform matching directly on the image data. This bypasses the need for OCR and reduces the associated storage requirement. Any necessary invariance to image distortions are modeled at their source rather than one step removed as they are if OCR data is utilized.

Various solutions have been proposed for matching queries to database entries when both are images. Top-down structural methods have been used in which images are segmented into a number of objects and the identities of the objects are used as an *iconic index* for the image. Several versions of a representation known as a two-dimensional string have been used to express the geometric relationships between the objects [2, 4, 7, 8]. These methods use a string matching algorithm to locate images in a database that match a query.

Hashing has been used to speed up matching in the two-dimensional string approach [3]. In this algorithm both the query and database images are represented by sets of ordered triples. Each triple contains the identity of two objects in the image and the spatial relation (one of nine direction codes) between them. Each unique triple points to the database images that contain it. The database images that match a query are determined by intersecting the lists of images pointed to by each triple in the query. A query is satisfied if the intersection is not empty. While this top-down strategy is useful as a fast adaptation of the two-dimensional string, it is sensitive to errors in segmentation and recognition of the objects in the image. In fact, a single error in either process may cause a query to fail. Unfortunately, it is precisely this sensitivity to noise that must be overcome to guarantee reliable performance.

Bottom-up featural information has been used to overcome some of the disadvantages of a top-down method. In a technique known as *geometric hashing* the *interesting points* extracted from a query image are matched to interesting points extracted from database images [5, 6]. Interesting points are located by an application-specific operator that may, for example, find small areas of high gray level variance. The assumption is that the same points will be located in different versions of the same image even though it may have been distorted by noise.

The interesting points from two images are matched to each other by first using each pair of points from both images to generate normalized versions of both point sets. The normalization process corrects for translation, rotation, and scaling by transforming the chosen pair of points to (0,0) and (1,0) and applying that transformation to the rest of the point set.† The two images are *equivalent* if an acceptable number of points are in one-to-one correspondence after some pair of normalizations.

An algorithm for text image matching is discussed in this paper that combines several aspects of both the top-down and bottom-up methods. The method discussed here is similar to that proposed in [1] that was concerned with the recognition of two-dimensional shapes. Our approach addresses the concerns of a document image retrieval system that contains primarily images of text.

Features calculated from groups of consecutive words are used as descriptors for a passage of text. The identity of the passage from which each descriptor was extracted is stored in a hash table. At run-time, the descriptors extracted from an input image are used to access the hash

† Other distortions can be accounted for by using more than two points in the normalization.

setup time (compile the database)

for each image in the database
1. locate features in the image
2. calculate descriptors from groups of features
3. enter passage identifier and location in a hash table

run time (match a query image to the database)

1. locate features in the input image
2. construct descriptors from groups of features
3. for each descriptor
   increment accumulator cell corresponding to each passage identifier

Figure 1. Algorithm for text image matching.

table and vote for the database entries that contain it. A match occurs if a database document receives an acceptable number of votes. The descriptors are extracted from a number of adjacent words and are invariant to changes in translation, rotation, and scaling. The embedding of the invariance to distortion directly in the descriptors rather than in the matching process results in an algorithm with an O(N) run-time complexity for N words in a query image.

The rest of this paper presents the text image matching algorithm in more detail. A statement of the algorithm is given along with an analysis of its expected performance in the presence of noise. Experimental results are given that illustrate several aspects of the algorithm including a method for choosing less than the maximum number of descriptors for a given document and its effect on matching performance. Experiments on an image database illustrate the ability of the algorithm to operate in the presence of noise.

2. Algorithm Description

The algorithm for text image matching is presented in Figure 1. The algorithm contains two basic steps. A setup time phase is used in which a hash table is constructed that represents a database of documents. This is done by locating the regions in each image that contain text. The individual words in those regions are isolated and the left-to-right reading order of the words is determined. A feature description is calculated from each word and *descriptors* are computed from groups of adjacent words. Each descriptor is then input to a hashing function that returns an address in a hash table. Each location in the hash table contains a linked list of

identifiers for text passages. Each passage identifier also contains the location of the descriptor in the original image.

At run time an input or *query* image is matched to the database and a list of images are returned that are close matches to the input. This is done by first locating features and constructing descriptors with the same algorithms that were used to construct the hash table. Each descriptor from a query document is then input to the hash function and the passage identifiers stored at the corresponding location are returned. The accumulator cell that corresponds to each of these passage identifiers is then incremented. After processing an entire query image, the accumulator measures the number of descriptors that are in common between the query and each text passage in the database.

A diagram of the operation of the algorithm is presented in Figure 2. The example data shows four descriptors that were extracted from four groups of five consecutive words. Three of the descriptors hash to addresses 1, 4, and 7. The fourth descriptor, which is different from the other three, also hashes to address 4. This results in a *collision.* The accumulator values demonstrate that document four received four votes, one from each descriptor. The next best match is document seven that received two votes. In this case, a decision could be output that document four is equivalent to the query.

2.1. Descriptors

The descriptors provide a distortion-invariant representation for local features in an image. The objective is to make each descriptor an independent source of information. Then by using a number of descriptors for each image, the inter-descriptor redundancy can compensate for noise that may occur in the feature calculation. Thus, even though a percentage of the descriptors extracted from a document may be incorrect, there should still be more descriptors in common between two equivalent passages than between two different documents.

The use of distortion-invariant descriptors requires that the feature extraction stage be specialized for this requirement. In text images this means that the reading order of the words must be determined. However, this results in a simpler matching algorithm than previous methods that postponed the correction for geometric distortion until after the descriptors had been calculated.

The use of variable-size groups of local features as descriptors for an image provides a method for tuning the amount of contextual information that is utilized. Increasing the number of words used to form each descriptor increases the number of possible descriptors. This improves performance by decreasing the number of descriptors that will occur by chance in a text passage that does not match a query. However, this improved performance comes at the expense of an increased storage requirement for the hash table.

2.2. Hashing System

The hashing system consists of the hash function, the hash table, the linked lists of passage identifiers, and an array of accumulators. The primary concerns in the design of the hashing system are to ensure a high rate of dispersion by the hash function and to minimize the storage required for the data structures. Ideally, the descriptors input to the hash function should result in addresses that are uniformly distributed over the address space of the hash table.

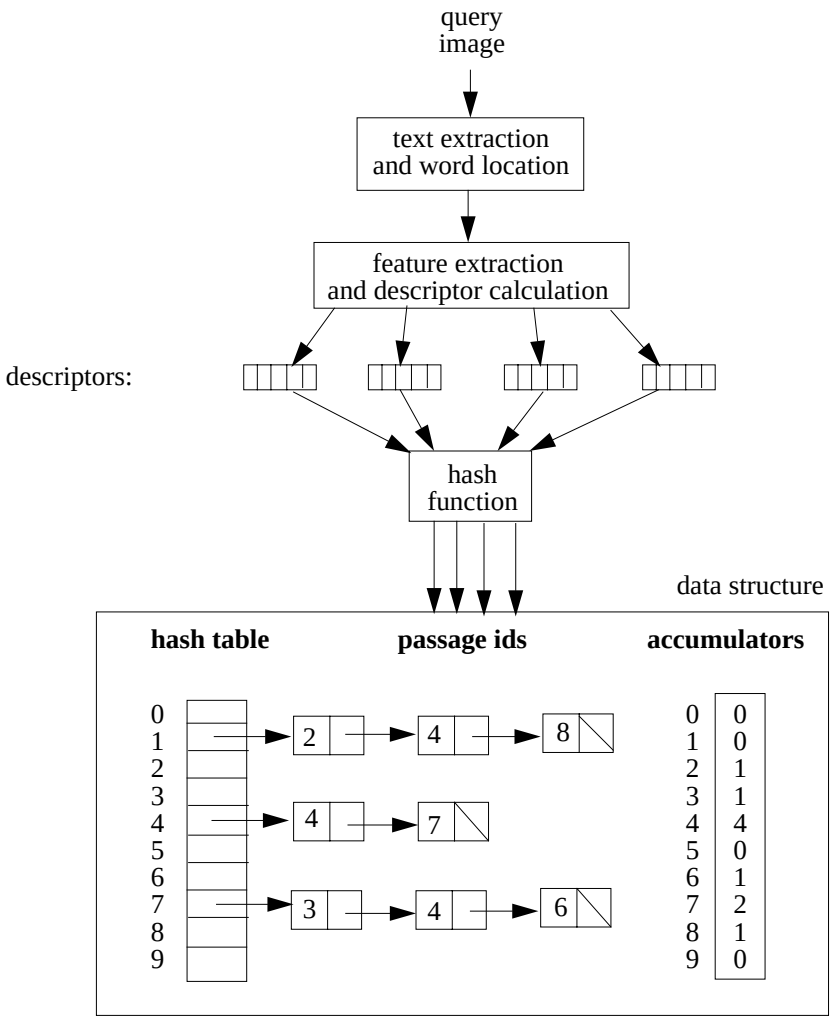


Figure 2. Text matching algorithm.

Also, the storage used for the hash table and the linked lists should be minimized. Reducing the storage overhead to be less than that required for the complete ASCII representation for the document would be an additional advantage for the proposed approach over a text image matching algorithm that used OCR results.

The hash function is given a descriptor and it returns an integer address in the range [0..N-1]. Typically, N is chosen to be a large prime number. Many hash functions have been

proposed in the literature. An example is using a random number generator with the seed of the random number generator equal to the descriptor. The objective in choosing a hash function is to minimize the number of collisions it produces on the data it encounters.

The hash table is implemented as a linear array with N locations. Each location holds a pointer to a passage identifier node. A concern in implementing a hash table is the resolution of collisions. This requires that the descriptor be stored at each location it hashes to. If two different descriptors hash to the same location, then some form of a linked list of descriptors is maintained to differentiate the data that is stored in the table. In the present application, because of the redundancy between descriptors, a collision resolution scheme was deemed unnecessary. Instead, the passage identifiers were stored at whatever location in the hash table the corresponding descriptor hashed to.

The linked lists of passage identifiers are implemented in contiguous memory locations. This is done by first constructing the linked lists using dynamic memory allocation. The data stored in each list is then written into consecutive memory and the location of the first data element in the list is stored in the hash table. Thus, the storage needed for each node is given by the number of bits needed to encode the largest passage identifier. A separate tag bit is used to indicate the end of a linked list.

The array of accumulators requires a number of locations equal to the number of passages that will be represented in the database. Each location needs a number of bits that will represent the most descriptors extracted from any query.

2.3. Reduced Number of Descriptors

Storing fewer than the maximum number of descriptors for each passage is one method of reducing the storage cost. However, this should be done without affecting the matching performance. This requires a method for choosing descriptors that discriminate one passage from another and can be calculated reliably in the presence of noise.

There are several methods for choosing a subset of descriptors for a given passage. One is similar to some techniques for choosing keywords to describe the content of a passage of text [10]. The adaptation of this approach to this application weights each descriptor by the number of database documents that contain it. The descriptors are then sorted by this weight and a given number of descriptors with the minimum weights are chosen.

A potential difficulty with this method for choosing descriptors is that it ignores information that may be available about the reliability with which a given descriptor may be calculated. A solution is to use the probability that descriptor D is correct as the criterion for choosing the descriptors that are retained. This probability can be estimated by the product of the probabilities that each of its components is correct. This is applicable in the present algorithm since each descriptor is composed of feature descriptions from S adjacent words.

3. Analysis

The tolerance of the proposed algorithm to noise in the feature calculation is an important consideration. If the probability of correctly determining the feature description for a word is p, then the probability of correctly determining the feature description for S consecutive words is

p^S. For example, with $p = 0.90$, and $S = 5$, $p^S = 0.59$. Thus, 59 percent of the descriptors extracted from a given passage will be correct under these conditions. All of them will occur in the correct matching passage. Some percentage of these descriptors (called *false positives*) will also occur in other passages in the database.

The other 41 percent of the descriptors (the errors) will be distributed over the other K^S descriptors, assuming a maximum word length of K characters. Some of those errors will be *detectable* because they occur in no other passages in the database. These descriptors will most likely produce an address that is not contained in the hash table when they are provided to the hash function. Other errors will be *undetectable* because they occur elsewhere.

The detectable error descriptors are of no concern since they result in incrementing no counters. Error descriptors that increment counters associated with the correct passage are usually not detectable and are of no concern. However, undetectable errors as well as false positives that increment counters for other passages could lead to errors in the final decision or an inability to reach a decision if the errors are too numerous.

The probability that a feature description is determined correctly (p) is influenced by the quality of the original text image. The objective in constructing software for this task is to obtain a value for p that is as high as possible.

The choice of S provides the most direct control over the number of detectable and undetectable errors as well as the number of false positives that are generated. Increasing S increases the number of detectable errors since an error is more likely to result in a descriptor that does not exist in a text database. This is because of the exponential increase in the number of possible descriptors that occurs as S is increased. The number of undetectable errors and false positives are decreased for the same reason. However, increasing S increases the overall number of descriptors that are generated for a database of text passages and increases the storage needed for the lists of passage identifiers.

Thus, a value for S should be chosen such that $p^s \times num\_descriptors\_per\_passage$ is significantly greater than the most undetectable errors and false positives that will occur in any passage. The storage cost incurred for increases in S can be mediated by retaining a fixed number of descriptors per passage instead of all possible descriptors. This will provide a predictable linear increase in storage as passages are added to the database. Also, there is a potential to improve performance if the descriptors removed from the hash table are those that result in undetectable errors or false positives.

4. Example

An example is now presented that demonstrates various aspects of the text image matching algorithm. The two passages of text used in this case are sentences from separate pages. The page images are samples A001 and A003 from the *UW English Document Image Database I* [9]. The ASCII for those sentences are shown in Figure 3(a).

The number of characters in each word were used as their feature descriptions. The descriptors were calculated from each group of five consecutive words in the two passages. A maximum word length of 15 characters was supported. This allows each feature to be represented in 4 bits and each descriptor D to be represented as a 20-bit integer. The hash function is directly applied to this number. In this example 23 locations were used in the hash table

and the hash function was $D \bmod 23$. The descriptors for the two ASCII passages and the hash table address of each descriptor are shown in Figure 3(b). Figure 3(c) shows the hash table that results from this data. Of the 45 descriptors that were input, 20 of the 23 hash table addresses were used and there were nine collisions that resulted in more than one passage identifier being stored in a single hash table bin.

Figure 4(a) shows the image data from passage 0 that was used to test the example data structures. The word images in this passage were manually extracted and saved in individual files. The number of characters in each word were estimated by a procedure that counted the

---

```
    The Preisach Model for ferromagnets is generalized and adapted for the
description of the hysteretic behaviour of a polycrystalline specimen of shape-
                                memory alloys.

                    We hope that this work is a step toward
understanding how people generate language as rapidly as they do and building practical
        language-generation systems that achieve similarly impressive speeds.
```

(a)

```
3-8-5-3-12=7    8-5-3-12-2=1    5-3-12-2-11=8   3-12-2-11-3=20  12-2-11-3-7=8
2-11-3-7-3=21   11-3-7-3-3=21   3-7-3-3-11=22   7-3-3-11-2=12   3-3-11-2-3=0
3-11-2-3-10=5   11-2-3-10-9=15  2-3-10-9-2=1    3-10-9-2-1=19   10-9-2-1-15=21
9-2-1-15-8=8    2-1-15-8-2=5    1-15-8-2-5=15   15-8-2-5-6=6    8-2-5-6-6=6

2-4-4-4-4=7     4-4-4-4-2=14    4-4-4-2-1=1     4-4-2-1-4=20    4-2-1-4-6=9
2-1-4-6-13=21   1-4-6-13-3=21   4-6-13-3-6=15   6-13-3-6-8=8    13-3-6-8-8=6
3-6-8-8-2=19    6-8-8-2-7=18    8-8-2-7-2=13    8-2-7-2-4=9     2-7-2-4-2=8
7-2-4-2-3=13    2-4-2-3-8=12    4-2-3-8-9=4     2-3-8-9-15=20   3-8-9-15-7=21
8-9-15-7-4=17   9-15-7-4-7=19   15-7-4-7-9=4    7-4-7-9-10=16   4-7-9-10-6=10
```

(b)

```
hash[0] = 0     hash[7]  = 0:1  hash[13] = 1     hash[18] = 1
hash[1] = 0:1   hash[8]  = 0:1  hash[14] = 1     hash[19] = 0:1
hash[4] = 1     hash[9]  = 1    hash[15] = 0:1   hash[20] = 0:1
hash[5] = 0     hash[10] = 1    hash[16] = 1     hash[21] = 0:1
hash[6] = 0:1   hash[12] = 0:1  hash[17] = 1     hash[22] = 0
```

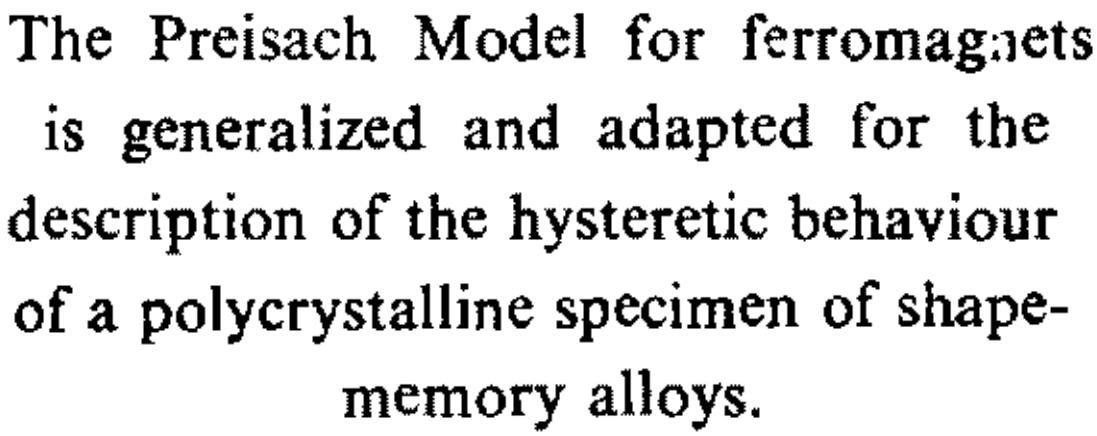
(c)

Figure 3. Two passages of text (a), the descriptors extracted from each passage (b), and the hash table as well as lists of passage identifiers (c). The five word-lengths in each descriptor are shown separated by hyphens followed by their hash address after the equals sign.

---

number of connected components. The descriptors that were calculated by this procedure are shown in Figure 4(b) along with their hash table addresses. Those addresses were used to access the hash table and tally votes in the accumulator. The resultant accumulator values are shown in Figure 4(c).

There was one error in feature calculation that mis-estimated the number of letters in `ferromagnets` as 15 instead of 12 because of several broken characters. This resulted in five incorrect descriptors. When the hash table was accessed with this data, two of the addresses were empty and thus the errors were *detectable.* The other three errors were *undetectable.*



(a)

```
3-8-5-3-15=NULL   8-5-3-15-2=7     5-3-15-2-11=17   3-15-2-11-3=22   15-2-11-3-7=NULL
2-11-3-7-3=21     11-3-7-3-3=21    3-7-3-3-11=22    7-3-3-11-2=12    3-3-11-2-3=0
3-11-2-3-10=5     11-2-3-10-9=15   2-3-10-9-2=1     3-10-9-2-1=19    10-9-2-1-15=21
9-2-1-15-8=8      2-1-15-8-2=5     1-15-8-2-5=15    15-8-2-5-6=6     8-2-5-6-6=6
```

(b)

```
acc[0] = 17
acc[1] = 13
```

(c)

Figure 4. Image of the example test passage (a), descriptors extracted from the image (b), and the accumulator values (c).

The accumulator values shown in Figure 5(c) indicate 17 votes for passage 0 and 13 votes for passage 1. This resulted from five hash addresses that contained pointers only to passage 0 and 13 addresses that pointed either to passage one or to both passages (i.e., *false positives*). In this case, a decision in favor of passage 0 would have been supported.

Reducing the number of descriptors stored in the hash table by retaining those that point only to one passage would have left 11 filled hash table entries (three that point to passage 0 and eight that point to passage 1). This would have resulted in five votes for passage 0 and one vote for passage 1 when the recognized descriptors were matched against the hash table. Thus, the matching performance would have been improved since only one false positive would have occurred. Also the storage needed for the passage identifiers would have been reduced by 62 percent because 18 out of the 29 linked list entries would have been removed.

5. Experimental Results

Experiments were conducted to investigate several aspects of the text image matching algorithm. Three potential implementation scenarios were investigated. One was an application where both the query and the database documents were not corrupted by noise. This is pertinent if clean ASCII data is to be matched versus other clean ASCII data and would be useful if applied to a large collection of computer files. Another scenario is matching corrupted data to clean ASCII text. This is useful in an application where an image of a document is input and the original ASCII data from which that document was generated is available. The third scenario is when both the query and the database are corrupted. This would be useful when both the queries and the database are generated by two independent image processing algorithms.

The number of descriptors extracted from an input document that were matched against the hash table were varied. This was done to demonstrate that reliable performance can be obtained with descriptors that are generated from a limited portion of a document. This is an important consideration since the runtime for the matching operation is expected to be largely determined by the image processing operations needed to estimate the number of characters in a word.

An additional set of experiments investigated the usefulness of the frequency-weighted and probabilistic methods for choosing subsets of descriptors for entry into the hash table. These experiments matched corrupted queries to a database generated from corrupted text. Reliable performance in these experiments is important since the storage needed for the hash table is determined mostly by the number of descriptors stored for each passage.

5.1. Database

A database of page images was used to generate queries that were matched against a text database. The image database was collected from 17 articles in 14 different journals. Appendix 1 contains a list of the articles in the database. There were on average seven pages in each article. This provided 115 page images that were scanned at 300 ppi in binary mode.

The words on each page were located automatically and the identity of each word was manually entered by a human operator. Errors in word location from the image were not corrected. Thus, some images may contain only a portion of a word or more than one word. This was done to provide test data that reflected the performance of a document segmentation algorithm. Overall, 65,565 individual word images were used for testing. On average there were 570 words on each page. The original ASCII documents contained 64,475 words. Thus, the word location algorithm had approximately a 1.7 percent error rate.

The text database was constructed from the ASCII for the 115 page images plus the ASCII for 882 document pages in the University of Washington (UW) English Document Image Database [9]. This included pages that contained at least one zone that was classified as ‘‘body text.’’ The ASCII data was pre-processed to eliminate line breaks. Words split across lines were joined and the intermediate hyphen was removed. This was done to approximate the appearance of the ASCII data before it was formatted.

5.2. Descriptors

The descriptors were calculated by appending the number of characters in four, five, and six adjacent words. The number of characters in each word image was estimated by counting the number of connected components that it contained. Small components that contained fewer than a fixed number of pixels that occurred above and at the and of words were not considered. This removed most of the common punctuation marks such as periods, commas, and so on.

The results of word length calculation are shown in Table 1. The number of words in the ASCII text for each passage and the number of words segmented from the image data are shown as well as the percentage of the words in the image data whose length was estimated correctly. The article numbers refer to the database entries listed in Appendix 1.

The number of characters in an ASCII word were estimated by stripping all the punctuation (except for the characters (,),&,#,+,<,>,{,},[,]) and counting the number of contiguous characters that remained. This models the performance of the word length estimation algorithm presented above.

The descriptors used to generate the corrupted version of the database were computed by a noise model that simulated imperfections in the word length calculation. The sequence of word lengths from a passage of text were input to a uniform random number generator that chose a fixed percentage of them (for example, ten percent). Those word lengths were then corrupted by adding a random deviate from a N(0,1) distribution. The descriptors for a passage of text were then computed from this data stream.

5.3. Three Scenarios

The three implementation scenarios mentioned earlier were designated clean-clean, noisy-clean, and noisy-noisy. Clean-clean corresponds to matching descriptors from the original ASCII document to the same descriptors stored in the hash table. It is expected that all the input descriptors will be located in the correct document. It is of interest to determine how many descriptors will occur in the next most frequently voted-for document since this indicates the reliability of a threshold.

perf. metric	article 1	2	3	4	5	6	7	8	9
words in ASCII	3461	4403	4254	4286	2778	3421	3069	3097	6658
words seg.	3688	4412	4386	4280	2809	3500	3155	3182	6791
%corr. word length	89%	98%	98%	95%	93%	90%	97%	94%	93%
perf. metric	article 10	11	12	13	14	15	16	17	
words in ASCII	4718	1066	4773	3404	5500	4236	2372	2979	
words seg.	4839	1050	4716	3405	5532	4312	2450	3058	
%corr. word length	95%	93%	92%	96%	91%	93%	93%	96%	

Table 1. Number of words in the preprocessed ASCII truth, number of words segmented from each article, and performance of word length estimation.

The noisy-clean case corresponds to matching descriptors computed from the test images to clean descriptors extracted from the ASCII documents. It is expected that the number of descriptors that match the correct document will be decreased by a fixed percentage proportional to the accuracy of the length calculation raised to the power S (i.e., p^S) where S is the number of word lengths in each descriptor. The number of descriptors that match the next-best document are expected to remain about the same as in the clean-clean case. This is because the input descriptors that do not match should be randomly distributed across the other documents in the database.

The noisy-noisy case corresponds to matching descriptors from the test images to descriptors from the ASCII documents that had been corrupted by the noise simulation described above. The number of descriptors from the input document that match the correct database entry is expected to be $p^S \times p^S$ since the the input and the database entries are generated by independent sources. The closeness of this value to the number of votes received by the next best choice is of concern.

The hash tables (clean and noisy) needed to perform the experiments were generated from the clean and noisy descriptors for the 997 ASCII database documents. In these experiments 100,003 locations were used in the hash table and collisions were not resolved. That is, no distinction was maintained between two different descriptors that hashed to the same location. Altogether, six different hash tables were generated using three sequence lengths (4,5, and 6) and two conditions on the data (clean and noisy).

The number of descriptors from the input documents that were matched to the hash tables was varied from all the descriptors (570 on average) to 50, 25 and 10 descriptors. When fewer than all the descriptors were used, the first N (N=50,25,10) that occurred were used to access the database. In a real application the rest of the document would not be processed. Thus, the performance achieved will reflect anywhere from a ten times to a fifty times speedup in image processing cost.

Performance was measured by counting the number of descriptors from the input documents that occurred in each database document. The five database entries that received the most votes were determined as well as whether each one was the same as the input document. The number of votes received by a document is expressed as the percentage of input descriptors that it contained. Thus, an input document that is a subset of a much larger document will result in 100 percent of its sequences being located (assuming there is no noise).

The experimental results are shown in Table 2. The percentages shown are averaged across the 115 test documents. The average percentage of instances where the document that received the most votes is correct is shown as well as the average percentage correct rate in the top five choices. The average percentage of input descriptors are shown that occurred in the two documents that received the most votes.

seq. length	condition	test data: all seqs. %cor	%cor5	avg top	avg 2nd	50 seqs. %cor	%cor5	avg top	avg 2nd
4		100%	100%	100%	19%	100%	100%	100%	25%
5	clean-clean	100%	100%	100%	5%	100%	100%	100%	11%
6		100%	100%	100%	3%	100%	100%	100%	7%
4		100%	100%	68%	19%	100%	100%	69%	26%
5	noisy-clean	100%	100%	60%	5%	100%	100%	62%	11%
6		100%	100%	55%	3%	100%	100%	56%	7%
4		100%	100%	48%	18%	90%	97%	43%	25%
5	noisy-noisy	100%	100%	37%	5%	96%	97%	32%	10%
6		100%	100%	30%	3%	96%	97%	25%	6%
		test data: 25 seqs. %cor	%cor5	avg top	avg 2nd	10 seqs. %cor	%cor5	avg top	avg 2nd
4		100%	100%	100%	31%	100%	100%	100%	46%
5	clean-clean	100%	100%	100%	15%	100%	100%	100%	26%
6		100%	100%	100%	10%	100%	100%	100%	17%
4		93%	99%	71%	32%	75%	84%	76%	46%
5	noisy-clean	98%	99%	64%	15%	81%	88%	66%	25%
6		99%	99%	58%	10%	77%	83%	60%	17%
4		86%	90%	53%	30%	56%	67%	57%	45%
5	noisy-noisy	91%	95%	41%	14%	68%	73%	40%	24%
6		88%	91%	33%	10%	63%	70%	31%	17%

Table 2. Performance averaged over 115 query documents.

The results show that perfect performance is obtained in every case where all the sequences in the test document were matched against the database. That is, the top choice is correct in every case. The number of descriptors from the input that match the correct document behaves almost exactly as predicted assuming that the word length calculation was about 90 percent correct on average. Also, the difference between the percent of votes received by the first and second best choices was dramatic enough to make a fixed threshold adequate.

When fewer than all the input descriptors are matched against the hash tables, perfect performance (top choice always correct) was obtained in the clean-clean condition. This is especially encouraging for applications such as content-based addressing of ASCII files since it says a unique ''signature'' may be derived from as few as ten sequential words. Of course, the number of words needed to derive such a signature increases as the number of documents in the database increases.

The results obtained when fewer than all the input descriptors are used and those descriptors are corrupted show that reasonable performance (top five choices about 90 percent correct or better) is obtained with as few as 25 descriptors. This is significant for an application in which the user can manually choose the correct document from among a limited number of choices.

5.4. Choosing Subsets of Descriptors

Storing fewer than the maximum number of descriptors from a passage in the hash table is a way to reduce the storage cost for the data structure. The two methods proposed earlier chose the descriptors for a passage based on their frequency of occurrence in the rest of the database or based on their expected probability of correctness. These techniques are referred to as the frequency weighting algorithm and the probabilistic choice method.

Both algorithms were implemented and their effect on performance as well as storage cost was determined. The data from the noisy-noisy case were used to test the techniques since it represents the most ambitious application. The best 100, 50, and 25 descriptors were chosen for each database document when sequence lengths of 4, 5, and 6 word lengths were used to form the descriptors.

Performance was measured as before by calculating the average percentages of correct, correct in the top five, and percentages of votes received by the first and second closest matching documents. The storage cost was calculated by measuring the number of bytes needed for the passage lists. The storage for the hash tables is constant for a fixed address space. The storage needed for the passage lists is directly affected by the use of fewer than the maximum number of descriptors.

Table 3 illustrates the performance of the two methods for choosing subsets of descriptors. It is seen that perfect performance (100% correct top choice) is obtained when 100 five-word and six-word descriptors are used. The usefulness of the probabilistic method is illustrated when 50 and 25 descriptors are used. The probabilistic technique has an equal or higher correct rate than the frequency weighting algorithm.

The reduction in storage achieved by both methods is dramatic. By way of comparison, when all the descriptors are stored for a every document, 628KB are needed for the four-word sequences, 748KB for the five-word sequences, and 784KB for the six-word sequences. The

parameters		subset choice algorithm									
		freq. weight					probabilistic				
no. seqs. stored	seq. len	%cor	%cor5	avg top	avg 2nd	stor KB	%cor	%cor5	avg top	avg 2nd	stor KB
100	4	83%	91%	9%	4%	117	98%	98%	14%	5%	101
	5	99%	100%	8%	1%	147	100%	100%	10%	2%	120
	6	100%	100%	7%	1%	148	100%	100%	8%	1%	126
50	4	62%	83%	4%	2%	73	87%	92%	7%	3%	55
	5	98%	98%	4%	1%	84	98%	99%	5%	1%	64
	6	99%	100%	4%	1%	83	99%	100%	4%	1%	67
25	4	46%	67%	2%	1%	42	57%	78%	3%	2%	29
	5	89%	96%	2%	1%	44	91%	97%	2%	1%	33
	6	94%	95%	2%	1%	44	96%	99%	2%	1%	35

Table 3. Performance of frequency weighting and probabilistic method for choosing subsets of descriptors for a document.

use of 100 sequences provides an 84% reduction in storage cost (784KB to 126KB) for six-word sequences with no loss in matching performance.

5.5. Run time

An additional experiment was performed to measure the run time of the retrieval algorithm. A hash table was constructed from the 997 images used for the earlier experiments plus the ASCII from 10,000 ‘‘pages’’ extracted from the Wall Street Journal (WSJ). The WSJ pages were constructed by appending 500 consecutive words from an online collection of ASCII text. Each WSJ page on average contained the same number of words as the 997 pages. The descriptors were constructed from the number of characters in six adjacent words. The word lengths were corrupted with the same noise model that was described previously.

Hash tables were constructed from all the descriptors in every page as well as from the 100 descriptors from each page that had the highest a-priori probability of being calculated correctly. When all the descriptors were used, the hash table required 25 MB of storage and when 100 descriptors were stored for each document about 8MB were needed.

The same 115 test images used earlier were matched against the hash tables. The number of descriptors from each query page were varied and the correct rate as well as the run time for each page were measured.

The results of these experiments are shown in Tables 4(a) and 4(b). When 50 descriptors from each input page were used to access the hash table calculated from all the descriptors, 90 percent of the test images were located correctly in the top choice and 94 percent were located correctly in the top five choices. This improved to 100 percent correct in the top choice when 100 or more descriptors were used. Only 12 milliseconds were needed to compute the matches for each page. When all the descriptors from each test image were matched against the hash table constructed from 100 descriptors chosen from each database image, 100 percent of the matches were correct and 12 milliseconds were needed for each match. It should be noted that this speed was calculated on a Sun Sparcstation 2 and includes neither input/output nor image processing costs.

no. descs. per query page	correct rate top 1 / top 5	time per query page
50	90% / 94%	12 ms
100	100% / 100%	23 ms
all (~500)	100% / 100%	102 ms

(a).

no. descs. per query page	correct rate top 1 / top 5	time per query page
all (~500)	100% / 100%	12 ms

(b)

Table 4. Performance on 10,987 document image database, (a) all the descriptors in the hash table, and (b) 100 descriptors from each document in the hash table.

6. Discussion and Conclusions

An algorithm for image database organization and retrieval was proposed in which the queries are images and the objective is to retrieve instances of those images. The query images may be subsets of database entries and both the queries and the database entries may be corrupted by noise.

The proposed solution describes each image (both query and database) as a set of descriptors. Each descriptor contains features that occur nearby one another in the image. The descriptors are themselves invariant to various geometric distortions and provide a redundant description of an image. An input image is matched to the database by counting the number of descriptors that are in common between the input and each database entry. A hash table data structure implements this operation efficiently.

An adaptation of this strategy for images of text was described in which the features are the lengths (number of characters) of words and the descriptors are composed of the lengths of given numbers of consecutive words. This strategy was applied to a database of approximately 1000 document images. Experimental results showed that reliable performance was achieved when various numbers of descriptors were extracted from both the input and the database documents.

7. Acknowledgments

Dr. Peter Hart contributed valuable advice during the development of this work.

References

1. A. Califano and R. Mohan, ''Multidimensional indexing for recognizing visual shapes'', *IEEE Transactions on Pattern Analysis and Machine Intelligence 16*, 4 (April, 1994), 373-392.
2. S. K. Chang, Q. Y. Shi and C. W. Yan, ''Iconic indexing by 2-D Strings'', *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-9*, 3 (May, 1987), 413-428.
3. C. C. Chang and S. Y. Lee, ''Retrieval of similar pictures on pictorial databases'', *Pattern Recognition 24*, 7 (1991), 675-680.
4. G. Costagliola, G. Tortora and T. Arndt, ''A unifying approach to iconic indexing for 2-D and 3-D scenes'', *IEEE Transactions on Knowledge and Data Engineering 4*, 3 (June, 1992), 205-222.
5. D. M. Gavrila and F. C. A. Groen, ''3D Object recognition from 2D images using geometric hashing'', *Pattern Recognition Letters 13* (April, 1992), 263-278.
6. Y. Lamdan and H. J. Wolfson, ''Geometric hashing: A general and efficient model-based recognition scheme'', *Second International Conference on Computer Vision*, 1988, 238-249.
7. S. Y. Lee and F. J. Hsu, ''2D C-string: A new spatial knowledge representation for image database systems'', *Pattern Recognition 23*, 10 (1990), 1077-1087.
8. S. Y. Lee and F. J. Hsu, ''Spatial reasoning and similarity retrieval of images using 2-D C-string knowledge representation'', *Pattern Recognition 25*, 3 (1992), 305-318.
9. I. T. Phillips, S. Chen and R. M. Haralick, ''CD-ROM document database standard'', *Proceedings of the Second International Conference on Document Analysis and Recognition*, Tsukuba Science City, Japan, October 20-22, 1993, 478-483.
10. G. Salton, *Automatic text processing*, Addison Wesley, 1988.

Appendix 1: Document Image Database

no.	Publication	Article Title	pages scanned	words seg.	split words	merged words
1	BioScience v.40 no.10 nov. 1990 pp.738-742	Humoral immunity in insects	5	3688	228	0
2	Business and Society Review winter 1991 pp.25-32	Industrial espionage - what you don't know can hurt you	8	4412	13	4
3	Communications of the ACM v.33 no.3 March 1990 pp.281-287	Scaling up - a research agenda for software engineering	7	4386	160	26
4	CVGIP-Graphical Models and Image Processing v.53 no.6 Nov. 1991 pp. 522-528	Winding and euler numbers for 2D and 3D digital images	7	4280	87	86
5	CVGIP-Image Understanding v.53 no.1 Jan.1991 pp. 1-7	Rigid body motion from range image sequences	6	2809	82	46
6	Geotectonics v.25 no.5 1991 pp.411-415	Principles of faulting in the crust	5	3500	98	14
7	IEEE Computer Graphics and Applications March 1993 pp.28-33	Landscape visualization with Emaps	6	3155	105	9
8	IEEE Computer March 1991 pp. 45-55	Neuronet A distributed real-time system for monitoring neurophysiologic function in the medical environment	11	3182	133	18
9	IEEE Expert April 1993 pp.13-18	Using statistical methods to improve knowledge-based news categorization	6	6791	270	59
10	IEEE Expert April 1993 pp.25-34	Generating, integrating, and activating thesauri for concept-based document retrieval	10	4839	209	37
11	Pattern Recognition Letters v.7 1988 pp.9-12	A heuristic noise reduction algorithm applied to handwritten numeric characters	4	1050	28	32
12	Pattern Recognition v.26 no.3 1993 pp.419-429	Normalizing and restoring on-line handwriting	11	4716	51	92
13	Proceedings of the IEEE v.78 no.3 March 1990 pp. 512-525	Ionospheric effects on modern electronic systems	6	3405	105	27
14	Proceedings of the IEEE v.79 no.4 April 1991 pp 403-414	Memory systems for highly parallel computers	8	5532	40	7
15	Journal of Spacecraft and Rockets v.29 no.4 July-August 1992 pp.437-443	Mobile transporter concept for extravehicular assembly of future spacecraft	7	4312	88	4
16	TREE v.7 no.12 Dec. 1992 pp. 417-419	Plant senescence	3	2450	85	6
17	Virology v.194 1993 pp.277-281	Purification of the IDIR strain of group B rotavirus and identification of viral structural proteins	5	3058	93	14
	totals		115	65565	1875	481