String Techniques for Duplicate Document Detection[*]

Daniel P. Lopresti
`dpl@research.bell-labs.com`

Bell Laboratories
Lucent Technologies, Inc.
600 Mountain Avenue, Room 2C-552
Murray Hill, NJ 07974

Abstract

Detecting duplicates in document image databases is a problem of growing importance. The task is made difficult by the various degradations suffered by printed documents, and by conflicting notions of what it means to be a "duplicate." To address these issues, this paper describes a framework for clarifying and formalizing the duplicate detection problem. Four distinct models are presented, each with a corresponding algorithm for its solution adapted from the realm of approximate string matching. The robustness of these techniques is demonstrated through a set of experiments using data derived from real-world noise sources.

1 Introduction

As information management and networking technologies continue to proliferate, databases of document images and their associated meta-data are growing rapidly in size and importance. A key problem facing such systems is determining whether duplicates already exist in the database when a new document arrives. This is challenging both because of the various ways a document can become degraded and because of the many possible interpretations of what it means to be a "duplicate."

For example, one document might be a photocopy of another, or a fax. The copies could be visually identical, or one might have additional handwritten notes appended to it. If the original document was generated on-line, a duplicate could contain exactly the same text, only formatted in a different way (changes in font, line spacings and lengths, etc.). A duplicate might possess substantially the same content, but with minor alterations due to editing (*i.e.*, earlier or later versions of the same document). Of course, in any of these cases the scanned image of either or both of the documents may contain significant "noise" due to the way the hardcopy was handled or anomalies in the scanning process. All of these interpretations are reasonable; later a framework is described for clarifying and formalizing them.

Whatever the definition, the process of determining whether one document is a duplicate of another involves two steps:

1. Extracting appropriate information (features) from the incoming document image.

2. Comparing the features against those previously extracted from documents in the database.

What features to use, and how they are compared, are the two primary issues to be resolved. Different choices lead to models which will be appropriate for different applications.

Previous work on detecting duplicates (*e.g.*, [2, 6, 7, 19]) has concentrated mostly on exploring the first step above, turning to more traditional measures when it comes to the second. In this paper, the focus is on the models and algorithms associated with comparing document representations (*i.e.*, the second step), while features are taken to be the uncorrected text output from a commercial OCR package. A framework is given for categorizing and studying different kinds of duplicates, along with algorithms that extend the range of techniques available for searching document image databases. These methods prove to be extremely robust, even in the presence of low OCR accuracies.

The remainder of this paper is organized as follows. Section 2 presents four distinct but related models for the duplicate detection problem motivated in part by the literature for approximate string matching. Each of these is solved optimally using a dynamic programming algorithm, as described in Section 3. Implementation issues are considered in Section 4. Section 5 presents experimental results that demonstrate the robustness of these techniques across models and in the presence of real-world noise.

[*] Presented at the *Symposium on Document Image Understanding Technology*, Annapolis, MD, April 1999.

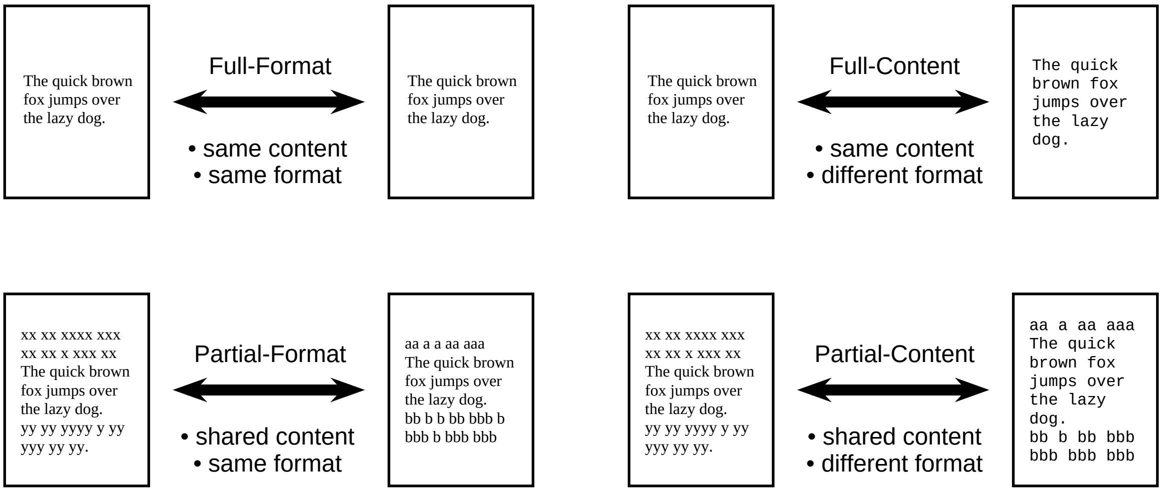
The quick brown fox jumps over the lazy dog.
Full-Format
• same content
• same format
The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.
Full-Content
• same content
• different format
The quick brown fox jumps over the lazy dog.
xx xx xxxx xxx xx xx x xxx xx The quick brown fox jumps over the lazy dog. yy yy yyyy y yy yyy yy yy.
Partial-Format
• shared content
• same format
aa a a aa aaa The quick brown fox jumps over the lazy dog. bb b b bb bbb b bbb b bbb bbb
xx xx xxxx xxx xx xx x xxx xx The quick brown fox jumps over the lazy dog. yy yy yyyy y yy yyy yy yy.
Partial-Content
• shared content
• different format
aa a aa aaa The quick brown fox jumps over the lazy dog. bb b bb bbb bbb bbb bbb

Figure 1: The four duplicate classes discussed in this paper.

Related work is reviewed in Section 6. Finally, conclusions and possible future research directions are discussed in Section 7.

2 Models

For the purposes of this paper, the assumption is that the documents of interest, while in image form, are primarily textual in content. Viewed abstractly, such a page is a series of lines, each consisting of a sequence of symbols. In this *string-of-strings* viewpoint, the term "symbol" can be defined quite liberally. It could be interpreted as meaning characters, of course, but representations at higher levels (*e.g.*, words) or lower levels (*e.g.*, basic features computed from image components) are also possible.

What, then, is a duplicate? Rather than start enumerating possibilities in an ad hoc manner, some structure can be obtained by first partitioning the problem along two dimensions: whether the duplication is full or partial, and whether the layout of text across lines is maintained or not. The reasons for this particular classification scheme are rooted in the string formalisms to be described in the next section. For now, the four possibilities are illustrated with real-world examples and to introduce the terminology:

1. If two documents are visually identical, one is a photocopy or a fax of the other, say, they are *full-format* duplicates. This category also covers documents distributed electronically (*e.g.*, as PDF or PostScript) and printed without further editing.

2. If two documents have identical textual content, but not necessarily the same formatting, they are *full-content* duplicates. This includes, for example, the same e-mail message sent to two people and printed using different-sized fonts, or an HTML document downloaded from the WWW and printed using different margin settings.

3. If two documents share significant content with the same formatting, they are *partial-format* duplicates. Exactly how long the similar regions must be will depend, in general, on the application. Two instances of this are the copy-and-pasting of whole paragraphs from one document into another, and "redacting," the editing of a hardcopy document by obscuring portions of the text so that it is no longer legible.

4. If two documents share content but their formatting is not necessarily the same, they are *partial-content* duplicates. This arises in the copy-and-pasting of individual sentences or groups of sentences. A later version of a document that has undergone several editing passes is likely to be a partial-content duplicate.

These various types of duplication are shown in Figure 1. In the next section, algorithms specialized to each of these cases are given. Note that although the text used to illustrate the figure is "clean," it will be necessary to handle a full range of document recognition errors, include characters that have been misrecognized, omitted, or added, words that have been improperly segmented, complete lines that have been missed or inserted, etc.

Before moving on, it may be instructive to consider briefly the relationships between the various kinds of duplicates. This "universe" is depicted in Figure 2, where several example data-points have also been plotted. Note that there is overlap between the classes, with partial-content duplicates encompassing all the other types.

Clearly, every format duplicate is also a content duplicate; the former is a special case of the latter.

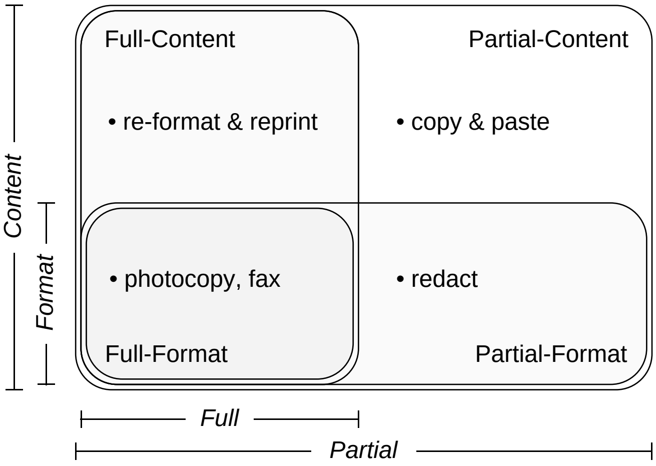

Figure 2: The universe of duplicates.

From a formal standpoint, the distinction is whether the page is treated as a 2-D stream consisting of lines made up of characters, or as a 1-D stream of characters in reading order. Note that the 2-D representation can be converted into a 1-D representation by treating the new line character as a space [19]. This implies that any algorithm for detecting content duplicates can also be used to detect format duplicates. There will undoubtedly be cases, however, where a search can be confined to, say, possible photocopies of a document. Here, an algorithm specialized to finding format duplicates will yield higher precision (*i.e.*, fewer false "hits") than the more general algorithm, which also returns potential content duplicates.

Note also that any full duplicate is also a partial duplicate. Again, there are benefits in maintaining the distinction, both in terms of retrieval precision and because the special case admits heuristics that greatly speed the computation, as is discussed in another paper [11].

3 Basic Algorithms

If it were possible to assume that OCR was perfect or nearly so, the problem of locating duplicates would be relatively straightforward. At best, this is a highly optimistic assumption. Instead, it is safer to acknowledge that OCR can be arbitrarily bad, with no specific guarantee that any n consecutive characters will come through unscathed. If, for example, the accuracy rate were 75% (a reasonable assumption in the case of faxes, small fonts, etc.) and errors are independent, the probability that a given n-gram will survive is 0.24 for $n = 5$, and 0.056 for $n = 10$. The chance that a complete sentence would make it through without errors is miniscule. Hence, schemes that depend on a majority of words or sentences being recognized correctly, while working reasonably well for clean input, may break down in the case of degraded documents.

Fortunately, the literature on approximate string matching is rich with techniques for addressing such concerns [5, 17, 20]. Moreover, the model correlates well with the physical processes that result in errors, so as a measure of similarity it is supported by intuition. Drawing from this body of work, algorithms are given for each of the four variants of duplicate detection. In the context of their respective models, all are guaranteed to return optimal solutions.

Beginning with some definitions, a *string*, $D = d_1 d_2 \dots d_n$, is a finite sequence of symbols chosen from a finite alphabet, $d_i \in \Sigma$. String $S = s_1 s_2 \dots s_m$ is a *substring* of string $D = d_1 d_2 \dots d_n$ if $m \leq n$ and there exists an integer k in the range $[0, m-n]$ such that $s_i = d_{i+k}$ for $i = 1, 2, \dots, m$. The set of all possible substrings of D is denoted D^*. In the 1-D case (*i.e.*, content duplicates), a document is simply a string. In the 2-D case (*i.e.*, format duplicates), a document is a sequence of strings, $D = D^1 D^2 \dots D^m$ where $D^i = d^i_1 d^i_2 \dots d^i_n$.

A standard measure for approximate string matching is provided by *edit distance* [8]. In the simplest case, the following three operations are permitted: (1) delete a symbol, (2) insert a symbol, (3) substitute one symbol for another. Each of these is assigned a cost, c_{del}, c_{ins}, and c_{sub}, and the edit distance is defined as the minimum cost of any sequence of basic operations that transforms one string into the other.

3.1 Full-Content Duplicates

As it relates to full-content duplicates, this optimization problem can be solved using a well-known dynamic programming algorithm [15, 21]. Let $Q = q_1 q_2 \dots q_m$ be the query document, $D = d_1 d_2 \dots d_n$ be the database document, and define $dist1_{i,j}$ to be the distance between the first i characters of Q and the first j characters of D. The initial conditions are:

$$
\begin{array}{lcll}
dist1_{0,0} & = & 0 & \\
dist1_{i,0} & = & dist1_{i-1,0} + c_{del}(q_i) & 1 \leq i \leq m \\
dist1_{0,j} & = & dist1_{0,j-1} + c_{ins}(d_j) & 1 \leq j \leq n
\end{array}
\tag{1}
$$

and the main dynamic programming recurrence is:

$$
dist1_{i,j} = \min \left\{ \begin{array}{lcl}
dist1_{i-1,j} & + & c_{del}(q_i) \\
dist1_{i,j-1} & + & c_{ins}(d_j) \\
dist1_{i-1,j-1} & + & c_{sub}(q_i, d_j)
\end{array} \right.
\tag{2}
$$

for $1 \leq i \leq m$, $1 \leq j \leq n$. The computation builds a matrix of distance values working from the upper left corner ($dist1_{0,0}$) to the lower right ($dist1_{m,n}$), as illustrated in Figure 3. Once it has completed, a sequence of editing decisions that achieves the optimum can be determined via backtracking.

As indicated above, the costs in general can be a function of the symbol(s) in question. As a rule, the deletion and insertion costs are assumed to be greater than 0, while the substitution cost is greater than 0 if the symbols do not match and less than or equal to 0 if they do. In the event constant costs are used, it is convenient to refer to them as simply c_{del}, c_{ins}, and c_{sub} (when the two symbols are different) or c_{mat} (when they are the same). It is possible, and indeed sometimes desirable, to specify cost functions that are quite sophisticated. Moreover, the set of basic editing operations can be supplemented as appropriate. Both of these issues will be covered in a later section.

Algorithm *dist1* provides the basis for a solution to the full-content duplicate problem; the smaller the distance, the more similar the two documents. While OCR errors will raise this value somewhat, to the extent they are modeled by symbol deletions, insertions, and substitutions, they will be accounted for.

3.2 Partial-Content Duplicates

The previous formulation requires the two strings to be aligned in their entirety. For the partial duplicate problem, what is needed is the best match between any two substrings of Q and D. Conceptually, this corresponds to generating all substring pairs in $\{Q^* \times D^*\}$ and then comparing them using algorithm *dist1*. In practice, however, this would be too inefficient.

Fortunately, the original computation can be modified so that shorter regions of similarity can be detected in two longer documents with no increase in time complexity. The edit distance is made 0 along the first row and column of the matrix, so the initial conditions become:

$$
\begin{array}{lcll}
sdist1_{0,0} & = & 0 & \\
sdist1_{i,0} & = & 0 & 1 \leq i \leq m \\
sdist1_{0,j} & = & 0 & 1 \leq j \leq n
\end{array}
\tag{3}
$$

In addition, another term is added to the inner-loop recurrence capping the maximum distance at any cell to be 0. This has the effect of allowing a match to begin at any position between the two strings. The recurrence is:

$$
sdist1_{i,j} = \min \left\{ \begin{array}{lcl}
0 & & \\
sdist1_{i-1,j} & + & c_{del}(q_i) \\
sdist1_{i,j-1} & + & c_{ins}(d_j) \\
sdist1_{i-1,j-1} & + & c_{sub}(q_i, d_j)
\end{array} \right.
\tag{4}
$$

for $1 \leq i \leq m$, $1 \leq j \leq n$. Finally, the resulting distance matrix is searched for its smallest value. This reflects the end-point of the best substring match. The starting point can be found by tracing back the sequence of optimal editing decisions. Note there is an added requirement that the cost when two symbols match be strictly less than zero, otherwise every entry in the matrix will be 0. This computation is illustrated in Figure 4.

Algorithm *sdist1* solves the partial-content duplicate problem by computing

$$
\min\{ dist1(A, B) \mid A \in Q^*,\ B \in D^* \}
$$

In other words, it locates the best-matching regions of similarity between the two documents Q and D. A and B, the two matching subregions, can be recovered if so desired.

3.3 Full-Format Duplicates

For the 2-D models (*i.e.*, format duplicates), another level is added to the optimization. The problem is still one of editing, but at the higher level the basic entities are now strings (lines). At the lower level, as before, they are symbols. Say that $Q = Q^1 Q^2 \ldots Q^k$ and $D = D^1 D^2 \ldots D^l$, where each Q^i and D^j is itself a string. For full-format duplicates, the inner-loop recurrence takes the same general form as the 1-D case:

$$
dist2_{i,j} = \min \left\{ \begin{array}{lcl}
dist2_{i-1,j} & + & C_{del}(Q^i) \\
dist2_{i,j-1} & + & C_{ins}(D^j) \\
dist2_{i-1,j-1} & + & C_{sub}(Q^i, D^j)
\end{array} \right.
\tag{5}
$$

for $1 \leq i \leq k$, $1 \leq j \leq l$, where C_{del}, C_{ins}, and C_{sub} are the costs of deleting, inserting, and substituting whole lines, respectively. The initial conditions are defined analogously to Equation 1.

Since the basic editing operations now involve full strings, it is natural to define the new costs as:

$$
C_{del}(Q^i) \equiv dist1(Q^i, \phi)
$$

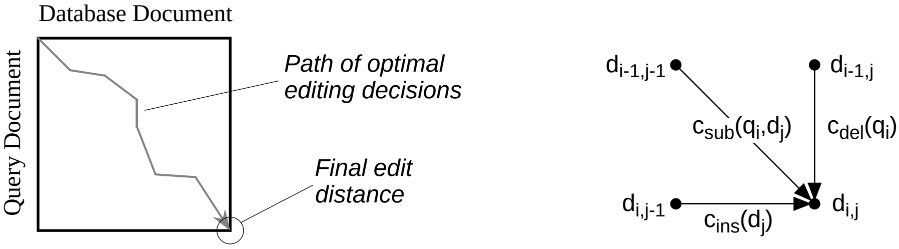

Figure 3: The basic algorithm for string edit distance (*dist1*).

$$\begin{aligned} C_{ins}(D^j) &\equiv \mathit{dist1}(\phi, D^j) \\ C_{sub}(Q^i, D^j) &\equiv \mathit{dist1}(Q^i, D^j) \end{aligned} \tag{6}$$

where ϕ is the null string. Hence, the 2-D computation is defined in terms of the 1-D computation. This is illustrated in Figure 5.

All else being equal, it can be shown that $\mathit{dist2}(Q, D) \geq \mathit{dist1}(Q, D)$ for any two documents Q and D. As noted earlier, *dist1* admits a larger class of duplicates (full-content), while *dist2* may provide higher precision for the class it is intended for (full-format).

3.4 Partial-Format Duplicates

Lastly, the extension for partial-format duplicates combines the modifications for the partial (Equation 4) and format (Equation 5) problems:

$$\mathit{sdist2}_{i,j} = \min \begin{cases} 0 & \\ \mathit{sdist2}_{i-1,j} & + \; C_{del}(Q^i) \\ \mathit{sdist2}_{i,j-1} & + \; C_{ins}(D^j) \\ \mathit{sdist2}_{i-1,j-1} & + \; C_{sub}(Q^i, D^j) \end{cases} \tag{7}$$

for $1 \leq i \leq k$, $1 \leq j \leq l$. Note that C_{del}, C_{ins}, and C_{sub} are defined as before in terms of *dist1* (*i.e.*, Equation 6), not in terms of the 1-D substring computation as might be expected. The granularity of this matching is whole lines. As before, the resulting matrix must be searched for its smallest value, and then traced back to find where the match starts.

At this point four different algorithms have been presented, one for each of the models described in Section 2.

4 Implementation Issues

In this section, a number of issues associated with implementing the algorithms of the previous section are addressed. The inner loops are straightforward to code. Even so, there are numerous degrees of freedom and possible extensions that, while they do not change the underlying algorithm, do alter the nature of the computation in interesting and possibly useful ways.

4.1 Input Alphabet

Generally, string algorithms are viewed as operating on character data. While this provides a natural link to the output from OCR, the algorithms are more general than this and can be used on any representation that obeys a 1-D or 2-D string model. The former views a document as a stream of symbols in reading order, where "symbol" could be any of a variety of features that might be computed from the image including characters, shape codes, word lengths, etc. The latter just adds to this a notion of lines, each a sequence of symbols, again in some reading order. The choice of which set of features to use in a given application will depend on the speed and/or robustness with which it can be computed.

4.2 Cost Assignments

The selection of an algorithm determines the editing model. However, within the context of a single algorithm, the choice of cost functions can have a significant impact. While it is fairly common for implementations of Equations 1-4 to employ constant editing costs, the general way in which the algorithms are formulated is much more powerful than this.

To illustrate, consider the question of white-space errors which are common in OCR. By setting $c_{del}(sp) = c_{ins}(sp) = 0$, in effect not charging for such events, unimportant differences between two OCR'ed versions of the same documents can be ignored. Through an appropriate choice of cost functions, the distinction between various input representations is also eliminated. For example, characters and shape codes will yield identical results if the cost of character substitutions is determined based on shape code classes (*e.g.*, $c_{sub}(q_i, d_j) = 0$ for $q_i, d_j \in \{g, p, q, y\}$, the set of descender characters).

If the distribution of the OCR errors can be estimated *a priori* (*e.g.*, via a confusion matrix), this can be exploited by setting the editing costs to be inversely proportional to the frequencies of the error patterns in question. So, for example, if the substitution $e \rightarrow c$ is ten times more likely to occur than $M \rightarrow W$, its cost is made one tenth as much. This will yield a more sensitive comparison; values closer

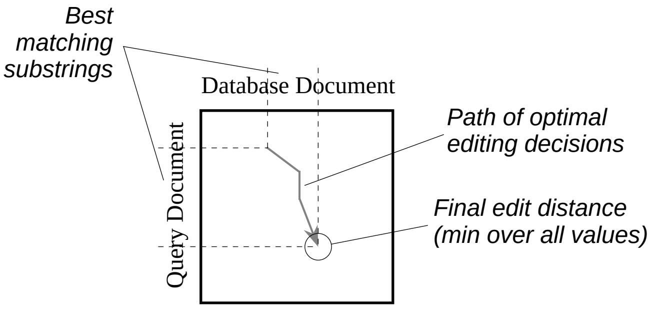
Best matching substrings
Database Document
Query Document
Path of optimal editing decisions
Final edit distance (min over all values)

Figure 4: The substring algorithm for edit distance (*sdist1*).

to the minimum when the documents are indeed duplicates under the model in question (differences due to common OCR errors), and further away from the minimum when they are not (true differences).

4.3 New Editing Operations

While the three basic editing operations (deletion, insertion, and substitution) are sufficient to capture all possible differences between two strings, the set can be supplemented with more sophisticated operations to better model an underlying error process. In the case of OCR, it may be desirable to add "split" and "merge" operations to account for mistakes in symbol segmentation [3]. The recurrence for *dist1*, for example, would then become:

$$dist1_{i,j} = \min \left\{ \begin{array}{lcl} dist1_{i-1,j} & + & c_{del}(q_i) \\ dist1_{i,j-1} & + & c_{ins}(d_j) \\ dist1_{i-1,j-1} & + & c_{sub}(q_i, d_j) \\ dist1_{i-1,j-2} & + & c_{split}(q_i, d_{j-1}d_j) \\ dist1_{i-2,j-1} & + & c_{merge}(q_{i-1}q_i, d_j) \end{array} \right. \tag{8}$$

for $1 \le i \le m,\ 1 \le j \le n$.

Other operations such as transpositions can also be supported. In general, as long as the number of symbols involved (the "look-back") is bounded, the recurrence can be augmented without changing the computational complexity of the algorithm.

4.4 Normalization

For exact duplicates, the distance returned by any of the four algorithms of Section 3 will either be 0 or a negative number that grows smaller as the lengths of the documents increase. For dissimilar documents, the maximum distance grows larger as the lengths increase. It is always the case that, for a given query, a smaller distance corresponds to a better match. In order for the results for different queries to be comparable, however, it is necessary to normalize the distances.

If the target interval is $[0, 1]$, where 0 represents a perfect match and 1 a complete mismatch, then the following formula provides an appropriate mapping:

$$normdist = \frac{dist - mindist}{maxdist - mindist} \tag{9}$$

where *mindist* and *maxdist* are, respectively, the minimum and maximum possible distances for the comparison in question.

Assuming a full-duplicate computation, and making certain reasonable assumptions about the cost functions, the minimum is obtained when all of the characters in the query match the database document and there are no extra, unmatched characters. If the query is $Q = q_1 q_2 \ldots q_m$, then:

$$mindist = \sum_{i=1}^{m} c_{sub}(q_i, q_i) \tag{10}$$

Or, more simply, $mindist = m \cdot c_{mat}$ when the costs are constant.

The maximum distance, on the other hand, is determined by the query and the set of all strings with the same length as the database document. If the cost functions are unconstrained, this in itself becomes an optimization problem. Fortunately, for constant costs there is a simple closed-form solution. Without loss of generality, let the query be the shorter of the two strings (*i.e.*, $m \le n$). There are two possible "worst-case" scenarios: either all of the symbols of the query are substituted and the remaining symbols of the database string are inserted, or all of the query symbols are deleted and the entire database string is inserted. Thus:

$$maxdist = \min \left\{ \begin{array}{l} m \cdot c_{sub} + (n - m) \cdot c_{ins} \\ m \cdot c_{del} + n \cdot c_{ins} \end{array} \right. \tag{11}$$

The partial-duplicate computations are normalized similarly.

4.5 Searching Databases

The algorithms given earlier are phrased in terms of quantifying the similarity between strings (documents). The problem of searching a database for duplicates can be cast in two ways:

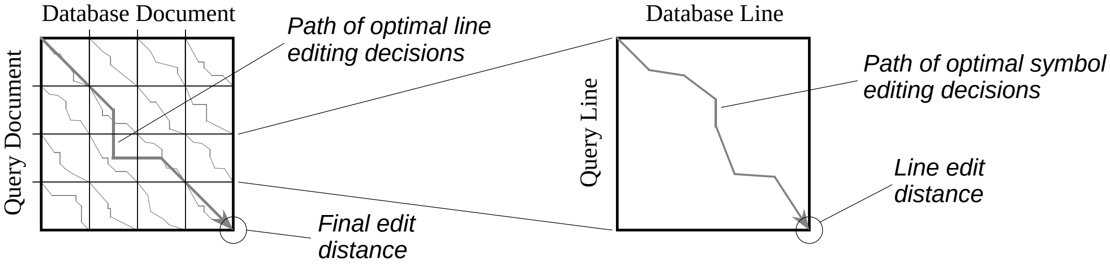

Figure 5: The 2-D algorithm for edit distance (*dist2*).

1. Return the top n matches (in ranked order).
2. Return all documents with distances below a threshold τ.

Note that the first of these requires the computation to complete before any results can be returned to the user. The second can report potential matches as they are encountered (and therefore hide some of the computational latency), but requires setting a threshold in advance. Both policies employ edit distance as a subroutine, and hence can make use of the techniques described to this point.

4.6 Speeding Things Up

Algorithms *dist1*, *sdist1*, *dist2*, and *sdist2* are optimal in the sense they return min-cost solutions to their respective problems. All require time proportional to the product of the lengths of the two documents being compared. In situations where the resulting database search is too slow, there are a variety of ways to speed things up. These include:

- Computing edit distance faster.
- Avoiding having to compute edit distance for every document in the database.
- Computing an approximation to edit distance.

These approaches can, of course, be used in combination.

Asymptotically faster algorithms and parallel VLSI architectures (*e.g.*, [9]) fall in the first category. Database indexing techniques occupy the second. The third is represented by a well-known heuristic based on the observation that, if two strings are similar, the path of optimal editing decisions must remain near the main diagonal (recall Figure 3). Hence, the computation can be restricted to a band close to the diagonal. Should the edit distance fall below some threshold as determined by the width of the band, the heuristic will return its true value, otherwise it returns a value possibly greater than the true distance (as a path other than the optimal has been chosen). This basic concept, illustrated in Figure 6, has been exploited to speed up the computation in the fields of speech recognition [16] and molecular biology [4].

Note that this heuristic applies only in the case of the full-duplicate versions of the problem, as it assumes the optimal editing path starts at $(0,0)$ and ends at (m,n). It can be shown, however, that this approach will never miss a duplicate that would have been returned by the slower, optimal algorithms.

Several new techniques for obtaining substantial speed-ups (up to two orders of magnitude) for which similar proofs-of-correctness can be given are presented elsewhere [11].

5 Experimental Results

To investigate the performance of the algorithms described in this paper, two sets of experiments were designed to explore different aspects of the problem space. The first examined duplicate detection in the presence of several real-world noise sources, while the second studied the four duplicate models and algorithms and how they relate.

For reasons of convenience, the same database was used as in previous retrieval experiments [10, 13]. This consisted of 1,000 professionally written news articles collected from Usenet. The shortest document was 364 characters long, the longest $8,626$, and the average $2,974$. Hence, the total size of the database was approximately 3 megabytes.

The database was used as-is (*i.e.*, no attempt was made to inject OCR errors, either real or synthetic). The query documents, however, and the intended duplicates were all "authentic": pages that had been printed, scanned, and OCR'ed. These documents were formatted in 11-point Times font with a 13-point line spacing using Microsoft Word. Each page was printed on one of two laserprinters, subjected to a noise source in most cases, scanned at 300 dpi using a UMAX Astra 1200S scanner, and then OCR'ed with Caere OmniPage Limited Edition.

For the full-duplicate computations, the edit costs were set to be $c_{del} = c_{ins} = c_{sub} = 1$ and $c_{mat} = 0$. For the partial-duplicate computations, the match cost was $c_{mat} = -1$. The study of more complex

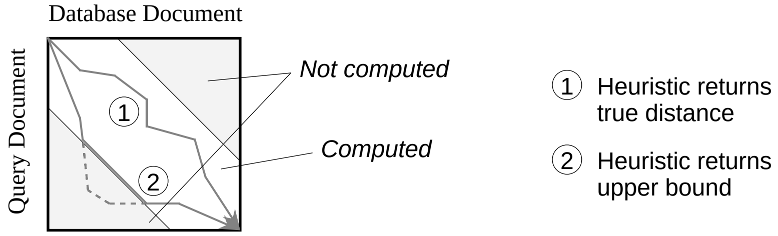
Database Document
Query Document
Not computed
Computed
1
2
1 Heuristic returns true distance
2 Heuristic returns upper bound

Figure 6: A heuristic for string edit distance.

costs assignments (*e.g.*, those based on confusion matrices) is left to a future paper.

5.1 Experiment 1

The goal of this experiment was to study duplicate detection under various noise conditions: copier degradations (multiple generations, excessively light or dark), faxing, and handwritten mark-up (redaction). The source document was 1,395 characters long (26 lines, 203 words). Two sets of six pages were created, one set to be inserted into the database as the intended duplicates, and the other to serve as the queries. The first set was printed on an HP LaserJet 4MPlus laserprinter, the second on an HP LaserJet 4MV. Within each set, one page was used as-is and the others were subjected to one of five different noise sources:

Faxed The page was faxed in standard mode from a Xerox Telecopier 7020 fax machine to a Xerox 7042.

3rd Generation The page was copied to the third generation on a Xerox 5034 copier.

Light The page was copied on the same copier with the contrast set to the lightest possible setting.

Dark The page was copied with the contrast set to the darkest possible setting.

Annotated Five separate text lines on the page were completely obscured using a thick blue marker pen. Different lines were excised in the query and database documents. Also, "This is important!" was handwritten in the margin.

The pages were then scanned and OCR'ed. In addition, the original ASCII text for the query document was left in the database. Hence, each of six queries was run against a database of 1,000 documents containing seven intended duplicates (six that had been OCR'ed, plus the original).

Table 1 below shows the OCR accuracies. Note that the rates range widely, dropping as low as 73.5%. While the two different versions from the same noise source are usually fairly close, they are by no means identical. As expected, a large variety of OCR errors were encountered. Beyond this, other kinds of degradations arose as well. For example, the standard headers prepended to faxes were transcribed (albeit with numerous mistakes), and the lines that had been crossed-out were completely missing from the annotated pages.

Table 1: OCR accuracies for Experiment 1.

	OCR Accuracy	
Document Type	Database	Query
OCR	96.2%	96.0%
Faxed	77.7%	83.9%
3rd Generation	95.9%	96.1%
Light	86.1%	77.8%
Dark	94.0%	95.3%
Annotated	75.6%	73.5%

Since the query documents and their intended matches have the same format, this is a full-format duplicate detection problem and the *dist2* algorithm is most appropriate. The charts in Figures 7-12 plot, for each query, the normalized edit distance for every document in the database. Note that there is always a clear distinction between true duplicates and everything else. This demonstrates that the technique is robust when faced with the sorts of OCR errors seen in practice.

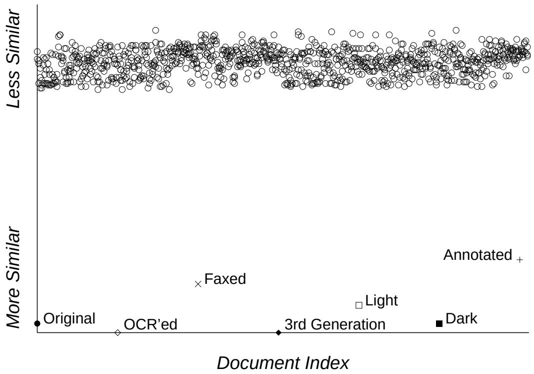

Figure 7: Full-format detection for OCR'ed query.

Studying the data further, it should come as no surprise that the annotated documents yielded the worst-case scenario. Recall that about 20% of the text was completely obscured, a figure that places

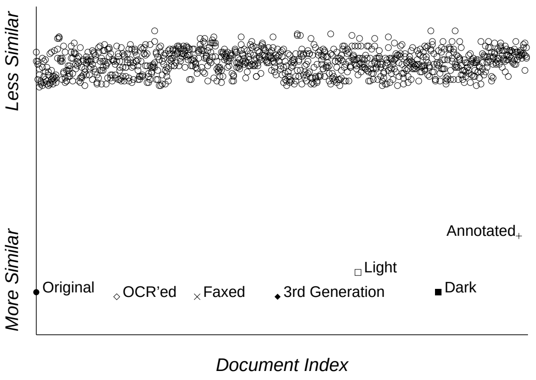

Figure 8: Full-format detection for faxed query.

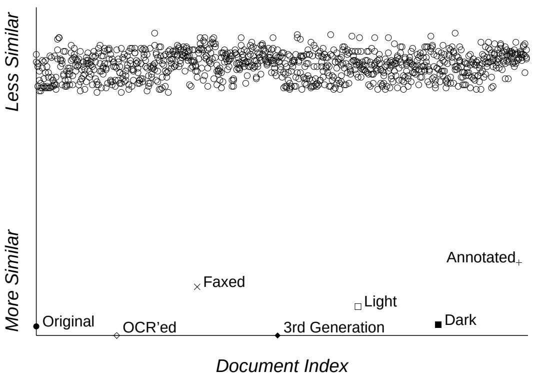

Figure 9: Full-format detection for 3rd generation query.

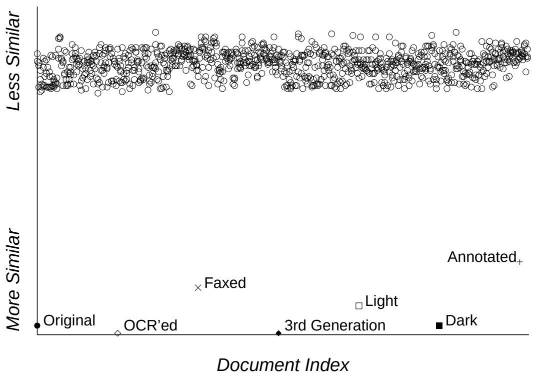

Figure 11: Full-format detection for dark query.

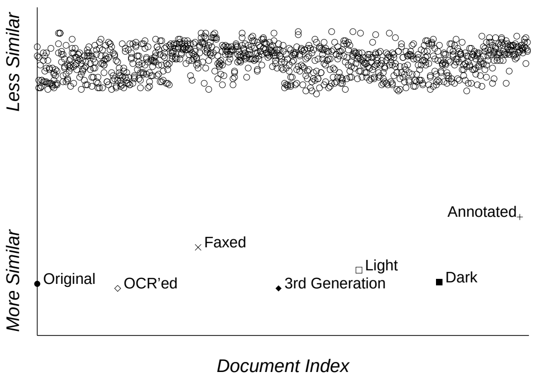

Figure 12: Full-format detection for annotated query.

severe constraints on the performance of any comparison measure. Still, the normalized edit distance in most of the charts is not much greater than this value. When the annotated documents were compared to each other (Figure 12), the amount of text missing between the two amounted to 40%. Even so, and despite all the other OCR errors that must have occurred, it is possible to distinguish the duplicates from non-duplicates.

It is also interesting to note that query and database documents produced using the same noise source are usually a slightly better match (the notable exception being the case of the annotated pages). Whether it is possible to exploit this is a topic for future research.

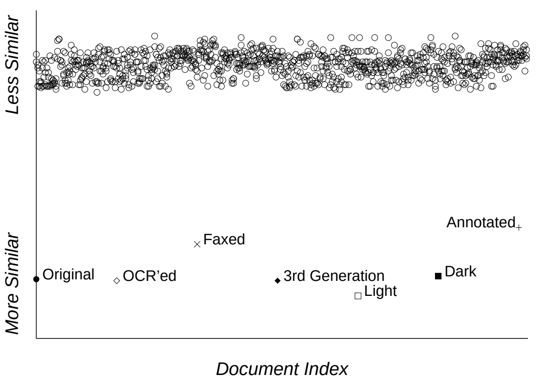

Figure 10: Full-format detection for light query.

5.2 Experiment 2

The purpose of this experiment was to determine how the different duplicate models relate empirically. The four algorithms described in Section 3 were run using the same source document as in the previous experiment. Duplicates were constructed from the query by:

1. Changing the line breaks to create a document that was a full-content duplicate but not a full-format duplicate.

2. Appending roughly equal amounts of unrelated text to the beginning and end of the document to create a partial-format duplicate approximately twice as long as the original.

3. Combining these first two steps to create a partial-content duplicate.

The pages were then printed, scanned, and OCR'ed. The OCR accuracies appear in Table 2. As before, the original source text was left in the database to serve as a second full-format duplicate of the query. Hence, there were between two and five duplicates in the database, depending on the model.

Table 2: OCR accuracies for Experiment 2.

Document Type	OCR Accuracy	
	Database	Query
Full-format	96.0%	95.9%
Full-content	96.1%	n/a
Partial-format	94.9%	n/a
Partial-content	96.0%	n/a

The results for this experiment are shown in Figures 13-16. Since there is a fair amount of residual similarity even in the non-matching cases, the normalized edit distances are lower than for purely random documents. Note that, as expected, algorithm *dist2* works best for full-format duplicates, and *dist1* adds to this full-content duplicates (Figures 13 and 14). The partial-format algorithm *sdist2* can detect full- and partial-format duplicates, while *sdist1* covers all four duplicate classes (Figures 15 and 16).

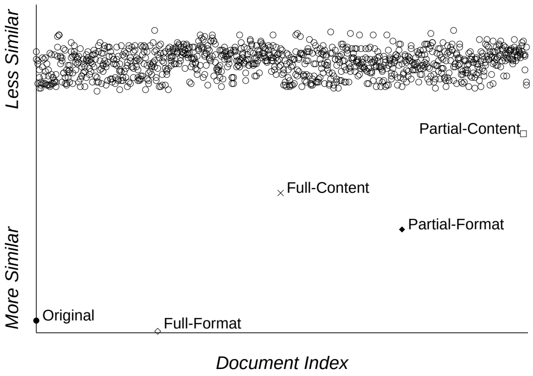

Figure 13: Duplicate detection using *dist2*.

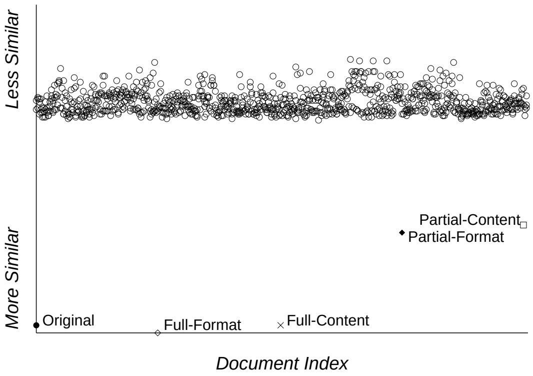

Figure 14: Duplicate detection using *dist1*.

6 Related Work

For the most part, past work on the subject has concentrated on identifying which features to extract (the first step mentioned in Section 1) and not so much on the different ways they might be compared (the second step). The latter is typically handled using one or another of the techniques from the literature.

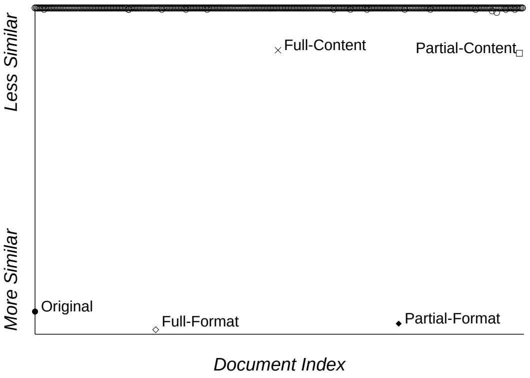

Figure 15: Duplicate detection using *sdist2*.

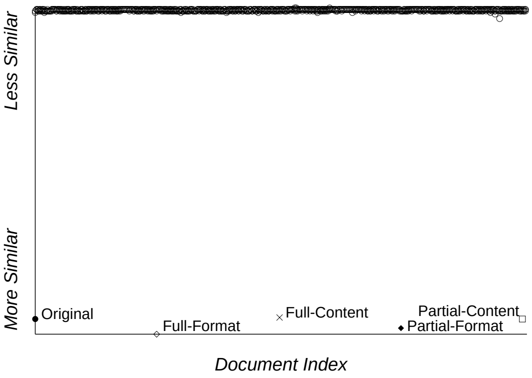

Figure 16: Duplicate detection using *sdist1*.

Spitz, for example, employs character shape codes as features and compares them using the standard string matching algorithm (*i.e.*, Equation 1) [19]. In the taxonomy presented in Section 2, this corresponds to the full-content problem. Doermann, et al., also use shape codes, but extract n-grams for a specific text line to index into a table of document pointers [2]. Since this signature is computed from a single line, it does not explicitly measure the similarity of complete pages. The intention, though, is that this is a method for addressing the full-format problem. Hull, et al., describe three techniques: one based on decomposing the page into a grid and counting connected components within each cell, another using word lengths as a hash key, and one comparing image features (pass codes arising from fax compression) under a Hausdorff distance measure [7]. More details on the last method appear in [6]. The first and third of these fall in the full-format category, while the second can be classified as searching for full-content duplicates.

Also seemingly related is the general copy detection problem. There are significant differences, however, owing to the noise effects suffered by printed pages and the OCR errors they induce. Methods predicated on finding long strings of perfect similarity may not work as reliably in practice when noisy documents are included in the database. Some of the better-known schemes in this category include COPS [1] which is sentence-based, SCAM [18] which is word-based, and various algorithms for searching by computing checksums in predetermined "windows" [14].

7 Conclusions and Future Research

This paper has examined a number of issues related to the detection of duplicates in document image databases. Four distinct models for formalizing the problem were presented, along with algorithms for determining the optimal solution in each case. Experimental results demonstrate that the models match the real world, and the algorithms are robust with respect to the kinds of OCR errors that are likely to be encountered. Table 3 enumerates these classes one last time. A solid dot highlights the algorithm most suited to a particular problem, while a hollow dot indicates that the algorithm will find not only such duplicates but other types as well.

Since some of the problems seem to subsume others, an obvious question is "Why bother with the less general ones?" The answer lies in increased precision for those situations where admitting a larger class of duplicates is undesirable (*e.g.*, when the targeted duplicates are known to be photocopies). Special cases also make it possible to develop more efficient algorithms.

There are numerous ways this work could be extended. For example, there exists yet another model for approximate string matching known as "word-spotting" that applies when one of the strings must be matched in its entirety and the other is allowed the flexibility of choosing its most similar substring. This might arise when a paragraph is copied out of one document and used to query the database for other pages that contain it. Again, there is a dynamic programming algorithm along the lines of Equations 2 and 4 that solves the problem. Although the *sdist* algorithms can also catch such duplicates, they do so at a potentially lower precision.

Finally, there may be advantages to adding more levels to the symbol/line hierarchy. This could include text blocks as a collection of lines, columns as a collection of text blocks, and pages as a collection of columns. These would add new dimensions to the optimization problem, but the techniques already discussed may be generalizable. The most serious issue appears to be the requirement the system follow a unidirectional editing process at each level. Allowing arbitrary block motion overcomes this, however, and is addressed in another paper [12].

Acknowledgements

The trademarks mentioned in this paper are the properties of their respective companies.

References

[1] S. Brin, J. Davis, and H. Garcia-Molina. Copy detection mechanisms for digital documents. In *Proceedings of the ACM SIGMOD Annual Conference*, San Francisco, CA, May 1995.

[2] D. Doermann, H. Li, and O. Kia. The detection of duplicates in document image databases. In *Proceedings of the Fourth International Conference on Document Analysis and Recognition*, pages 314–318, Ulm, Germany, August 1997.

[3] J. Esakov, D. P. Lopresti, and J. S. Sandberg. Classification and distribution of optical character recognition errors. In *Proceedings of Document Recognition (IS&T/SPIE Electronic Imaging)*, pages 204–216, San Jose, CA, February 1994.

[4] J. W. Fickett. Fast optimal alignment. *Nucleic Acids Research*, 12(1):175–179, 1984.

[5] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, Cambridge, UK, 1997.

[6] J. J. Hull. Document image similarity and equivalence detection. *International Journal on Document Analysis and Recognition*, 1(1):37–42, February 1998.

[7] J. J. Hull, J. Cullen, and M. Peairs. Document image matching and retrieval techniques. In *Proceedings of the Symposium on Document Image Understanding Technology*, pages 31–35, Annapolis, MD, April-May 1997.

[8] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics and Control Theory*, 10(8):707–710, 1966.

[9] R. J. Lipton and D. P. Lopresti. A systolic array for rapid string comparison. In H. Fuchs, editor, *Proceedings of the 1985 Chapel Hill Conference on Very Large Scale Integration*, pages 363–376. Computer Science Press, 1985.

[10] D. Lopresti. Robust retrieval of noisy text. In *Proceedings of the Third Forum on Research and Advances in Digital Libraries*, pages 76–85, Washington, DC, May 1996.

[11] D. Lopresti. String techniques for detecting duplicates in document databases. Submitted for publication, 1999.

Table 3: The algorithms and where they apply.

Duplicate		Algorithm			
Type	Examples	*dist2*	*dist1*	*sdist2*	*sdist1*
Full-format	photocopies, faxes	●	○	○	○
Full-content	printed HTML		●		○
Partial-format	redaction			●	○
Partial-content	copy-and-paste				●

[12] D. Lopresti and A. Tomkins. Block edit models for approximate string matching. *Theoretical Computer Science*, (181):159–179, 1997.

[13] D. Lopresti and J. Zhou. Retrieval strategies for noisy text. In *Proceedings of the Fifth Annual Symposium on Document Analysis and Information Retrieval*, pages 255–269, Las Vegas, NV, April 1996.

[14] U. Manber. Finding similar files in a large file system. In *Proceedings of USENIX*, pages 1–10, San Francisco, CA, January 1994.

[15] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino-acid sequences of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

[16] L. R. Rabiner, A. E. Rosenberg, and S. E. Levinson. Considerations in dynamic time warping algorithms for discrete word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-26(6):575–582, December 1978.

[17] D. Sankoff and J. B. Kruskal, editors. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.* Addison-Wesley, Reading, MA, 1983.

[18] N. Shivakumar and H. Garcia-Molina. SCAM: A copy detection mechanism for digital documents. In *Proceedings of the Second International Conference on Theory and Practice of Digital Libraries*, Austin, TX, 1995.

[19] A. L. Spitz. Duplicate document detection. In *Proceedings of Document Recognition IV (IS&T/SPIE Electronic Imaging)*, pages 88–94, San Jose, CA, February 1997.

[20] G. A. Stephen. *String Searching Algorithms.* World Scientific, Singapore, 1994.

[21] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21:168–173, 1974.