

GENERALIZING EDIT DISTANCE TO INCORPORATE DOMAIN INFORMATION: Handwritten text recognition as a case study *

Giovanni Seni, V Kripásundar, Rohini K. Srihari

CEDAR/SUNY at Buffalo
226 Bell Hall
Buffalo, NY 14260

ABSTRACT

In this paper the Damerau-Levenshtein string difference metric is generalized in two ways to more accurately compensate for the types of errors that are present in the script recognition domain. First, the basic dynamic programming method for computing such a measure is extended to allow for merges, splits and two-letter substitutions. Second, edit operations are refined into categories according to the effect they have on the visual “appearance” of words. A set of recognizer-independent constraints is developed to reflect the severity of the information lost due to each operation. These constraints are solved to assign specific costs to the operations. Experimental results on 2,335 corrupted strings and a lexicon of 21,299 words show higher correcting rates than with the original form.

Keywords: string distance, string matching, spelling error correction, word recognition and correction, text editing, script recognition and post-processing

1 INTRODUCTION

Since the goal of text recognition devices is to accurately transcribe input text, their word recognition modules commonly need to look up their output interpretation string(s) in a pre-specified list of words (lexicon) for validation. Very often, however, there are errors in the interpretation string due to a multitude of factors such as poor input quality, insufficient training, *etc.* The characteristics of different applications impose different constraints on the design of error-correctors. For instance, in an interactive pen-based application, one could simply allow the user to write over an error. Alternatively, a search technique can be used to determine the lexicon-entry which has the maximum likelihood of being the “true” value of the given string. We can think of the search technique as an error-correcting post-processor.

In general, the task of (isolated-word) error correction involves three steps:⁽¹⁾ error detection, generation of candidate corrections, and ranking of candidates. These steps are commonly combined into a single one by computing a similarity measure between the input string and each word in the lexicon. The Damerau-Levenshtein^(2,3) metric is such a measure; it computes the distance between two strings as measured by the

The ideas for the research reported here were conceived independently by each of the authors, who then collaborated on their development.

minimum-cost sequence of “edit operations” — namely, deletions, insertions, and substitutions — needed to change the first string into the second. In this context, the use of the word distance does not always correspond with the precise mathematical definition. The term minimum edit distance was introduced by Wagner and Fischer,⁽⁴⁾ who, simultaneously with Okuda *et al.*,⁽⁵⁾ proposed a dynamic-programming recurrence relation for computing the minimum-cost edit sequence.

Minimum edit distance techniques have been used to correct virtually all types of non-word mis-spellings, including typographic errors (*eg*, mis-typing letters due to keyboard adjacency), spelling errors (*eg*, doubling of consonants), and OCR errors (*eg*, confusion of individual characters due to similarity in feature space). OCR-generated errors, however, do not follow the pattern of human errors.⁽¹⁾ Specifically, a considerable number of the former are not one-to-one errors⁽⁶⁾ but rather of the form $x_i \dots x_{i+m-1} \rightarrow y_j \dots y_{j+n-1}$ (where $m, n \geq 0$). This is particularly true in the script recognition domain, where the need for letter segmentation (partitioning the word into letters) and the presence of ligatures give rise to *splitting*, *merging* and *pair-substitution* errors. Figure 1 illustrates these type of errors: the letter sequence ‘cl’ has been merged into letter ‘d’, ‘a’ has been split into the sequence ‘ci’ and the letter sequence ‘ly’ has been mis-recognized as ‘bj’.

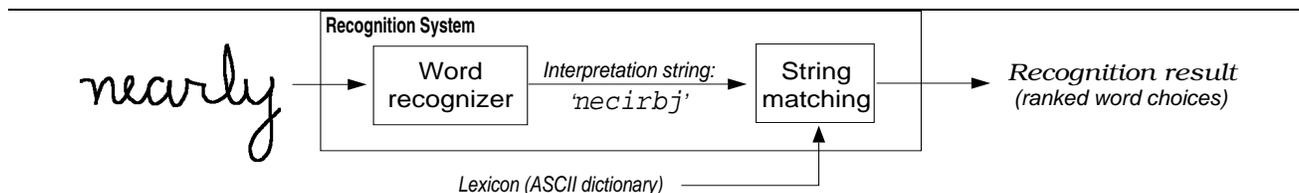


Figure 1: **The role of a string matching algorithm in a handwritten text recognition application:** The intended word is ‘clearly’ but the interpretation string produced by the classifier is ‘decirbj’ after a MERGE, a FRACTURE (or SPLIT) and a PAIR-SUBSTITUTE error.

Different extensions of the basic Damerau-Levenshtein metric are reported in the literature. Lowrance and Wagner⁽⁷⁾ extend the metric to allow reversals of characters. Kashyap and Oommen⁽⁸⁾ present a variant for computing the distance between a misspelled (noisy) string and every entry in the lexicon “simultaneously” under certain restrictions. Veronis⁽⁹⁾ suggests a modification to compensate for phonographic errors (i.e., errors preserving pronunciation). In this research we are concerned with the correction of errors due to improper text segmentation on which relatively little work has been done. We are only aware of Bozinovic’s attempt to model merge and split errors using a probabilistic finite state machine⁽¹⁰⁾ (instead of through minimum edit distance methods). He, however, points out that the model “only approximates merging errors since it does not reflect what symbol the deleted one merged with”.

Another explored direction of generalization to the Damerau-Levenshtein metric comes from assigning different weights to each operation as a function of the character or characters involved. Thus, for example, $W_S('v'/'u')$ (the cost associated with the edit operation ‘u’ \rightarrow ‘v’) could be smaller than $W_S('v'/'q')$. Tables of confusion probabilities modeling phonetic similarities, letter similarities, and mis-keying have been published. For a particular OCR device this probability distribution can be estimated by feeding the device a sample of text and tabulating the resulting error statistics. However, the need remains for a framework permitting the analysis of how the various types of errors should be treated. That is, how does the cost of each operation relate to those of the others? Should $W_S(u/v)$ be less or greater than $W_M(z/xy)$ (the cost associated with the edit operation $xy \rightarrow z$) for any characters u, v, x, y and z ?

This paper shows how the Damerau-Levenshtein metric can be tailored to more accurately compensate for the types of errors characteristic of the script recognition domain. We first discuss the role of *downward strokes* as a significant perceptual element for the recognition process. We then develop a domain-independent framework for modeling the various types of errors, including segmentation ones, where operations are refined into categories

according to the effect they have on the visual form of words. We identify a set of recognizer-independent constraints that reflect the severity of the information lost due to each operation. These constraints are solved simultaneously to assign specific costs to the operations. Finally, the technique is tested on simulated mis-spellings in words (corrupted strings) of length 2 to 15 characters.

2 STROKE-BASED REPRESENTATION OF WORDS

Previous work by Seni and Srihari^(11,12) has demonstrated that the visual configuration of a word written in cursive script can be captured by a stroke description string. The vocabulary of the description string corresponds to the different types of downward strokes* made by a writer in writing the word. Downward strokes constitute a simple but robust cue that contributes significantly to letter identification; furthermore, they allow for a compact description of the overall shape of a word without having to consider its internal details. This idea has been used by the authors to good effect in a lexicon-filtering module for an on-line script recognition application that operates on lexicon sizes of over 20,000 words. It has also provided formal grounding for the notion of visually similar neighborhood (VSN); in Figure 2 an image of the word ‘play’ is shown with its extracted downward strokes and a description of its shape as captured by the string that results of concatenating them. The visually similar words ‘plug’ and ‘ploy’ are also described by this string.

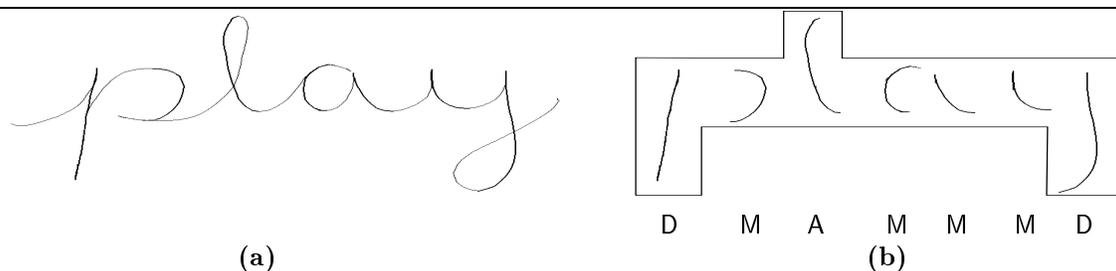


Figure 2: **Stroke-based representation of a word:** (a) a word image, and (b) the extracted downward strokes. The string of concatenated stroke primitives (‘DMAMMMMD’) provides a coarse representation of the word-shape.

The original stroke description scheme identifies 9 different types of strokes, some of which capture spatio-temporal information such as retrograde motion of the pen. To avoid limiting the scope of this work to those instances where temporal information is available, we simplify the stroke description scheme such that it describes any character solely by the length of its strokes. The simplified vocabulary then consists of three different strokes, namely Ascender (A), Median (M), and Descender (D). There is ample evidence for the perceptual relevance of ascending and descending extensions.⁽¹³⁾ The stroke descriptions for the letters ‘p’, ‘l’, ‘a’ and ‘y’ would then be “DM”, “A”, “MM” and “MD” respectively. The stroke description for the word ‘play’ would simply be the concatenation “DM·A·MM·MD”. The dots represent character segmentation points. The importance of the concatenation operation should be highlighted; although character-pairs like ‘p’-‘q’ and ‘d’-‘b’ share the same strokes — namely “D”, “M” and “A”, “M” respectively — they should not be judged similar because these strokes are combined in different ways. Any difficulties in discriminating among these letters could be attributed to recognition at too low a level.⁽¹³⁾

We will refer to the stroke description of a word as its *shape*. It is noteworthy that ascenders and descenders together define the shape of a word by their relative rarity and perceptual salience. We will therefore refer to them collectively as *prominent strokes*.

*These correspond to pieces of the drawing between pairs of consecutive y-maxima and y-minima in the pen trace.

3 EXTENSION OF THE DAMERAU-LEVENSHTEIN METRIC

Let A be a finite set of symbols (the alphabet), and A^* be the set of strings over A . We will denote the null or empty string by λ . Let $X = x_1x_2 \cdots x_n$ and $Y = y_1y_2 \cdots y_m$ be any two strings in A^* (we can assume Y to be a noisy version of X). Let $\alpha, \beta \in A^*$, $1 \leq p \leq n$, $1 \leq q \leq m$. Consider all possible ways of transforming Y into X . Now suppose that the i^{th} edit-sequence consisted of:

1. $\#S_i$ SUBSTITUTE operations of the form $y_q \rightarrow x_p$ (where $X = \alpha x_p \beta$ and $Y = \alpha y_q \beta$);
2. $\#D_i$ DELETE operations of the form $y_q \rightarrow \lambda$ (where $X = \alpha \beta$ and $Y = \alpha y_q \beta$);
3. $\#I_i$ INSERT operations of the form $\lambda \rightarrow x_p$ (where $X = \alpha x_p \beta$ and $Y = \alpha \beta$);

The Damerau-Levenshtein metric (DLM) computes a similarity value between X and Y as follows:

$$DLM(X/Y) = DLM(Y \rightarrow X) = \min_i (\#S_i \times W_S + \#D_i \times W_D + \#I_i \times W_I) \quad (1)$$

where $\#S_i$, $\#D_i$, and $\#I_i$ are the number of SUBSTITUTE, DELETE, and INSERT operations in the i^{th} edit-sequence, and W_S , W_D , and W_I are the non-negative costs associated with each of these operations. We use the notation $DLM(X/Y)$ instead of $DLM(X, Y)$ to emphasize that the problem is not symmetrical in general. Computing the DLM measure can be formulated as an optimization problem and a Dynamic Programming technique can be applied. The computation is carried out using the following recurrence relation:^(4,5)

$$d_{i,j} = \min \begin{cases} d_{i-1,j-1} + W_S(x_i/y_j) \\ d_{i,j-1} + W_D(y_j) \\ d_{i-1,j} + W_I(x_i) \end{cases} \quad (2)$$

with the basis equations $d_{0,0} = 0$, $d_{i,0} = \sum_{k=1}^i W_D(y_k)$, and $d_{0,j} = \sum_{k=1}^j W_I(x_k)$. $DLM(X/Y)$ is then computed as $DLM(X/Y) = d_{m,n}$. The algorithm requires time proportional to the product of the lengths of the two strings (i.e., $O(nm)$) which is not prohibitive. Shortcuts, however, must be devised if comparing the corrupted string with every word in a large lexicon (a problem out of the scope of this paper but documented in the literature).⁽⁸⁾

Although the above three edit operations (henceforth termed SUBSTITUTE, DELETE, and INSERT) have a strong data-transmission “flavor”, since they were originally motivated by applications such as automatic detection and correction of errors in computer networks, they also suit the type of errors introduced by OCR’s and other automatic reading devices. Insertions are needed to compensate for characters in the input which did not exceed a minimal recognition threshold; deletions are needed to get rid of false-positive responses — ligatures are a large source of false-positives in the script recognition domain — and substitutions are needed to compensate for likely character confusions. The types of errors that these operations correct can be considered recognition errors. A different type of error occurs when adjacent characters are merged or split due to improper character segmentation. To capture the fact that the merging of two characters into a third is not quite the same phenomenon as a substitution plus a deletion[†], we explicitly introduce the MERGE and SPLIT operations. In *sans serif* text, for instance, the sequence `rn` can easily be merged into an `m`. This cannot be modeled meaningfully by a (context-free) substitution of `r` for `m` and a parallel deletion of `n` (cf. Figure 3). The recurrence relation in Equation (2) can be easily extended to cope with merging and splitting errors by adding the minimization terms:

$$\begin{aligned} d_{i-2,j-1} &+ W_{\text{merge}}(x_{i-1}x_i/y_j) \\ d_{i-1,j-2} &+ W_{\text{split}}(x_i/y_{j-1}y_j) \end{aligned}$$

We introduce here another edit operation that we term PAIR-SUBSTITUTE, to model a different type of phenomenon. MERGE and SPLIT model segmentation errors on the part of the recognizer — specifically, errors of omission and insertion of segmentation points. A third kind of segmentation error occurs when there is a *movement* of the segmentation point (cf. Figure 1). PAIR-SUBSTITUTE models this by substituting a pair of characters by another pair. The following term is thus also added to Equation (2):

$$d_{i-2,j-2} + W_{\text{pair-substitute}}(x_{i-1}x_i/y_{j-1}y_j)$$

[†] In fact, a substitution can itself be modeled as a deletion plus an insertion, and has been treated as such in many early applications.

Just as a MERGE cannot be replaced by a substitution-plus-deletion, so a PAIR-SUBSTITUTE operation cannot be replaced by two parallel, but non-conjoined substitutions. All three operations can be thought of as capturing (a limited amount of) context-sensitivity, and so cannot be reduced to any set of “simpler” operations.

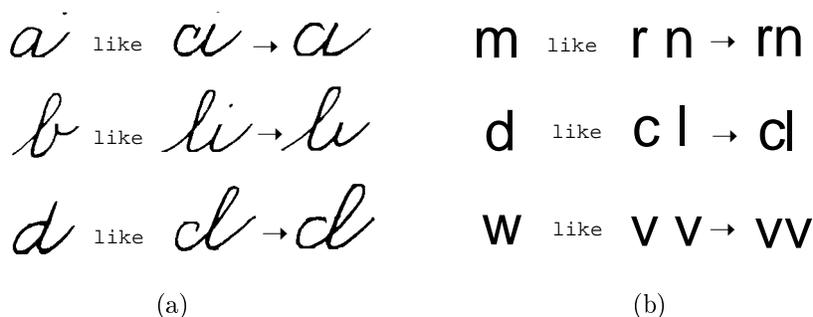


Figure 3: **Examples of common merging errors:** Errors occurring in (a) cursive handwriting, and (b) machine-printed text. These errors cannot be modeled meaningfully by a (context-free) substitution and a parallel deletion.

Wagner and Fischer⁽⁴⁾ used the notion of trace to prove the correctness of Equation (1). Their proof hinges on the fact that no two lines in the trace cross. Their notion of trace is easily (and naturally) extended to incorporate the three new operations without introducing crossings. Hence, the optimality of the extended recurrence holds.

With the addition of the three new operations, we are faced with a compelling need to develop a rationale for any decisions we make about the relative costs associated with the operations. It turns out that stroke descriptions provide valuable information in comparing a test string (the string generated by the recognizer) and a reference string (usually an entry in the lexicon). In the rest of the paper, we will use the letters S, D, I, and M to refer to SUBSTITUTE, DELETE, INSERT, and MERGE respectively. We will use F to denote a SPLIT (mnemonic: a SPLIT is a FRACTURE) and P to denote a PAIR-SUBSTITUTE.

4 A TYPOLOGY OF RECOGNIZER ERRORS

Since we are interested in finding a minimum-cost edit sequence for transforming the test string into the reference string, we define the relative cost of an edit operation in terms of its effect on the visual appearance of the test string. More precisely, we can outline three primary criteria on the basis of which we refine the six categories of operations into Likely, Unlikely and Impossible operations. We also achieve a basic ordering of the refined set of operations based on these three criteria. We will then apply these primary criteria, and other secondary criteria, to further restrict the legal cost-ranges for the various operations.

Based on the stroke description scheme, we define (in order of decreasing significance) three measures that help us judge the quantity and quality of the “damage” that a particular edit operation inflicts on the shape of the test-string.

- (M1) the number and positions of prominent strokes
- (M2) the number and positions of *all* strokes together
- (M3) the number and positions of characters

The primary criteria are defined as the changes caused by the edit operation to M1, M2 and M3. Thus, a SUBSTITUTE would *a priori* be considered less damaging than a DELETE — an S would maintain all three variables at approximately the same value, whereas a D would damage M2 and M3 at the very least. Similarly

a FRACTURE would be rated as being less expensive than an INSERT, since an F increases M3 alone, while an I increases both M2 and M3.

4.1 Refining the operations

We refine the six basic operations using the measures M1, M2 and M3. We will use the prefixes ‘VL’, ‘L’ and ‘U’ to denote ‘Very Likely’, ‘Likely’, and ‘Unlikely’ operations, respectively. Table 1 summarizes the definitions.

Basic operation	Refinement	Definition (based on change in word shape)	Measures affected	Example
SUBSTITUTE	VLS	No change in shape	None	‘n’ → ‘u’
	LS	Prominent strokes static	M2	‘b’ → ‘l’
	US	Prominent stroke(s) demoted	M1, M2	‘h’ → ‘n’
PAIR-SUBST	PS	No change in shape	None	‘hi’ → ‘lu’
MERGE	LM	No change in shape	M3	‘ij’ → ‘y’
	UM	Prominent strokes static	M2, M3	‘uj’ → ‘y’
DELETE	VLD	Single Median deletion	M2, M3	‘r’ → ‘λ’
	LD	Prominent strokes static	M2, M3	‘a’ → ‘λ’
	UD	Everything else	All three	‘y’ → ‘λ’
FRACTURE (SPLIT)	LF	No change in shape	M3	‘u’ → ‘ii’
	UF	Prominent strokes static	M2, M3	‘d’ → ‘ch’
INSERT	LI	Single Median insertion	M2, M3	‘λ’ → ‘i’
	UI	Everything else	All three	‘λ’ → ‘q’

Table 1: **Refining the basic operations:** The prefixes ‘VL’, ‘L’ and ‘U’ qualify each operation as being ‘Very Likely’, ‘Likely’, and ‘Unlikely’, respectively. The cases not covered here are deemed Impossible.

The cases that are not covered by the definitions in the table are defined to be Impossible operations. For instance, a MERGE of the form ‘pd’ → ‘q’ simply cannot happen under any normal circumstances. The distinction between Unlikely and Impossible operations is that Unlikely operations could happen in very noisy situations or when there are serious generation problems, while Impossible operations cannot be conceived of even under such circumstances. The Impossible operations do not figure in any of our analysis, because their cost is set to ∞ in order to prevent them from being included in any minimum-cost edit sequence. We will therefore refer to the set {VLS, LS, US, PS, LM, UM, VLD, LD, UD, LF, UF, LI, UI} as the set of refined operations.

It is worth noting that secondary criteria such as (non-)closure of loops may motivate occasional deviations from the definitions set forth in Table 1. For instance, one could decide not to categorize ‘y’ → ‘q’ as a VLS despite the similarity in the stroke descriptions for the two characters. The secondary criteria thus provide a basis for fine-tuning of decisions concerning the refined operations.

The role of statistics: As discussed in the Introduction, the weights used in a certain application can be customized by estimating the recognizer’s error probabilities based on error statistics (derived by observing the behavior of the recognizer on sample text). Quantizing the error histograms from such statistics would result in a set of refined operations similar to those presented in Table 1. However, this method has several difficulties:

1. It is prone to error when there is insufficient training data. Statistical methods are notorious for their sparse data problems.
2. The method requires considerable truthing effort for a large training set. Further, the truthing can become skewed due to systematic judgement errors on the part of the truther.

3. Thresholds for quanta have no intrinsic “meaning”, and so no theoretical significance.

The stroke description scheme, along with the measures M1, M2 and M3, provides a solid foundation for the refinement of the basic operations. Error statistics, when reliable, can and should be incorporated into individual decisions about membership in the various categories of operations.

Thus, Table 1 provides a theoretically motivated refinement of the set of basic operations. However, assigning relative costs to this set of finer divisions among the operations is not a straightforward process. Nor is it easy to justify any intuitions about the ordering of the basic operations. Here, too, our three measures M1, M2, and M3 come to the rescue. In the rest of this section, we will look at the implications of our three criteria in assigning positions and values to the operations.

4.2 The basic ordering

In measuring the distance of the test string from the reference string, the system is in fact attempting to recover from (potential) recognizer errors. The nature of cursive handwritten text is such that the recognizer is more liable to treat non-strokes (such as ligatures and connection strokes) as valid strokes, than to overlook valid strokes. Therefore, in this domain, a DELETE operation should be penalized less than an INSERT should. (The opposite may be true in the domain of machine-printed text.)

Given this, and on the basis of the three measures and their importance relative to each other, we can draw certain general conclusions about the various operations. These provide us with a basic ordering of the set of refined operations. The definitions in Table 1 can be consulted in order to follow the analysis here.

First, we note that every Unlikely operation must be penalized more than any Likely operation should. A glance at the table will reveal that Unlikely operations tend to upset a superset of the measures affected by Likely operations. Indeed, most of the Unlikely operations directly affect the value of M1. We denote this “meta-ordering” by the formula:

$$L\mathcal{X} < U\mathcal{Y} \tag{3}$$

where the \mathcal{X} and \mathcal{Y} stand as place-holders for any of the six basic operations. Table 1 also confirms that the six operations can be grouped based on their effect on the three measures:

- SUBSTITUTE & PAIR-SUBSTITUTE preserve M3, while none of the others do; further, VLS preserves the recognizer’s segmentation decisions while PS over-rules them.
- MERGE & DELETE decrease M3; DELETE decreases M2 also. (MERGE only re-groups strokes, and so does not affect M2.)
- SPLIT & INSERT increase M3; INSERT increases M2 also. (SPLIT, like MERGE, also re-groups strokes while preserving M2.)

Consequently, and because of the (domain-specific) assumption concerning DELETE’s *vs.* INSERT’s, we can conclude that:

$$\text{SUBSTITUTE} < \text{PAIR-SUBSTITUTE} < \text{MERGE} < \text{DELETE} < \text{SPLIT} < \text{INSERT} \tag{4}$$

Based on Inequalities (3) and (4) we can now refine the ordering as follows:

$$\begin{aligned} \text{VLS} < \text{PS} < \text{LS} < \text{LM} < \text{VLD} < \text{LD} < \text{LF} < \text{LI} \\ < \text{US} < \text{UM} < \text{UD} < \text{UF} < \text{UI} \end{aligned}$$

We will refer to this as the **basic ordering** of the set of refined operations. As noted above, PS gets placed after VLS because the former does not preserve segmentation information. PS gets positioned before LS because, unlike LS, it preserves M2.

4.3 Additional constraints

We now apply secondary constraints to further restrict the ranges of costs that the various operations can be assigned. These secondary constraints are specified in an attempt to model phenomena that are not captured by the basic ordering above. The first pair of such constraints are:

$$\text{LM} < \text{VLS} + \text{VLD} \quad (5)$$

$$\text{LF} < \text{VLS} + \text{LI} \quad (6)$$

These constraints capture the fact that a MERGE (such as ‘cl’ → ‘d’) is not just a DELETE plus a SUBSTITUTE, and therefore should not be penalized to the same extent. Although these are trivially satisfied by the basic ordering, they help to illustrate the types of relationships that the additional constraints try to capture. Next, we specify:

$$\text{LD} < 2 \times \text{VLD} \quad (7)$$

This is motivated by the fact that an LD “corresponds” to two VLD’s (in that a VLD deletes a single median stroke, while an LD deletes two), but damages M3 (the character count) less than a pair of VLD’s. We then add two complementary constraints as follows:

$$\text{LM} + \text{LI} < \text{US} + \text{VLS} \quad (8)$$

$$\text{LF} + \text{VLD} < \text{US} + \text{VLS} \quad (9)$$

These model the idea that an edit sequence where a Likely MERGE was coupled with a Likely INSERT would be preferred to a sequence where an Unlikely SUBSTITUTE was forced. Thus, for example, in comparing the test string ‘suli’ with the reference string ‘such’, we would prefer the edit sequence [‘λ’ → ‘c’; ‘li’ → ‘h’] over the sequence [‘l’ → ‘c’; ‘i’ → ‘h’].

4.4 Solving for the cost ranges

In order to find values for the costs that must be associated with the various edit operations, we solve the set of inequalities formed by adding the additional constraints to the basic ordering. We used the simplex method of linear programming to solve the set of inequalities. The objective function that we maximized was:

$$(\text{US} - \text{LI}) + ^-(\text{UI} - \text{US}) + ^-(\text{LI} - \text{VLS})$$

Each of the three terms in the objective function captures a specific aspect of the structure of the set of refined operations. Maximizing the first term (US – LI) corresponds to stating that the Unlikely operations should be placed as far from the Likely operations as possible. Maximizing $^-(\text{UI} - \text{US})$ is the same as minimizing (UI – US) and so states that the Unlikely operations should be grouped together. Similarly, the third term $^-(\text{LI} - \text{VLS})$ implies that the Likely operations should also be clustered.

This objective function increases monotonically with the Unlikely operations, and so we need to specify an upper bound for the costs of the various operations. Therefore, we bound the operations from above by adding the constraint $\text{UI} \leq 1.5$. We further specify that $\text{US} = 1.0$ based on the reasoning that this corresponds to the “traditional” notion of SUBSTITUTE. Additionally, we set the step-size for the costs to be 0.05 and specify a minimum cost of 0.2 for any operation. These numbers constitute reasonable estimates, and are certainly open to refinement based on feedback through performance figures. With these constraints, we obtained the cost assignment in Table 2.

Notice that the ratio between the highest and the lowest costs for the Likely operations (0.55 & 0.2) is approximately 3. Without the preceding analysis, we would have no basis for specifying any bounds on this ratio.

VLS	PS	LS	LM	VLD	LD	LF	LI	US	UM	UD	UF	UI
0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	1.0	1.05	1.1	1.15	1.2

Table 2: **Cost assignment for the refined set of operations:** Two clusters are clearly noticeable here: they correspond to the Likely and Unlikely operations.

5 TESTING THE STRING MEASURE

Having determined costs for the set of refined operations, the next step is to decide which character or characters will be involved in each type of operation. These decisions are initially made by simply looking at the stroke description of each character. As pointed out in §4.1, such an assignment can then be refined based on secondary criteria and tabulated error statistics. Tables 3, 4, and 5 show our current letter assignment intended for a script recognition application. It is not claimed that this assignment represents an optimal or even a complete set. It has been chosen to illustrate the proposed methodology and to allow for a comparison with the traditional Damerau-Levenshtein metric (*ie*, $W_S = W_D = W_I = 1$ in Equation (1) on page 4).

x_i	VLS	LS	x_i	VLS	LS	x_i	VLS	LS
a	u	o	j	—	—	s	—	l
b	—	l	k	—	h	t	—	—
c	—	—	l	—	—	u	v,n,a	o
d	—	l	m	w	n	v	u	—
e	i	—	n	u	v	w	m	u, v
f	—	—	o	—	a, u	x	—	—
g	y	j, q	p	—	—	y	g	j, q
h	—	k, l	q	g	y	z	—	—
i	e	r	r	—	i			

Table 3: **The SUBSTITUTE table:** Each entry in the table specifies the category (and cost) of that particular SUBSTITUTE. *eg*: ‘l’ \in SUBSTITUTE(‘h’, LS) means that $W_S(\text{‘h’}/\text{‘l’}) = \text{LS}$.

SPLIT ↓	a	b	d	h	k	u	u	w	w	y
MERGE ↑	ci	li	cl	li	lc	ee	ii	ev	iv	ij

Table 4: **The SPLIT/MERGE table:** The entries in the table are bi-directional. (This is not necessarily the case in every domain.) Going from the top row to the bottom row corresponds to a SPLIT, while the reverse corresponds to a MERGE.

5.1 Test strings

We have used an automatic string corruption scheme, to test the extended DLM against the traditional DLM. The primary motivation for the use of such a scheme is its speed. A secondary benefit of using this corruption scheme is that, unlike an algorithm that generates corruptions randomly, it allows us to ensure that the “recognition” results are comparable to those of a real (*ie*, lexicon-independent) recognizer.

{he, hi} ⇔ lu	{em, im} ⇔ un
{me, mi} ⇔ nu	{ey, iy} ⇔ uj
{ue, ui} ⇔ {eu, iu}	{ew, iw} ⇔ uv
{mn, mu} ⇔ nm	
{hn, hu} ⇔ lm	{bj, hj} ⇔ ly
hv ⇔ lw	
mv ⇔ nw	mj ⇔ ny
wv ⇔ uw	wj ⇔ uy

Table 5: **Valid PAIR-SUBSTITUTE possibilities:** Very few PAIR-SUBSTITUTE’s are plausible, and even these can be grouped on the basis of their participating letters. Letter-pairs such as ‘e’-‘i’ and ‘n’-‘u’ form the “basis” for these groups. Like SPLIT/MERGE’s, PAIR-SUBSTITUTE’s tend to be bi-directional also.

The corruption scheme works through a Handwritten Word Recognizer (HWR) simulator. As part of the research at CEDAR in the use of language models for contextual post-processing of recognizer output, a two-stage HWR simulator,⁽¹⁴⁾ has been previously developed. In the first stage, the simulator uses probability-estimates of segmentation and recognition errors (derived from an actual recognizer) to carry out the corruption process. In its second stage, the simulator then uses a lexicon to come up with the words that are closest to the corrupted string.

For the results reported here, we have generated the test data using the first stage of the simulator to come up with plausible corruptions of actual lexicon entries. The input to the simulator was a piece of running English text, and so reflects word-frequency phenomena present in natural language. The output was a set of 2,335 corrupted strings, ranging in length from 2 to 15 characters. The output strings contained from 1 to 12 errors per string, with an average of 4.

5.2 Results

In the context of word recognition applications, three different measures can be used to determine the effectiveness of a string similarity function: **accuracy**, which is the expected value of the position at which the correct word appears in the ranked lexicon; **granularity**, which measures the average number of words of the ranked lexicon which have an equal-score to the correct word; and **speed**, which is the average time taken to carry out the matching process. In practice, however, the recognition module would have to output a list of M words (where $M = \text{position} + \text{equal-score}$) to make sure that the correct word is present; we term this list the “match” set.

Figure 4 illustrates some statistics about the size M of the match set for the 2,335 corrupted strings and a lexicon of 21,299 words: the size of the match set at the p^{th} percentile indicates the maximum number of words that would need to be examined to achieve a $p\%$ “hit-ratio” on the truth. The size of the match set is thus expected to grow with the required $p\%$ hit-ratio. The superior performance (as measured by the size of the match set) of the the extended DLM over the traditional metric is clearly noticeable.

For purposes of comparison, we have also included the performance of Proximity on the same data-set. Proximity is a function, which has previously been used at CEDAR, based on a “cumulative” implementation of Longest Common Subsequence (LCS):⁽¹⁵⁾

$$Proximity(X, Y) = \frac{2 \times P(X, Y)}{P(X, X) + P(Y, Y)}$$

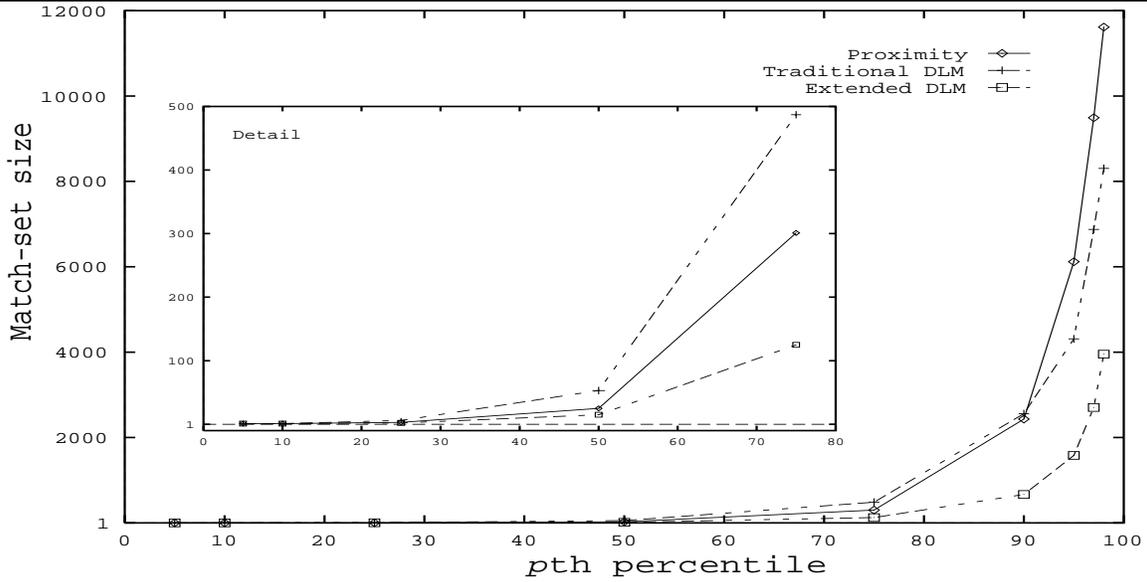


Figure 4: **Performance evaluation of the Extended DLM:** The graph depicts the sizes of the match sets at various p^{th} percentile levels. The size of the match set returned by Extended DLM is always smaller than with Traditional DLM and Proximity.

with

$$P(X, Y) = \sum_i LCS(X_{1:i}, Y_{1:i}) + \sum_i LCS(X_{1:i}^R, Y_{1:i}^R)$$

where $X_{1:i}$ denotes the string obtained from X by keeping its first i -elements, and X^R is the reversal (left-right flip) of X . Extended DLM compares favorably with Proximity also.

Details of the computed statistics about the size of the match set are given in Table 6. The table also shows that the extended DLM achieves an approximate three-fold improvement in performance over the traditional metric.

Metric/Statistic	P5	P10	Q1	H	Q3	P90	P95	P97	P98	Range	Mean	Z
Proximity	1	1	3	25	301	2432	6118	9491	11614	1–21299	994.31	1
Traditional DLM	1	1	6	53	487	2557	4312	6874	8303	1–20141	825.21	1
Extended DLM	1	1	2	15	125	667	1581	2704	3953	1–14503	309.28	1

Table 6: **Performance evaluation of the Extended DLM:** The table summarizes statistics for the match sets for 2,335 corrupted strings and a lexicon of 21,299 words. (H = Median, P = Percentile, Q = Quartile, Z = Mode)

5.3 Discussion

The significant performance improvement seen in Table 6 is a strong indication of the validity of the proposed scheme. We should emphasize that we made no attempt to unveil the corruption process carried out by the HWR simulator. The decisions as to what letters to include in each operation type were made only on the basis of the notion of visual similarity of letters.

Previous literature on the subject has offered no suggestions regarding the ordering or the relative costs of a refined set of operations. Indeed, even deciding the *number* of degrees of likelihood that are relevant to a particular domain has been left to the user’s whim. Our analysis for the script domain is still open to debate and refinement, but it provides a basis for decisions such as $VLS < VLD$.

We believe that the central ideas discussed here are applicable in several related domains. In effect, the idea of stroke descriptions and visual similarity comprise a model of noise faced by a handwritten text recognition post-processor. In a different domain, a different model of noise and a corresponding set of constraints can be applied to yield similar improvements in string distance measurement. Our research here demonstrates that a theoretically grounded approach to refining edit operations, and to cost assignment, is possible and is beneficial.

6 SUMMARY AND FUTURE WORK

The Damerau-Levenshtein metric is quite possibly one of the most widely used measures for the string-to-string similarity problem. Although originally designed to correct simple data transmission errors, the metric can be easily extended and customized to accurately compensate for the type of errors present in a given application domain. In the context of handwritten text recognition, error characterization is achieved using the notion of visual form of words as captured by a stroke description string. It provided the foundation to develop a hierarchy of errors, where errors altering word shape are assigned high cost, whereas errors preserving word shape are assigned a lower cost.

Further characterization of the error process is achieved by introducing three new operations — namely, MERGE, SPLIT, and PAIR-SUBSTITUTE. Relationships between the different types of errors are specified as a set of linear constraints which is solved to derive costs for the distinct error classes. The notion of visual form of letters is parallel to that of the visual form of words; it permits subsequent assignment of letters to the different error classes based on visual similarity. Such assignment can be additionally tailored to a specific application using statistical findings when reliable.

Finally, a word is in order about the strength of the extended DLM as a post-processor in a handwritten text recognition application for large vocabularies (*eg*, $> 20,000$ words). Clearly, a match-set size of more than a few words would frustrate any user, since no practical procedure can recover the truth from such a set. There is, thus, the need for lexicon filters that reduce the size prior to the matching process. The main limiting factor of any string-distance measurement based on edit sequences, including DLM’s, is the “local” nature of its computation. Such metrics have a restricted view of the two strings being compared. It may therefore be fruitful to use a global metric (such as techniques based on Longest Common Subsequence) in conjunction with the extended DLM.

Despite all these attempts to improve error-correction performance, the task remains challenging. Not all errors can be corrected using intra-word information. Errors that result in other valid words, and errors that have several potential corrections, can only be corrected automatically when the global context of the word in the running text is taken into account. Research is in progress at our group to develop a formal model of linguistic context that would help in the disambiguation of the recognizer’s output.

ACKNOWLEDGMENT

The authors wish to acknowledge financial support from the National Science Foundation (NSF) through grant IRI-9315006 under the Human Language Technology program.

References

- [1] K. Kukich. Automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439, 1992.
- [2] F.J. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964.
- [3] V.I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics-Doklady*, 10(8):707–710, 1966.
- [4] R.A. Wagner and M.J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.
- [5] T. Okuda, E. Tanaka, and T. Kasai. A method for the correction of garbled words based on the Levenshtein metric. *IEEE Transactions on Computers*, 25(2):172–178, 1976.
- [6] M.A. Jones, G.A. Story, and B.W. Ballard. Integrating multiple knowledge sources in a Bayesian OCR post-processor. In *ICDAR-91*, pages 925–933, St. Malo, France, 1991.
- [7] R. Lowrance and R.A. Wagner. An extension of the string-to-string correction problem. *Journal of the ACM*, 23(2):177–183, 1975.
- [8] R.L. Kashyap and B.J. Oommen. An effective algorithm for string correction using generalized edit distances. *Information Sciences*, 23:123–142, 1981.
- [9] J. Veronis. Computerized correction of phonographic errors. *Computers and the Humanities*, 22:43–56, 1988.
- [10] R. Bozinovic and S.N. Srihari. A string correction algorithm for cursive script recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4(6):655–663, 1982.
- [11] G. Seni, N. Nasrabadi, and R.K. Srihari. An on-line cursive word recognition system. In *Computer Vision and Pattern Recognition (CVPR)*, pages 404–410, Seattle, Washington, 1994. IEEE.
- [12] G. Seni and R.K. Srihari. A hierarchical approach to on-line script recognition using a large vocabulary. In *IWFHR-IV*, pages 472–479, Taipei, Taiwan, 1994.
- [13] H. Bouma. Visual recognition of isolated lower-case letters. *Vision Research*, 11:459–474, 1971.
- [14] R.K. Srihari, Stayvis Ng, Charlotte M Baltus, and Jackie Kud. Use of language models in on-line sentence/phrase recognition. In *IWFHR-III*, pages 284–294, Buffalo, New York, 1993.
- [15] Proximity Technology Inc. *PF474 - PROXIMITY Technical Manual*, 1984.