INDIRECT SYMBOLIC CORRELATION APPOROACH TO UNSEGMENTED TEXT RECOGNITION

G. Nagy[1], S. C. Seth[2], S. K. Mehta[3], Y. Lin[2]

[1]Rensselaer Polytechnic Institute, Troy, NY (nagy@ecse.rpi.edu)

[2]University of Nebraska-Lincoln (<seth, ylin>@cse.unl.edu)

[3]Indian Institute of Technology, Kanpur, India (skmehta@cse.iitk.ac.in)

Introduction

During the last twenty years, most recognition engines for difficult to segment scripts have been built around Hidden Markov Models (HMMs). Parametric recognizers for unsegmented signals, like HMMs, are hard to train. In contrast, non-parametric classifiers, like Nearest-Neighbor, require only a labeled reference list. In this paper, we provide preliminary results in support of an entirely new method for non-parametric classification of unsegmented text. *Indirect symbolic correlation* is a general method for bringing lexical context into the recognition of unsegmented signals that represent words or phrases in printed form. It is applicable wherever segments of lexically labeled reference signals can be compared to unlabeled signals. The signal need only preserve the ordering of the alphabetic units within a word (and of words within a phrase).

The method is general in the sense that it does not depend on the signal representation or signal-matching algorithm except for the above constraint. *Indirect* means that the unknown signals can represent words for which no labeled signals are available. *Symbolic* means that the label of the unknown signal is determined by comparing the signal-level matches with lexical matches precomputed between the labels of the reference signals and a lexicon of admissible words. *Correlation* refers to both the signal-level and the lexical tally of ordered matches, which is usually accomplished by shifting one signal with respect to the other.

The nature of the underlying features is immaterial for the graph-level comparisons. Errors at the feature level can be compensated by extending the reference signal to increase the number of potential matches for each segment of the unknown signal.

Indirect Symbolic Correlation promises far-reaching benefits over HMM. It avoids parameter estimation with unstable Expectation Maximization. It can use the reference sample more efficiently than HMM, and immediately incorporate new samples into the recognition process.

Approach

In contrast with most pattern recognition methods, which compare an unknown signal with a labeled reference signal, we propose a graph-based *comparison of sequence comparisons*. To illustrate the concepts, consider an idealized example of cursive English script with only a six-letter subset of the alphabet.

Unknown words must belong to a lexicon of acceptable words. This list is available in some computer representation, such as UNICODE symbols. Here the *lexicon* consists of 9 words:

low, me, mole, mule, moll, mellow, wool, loom, we.

Written samples of a subset of the lexicon are available. These samples are correctly labeled. Such a sample is usually called a training set, but here it is used only for comparisons, and called the *reference set*. There are five samples in the reference set. Every sample happens to be a different word. The reference sample needs not to be segmented at the word level; we show the inter-word blanks only for the sake of legibility.

The script font of the reference set is intended to suggest handwriting:

loom, mole, me, mule, moll.

The transcript of the reference set is also available:

loom, mole, me, mule, moll.

We wish to recognize an unknown word sample, such as

mellow.

This word does not appear in the reference set, but segments of the word, such as

m e me l ll o lo

will be similar to the corresponding segments of the reference set. In fact, some of the segments (here only individual letters) appear multiple times in the reference set. With a large reference set, there will be more matches that are multi-symbol. It is precisely the number of redundant comparisons that increase with the length of the reference set, which is the strength of this method. Even if many of the matches fail to be recognized, or there are false matches, adding samples to the reference list will reduce the error rate.

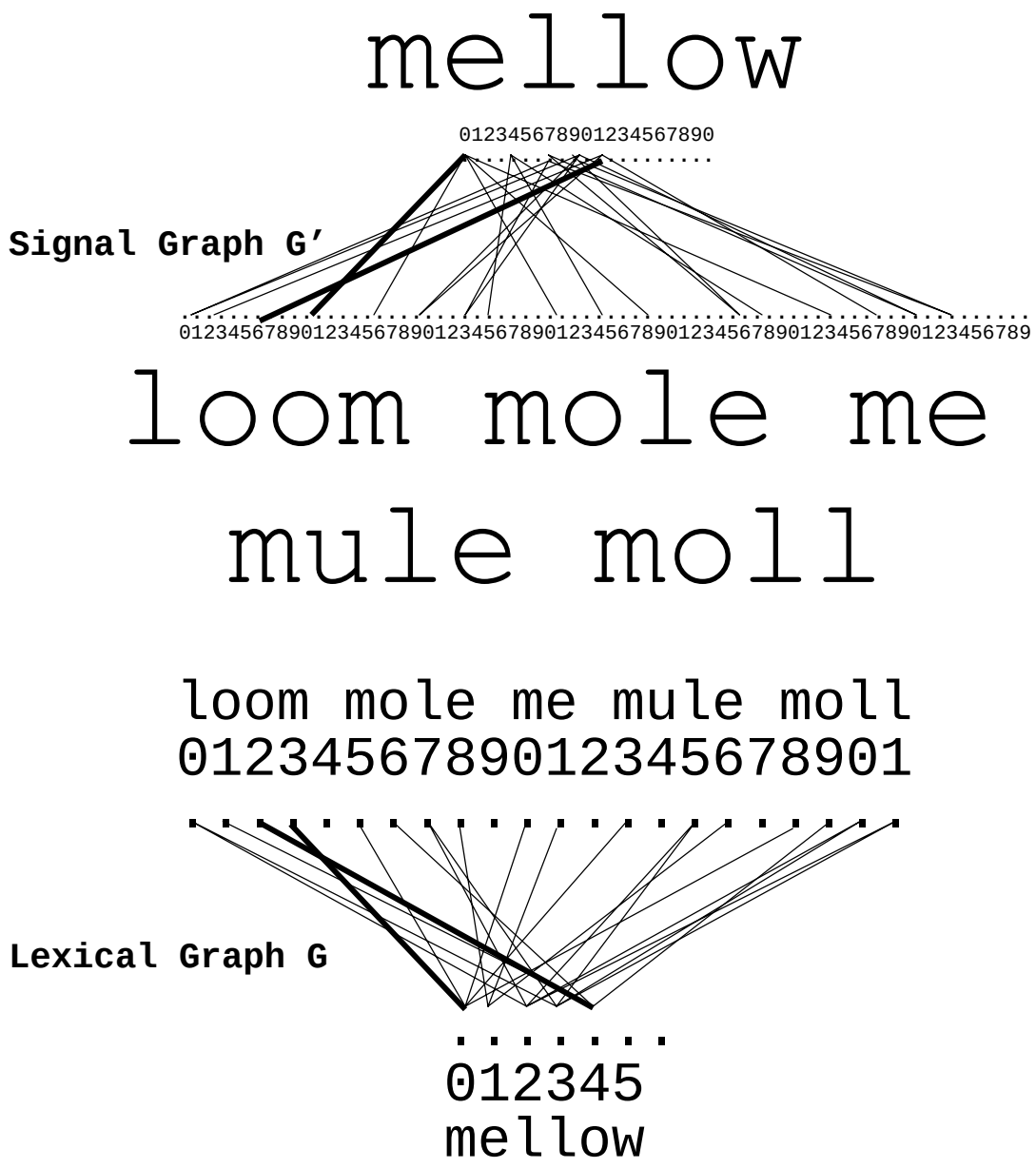
If one knew exactly where each match occurred then one could immediately identify each symbol, and hence the unknown word. However, the position of the matches cannot be known unless both the reference words and the unknown pattern are *segmented* at the character level. It is known from experience that accurate segmentation prior to recognition is a nearly impossible task in OCR.

Match Graphs

Figure 1 shows the two bipartite graphs that result from the feature-level and lexical comparisons of the same word against the reference string. The top graph *(G′)* shows error-free matching of the feature string of the unknown word ("mellow") against the stored reference feature string. The bottom graph *(G)* shows the same comparison at the lexical level.

We next compare the signal-match graph G' to the lexical graph G. G' represents the correspondence of matching segments between the features of the unknown signal and a reference signal. G represents the lexical correspondences between the labels of the reference signal and a specific word of a potentially very large lexicon. New words are recognized because their constituent parts are matched by portions of the reference signal, and the order of the matches is unscrambled through the lexical comparison of the reference labels with the lexicon. An entire set of graphs similar to G are precomputed, one for each word in the lexicon. The unknown pattern is identified with the label of the lexical graph that best matches the signal graph G'.

The discretization at the feature level is arbitrary because different letters may have different widths. At the lexical level, each character has a count of 1. Although the numbering is different, ideally the *sequence* of matches is the same in the two comparisons of the same word. However, the lexical transcripts preserve only the *relative order* of the features, not their linear scale. We must therefore adopt a graph-theoretic approach for sequence comparisons instead of vector space operations.



Lexicon L	{low, me, mole, mule, moll, mellow, wool, loom, we}
Unkown q(t)	mellow
Ref. Signal r(t)	low me mole mule moll
Ref. String r_s	low me mole mule moll
X′ (q)	{(0,7), (0,9), (2,11), **(6,11), (10,0)**, (15,0), (19,11)…}
X_{mellow}	{(0,2), (0,3), (1,4), **(2,4), (3,0)**, (5,0), (6,4), (7,2), …}
X_{wool}	{(0,3), (1,1), (1,2), (2,1), (2,2), (6,1), (6, 2), (7,3), …}
f(q)	mellow

Figure 1: example of feature-level and lexical comparisons.

Recognizing the unknown sample requires finding a word in the lexicon that has an identical (or similar) sequence of matches as that of the unknown word with the reference feature string. In other words, when the match function is error-free, the string match set X_{mellow} is order-isomorphic to the signal match set $X'_{q,}$ whereas the string match set of a different word, such as $X_{wool,}$ is not. In Figure 1, the lexical comparison for "`wool`" is listed without showing its lexical graph.

The notion of *order isomorphism* can be interpreted as follows. The matches (2, 4) and (3, 0) in the string match set have the same order relation as the matches (6, 11) and (10, 0) in the signal match set. Thus, identifying (2, 4) with (6, 11) is *compatible* with identifying (3, 0) with (10, 0). (The corresponding graph edges are shown in bold in Fig. 1.) If every pair of pairs is compatible, then the two sequences are order isomorphic.

Order Isomorphism and Permutations

The order isomorphism problem can be restated as finding common patterns in two permutations. Graphically, a permutation of integers 1 to n can be represented as a bipartite graph, e.g. the permutation (2 4 1 3 6 7 5) of 1 to 7 will have the representation shown in Figure 2.

In general it is not necessary to use contiguous set of numbers to describe a permutation as long as we keep in mind the natural ordering of the numbers. So the same graph can also be represented by (5 11 3 8 16 20 14).

Both our signal and lexical graphs are bipartite but not, in general, permutations because a node can have zero or multiple edges incident on it. However, we can convert them to permutations by splitting nodes with multiple edges and eliminating nodes with zero degree, as illustrated in the example in Figure 3.

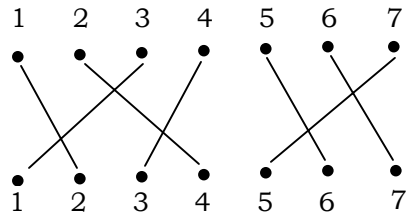


Figure 2: Permutation represented as a bipartite graph.

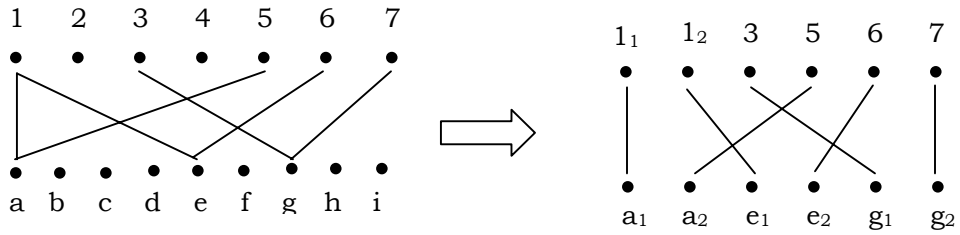


Figure 3: Conversion of a bipartite graph to permutation.

The conversion process preserves the order isomorphism because the left-to-right order of edges encountered (at the top or bottom) does not change.

A sub-permutation is a subsequence of the permutation sequence. So (11 3 16 14) is a sub-permutation of (5 11 3 8 16 20 14). It is easy to observe that the graph associated with a sub-permutation is a sub-graph. Actually, there is a one-to-one correspondence between the sub-graphs and sub-permutations.

Now the order isomorphism problem can be restated as follows: Given two permutations, G1 and G2, not necessarily of the same size, find the largest common sub-graph of G1 and G2, where the size of a permutation graph is the number of its edges.

We will first restate the problem in the language of permutations:

> Given two permutations T = (T_0, T_1, …, T_{n-1}) and P = (P_0, P_1, …, P_{m-1}) (corresponding to graphs G1 and G2 respectively). Find the *longest* sequence of pairs, <T_{i1},P_{j1}>, <T_{i2},P_{j2}>, …, <T_{ik},P_{jk}>, with the following properties:
>
> (a) $0 \le i1 < i2 < \cdots < ik \le m-1$, $0 \le j1 < j2 < \cdots < n-1$, and
> (b) Permutations T' = (T_{i1},T_{i2},…,T_{ik}) and P' = (P_{j1}, P_{j2},…, P_{jk}) are isomorphic.

where, T' (equivalently P') is the desired permutation. The pair of permutations will be said to represent the *best matching* permutation. In the sequel, without loss of generality, we will assume that $m \le n$.

Returning to our example in Figure 1, we obtain a compact permutation corresponding to X'_q in steps as follows:

$$(7\ 9\ 11_1\ 11_2\ 0_1\ 0_2\ 11_3) \Rightarrow (3\ 4\ 5\ 6\ 1\ 2\ 7)$$

Similarly, the permutations for X_{mellow} and X_{wool} will be (3 5 6 7 1 2 8 4) and (7 1 4 2 5 3 6 8). It can be verified that the best matching permutation between X'_q and X'_q is of length 7 (all of X'_q) whereas the best matching permutations between X'_q and X_{wool} is of length 5 (e.g. the sub-permutation (3 4 5 6 7) of X'_q).

Algorithm

Our algorithm systematically generates sequence of pairs that satisfy condition (a) by traversing a *host tree* (see Figure 4) in the depth-first order. Note that a node, such as (x, -), in the tree denotes a null matching. For every generated pair, condition (b) is checked for using the following observation about permutations:

Observation: Given two permutations A= (a_1,a_2,...,a_k) and B = (b_1,b_2,...,b_k), let t_q be the number of integers to the left of a_q which are greater that a_q, and s_q be the number of integers to the left of b_q which are greater than b_q. Then A and B are isomorphic iff $t_q = s_q$, for all q.

This observation also implies that if (a_1,a_2,...,a_k) and (b_1,b_2,...,b_k) are isomorphic then so are their corresponding sub-permutations. In particular, (a_1,a_2,...,a_r) and (b_1,b_2,...,b_r) are also isomorphic for any $r \leq k$. Now our approach would be to generate the host-tree from top to bottom (in depth-first order) and never generate a node that gives a sequence corresponding to non-isomorphic initial sub-permutations.

The example in Figure 5 shows the pruned host tree for permutations T = (6 3 2 5 1 7 4) and P = (1 5 3 4 2). In this tree we also did not generate nodes that would have resulted in a shorter than a valid sequence found earlier. Single integer in the parentheses is the length of the sequence associated with that leaf node.

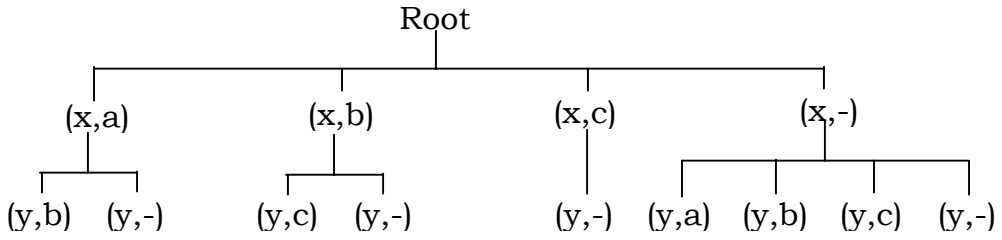


Figure 4: A host tree used for matching of permutations.

The pruning of the above tree occurs at the nodes marked as *no gain.* The host tree is pruned at *no gain 1* because already three "-" pairs are formed and there is no possibility of getting a sequence of length greater than 2. In the *no gain 2* case, a sequence of length 4 has already occurred and since a parent of this node has a "-"node, no sequence of size 5 will occur in this sub-tree. In the *no gain 3* case, 1 has been matched with 5 and there are only 3 more integer after 5 in T so the sequences in this sub-tree can not be of length more than 4 and we already have seen a sequence of length 4. Thus, the resulting longest isomorphic sub-permutations in the example are T' = (2 5 7 4) and P' = (1 3 4 2).

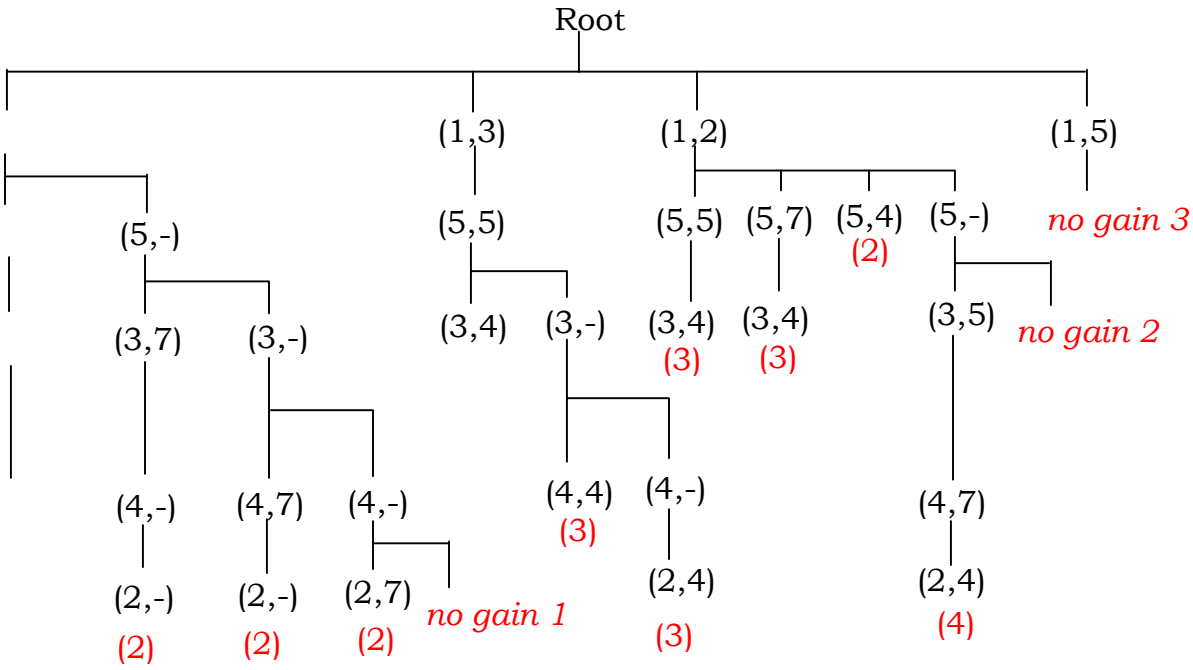


Figure 5: Pruned host tree for permutations T = (6 3 2 5 1 7 4) and P = (1 5 3 4 2).

The order-isomorphism problem can be shown to be NP-complete by reducing the pattern matching for permutations problem [Bose93] to it. Our algorithm, however, is practical as long as the sizes of the two graphs are not too large.

Experimental Design

We report on preliminary experiments on a synthetic data set for the signal graphs designed to evaluate the proposed approach under the following assumptions:

(a) The signal graph represents the result of a bigram-matching process
(b) The matching process is relatively reliable -- although it may miss or misclassify some bigrams, it can identify a large fraction of them correctly.

In a related lexicon-based, indirect symbolic approach, El-Nasan and Nagy [El-Nasan 01] used bigram *occurrences* as the basis for unsegmented text recognition. The approach proposed here imposes a stricter matching criterion than bigram occurrences: the bigrams should not only be common between the two words but also occur in the same order in the two graphs. This additional constraint can result in significantly shorter reference set.

Selection of Lexicon: For our initial experiments, we used a 151-word lexicon from El-Nasan and Nagy [El-Nasan 02], converting the words with both upper and lower cases to the lower case and removing the duplicates. This reduced the set size from 151 to 146 words.

Selection of Reference String: Candidate words for the reference string were chosen from a Unix spell-check dictionary, after removing entries with numerals and upper-case characters and choosing only words of length 5 or more. From the remaining 17,687 words, we randomly selected 200 words for the reference set. The distribution of the lexical-graph size for a reference set of 200 words is shown in Figure 6. The minimum, maximum, average graph sizes of this distribution are 12, 148, and 61 respectively. The 200 words were concatenated in the order of selection to form the *reference string*.

Noise Model: The query graph for a word (assumed to be in the lexicon) was created as a noisy version of the lexical graph for that word using a simple noise model with one normalized parameter p with real value between 0 and 1 representing the magnitude of error. For a lexical graph with e edges the noise model will introduce pe noisy edges as follows: We toss an unbiased coin pe times. Each time, if it is *head,* we randomly select an edge (with index in the range 1 through e) in the original lexical

graph and delete it; if it is *tail*, we randomly select a missing edge from the complementary graph of the lexical graph and insert it. If a selected edge for deletion (insertion) was already deleted (inserted) in a previous operation, we do nothing as these situations will occur only rarely.

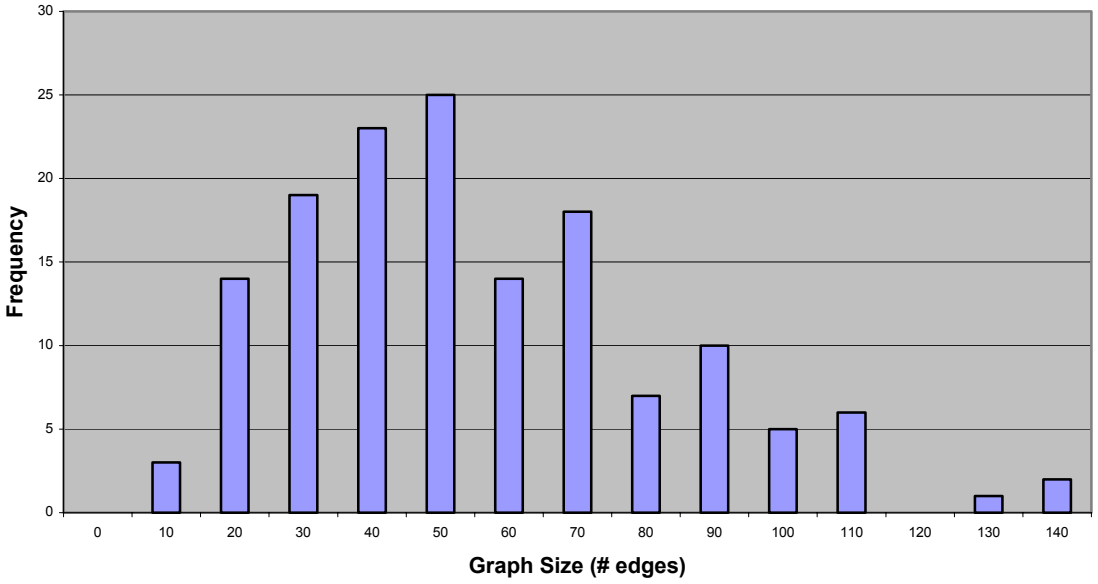


Figure 6: Distribution of the graph size of words in the lexicon.

Match Algorithm: The time complexity of our permutation-matching algorithm is highly data dependent. Generally, the matching time grows non-linearly with the sizes of the two graphs, as well as with their size differences. This observation suggests an iterative approach in which the reference string size grows progressively larger. Further, by choosing a prefix of the reference string initially and extending it to the right by a fixed amount in the subsequent iteration, we ensure that the two graphs used for matching in the previous step are subgraphs of the new graphs. In the results reported below, the initial reference string is chosen to be 10 words long and it is extended by another 10 words if it is necessary to take the next iterative step. At each step, only those lexicon words are matched that were not eliminated as mismatches during a previous step. Thus, this *ambiguity set* of candidate matches can only diminish after each step.

The matching process starts by performing a simple check to eliminate matching two graphs of substantially different sizes. Further pruning of the candidate matches occurs after the matching process. Every candidate has an associated matched subgraph associated with it. If the matched subgraph of a word is too small compared to the best match, that word is also eliminated from further matching.

The criteria stated in the previous paragraph introduce parameters that can be tuned to improve the performance of the algorithm.

Experimental Results and Discussion

Table 1 shows the results of our preliminary experiments. With 10% noise (p=0.1) added to the query graph, the correct recognition rate for the query word was 79.6%, that is, in these cases, the ambiguity set contained only the query words. This recognition rate was achieved with a reference string ranging in size from 10 to 160 words with an average value of 46.5 words. In 19 (12.9%) cases, the query word was eliminated from the ambiguity set(mis-classification). This happened due to too small a size of the (noisy) query graph (fewer than 9 edges with an average size of 4.7 edges.) We conjecture that the error rate can be arbitrarily reduced by increasing the size of the reference string. With 20% noise, the correct recognition rate dropped to 67.3% while the misclassification rate rose to 19.7%.

We expect to be able to include results with larger reference string sizes in the final version of the paper.

Table 1: Results from matching noisy query

% noise (p)	**correctly recognized** (word uniquely in ambiguity set)	**required ref string size (words)**			**misclassified** (word not in ambiguiry set)
		min	max	average	
0.1	117 (79.6%)	10	160	46.5	19 (12.9%)
0.2	99 (67.3%)	10	120	48.7	29 (19.7%)

References

[Bose 93] P. Bose, J. F. Buss, and A. Lubiw, “Pattern Matching for Permutations”, Proc. Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science 709, , Springer Verlag pp. 200-209, New York, 1993.

[El-Nasan 01] A. El-Nasan, S. Veeramachaneni, and G. Nagy, “Word Discrimination Based on Bigram Co-occurences”, 6th ICDAR, pp. 149-153, 2001.

[El-Nasan 02] A. El-Nasan and G. Nagy, “On-Line Handwriting Recognition Based on Bigram Co-occurrences”, ICPR-02.