SYMBOLIC CHANNEL MODELLING FOR NOISY CHANNELS WHICH PERMIT ARBITRARY NOISE DISTRIBUTIONS

B. J. Oommen **and** **R. L. Kashyap**
School of Computer Science, Carleton University, Ottawa ; ONT : K1S 5B6, CANADA, oommen@scs.carleton.ca
School of Electrical Engineering, Purdue University, W. Lafayette ; IN : 47907, USA, kashyap@ecn.purdue.edu

ABSTRACT

In this paper we present a new model for noisy channels which permit arbitrarily distributed substitution, deletion and insertion errors. Apart from its straightforward applications in string generation and recognition, the model also has potential applications in speech and uni-dimensional signal processing. The model is specified in terms of a noisy string generation technique. Let A be any finite alphabet and A* be the set of words over A. Given any arbitrary string U A*, we specify a stochastically consistent scheme by which this word can be transformed into any Y A*. This is achieved by specifying the process by which U is transformed by performing substitution, deletion and insertion operations. The scheme is shown to be Functionally Complete and stochastically consistent. The probability distributions for these respective operations can be completely arbitrary. Apart from presenting the channel in which all the possible strings in A* can be potentially generated, we also specify a technique by which Pr[Y|U], the probability of receiving Y given that U was transmitted, can be computed in cubic time. This procedure involves dynamic programming, and is to our knowledge, among the few non-trivial applications of dynamic programming which evaluate quantities involving relatively complex combinatorial expressions and which simultaneously maintain rigid probability consistency constraints.

I. INTRODUCTION

Syntactic and structural pattern recognition are distinct from statistical pattern recognition because, unlike in the latter, in the former two areas, the processing of the patterns is achieved by representing them symbolically using primitive or elementary symbols. The pattern recognition system essentially symbolically models noisy variations of typical samples of the patterns, and these models are utilized in both the training and testing phases of the system. Thus, the fundamental question in many of these systems is essentially one of modelling the structural behaviour of the patterns. From the point of view of reverse engineering or black-box modelling, this question is, indeed, one of specifying how the individual patterns from the various classes could have been generated. This is the problem studied in this paper.

In this paper we shall present a new model for noisy channels which transfer (or rather, carry) symbolic data, garbling it with ***arbitrarily distributed*** substitution, deletion and insertion errors. To our knowledge, this is the first generalized model of its type. Apart from its straightforward applications in string generation and recognition, like its predecessor [1], the model also has potential applications in speech and uni-dimensional signal processing.

All of text processing involves manipulating the symbols of an alphabet and in almost all cases this alphabet is finite. Once the alphabet for a text processing problem has been defined, the next question of importance is one of understanding the nature of the individual strings that will be processed. In many applications such as natural languages, telephone directories, and the vocabulary used by handicapped individuals the dictionary used is finite. But when the dictionary is prohibitively large, problem analysts tackle the problem by modelling the dictionary appropriately. Typically, it is represented using a stochastic string generation mechanism. The most elementary model (also referred to as the Bernoulli Model) is the one in which only the unigram (single character) probabilities of the dictionary are required [2,6,9,12]. A word in the dictionary is then modelled as a sequence of characters independently drawn from the unigram distribution. Typically, these unigram probabilities approximate the probabilities of the symobls in the original language. A generalization of this is the Markovian Model [1,2,6,8,9, 12,13] where the probability of a particular symbol occurring depends on the previous symbol. Essentially, this model is identical to the one which models the dictionary using the bigrams of the language. Both the Bernoulli Model and the Markovian Model have been used to analyze various pattern matching and keyboard optimization algorithms, and the associated data structures that are encountered, such as suffix trees and their generalizations (See the references listed above).

The problem of modelling the language can be viewed from an entirely different perspective, which is one of viewing the language to be the output of a sequence generator whose input is a string or language itself. Thus, if we permit the system to be operating without any input (or in an "unexcited" mode, as a systems theorist would say) all the above scenarios can be appropriately modelled. Indeed, a finite dictionary is obtained when the unexcited source generator randomly outputs an element from a predefined set of strings. Similarly, the Bernoulli model is obtained when the unexcited source generator generates a sequence characterized by a single probability distribution. Finally, the Markovian Model is obtained when the unexcited source generator generates a sequence characterized by a probability distribution (the distribution of the first character) and a finite stochastic matrix which constitutes the "next character" information.

In this paper, we shall consider the channel as an excited random string generator. Explicitly, we shall consider the channel as a generator whose input is a string U and whose output is the ***random*** string Y. The model for the channel is that Y is obtained by mutating the input string with an arbitrary sequence of string deforming operations. The operations which we shall consider in this paper are the substitution, deletion and insertion operations of the individual symbols of the alphabet. In the literature, these operations are the most popular, because the general string editing problem has been studied using them [1,5,7,11,14], and furthermore, they can also be used to study problems involving subsequences and supersequences [7,10,11,14]. Viewed from a philosophical perspective, our model is a "distant" relative of the ones using the Viterbi algorithm [4,9,13]. It is a generalization of [1], with the advantage that it is functionally complete even though the distribution for the number of insertions is not a mixture of geometric distributions.

Using the notation that U is the input to the channel and that Y is its random output, we list below the novel, salient features of our nodel :

(i) The strategy is Functionally Complete because it involves **all** the ways by which U can be mutated into Y using the edit operations. It is also stochastically consistent.

(ii) The distributions for the various garbling operations can be arbitrary.

(iii) The probability of U being transformed into the same word, Y, even twice can be arbitrarily small.

(iv) For a given U, the length of Y is a random variable whose distribution does not have to be a mixture of geometric distributions.

Apart from its straightforward applications in string generation and string comparison, we believe that just like its "predecessor" [1,13], this scheme will have applications in speech and phoneme generation and processing.

II. NOTATION

Let A be a finite alphabet, and A^* be the set of strings over A. λ A is the null symbol. A string X A^* of the form $X=x_1x_2...x_N$ is said to be of length $|X| = N$. Its prefix of length i will be written as X_i, where $i < N$. Upper case symbols represent strings, and lower case symbols, elements of the alphabet under consideration. The symbol $\approx$ represents the set union operator.

II.1 The Null Symbols ξ and λ and the Compression Operators C_I and C_O

Let Y' be any string in $(A \approx \{\lambda\})^*$, the set of strings over $(A \approx \{\lambda\})$. The string Y' is called an output edit sequence. The operation of transforming a symbol a A to λ will be used to represent the deletion of the symbol a. To differentiate between the deletion and insertion operation, the symbol ξ is introduced. Let X' be any string in $(A \approx \{\xi\})^*$, the set of strings over $(A \approx \{\xi\})$. The string X' is called an input edit sequence . Observe that ξ is distinct from λ, the null symbol, but is used in an analogous way to denote the insertion of a symbol. Thus, the operation of transforming a symbol ξ to b A will be used to represent the insertion of b.

The Output Compression Operator, C_O is a function which maps from $(A \approx \{\lambda\})^*$ to A^*. $C_O(Y')$ is Y' with all the occurrences of the symbol λ removed. Note that C_O preserves the order of the non-λ symbols in Y'. Thus, for example, if Y'=fλoλr, $C_O(Y')$=for. Analogously, the Input Compression Operator, C_I is a function which maps from $(A \approx \{\xi\})^*$ to A^*. $C_I(X')$ is X' with the occurrences of ξ removed. Note that C_I preserves the order of the non-ξ symbols in X'.

II.2 The Set of all Possible Edit Operations : $\Gamma(U,Y)$

For every pair (U,Y), U,Y A^*, the finite set $\Gamma(U,Y)$ is defined by means of the compression operators C_I and C_O, as a subset of $(A \approx \{\xi\})^* \times (A \approx \{\lambda\})^*$ as :

$\Gamma(U,Y) = \{(U', Y') \mid (U', Y') \quad (A \approx \{\xi\})^* \times (A \approx \{\lambda\})^*$, and each (U',Y') obeys (i) - (iii)$\}$ (1)

(i) $C_I(U') = U$; $C_O(Y') = Y$

(ii) $|U'| = |Y'|$

(iii) For all $1 \le i \le |U'|$, it is not the case that $u'_i = \xi$ and $y'_i = \lambda$.

By definition, if (U', Y') $\Gamma(U,Y)$, then, $\text{Max}[|U|, |Y|] \le |U'| = |Y'| \le |U| + |Y|$.

The meaning of the pair (U', Y') $\Gamma(U,Y)$ is that it corresponds to one way of editing U into Y, using the edit operations of substitution, deletion and insertion. The edit operations themselves are specified for $1 \le i \le |Y'|$, as (u'_i,y'_i), which represents the transformation of u'_i, to y'_i. Indeed, if u'_i A and y'_i A, it represents the substitution of y'_i for u'_i, if u'_i A and $y'_i = \lambda$, it represents the deletion of u'_i, and if $u'_i = \xi$ and y'_i A, it represents the insertion of y'_i.

$\Gamma(U,Y)$ is an exhaustive enumeration of the set of all the ways by which U can be edited to Y using the three edit operations without destroying the order of the occurrence of the symbols. Note that we do not permit the channel to delete a symbol it has once inserted or substituted.

Lemma O.

The number of elements in the set $\Gamma(U,Y)$ is given by :

$$\left|\Gamma(U,Y)\right| = \sum_{k=Max(0,|Y|-|U|)}^{|Y|} \frac{(|U|+k)!}{(k!\,(|Y|-k)!\,(|U|-|Y|+k)\,!)} \qquad (2)$$

Proof : The Proof is included in [15].

$\rightarrow\rightarrow\rightarrow$

Note that the size of $\Gamma(U,Y)$ increases combinatorially with the lengths of U and Y. Also, observe that $\Gamma(U,Y)$ includes duplicate entries for the same edit operations. Thus, if U="f" and Y="go", the entries which represent the same edit operations {(fξξ, λgo),(ξfξ, gλo),(ξξf, goλ)} will be in $\Gamma(U,Y)$. The difference between them is the **sequence** in which the operations occur.

III. MODELLING -- THE STRING GENERATION PROCESS

We now describe the process of generating a string Y given an input string U A*.

First of all we assume that the model utilizes a probability distribution G over the set of positive integers. The random variable in this case is referred to as *Z* and is the number of insertions that are performed in the mutating process. G is called the ***Quantified*** Insertion Distribution, and in the most general case, can be conditioned on the input string U. The quantity G(z|U) is the probability that *Z* =z given that U is the input word. Since G is a distribution, the sum of G(z|U) over all valid values of z should be unity. Examples of G are the Poisson and Geometric Distributions. However, G can be arbitrarily general.

The second distribution that the model utilizes is the probability distribution Q over the alphabet under consideration. Q is called the ***Qualified*** Insertion Distribution. The quantity Q(a) is the probability that a A will be the inserted symbol conditioned on the event that an insertion operation is to be performed. Indeed, the sum of Q(a) over all the elements of A should be unity.

The final distribution which the model utilizes is a distribution S over A X (A ≈ {λ}) called the Substitution and Deletion Distribution. The quantity S(b|a) is the conditional probability that given the symbol a A in the input string is mutated by a stochastic substitution or deletion -- in which case it will it will be transformed into a symbol b (A ≈ {λ}). Since S is a distribution, the sum of S(b|a) over all b (A ≈ {λ}) should be unity.

Using the above distributions we now informally describe the model for the garbling mechanism (or equivalently, the string generation process). Let |U| = N. Using the distribution G, the generator randomly determines the number of symbols to be inserted. Let *Z* be random variable denoting the number of insertions that are to be inserted in the mutation. Based on the random choice of *Z* let us assume that *Z* takes the value z. The algorithm then determines the position of the insertions among the individual symbols of U. This is done by randomly generating an input edit sequence U' (A ≈ {λ})*. For the sake of simplicity, we assume that each of the (N+k)! / (N! k!) possible strings are equally likely.

Since $C_I(U')$ is U, the positions of the ξ in U' represent the positions where symbols will be inserted into U. The non-ξ symbols in U' are now substituted for or deleted using S. Finally, the occurrences of ξ are transformed independently into the individual symbols of A using Q.

This defines the string generation process completely. It is formally given as Algorithm GenerateString below, and a graphical model of the channel for these operations is given in [15].

Algorithm GenerateString
Input : The word U and the distributions G, Q and S.
Output : A random string Y which garbles U with substitution, insertion and deletion mutations.
Method:

1. Using G randomly determine z, the number of symbols to be inserted in U.
2. Randomly generate an input edit sequence U' $(A \approx \{\xi\})^*$ by randomly determining the positions of the insertions among the individual symbols of U.
3. Randomly independently substitute or delete the non-ξ symbols in U' using S.
4. Randomly independently transform the occurrences of ξ into symbols of A using Q.

END Algorithm GenerateString

We now derive the properties of the string generation process.

THEOREM I

Let |U| = N and |Y| = M, and let Pr[Y|U] be defined as follows :

$$\Pr[Y|U] = \sum_{z=Max(0,M-N)}^{M} \frac{G(z).\,(N!\,z!)}{((N+z)!)} \sum_{U'} \sum_{Y'} \prod_{i=1}^{N+z} p(y'_i|u'_i) \quad \text{, where,} \qquad (3)$$

(a) y'_i and u'_i are the individual symbols of Y' and U' respectively,

(b) $p(y'_i|u'_i)$ is interpreted as $Q(y'_i)$ if u'_i is ξ, and ,

(c) $p(y'_i|u'_i)$ is interpreted as $S(y'_i|u'_i)$ if u'_i is not ξ. (4)

Then the above definition is both functionally complete and consistent.

Sketch of Proof : The theorem is quite intricate and is proved in [15] . It involves computing the product of the probabilities of the individual elements of every single pair in $\Gamma(U,Y)$. Thus, for every element (U', Y') the product of the individual probabilities is its contribution to Pr[Y|U]. We thus exhaustively add all the probability contributions of the various ways by which U can be mutated to Y. This is then further summed for all elements of A^* by considering the cases for the various permissible values of *Z* . By performing the summations in a systematic way and grouping the cominatorial terms intelligently it can be proved that the total summation equals unity.

$\rightarrow\rightarrow\rightarrow$

We shall now describe how the probability Pr[Y|U] can be computed efficiently.

IV. COMPUTING P[Y|U] EFFICIENTLY

Consider the problem of editing U to Y, where |U|=N and |Y|=M. Suppose we edit a prefix of U into a prefix of Y, using exactly i insertions, e deletions and s substitutions. Since the number of edit operations are specified, this corresponds to editing $U_{e+s} = u_1 . . . u_{e+s}$, the prefix of U of length e+s, into $Y_{i+s}=y_1 . . . y_{i+s}$, the prefix of Y of length i+s. Let $\Pr[Y_{i+s}|U_{e+s} ; Z=i]$ be

the probability of obtaining Y_{i+s} given that U_{e+s} was the original string, and that exactly i insertions took place in garbling. Then, by definition,

$$\Pr[Y_{i+s}|U_{e+s} ; Z=i] = 1 \qquad \text{if } i=e=s=0$$

To obtain an explicit expression for the above for values of i, e and s which are nonzero, we have to consider all the possible ways by which Y_{i+s} could have been obtained from U_{e+s} using exactly i insertions. Let r=e+s and q=i+s. Let $\Gamma_{i,e,s}(U,Y)$ be the subset of the pairs in $\Gamma(U_r,Y_q)$ in which every pair corresponds to i insertions, e deletions and s substitutions. Since we shall be using the strings U and Y, $\Gamma_{i,e,s}(U,Y)$ will be referred to as $\Gamma_{i,e,s}$. Using (3) and (4),

$$\Pr[Y_{i+s}|U_{e+s} ; Z=i] = \frac{(s+e)!\ i!}{(s+e+i)!} \sum_{(U'_r,Y'_q)} \prod_{j=1}^{|U'_r|} p(y'_{qj}|u'_{rj}) , \qquad \text{if i, e or s} > 0 \quad (5)$$

where, (U'_r,Y'_q) is the arbitrary element of $\Gamma_{i,e,s}$, and has jth symbols u'_{rj} and y'_{qj} respectively.

Let W(. , . , .) be the array whose general element W(i,e,s) is the sum of the product of the probabilities associated with the general element of $\Gamma_{i,e,s}$ defined as below.

$$W(i,e,s) = 0, \qquad \text{if i,e or s} < 0$$

$$= \frac{(s+e+i)!}{i!\ (s+e)!} \Pr[Y_{i+s}|U_{e+s} ; Z=i] \qquad \text{otherwise} \qquad (6)$$

Using the expression for $\Pr[Y_{i+s}|U_{e+s} ; Z=i]$ we can obtain the explicit form of W(i,e,s) for all nonnegative values of i, e and s. The bounds for these indices are :

$$\text{Max}[0,M-N] \le i \le q \le M ; \qquad 0 \le e \le r \le N ; \qquad 0 \le s \le \text{Min}[M,N].$$

Triples (i,e,s) satisfying these constraints are termed "feasible" and satisfy Theorem II, and the recursively computable properties of W(.,.,.) are stated in Theorem III.

THEOREM II.

To edit U_r, the prefix of U of length r, to Y_q, the prefix of Y of length q, the set of feasible triples is given by $\{ (i, r-q+i, q-i) \mid \text{Max } [0, q-r] \le i \le q \}$.

Proof : The theorem is proved in [15].

$\rightarrow\rightarrow\rightarrow$

THEOREM III.

Let W(i,e,s) be the quantity defined as in (6) for any two strings U and Y. Then, for all feasible values of i,e and s,

$$W(i,e,s) = W(i-1,e,s).p(y_{i+s}|\xi) + W(i,e-1,s).p(\lambda|u_{e+s}) + W(i,e,s-1).p(y_{i+s}|u_{e+s})$$

where p(b|a) is interpreted as in (4).

Sketch of Proof : The proof of the result is quite involved. Let $\Gamma_{i,e,s}$ be the set of all ways by which U_{e+s} can be edited to Y_{i+s}. The proof of the theorem involves partitioning $\Gamma_{i,e,s}$ into three subsets each of which strips off the elements of the individual pairs as follows :

$$\Gamma^1_{i,e,s} = \{ (U'_r,Y'_q) \mid (U'_r,Y'_q) \quad \Gamma_{i,e,s}, u'_{rL} = u_r, y'_{qL} = y_q \} \quad (7)$$

$$\Gamma^2_{i,e,s} = \{ (U'_r,Y'_q) \mid (U'_r,Y'_q) \quad \Gamma_{i,e,s}, u'_{rL} = u_r, y'_{qL} = \lambda \} \quad (8)$$

$$\Gamma^3_{i,e,s} = \{ (U'_r,Y'_q) \mid (U'_r,Y'_q) \quad \Gamma_{i,e,s}, u'_{rL} = \xi, y'_{qL} = y_q \}. \quad (9)$$

After considerable manipulations, it can be shown that the contributions of each of the sets lead to the following quantities respectively :

$$\sum_{(U'_r,Y'_q)\ (\Gamma^1_{i,e,s})} \prod_{j=1}^{|U'_r|} p(y'_{qj}|u'_{rj}) = W(i,e,s-1).p(y_{i+s}|u_{e+s}),$$

$$\sum_{(U'_r,Y'_q)\ (\Gamma^2_{i,e,s})} \prod_{j=1}^{|U'_r|} p(y'_{qj}|u'_{rj}) = W(i,e-1,s).p(\lambda|u_{e+s}),$$

$$\sum_{(U'_r,Y'_q)\ (\Gamma^3_{i,e,s})} \prod_{j=1}^{|U'_r|} p(y'_{qj}|u'_{rj}) = W(i-1,e,s).p(y_{i+s}|\xi).$$

Resubstituting these expressions into the original expressions proves the result.

$\rightarrow\rightarrow\rightarrow$

The computation of $Pr[Y|U]$ from W(i,e,s) is done by evaluating a combination of the appropriate terms weighted by factors that are dependent only on the number of insertions. This is stated in Theorem IV, whose proof is found in [15]. Thus, to evaluate $Pr[Y|U]$ we make use of the fact that although the latter index itself does not seem to have any recursive properties, the index W(. , . , .), has the properties proved in Theorem III and subsequently utilize Theorem IV to yield $Pr[Y|U]$.

THEOREM IV

If $h(i)= G(i) \cdot \frac{N!\ i!}{(N+i)!}$, the quantity $Pr[Y|U]$ can be evaluated from the array W(i,e,s) as :

$$Pr[Y|U] = \sum_{i=Max(0,M-N)}^{M} h(i).W(i, N-M+i, M-i).$$

Proof : The proof of the theorem is given in [15].

$\rightarrow\rightarrow\rightarrow$

The Algorithm EvaluateProbabilities below evaluates the array W(. , . , .) for all permissible values of the variables i, e and s. Using the array W(i,e,s) it then evaluates $Pr[Y|U]$ by adding up the weighted contributions of the pertinent elements in W(., ., .) as specified by Theorem IV.

Algorithm EvaluateProbabilities

Input: The strings $U=u_1u_2. .\ u_N$, $Y=y_1y_2. .\ y_M$, and the distributions G, Q and S.

Output: The array W(i,e,s) for all permissible values of i, e and s and the probability Pr[Y|U].

Method :

R=Min[M,N]

W(0,0,0)=1

$\Pr[Y|U] = 0$

For i=1 to M **Do**

  $W(i,0,0) = W(i-1,0,0).\ Q(y_i)$

For e=1 to N **Do**

  $W(0,e,0) = W(0,e-1,0).S(\lambda|u_e)$

For s=1 to R **Do**

  $W(0,0,s) = W(0,0,s-1).S(y_s|u_s)$

For i=1 to M **Do**

  For e=1 to N **Do**

    $W(i,e,0) = W(i-1,e,0).Q(y_i) + W(i,e-1,0).S(\lambda|u_e)$

For i=1 to M **Do**

  For s=1 to M-i **Do**

    $W(i,0,s) = W(i-1,0,s).Q(y_{i+s}) + W(i,0,s-1).S(y_{i+s}|u_s)$

For e=1 to N **Do**

  For s=1 to N-e **Do**

    $W(0,e,s) = W(0,e-1,s).S(\lambda|u_{s+e}) + W(0,e,s-1).S(y_s|u_{s+e})$

For i=1 to M **Do**

  For e=1 to N **Do**

    For s=1 to Min[(M-i) , (N-e)] **Do**

      $W(i,e,s)=W(i-1,e,s).Q(y_{i+s})+W(i,e-1,s).S(\lambda|u_{e+s})+W(i,e,s-1).S(y_{i+s}|u_{e+s})$

For i=Min[0 , M-N] to M **Do**

$$\Pr[Y|U] = \Pr[Y|U] + G(i) \cdot \frac{N!\ i!}{(N+i)!} \cdot W(i,N-M+i,M-i)$$

END Algorithm EvaluateProbabilities

Obviously, the algorithm requires cubic time and space respectively. To compute $\Pr[Y|U]$, we can get a more efficient[1] (but more complicated) scheme, which requires only quadratic space. To do so, we take advantage of the fact that for a particular value of i, in order to compute W(i,e,s) for all permissible values of e and s, it is sufficient to store only the values of W (i-1, e, s) for all the corresponding permissible values of e and s. Thus the trellis is computed by successively evaluating the array W in planes parallel to the plane i = 0. The algorithm is given in [15], but omitted here in the interest of brevity.

From a naive perspective it is possible to consider the techniques applied here as mere application of dynamic programming algorithms [5,7,10,11,14] to the case when the operators utilized are the arithmetic addition and multiplication. This is not the case. The differences are listed in [15]. But in all brevity, we mention that to our knowledge, this is one of the few non-trivial applications of dynamic programming which evaluates quantities which involve relatively complex combinatorial expressions simultaneously satisfying rigid probability constraints. We are currently studying the use of this channel in speech recognition. Note that this a model, the entire question of "time warping" would be subsumed in appropriately modelling the distribution G.

---

[1]The next algorithm is more efficient in space. With respect to time the computational complexity of both are the same. However, for small values of M and N, the earlier algorithm is more efficient, because of the decreased overhead and book-keeping.

V. CONCLUSIONS

In this paper we have presented a new model for noisy channels which permit arbitrarily distributed substitution, deletion and insertion errors. This model has straightforward applications in string generation and recognition, and also potential applications in speech and uni-dimensional signal processing. The model is specified in terms of a noisy string generation technique. Given any arbitrary string U A^*, we specify a stochastically consistent scheme by which this word can be transformed into any Y A^* by causing substitution, deletion and insertion operations. The scheme has been shown to be Functionally Complete because it involves all the ways by which U can be mutated into Y using these three operations. The probability distributions for these respective operations can be completely arbitrary. Apart from presenting a scheme by which all the possible strings in A^* can be potentially generated, we also specify two cubic-time algorithm by which $\Pr[Y|U]$, the probability of receiving Y given that U was transmitted, can be computed. The first of these requires cubic space, and the second requires only quadratic space. For small values of M and N, the former is more efficient, because of the decreased overhead and book-keeping.

Acknowledgements : This work was partially supported by the Natural Sciences and Engineering Research Council of Canada

REFERENCES

1. R. L. Bahl and F. Jelinek, Decoding with channels with insertions, deletions and substitutions with applications to speech recognition, *IEEE Trans. Information Theory*, IT-21:404-411 (1975).
2. L. Devroye, W. Szpankowski and B. Rais, A note on the height of suffix trees, *SIAM J. of Computing*, 21:48-54, (1992)
3. G. Dewey, *Relative Frequency of English Speech Sounds*, Cambridge, MA, Harvard Univ. Press, (1923).
4. G.D. Forney, The Viterbi Algorithm, *Proceedings of the IEEE*, Vol. 61. (1973)
5. P. A. V. Hall and G.R. Dowling, Approximate string matching, *Comput. Surveys*, 12:381-402 (1980).
6. P. Jacquet and W. Szpankowski, Analysis of digital tries with markovian dependencies, *IEEE Trans. Information Theory*, IT-37:1470-1475 (1991).
7. W. J. Masek and M. S. Paterson, A faster algorithm computing string edit distances, *J. Comput. System Sci.*, 20:18-31 (1980).
8. S. L. Minneman, Keyboard optimization technique to improve output rate of disabled individuals, *Proc. of the 9th. Annual RESNA Conference*, Minneapolis, 402-404 (1986).
9. D. L. Neuhoff, The Viterbi algorithm as an aid in text recognition, *IEEE Trans. Information Theory*, 222-226 (1975).
10. B. J. Oommen, Recognition of noisy subsequences using constrained edit distances, *IEEE Trans. on Pattern Anal. and Mach. Intel.*, PAMI-9:676-685 (1987).
11. D. Sankoff and J. B. Kruskal, *Time Warps,String Edits and Macromolecules: The Theory and practice of Sequence Comparison*, Addison-Wesley (1983).
12. W. Szpankowski, Probabilistic analysis of generalized suffix trees, *Proceedings of CPM-92, The Third Annual Symposium on Combinatorial Pattern Matching*, 1-14 (1992).
13. A. J. Viterbi, Error bounds for convolutional codes and an asymptotically optimal decoding algorithm, *IEEE Trans. on Information Theory*, 260-26 (1967).

14. R. A. Wagner and M. J. Fisher, The string to string correction problem, *J. Assoc. Comput. Mach.*, 21:168-173 (1974).
15. B. J. Oommen and R. L. Kashyap, A consistent model for noisy channels permitting arbitrarily distributed substitutions, insertions and deletions. Submitted for Publication.