A CONSISTENT MODEL FOR NOISY CHANNELS PERMITTING ARBITRARILY DISTRIBUTED SUBSTITUTIONS, INSERTIONS AND DELETIONS

B.J. Oommen and R.L. Kashyap

TR-224, JUNE 1993

School of Computer Science, Carleton University
Ottawa, Canada, KIS 5B6

A CONSISTENT MODEL FOR NOISY CHANNELS PERMITTING ARBITRARILY DISTRIBUTED SUBSTITUTIONS, INSERTIONS AND DELETIONS+

B. J. Oommen[1] and R. L. Kashyap[2]

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 $U \in A^*$, we specify a stochastically consistent scheme by which this word can be transformed into any $Y \in A^*$. 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.

Keywords : Noisy channel modelling, String Generation, Unigram and Bigram Model, Sequence Generation, Markovian Sequence Generation.

+ Partially supported by the Natural Sciences and Engineering Research Council of Canada.

[1]Senior Member IEEE. Can be contacted at : School of Computer Science, Carleton University, Ottawa ; Canada : K1S 5B6.

[2]Fellow IEEE. Can be contacted at : School of Electrical Engineering, Purdue University, W. Lafayette ; Indiana : 47907.

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 [2], 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. For example, the most restricted alphabet is the binary set {0,1}, and the alphabet encountered for English text is the set of 26 characters {a...z}. To distinguish between the words of a language, customarily, various punctuation marks have been defined, the most common one being the delimiter "space". Of course, other punctuation marks such as the comma, semicolon, etc. are not only needed to distinguish the boundaries of the words, but also to impart a semantic implication to the text. In speech applications, the individual symbols are the set of phonemes [2,33] and in the recognition of noisy macro-molecules, the individual symbols are the underlying aminoacids [33].

Once the alphabet for a text processing problem (or application) has been defined, the next question that is of importance is one of understanding the nature of the individual words or strings that will be processed. We shall briefly catalogue each of the options reported in the literature.

In many real-life applications the dictionary used is finite. This is especially true in the case of natural languages, telephone directories, and even the vocabulary used by hospitalized handicapped individuals. Indeed, even in the case of written English text, various studies have been made which indicate that large proportions of the words used in English form a very small subset of the possible English words. In fact, Dewey [5] has compiled such a collection and claimed that this collection, consisting of 1023 words, comprises a very large proportion of written English text. Thus, in both string processing and string recognition it is not uncommon to represent the dictionary as a finite set of words, and using this model, string correction can be achieved using a suitable similarity metric [8,13-17,20-22,25,27-30,33,35]. The advantages of using a finite

dictionary in text recognition applications are many. First of all, the accuracy of the recognition is very high. Secondly, a noisy string is never recognized as a word which is not in the language, and thus, the question of "meaningless" decisions is irrelevant. Finally, the time complexity of the computation involved in the text recognition process is typically quadratic per word and is linear in the size of the dictionary. The quadratic complexity per word can be often decreased if the dictionary is modelled using a trie [16], and if the alphabet size is decreased [1,39].

When the dictionary is prohibitively large, problem analysts tackle the problem by modelling the dictionary (the output of the channel) differently. Typically, it is represented using a stochastic string generation mechanism. The most elementary model is the one in which only the unigram (single character) probabilities of the dictionary are required [4,12,26,36]. This model is also referred to as the Bernoulli Model. A word in the dictionary is then modelled as a sequence of characters, where each character is independently drawn from a distribution referred to as the unigram distribution. Typically, these unigram probabilities are chosen to be the probabilities of the letters occurring in the original language. A generalization of this is the Markovian Model [2,4,12,18,19,23,24,26, 34-37] 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. A word in the dictionary is modelled as a sequence of symbols where two subsequent symbols $x_i x_{i+1}$ occur with the probability with which they occur in 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 output of the channel 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 running 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 these operations [2,8,11,13-17,21,22,27,30,34,35,38,39], and furthermore, these operations can also be used to study problems involving subsequences and supersequences [1,10,11,22,33]. Also, since most errors found in text-based systems are substitution, deletion and insertion errors (and even the common transposition error can be modelled easily as a sequence of a deletion and insertion error) systems which correct these errors have been designed to recognize noisy strings and substrings [2,8,13-17,21,22,27,30,34,35,38] and even noisy subsequences [28,29,32]. Viewed from the perspective of these edit operations, our model is a "distant" relative of the ones which use the Viterbi algorithm [6,26,34,37].

Our paper is a generalization of the classic paper of Bahl *et. al.* [2]. In addition to the properties of the channel described in [2], ours is functionally complete even though the distribution for the number of insertions is not necessarily a mixture of geometric distributions.

Throughout this paper, for the sake of simplicity, we shall use the terms "model" and "generator" interchangeable. Using the notation that U is the input to the channel (string generator) and that Y is its random output, we list below the novel, salient features of our model :

(i) The model is Functionally Complete because it involves ***all*** the ways by which U can be mutated into Y using the three elementary edit operations. We shall show that the number of ways by which U can be transformed into Y is a combinatorially "explosive" large number. Furthermore, it is a stochastically consistent scheme, and thus,

$$\sum_{Y \in A^*} \Pr[Y|U] = 1.$$

(ii) The distributions for the various garbling operations can be completely arbitrary. These constitute the parameters of the generator (model). Note that these parameters are not merely "real numbers", but arbitrary distributions.

(iii) The probability of a particular word U being transformed into the same word Y even twice in the same way can be made 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.

(v) If the input U is itself an element of a finite dictionary, and the string generator is used to model the noisy channel, the technique for computing the probability Pr [Y|U] can be utilized in a Bayesian way to compute the *a posteriori* probabilities, and thus yield a minimum probability of error pattern classification rule.

Apart from its straightforward applications in string generation and string comparison, we believe that just like its "predecessor" [2,37], this scheme also has potential applications in speech and phoneme generation and processing. We shall first introduce the notation used.

II. NOTATION

Let A be a finite alphabet, and A^* be the set of strings over A. $\lambda \notin A$ is the null symbol. A string $X \in A^*$ 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 $\cup$ represents the set union operator.

II.1 The Input and Output Null Symbols : ξ and λ

Let Y' be any string in $(A \cup \{\lambda\})^*$, the set of strings over $(A \cup \{\lambda\})$. The string Y' is called an output edit sequence. The operation of transforming a symbol $a \in A$ 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 $(A \cup \{\xi\})^*$, the set of strings over $(A \cup \{\xi\})$. 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 $b \in A$ will be used to represent the insertion of the symbol b.

II.2 The Input and Output Compression Operators : C_I and C_O

The Output Compression Operator, C_O is a mathematical function which maps from $(A \cup \{\lambda\})^*$ 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 $Y'=f\lambda o\lambda r$, $C_O(Y')=for$.

Analogously, the Input Compression Operator, C_I is a mathematical function which maps from $(A \cup \{\xi\})^*$ to A^*. $C_I(X')$ is X' with all the occurrences of the symbol ξ removed. Again, note that C_I preserves the order of the non-ξ symbols in X'.

II.3 The Set of all Possible Edit Operations : $\Gamma(U,Y)$

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

$\Gamma(U,Y) = \{(U', Y') \mid (U', Y') \in (A \cup \{\xi\})^* \times (A \cup \{\lambda\})^*$, 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 $(U', Y') \in \Gamma(U,Y)$, then, $\text{Max}[\,|U|, |Y|\,] \le |U'| = |Y'| \le |U| + |Y|$.

The meaning of the pair $(U', Y') \in \Gamma(U,Y)$ 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. The cases below consider the three edit operations individually.

(i) If $u'_i \in A$ and $y'_i \in A$, it represents the substitution of y'_i for u'_i.

(ii) If $u'_i \in A$ and $y'_i = \lambda$, it represents the deletion of u'_i. Between these two cases all the symbols in U are accounted for.

(iii) If $u'_i = \xi$ and $y'_i \in A$, it represents the insertion of y'_i. Between cases (i) and (iii) all the symbols in Y are accounted for.

$\Gamma(U,Y)$ is an exhaustive enumeration of the set of all the ways by which U can be edited to Y using the edit operations of substitution, insertion and deletion without destroying the order of the occurrence of the symbols in U and Y. 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 :

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

Proof :

First of all note that $|\Gamma(U,Y)|$ depends only on $|U|$ and $|Y|$, and not on the actual strings U and Y themselves. Further, observe that the transformation of a symbol $a \in A$ to itself is also considered as an operation in the arbitrary pair $(U', Y') \in \Gamma(U,Y)$. With this in mind, it is easy to see that if k insertions are permitted, the number of possible strings U' is #(Possible U'), where,

$$\#(\text{Possible U'}) = \binom{|U|+k}{k}.$$

For each element U' which represents k insertions, any corresponding element Y' must contain exactly $(|U|-|Y|+k)$ deletions, and these must be chosen from among the symbols of U. This can be done in #(Possible Y') ways, where,

$$\#(\text{Possible Y'}) = \binom{|U|}{|U|-|Y|+k}.$$

The product of these two quantities yields the result for every value of k. The lemma follows by summing the above product for all permissible values of k. ♦♦♦

Remarks

1. Note that the size of $\Gamma(U,Y)$ increases combinatorially with the lengths of U and Y. Thus, it is interesting to note that whereas $\Gamma(3,3)$ has 63 elements, $\Gamma(4,4)$, $\Gamma(5,5)$ and $\Gamma(6,6)$ have 321, 1683 and 8989 elements respectively.

2. The reader must observe that we have chosen to use $\Gamma(U,Y)$ to represent the set of all ways by which U can be transformed to Y, and this set represents many duplicate entries in terms of the edit operations themselves. Thus, consider the case when U = "f" and Y = "go". Then,

$\Gamma(U,Y) = \{ (f\xi, go), (\xi f, go), (f\xi\xi, \lambda go), (\xi f\xi, g\lambda o), (\xi\xi f, go\lambda) \}$.

In particular, the pair $(\xi f, go)$ represents the edit operations of inserting the 'g' and replacing the 'f' by an 'o'. Notice that **all the last three pairs** represent the deletion of 'f' and the insertion of both 'g' and 'o'. The difference between the three is the **sequence** in which these operations are accomplished.

III. MODELLING -- THE STRING GENERATION PROCESS

We now describe the model by which a string Y is generated given an input string $U \in A^*$.

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. Thus, G has to satisfy the following constraint :

$$\sum_{z \geq 0} G(z|U) = 1. \tag{4}$$

Examples of the distribution G are the Poisson and the Geometric Distributions whose parameters depend on the word or the length of the input word. However, the distributions can be arbitrarily general.

Although, in general, Z is a random variable whose value depends on the input, U, for the sake of simplicity, for the rest of this paper we shall assume that Z is independent of U.

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 $a \in A$ will be the inserted symbol conditioned on the fact that an insertion operation is to be performed. Note that Q has to satisfy the following constraint :

$$\sum_{a \in A} Q(a) = 1. \tag{5}$$

Apart from G and Q, the final distribution which the model utilizes is a probability distribution S over $A \times (A \cup \{\lambda\})$. S is called the Substitution and Deletion Distribution. The quantity $S(b|a)$ is the conditional probability that given the symbol $a \in A$ in the input string is mutated by a stochastic substitution or deletion -- in which case it will it will be transformed into a symbol $b \in (A \cup \{\lambda\})$. Hence, $S(c|a)$ is the conditional probability of $a \in A$ being substituted for

by $c \in A$, and analogously, $S(\lambda|a)$ is the conditional probability of $a \in A$ being deleted. Observe that S has to satisfy the following constraint for all $a \in A$:

$$\sum_{b \in (A \cup \{\lambda\})} S(b|a) = 1. \tag{6}$$

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[3] 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 $U' \in (A \cup \{\xi\})^*$. For this paper we assume that each of the $\binom{N+k}{k}$ possible strings are equally.

Note that $C_I(U')$ is U and that the positions of the symbol ξ in U' represents the positions where symbols will be inserted into U. The non-ξ symbols in U' are now substituted for or deleted using the distribution S. Finally, the occurrences of ξ are transformed independently into the individual symbols of the alphabet using the distribution Q.

This defines the model completely. The process followed by the model is formally given as Algorithm GenerateString below.

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 $U' \in (A \cup \{\xi\})^*$ 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

A graphical display of the channel modelling the garbling process is shown in Figure I. An example will help clarify the string generating process.

[3]We assume that the user is capable of generating non-uniform random variables having the respective distributions G, Q and S. An excellent treatise on the subject is the one due to Devroye [3].

Example I.

Let U = "uli". Let the number of insertions, dictated by the distribution G, be 2. The positions of the two insertions is now randomly chosen out of the 10 possible positions. Let us suppose the resultant string is U'= "uξlξi". The non-ξ symbols of U' are now randomly substituted for or deleted using the distribution S. Let us suppose that 'u' gets transformed to 'u', 'l' gets transformed to 'd' and 'i' is deleted. The new string that is to be operated on is thus U'= "uξdξ". The ξ's in U' are now transformed into the symbols of the alphabet A using the distribution Q. Let us suppose the first ξ gets changed into a 'g' and the second ξ gets transformed into a 'k'. The resultant string Y, which is the final garbled version of U is thus Y = "ugdk". ♦♦♦

Let |U| = N and |Y| = M. Then, using the above notation, the following results are true :

THEOREM I

Let Pr[Y|U] be defined as follows :

$$\Pr[Y|U] = \sum_{z=\mathrm{Max}(0,M-N)}^{M} \frac{G(z).\,(N!\,z!)}{((N+z)!)} \sum_{U'} \sum_{Y'} \prod_{i=1}^{N+z} p(y'_i|u'_i)\,. \qquad (7)$$

where,

(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. (8)

Then the above definition is both functionally complete and consistent.

Proof :

The functional completeness is clear since the definition of the quantity Pr[Y|U] essentially involves computing the product of the probabilities of the individual elements of ***every single pair*** in Γ(U,Y). Thus, for every element (U', Y') the product of the individual probabilities is its contribution to Pr[Y|U].

The consistency of the definition is, however, a little more difficult to see. It involves proving that the value of the infinite summation :

$$\sum_{Y \in A^*} \Pr[Y|U]$$

is exactly unity. This may be an elementary exercise for an experienced probabilist, since, in one sense it follows from the basic laws of probability. However, it is definitely not obvious.

To prove the lemma formally, we shall first go through the mechanics of explicitly writing down the expression for the probability Pr[Y|U]. This is done by exhaustively summing up all the

probability contributions of the various ways by which U could have been transformed to Y. Notice that the information about this set of ways is contained in $\Gamma(U,Y)$. Let τ be the summation of this quantity over all the possible values of Y. We intend to prove that τ is exactly unity.

We shall prove this by successively considering the case when $Z = z$. Consider the set of strings that can be obtained by having z insertions occur in the mutation. Indeed, since inserted symbols cannot be subsequently deleted, whenever z insertions occur, the set of all strings that can be obtained is the union of the sets A^j, where j takes the value from z to N+z. Let ${}_zH_{N+z}$ be this set and let τ_z be :

$$\tau_z = \sum_{{}_zH_{N+z}} \frac{(N!\ z!)}{((N+z)!)} \Pr[Y|U\ ;\ Z=z]. \tag{9}$$

Indeed, if with no loss of generality we assume that z can be any non-negative integer, we have :

$$\tau = \sum_{Z=0}^{\infty} G(z)\ \tau_z. \tag{10}$$

Notice now that for each string $Y \in {}_zH_{N+z}$ the number of insertions must be bounded by Max(0,|Y|-N) and |Y|. Thus,

$$\sum_{Y\in {}_zH_{N+z}} \Pr[Y|U\ ;\ Z=z] = \sum_{Y\in {}_zH_{N+z}} \sum_{U'} \sum_{Y'} \prod_{i=1}^{N+z} p(y'_i|u'_i), \tag{11}$$

$$= \sum_{Y\in {}_zH_{N+z}} \sum_{(U',Y')\in(\Gamma_{U,Y})} \prod_{i=1}^{N+z} p(y'_i|u'_i)). \tag{12}$$

But for each U', the last product is over all the letters of the finite alphabet as in (8). Hence,

(i) $p(y'_i|u'_i)$ is $Q(y'_i)$ if u'_i is ξ, and, summed over all y'_i this quantity is unity, and,

(ii) $p(y'_i|u'_i)$ is interpreted as $S(y'_i|u'_i)$ if u'_i is not ξ, and, summed over all u'_i ***this*** is unity.

Hence, for every element U' this sums to unity, and since, for each z, there are $\binom{N+z}{z}$ elements of U', (12) has the value $\frac{(N+z)!}{N!\ z!}$. Hence, τ_z has the value unity. Consequently τ itself is unity because G is a valid distribution in itself. Hence the theorem ! ♦♦♦

We shall now describe how the probability Pr[Y|U] can be computed without explicitly individually evaluating the contribution of all the elements of $\Gamma(U,Y)$.

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 $U_{e+s} = u_1 \ldots u_{e+s}$, the prefix of U of length e+s, into $Y_{i+s}=y_1 \ldots y_{i+s}$, 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,

$$Pr[Y_{i+s}|U_{e+s} ; Z=i] = 1 \qquad \text{if } i=e=s=0 \tag{13}$$

To obtain an explicit expression for the above quantity 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 consistently be using the strings U and Y, $\Gamma_{i,e,s}(U,Y)$ will be referred to as $\Gamma_{i,e,s}$. Using (7) and (8),

$$Pr[Y_{i+s}|U_{e+s} ; Z=i] = \frac{(s+e)!\ i!}{(s+e+i)!} \sum_{(U'_r,Y'_q)} \prod_{j=1}^{|U'_r|} p(y'_{qj}|u'_{rj}), \qquad \text{if i, e or s} > 0 \tag{14}$$

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

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

$$= \frac{(s+e+i)!}{i!\ (s+e)!} Pr[Y_{i+s}|U_{e+s} ; Z=i] \qquad \text{otherwise} \tag{15}$$

Using the expression for $Pr[Y_{i+s}|U_{e+s} ; Z=i]$ we obtain the explicit form of W(i,e,s) for all nonnegative values of i, e and s as in (16).

$$W(i,e,s) = 1, \qquad \text{if i =e =s =0}$$

$$= \sum_{(U'_r,Y'_q)} \prod_{j=1}^{|U'_r|} p(y'_{qj}|u'_{rj}), \qquad \text{if i, e or s} > 0 \tag{16}$$

To obtain bounds on the magnitudes of the variables i, e and s, we observe that they are constrained by the lengths of the strings X and Y. Thus, if r=e+s, q=i+s and R=Min [M, N], these variables will have to obey the following obvious constraints :

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

Values of triples (i,e,s) which satisfy these constraints are termed as the feasible values of the variables. Let,

$$H_i = \{ j \mid \text{Max}[0,M-N] \le j \le M \},$$
$$H_e = \{ j \mid 0 \le j \le N \}, \text{ and}$$
$$H_s = \{ j \mid 0 \le j \le \text{Min}[M, N] \}. \qquad (17)$$

H_i, H_e and H_s are called the set of permissible values of i, e and s. Observe that a triple (i,e,s) is feasible if apart from $i \in H_i$, $e \in H_e$, and $s \in H_s$, the following is satisfied:

$$i + s \le M, \quad \text{and} \quad e + s \le N. \qquad (18)$$

The following result specifies the permitted forms of the feasible triples encountered on transforming U_r, the prefix of U of length r, to Y_q, the prefix of Y of length q.

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 :

Consider the constraints imposed on feasible values of i, e and s. Since we are interested in the editing of U_r to Y_q we have to consider only those triples (i, e, s) in which i+s=r and e+s=q. But the number of insertions can take any value from Max[0,q-r] to q. For every value of i in this range, the feasible triple (i,e,s) must have exactly q-i substitutions.

Similarly, since the sum of the number of substitutions and the number of deletions is r, the triple (i,e,s) must have exactly r-q+i deletions. Hence the result. ♦♦♦

The following theorem states the recursively computable property for the array W(.,.,.).

THEOREM III.

Let W(i,e,s) be the quantity defined as in (13) for any two strings U and Y. Then, for all nonnegative i,e and s,

$$W(i,e,s) = W(i-1,e,s).p(y_{i+s}|\xi) + W(i,e-1,s).p(\lambda|u_{e+s}) + W(i,e,s-1).p(y_{i+s}|u_{e+s}) \qquad (19)$$

where p(bla) is interpreted as in (8).

Proof :

The proof of the result is divided into three main divisions, Cases(a)-(c) respectively.

Case (a) : Any two of the three variables i, e and s are zero.

Case (b) : Any one of the three variables i, e and s is zero.

Case (c) : None of the variables i, e and s are zero.

The most involved of these cases is Case (c). Cases (a) and (b) are merely one and two parameter cases respectively of Case (c). To avoid repetition, we shall prove only Case (c). Thus, for the rest of the proof, we encounter only strictly positive values for the variables, i, e and s. Let r=e+s, q=i+s, $U_r=u_1 \ldots u_r$, and $Y_q=y_1 \ldots y_q$. Then, by definition,

$$W(i,e,s) = \sum_{(U'_r,Y'_q)} \prod_{j=1}^{|U'_r|} p(y'_{qj}|u'_{rj}), \tag{20}$$

where (U'_r,Y'_q) is the arbitrary element of the set $\Gamma_{i,e,s}$ with u'_{rj} and y'_{qj} as the jth symbols of U'_r and Y'_q respectively. In the above expression and in all the expressions used in this proof, we shall assume that p(b|a) is interpreted as defined by (8).

Let the lengths of the strings U'_r and Y'_q in the arbitrary element of $\Gamma_{i,e,s}$ be L. Then the last symbols of U'_r and Y'_q are u'_{rL} and y'_{rL} respectively. We partition the set $\Gamma_{i,e,s}$ into three mutually exclusive and exhaustive subsets.

$$\Gamma^1_{i,e,s} = \{ (U'_r,Y'_q) \mid (U'_r,Y'_q) \in \Gamma_{i,e,s}, u'_{rL} = u_r, y'_{qL} = y_q \} \tag{21}$$

$$\Gamma^2_{i,e,s} = \{ (U'_r,Y'_q) \mid (U'_r,Y'_q) \in \Gamma_{i,e,s}, u'_{rL} = u_r, y'_{qL} = \lambda \} \tag{22}$$

$$\Gamma^3_{i,e,s} = \{ (U'_r,Y'_q) \mid (U'_r,Y'_q) \in \Gamma_{i,e,s}, u'_{rL} = \xi, y'_{qL} = y_q \}. \tag{23}$$

By their definitions, we see that the above three sets are mutually exclusive. Further, since u'_{rL} and y'_{qL} cannot be ξ and λ respectively simultaneously, every pair in $\Gamma_{i,e,s}$ must be in one of the above sets. Hence these three sets partition $\Gamma_{i,e,s}$. Rewriting (20) we obtain,

$$W(i,e,s) = \Big[\sum_{(U'_r,Y'_q)\in(\Gamma^1_{i,e,s})} S' \Big] + \Big[\sum_{(U'_r,Y'_q)\in(\Gamma^2_{i,e,s})} S' \Big] + \Big[\sum_{(U'_r,Y'_q)\in(\Gamma^3_{i,e,s})} S' \Big] \tag{24}$$

$$\text{where, } S' = \prod_{j=1}^{|U'_r|} p(y'_{qj}|u'_{rj}). \tag{25}$$

Consider each of the terms in (24) individually. In every pair in $\Gamma^1_{i,e,s}$, $u'_{rL} = u_r$ and $y'_{qL} = y_q$. Hence,

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

$$= \Big[\sum_{(U'_r,Y'_q)\in(\Gamma^1_{i,e,s})} \prod_{j=1}^{|U'_r|-1} p(y'_{qj}|u'_{rj}) \Big] \cdot p(y_q|u_r). \tag{26}$$

For every element in $\Gamma^1_{i,e,s}$ there is a unique element in $\Gamma_{i,e,s-1}$ and vice versa. Hence, the first term in the above expression is exactly W(i,e,s-1). Since r=e+s and q=i+s,

$$\sum_{(U'_r,Y'_q)\in(\Gamma^1_{i,e,s})} \prod_{j=1}^{|U'_r|} p(y'_{qj}|u'_{rj}) = W(i,e,s-1).p(y_{i+s}|u_{e+s}). \tag{27}$$

Consider the second term in (24). In every pair in $\Gamma^2_{i,e,s}$, $u'_{rL} = u_r$ and $y'_{qL} = \lambda$. Hence,

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

$$= \Big[\sum_{(U'_r,Y'_q)\in(\Gamma^2_{i,e,s})} \prod_{j=1}^{|U'_r|-1} p(y'_{qj}|u'_{rj}) \Big].p(\lambda|u_r). \tag{28}$$

For every element in $\Gamma^2_{i,e,s}$ there is a unique element in $\Gamma_{i,e-1,s}$ and vice versa. Hence, the first term in the above expression is exactly W(i,e-1,s). Thus,

$$\sum_{(U'_r,Y'_q)\in(\Gamma^2_{i,e,s})} \prod_{j=1}^{|U'_r|} p(y'_{qj}|u'_{rj}) = W(i,e-1,s).p(\lambda|u_{e+s}). \tag{29}$$

Consider the third term in (24). In every pair in $\Gamma^3_{i,e,s}$, $u'_{rL} = \xi$ and $y'_{qL} = y_q$. Hence,

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

$$= \Big[\sum_{(U'_r,Y'_q)\in(\Gamma^3_{i,e,s})} \prod_{j=1}^{|U'_r|-1} p(y'_{qj}|u'_{rj}) \Big].p(y_q|\xi). \tag{30}$$

For every element in $\Gamma^3_{i,e,s}$ there is a unique element in $\Gamma_{i-1,e,s}$ and vice versa. Hence, the first term in the above expression is exactly W(i-1,e,s). As in the above cases,

$$\sum_{(U'_r,Y'_q)\in(\Gamma^3_{i,e,s})} \prod_{j=1}^{|U'_r|} p(y'_{qj}|u'_{rj}) = W(i-1,e,s).p(y_{i+s}|\xi). \tag{31}$$

Resubstituting (27), (29) and (31) into (24) proves the result. ♦♦♦

The computation of the probability Pr[Y|U] from the array W(i,e,s) merely involves weighting the appropriate terms by factors that are dependent only on the number of insertions.

THEOREM IV

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

$$Pr[Y|U] = \sum_{i=Max(0,M-N)}^{M} h(i).W(i,\ N-M+i,\ M-i). \qquad (32)$$

Proof :

Consider the constraints imposed by (14) on i, e and s. Since we are interested in the editing of the entire string U to the entire string Y, we have to consider only those elements of W(i,e,s) in which i+s=M and e+s=N. But the number of insertions can take any value from Max[O,M-N] to M. For every value of i in this range, the term in W(i,e,s) that will give a contribution to Pr[Y|U] must have exactly M-i substitutions. Since the sum of the number of substitutions and the number of deletions is N, the term in W(i,e,s) that will give a contribution to Pr[Y|U] must have exactly N-M+i deletions. Hence we are only interested in the terms of the form W(i, N-M+i, M-i) with i varying from Max[0, M-N] to M. The result is proved by noting that the weighting factor h(i) is ***only*** dependent on i, the number of insertions. ♦♦♦

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(. , . , .), which is closely related to it has the interesting properties proved in Theorem III. The Algorithm EvaluateProbabilities which we now present, evaluates the array W(. , . , .) for all permissible values of the variables i, e and s subject to the constraints given in (14). 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.

The evaluation of the array W(. , . , .) has to be done in a systematic manner, so that any quantity W(i,e,s) must be evaluated before its value is required in any further evaluation. This is easily done by considering a three-dimensional coordinate system whose axes are i, e and s respectively. Initially, the contribution associated with the origin, W(0,0,0) is assigned the value unity, and the contributions associated with the vertices on the axes are evaluated. Thus, W(i,0,0), W(0,e,0) and W(0,0,s) are evaluated for all permissible values of i, e and s. Subsequently, the i-e, e-s and i-s planes are traversed, and the contributions associated with the vertices on these planes are evaluated using the previously evaluated values. Finally, the contributions corresponding to strictly positive values of the variables are evaluated. To avoid unnecessary evaluations, at each stage, the variables must be tested for permissibility using the constraints of

(14). Finally, the quantity Pr[Y|U] is evaluated by adding the weighted contributions of W(.,.,.) associated with the points that lie on the three-dimensional line given by the parametric equation :

$$i = i \;\; ; \;\; e=N-M+i \;\; ; \;\; s=M-i$$

The process is formally given below.

Algorithm EvaluateProbabilities
Input: The strings $U=u_1u_2 \ldots u_N$, $Y=y_1y_2 \ldots y_M$, and the distributions G, Q and S. Let R=Min [M, N].
Output: The array W(i,e,s) for all permissible values of i, e and s and the probability Pr[Y|U] as defined by (7).
Method :

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

In the above algorithm, to compute Pr[Y|U], we have made use of the fact that though the latter index itself does not seem to have any recursive properties, the index W (.,.,.), which is closely related to it had the interesting properties stated in Theorem III. Obviously, the above process requires cubic time and space respectively. However, we shall now present a more efficient[4] process to compute the probability Pr[Y|U]. To do this, we shall take advantage of the

[4]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.

following fact . 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. This is true from Theorem IV, since the computation of any one quantity requires **at most** three previously computed quantities, namely W(i-1, e,s), W(i, e-1,s) and W(i, e, s-1). We shall utilize this face as follows.

Consider the three dimensional trellis described above. We shall successively evaluating the array W in planes parallel to the plane i = 0. Four arrays are maintained, namely,

(a) Wie: the plane in which s = 0,

(b) Wis: the plane in which e = 0,

(c) Wes0 : the plane parallel to i=0, maintained for the previous value of i, and,

(d) Wes1 : the plane parallel to i=1 maintained for the current value of i.

The algorithm, given formally on the next page, merely evaluates these arrays in a systematic manner. Initially, the quantities associated with the individual axes are evaluated. The i-e and i-s planes are then computed and stored in the arrays Wie and Wis respectively. The trellis is then traced plane by plane - always retaining only the current plane parallel to the plane i=0. Thus, prior to updating Wes0, its pertinent component required in the computation of Pr[Y|U] is used to update the latter.

Remarks :

(i) Observe that in the above algorithm the updating was performed using arrays. For large values of M and N it is more efficient to use pointers, in which case the updating of Wes0 from Wes1 requires only pointer manipulations and not a copying of the entire array.

(ii) A note about the *modus operandus* of the proof of computing Pr[Y|U] is not out of place. From a naive perspective it is possible to consider the techniques applied here as mere application of string editing dynamic programming algorithms [8-10,16,22,27,28,32,33,35, 38] to the case when the operators utilized are the arithmetic addition and multiplication respectively. This is, in fact, not the case. First of all, observe that the quantities computed are probabilities, and hence, at every point, the rigid constraints imposed by the laws of probability must be satisfied. There is a very fine point in which our proof differs from the proofs currently described in the literature. The fundamental difference is that we have tried to compute a quantity which has, by itself, no known recursively computable properties. But, we have been able to "discover" that there is a related quantity, namely, W(i,e,s) which can be recursively computed, and from whose values the quantity Pr[Y|U] can be evaluated. This makes the proof more interesting and a trifle more "intriguing". This is reminiscent of a control system in which various outputs are computed in terms of the **same** state variables by just using different "Output Functions". Furthermore, this is to our knowledge, one of the

few non-trivial applications of dynamic programming which evaluates quantities which involve relatively complex combinatorial expressions while simultaneously satisfying the rigid constraints imposed by the laws of probability.

Algorithm EfficientlyEvaluateProbabilities
Input & Output : Same as in **Algorithm EvaluateProbabilities**
Method :

Wis (0,0) = Wie (0,0) = Wes0(0,0) = 1
For s = 1 to Min[M,N] **Do** /* initialize s-axis */
  Wis(0,s) = Wes0(0,s) = Wis (0,s-1) .S (y_s| u_s)
For e = 1 to N **Do** /* initialize e-axis */
  Wie (0,e) = Wes0(e,0) = Wie (0,e-1). S(λ|x_e)
For i = 1 to M **Do** /* initialize i-axis */
  Wis(i,0) = Wie (i,0) = Wis (i-1,0).Q(y_i)
For i = 1 to M **Do** /* Compute Wie-plane */
  For e = 1 to N-M+i **Do**
    Wie (i,e) = Wie(i-1, e).Q (y_{i+s}) + Wie(i,e-1).S(λ| x_e)
For e = 1 to N **Do** /* Compute Wes-plane parallel to i=0 */
  For s = 1 to Min[M,N-e] **Do**
    Wes0(e,s) =Wes0 (e-1, s).S(λ| x_{e+s}) + Wes0(e,s-1).S(y_s| x_{e+s})]
Pr [Y | U] = G(0). Wes0(N-M, M) /* Initialize Pr [Y | U] */
For i = 1 to M **Do** /* Trace planes parallel to i=0 */
Begin
  For e = 1 to N-M + i **Do** /* Update line parallel to s-axis from Wis plane */
    Wes1 (0,s) = Wis (i,s)
  For s = 1 to Min [N, M-i] **Do** /* Update line parallel to e-axis from Wie */
    Wes1(e,o) = Wie (i,e) /* plane */
  For e = 1 to N-M + i **Do** /* Compute Wes1 using Theorem V */
    For s = 1 to Min [N-e,M-i] **Do**
      Wes1(e,s) = Wes1(e,s-1).S(y_{i+s}|x_{e+s}) + Wes1(e-1,s).S(λ|x_{e+s})
                + Wes0(e,s).Q(y_{i+s})

$$Pr[Y|U] = Pr[Y|U] + G(i) \cdot \frac{N!\ i!}{(N+i)!} \cdot Wes1(N-M+i, M-i)]$$

  For e = 1 to N **Do** /* Update Wes0 from Wes1 */
    For s = 1 to N-e **Do**
      Wes0(e,s) = Wes1 (e,s)
End

END Algorithm EfficientlyEvaluateProbabilities

We are currently studying the use of this channel in speech recognition. Note that with such a model, the entire question of "time warping" would be subsumed in appropriately modelling the distribution G.

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 $U \in A^*$, we specify a stochastically consistent scheme by which this word can be transformed into any $Y \in A^*$ 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.

REFERENCES

1. A. V. Aho, D.S. Hirschberg, and J. D. Ullman, Bounds on the complexity of the longest common subsequence problem, *J. Assoc. Comput. Mach.*, 23:1-12 (1976).
2. 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).
3. L. Devroye, *Non-Uniform Random Variate Generation*, Springer-Verlag, (1986).
4. L. Devroye, W. Szpankowski and B. Rais, A note on the height of suffix trees, *SIAM J. of Computing*, 21:48-54, (1992)
5. G. Dewey, *Relative Frequency of English Speech Sounds*, Cambridge, MA, Harvard Univ. Press, (1923).
6. G.D. Forney, The Viterbi Algorithm, *Proceedings of the IEEE*, Vol. 61. (1973)
7. J. Golic and M. Mihaljevic, A noisy clock-controlled shift register cryptanalysis concept based on sequence comparison approach, *Proceedings of EUROCRYPT 90,* Aarhus, Denmark, 487-491 (1990).
8. P. A. V. Hall and G.R. Dowling, Approximate string matching, *Comput. Surveys*, 12:381-402 (1980).
9. D. S. Hirschberg, Algorithms for longest common subsequence problem, *J. Assoc. Comput. Mach.*, 24:664-675 (1977).
10. D. S. Hirschberg, A linear space algorithm for computing maximal common subsequences, *Comm. Assoc. Comput. Mach.*, 18:341-343 (1975).
11. J. W. Hunt and T. G. Szymanski, A fast algorithm for computing longest common subsequences, *Comm. Assoc. Comput. Mach.*, 20:350-353 (1977).
12. P. Jacquet and W. Szpankowski, Analysis of digital tries with markovian dependencies, *IEEE Trans. Information Theory*, IT-37:1470-1475 (1991).
13. R. L. Kashyap and B. J. Oommen, A common basis for similarity and dissimilarity measures involving two strings, *Internat. J. Comput. Math.*, 13:17-40 (1983).
14. R. L. Kashyap and B. J. Oommen, Similarity measures for sets of strings, *Internat. J. Comput. Math.*, 13:95-104 (1983).
15. R. L. Kashyap and B. J. Oommen, The noisy substring matching problem, *IEEE Trans. Software Engg.*, SE-9:365-370 (1983).

16. R. L. Kashyap and B. J. Oommen, An effective algorithm for string correction using generalized edit distances -I. Description of the algorithm and its optimality, *Inform. Sci.*, 23(2):123-142 (1981).
17. R. L. Kashyap, and B. J. Oommen, String correction using probabilistic methods, *Pattern Recognition Letters*, 147-154 (1984).
18. S. H. Levine, An adaptive approach to optimal keyboard design for nonvocal communication, *Proc. of the Internat. Conference of Cybernetics and Society*, 334-337 (1985).
19. S. H.Levine, S. H., S. L. Minneman, C.O. Getschow, and C. Goodenough-Trepagnier, Computer disambiguation of multi-character text entry : An adaptive design approach, *Proc. of the IEEE Internat. Conference on Systems, Man and Cybernetics*, 298-301 (1986).
20. R. Lowrance and R. A. Wagner, An extension of the string to string correction problem, *J. Assoc. Comput. Mach.*, 22:177-183 (1975).
21. A. Levenshtein, Binary codes capable of correcting deletions, insertions and reversals, *Soviet Phys. Dokl.*, 10:707-710 (1966).
22. W. J. Masek and M. S. Paterson, A faster algorithm computing string edit distances, *J. Comput. System Sci.*, 20:18-31 (1980).
23. S. L. Minneman, A simplified Touch-Tone telecommunication aid for deaf and hearing impaired individuals, *Proc. of the 8th. Annual RESNA Conference*, Memphis, 209-211 (1985).
24. S. L. Minneman, Keyboard optimization technique to improve output rate of disabled individuals, *Proc. of the 9th. Annual RESNA Conference*, Minneapolis, 402-404 (1986).
25. S. B. Needleman and C. D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins, *J. Mol. Biol.*, 443-453 (1970).
26. D. L. Neuhoff, The Viterbi algorithm as an aid in text recognition, *IEEE Trans. Information Theory*, 222-226 (1975).
27. T. Okuda, E. Tanaka, and T. Kasai, A method of correction of garbled words based on the Levenshtein metric, *IEEE Trans. Comput.*, C-25:172-177 (1976).
28. B. J. Oommen, Recognition of noisy subsequences using constrained edit distances, *IEEE Trans. on Pattern Anal. and Mach. Intel.*, PAMI-9:676-685 (1987).
29. Oommen, B.J., Constrained string editing, *Information Sciences*, 40: 267-284 (1987).
30. J. L. Peterson, Computer programs for detecting and correcting spelling errors, *Comm. Assoc. Comput. Mach.*, 23:676-687 91980).
31. M. Regnier, A language approach to string searching evaluation, *Proceedings of CPM-92, The Third Annual Symposium on Combinatorial Pattern Matching*, 15-26 (1992).
32. D. Sankoff, Matching sequences under deletion/insertion constraints, *Proc. Nat. Acad. Sci. U.S.A.*, 69:4-6 (1972).
33. D. Sankoff and J. B. Kruskal, *Time Warps,String Edits and Macromolecules: The Theory and practice of Sequence Comparison*, Addison-Wesley (1983).
34. R. Shinghal, and G. T. Toussaint, Experiments in text recognition with the modified Viterbi algorithm, *IEEE Trans on Pat. Anal. and Mach. Intel.*, 184-192 (1979).
35. S. Srihari, *Computer Text Recognition and Error Correction*, IEEE Computer Society Press, (1984).
36. W. Szpankowski, Probabilistic analysis of generalized suffix trees, *Proceedings of CPM-92, The Third Annual Symposium on Combinatorial Pattern Matching*, 1-14 (1992).
37. A. J. Viterbi, Error bounds for convolutional codes and an asymptotically optimal decoding algorithm, *IEEE Trans. on Information Theory*, 260-26 (1967).
38. R. A. Wagner and M. J. Fisher, The string to string correction problem, *J. Assoc. Comput. Mach.*, 21:168-173 (1974).
39. C. K. Wong and A. K. Chandra, Bounds for the string editing problem, *J. Assoc. Comput. Mach.*, 23:13-16 (1976).

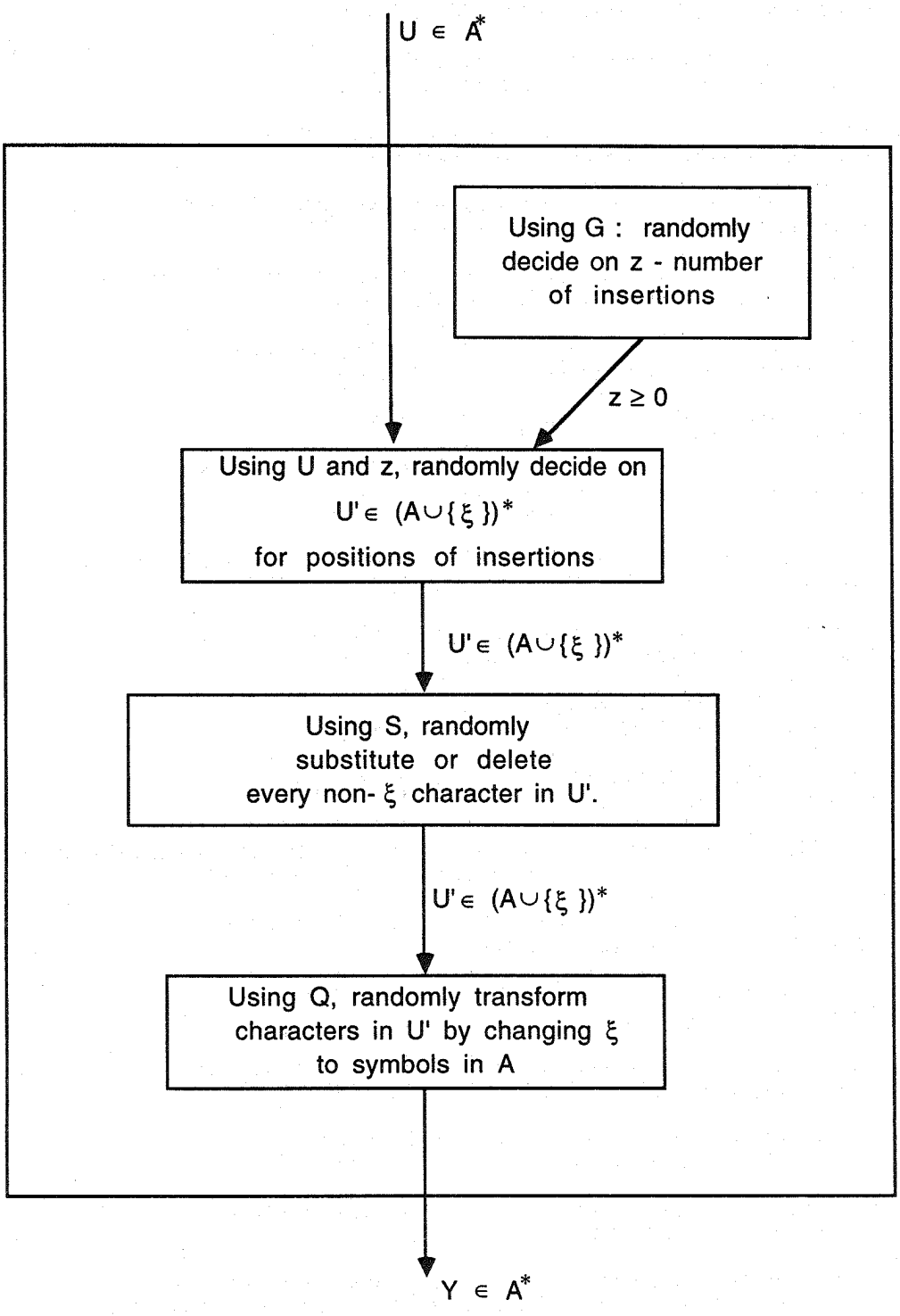

Figure I : A pictorial representation for the model for the channel. The input to the channel is the string U, and the output is the random string Y.

School of Computer Science, Carleton University
Recent Technical Reports

TR-204 **Recognition of Catastrophic Faults in Reconfigurable Arrays with Arbitrary Link Redundancy**
Amiya Nayak, Linda Pagli, Nicola Santoro, March 1992

TR-205 **The Permutational Power of a Priority Queue**
M.D. Atkinson and Murali Thiyagarajah, April 1992

TR-206 **Enumeration Problems Relating to Dirichlet's Theorem**
Evangelos Kranakis and Michel Pocchiola, April 1992

TR-207 **Distributed Computing on Anonymous Hypercubes with Faulty Components**
Evangelos Kranakis and Nicola Santoro, April 1992

TR-208 **Fast Learning Automaton-Based Image Examination and Retrieval**
B. John Oommen and Chris Fothergill, June 1992

TR-209 **On Generating Random Intervals and Hyperrectangles**
Luc Devroye, Peter Epstein and Jörg-Rüdiger Sack, July 1992

TR-210 **Sorting Permutations with Networks of Stacks**
M.D. Atkinson, August 1992

TR-211 **Generating Triangulations at Random**
Peter Epstein and Jörg-Rüdiger Sack, August 1992

TR-212 **Algorithms for Asymptotically Optimal Contained Rectangles and Triangles**
Evangelos Kranakis and Emran Rafique, September 1992

TR-213 **Parallel Algorithms for Rectilinear Link Distance Problems**
Andrzej Lingas, Anil Maheshwari and Jörg-Rüdiger Sack, September 1992

TR-214 **Camera Placement in Integer Lattices**
Evangelos Kranakis and Michel Pocchiola, October 1992

TR-215 **Labeled Versus Unlabeled Distributed Cayley Networks**
Evangelos Kranakis and Danny Krizanc, November 1992

TR-216 **Scalable Parallel Geometric Algorithms for Coarse Grained Multicomputers**
Frank Dehne, Andreas Fabri and Andrew Rau-Chaplin, November 1992

TR-217 **Indexing on Spherical Surfaces Using Semi-Quadcodes**
Ekow J. Otoo and Hongwen Zhu, December 1992

TR-218 **A Time-Randomness Tradeoff for Selection in Parallel**
Danny Krizanc, February 1993

TR-219 **Three Algorithms for Selection on the Reconfigurable Mesh**
Dipak Pravin Doctor and Danny Krizanc, February 1993

TR-220 **On Multi-label Linear Interval Routing Schemes**
Evangelos Kranakis, Danny Krizanc, and S.S. Ravi, March 1993

TR-221 **Note on Systems of Polynomial Equations over Finite Fields**
Vincenzo Acciaro, March 1993

TR-222 **Time-Message Trade-Offs for the Weak Unison Problem**
Amos Israeli, Evangelos Kranakis, Danny Krizanc and Nicola Santoro, March 1993

TR-223 **Anonymous Wireless Rings**
Krzysztof Diks, Evangelos Kranakis, Adam Malinowski, and Andrzej Pelc, April 1993

TR-224 **A consistent model for noisy channels permitting arbitrarily distributed substitutions, insertions and deletions**
B.J. Oommen and R.L. Kashyap, June 1993