A FORMAL THEORY FOR OPTIMAL AND INFORMATION THEORETIC SYNTACTIC PATTERN RECOGNITION[+]

B. J. OOMMEN[1] AND R. L. KASHYAP[2]

ABSTRACT

In this paper we present a foundational basis for optimal and information theoretic syntactic pattern recognition. We do this by developing a rigorous model, $\mathfrak{M}^*$, for channels which permit arbitrarily distributed substitution, deletion and insertion syntactic errors. More explicitly, if A is any finite alphabet and A^* the set of words over A, we specify a stochastically consistent scheme by which a string $U \in A^*$ can be transformed into any $Y \in A^*$ by means of arbitrarily distributed substitution, deletion and insertion operations. The scheme is shown to be *Functionally Complete* and *stochastically consistent*. Apart from the synthesis aspects, we also deal with the analysis of such a model and derive a technique by which $\Pr[Y|U]$, the probability of receiving Y given that U was transmitted, can be computed in cubic time using dynamic programming. One of the salient features of this scheme is that it demonstrates how dynamic programming can been applied to evaluate quantities involving complex combinatorial expressions and which also maintain rigid probability consistency constraints. Experimental results which involve dictionaries with strings of lengths between 7 and 14 with an overall average noise of 39.75 % demonstrate the superiority of our system over existing methods. Apart from its straightforward applications in string generation and recognition, we believe that the model also has extensive potential applications in speech and uni-dimensional signal processing.

Keywords : *Syntactic Pattern Recognition, Optimal and Information Theoretic Pattern Recognition, String Generation, Unigram and Bigram Model, Sequence Generation, Markovian Sequence Generation.*

[+] We would like to dedicate this paper to the memory of the late Prof. K. S. Fu who pioneered the field of *Syntactic Pattern Recognition*. Both authors remember their friend and colleague with respect. His contributions to the fields of Syntactic and Statistical Pattern Recognition are monumental.

[1]Senior Member IEEE. Address : School of Computer Science, Carleton University, Ottawa ; Canada : K1S 5B6. The work of this author was supported in part by the Natural Sciences and Engineering Research Council of Canada.

[2]Fellow IEEE. Address : School of Electrical Engineering, Purdue University, W. Lafayette ; Indiana : 47907. The work of this author was supported in part by the Office of Naval Research and BMD under Contract ONR N00014-91-J-4126.

I. INTRODUCTION

In the field of statistical Pattern Recognition (PR), the pattern are represented using numerical features. As opposed to this, in syntactic and structural PR the classifiers are designed to be trained and tested by representing the patterns symbolically using primitive or elementary symbols. The PR system essentially models the noisy variations of typical samples of the patterns symbolically, and these models are utilized in both the training and testing phases. In statistical PR, the noisy samples from a given class are modeled (either parametrically or non-parametrically) using probability distributions, the class conditional distributions. It is well known [7,9] that if these distributions are known, information theoretic, minimum probability of error classifiers can be designed.

In this paper we shall attempt to formally lay the foundation for optimal[3] *syntactic* PR systems which permit arbitrarily distributed noise[4]. To achieve this we have, in a novel way, "decoupled" the occurrence of noisy observations into two distinct modules. The first of these formally describes the module which accounts for the "dictionary" of ungarbled observations and the second describes the channel which syntactically corrupts error-free signals. This approach differs significantly from the original traditional way by which the problem was approached by the pioneer Fu[5][40] and his co-authors. The latter represented the set of possible garbled occurrences of possible patterns using stochastic string grammars from the well-known Chomsky hierarchy, or by web or tree grammars.

The alternate strategy for designing syntactic PR recognition systems works as follows. The system has a dictionary[6] which is a collection of all the ideal representations of the objects in question. When a noisy sample has to be processed, the system compares this sample with every element in the dictionary. This comparison is either done sequentially or using a grammatical parsing mechanism. The question of comparing patterns reduces to one of comparing their string representations, and this is typically achieved using three standard edit operations - substitution, insertion and deletion. To achieve this, one usually assigns a distance for the elementary *symbol* operations, and the inter-pattern distance is computed as a function of these elementary symbol edit distances. Recognition using distance criteria is obtained by essentially evaluating the string in the dictionary which is "closest" to the noisy one as per the metric under consideration..

The elementary distances can be assigned weights in a variety of ways. If $\mathbf{R}^+$ is the set of non-negative real numbers, the elementary edit distances are typically defined using three elementary functions $d_s(.,.)$, $d_i(.)$ and $d_e(.)$:

---

[3]By "optimal" we mean that we can obtain decisions which minimize the probability of error or which minimize the total expected risk. As we will see later, the conditions required for information theoretic syntactic PR are more stringent.

[4]In this paper we shall only deal with syntactic PR of patterns which are represented "linearly" as strings. The problem of developing information theoretic classifiers for PR systems using two-dimensional structures such as trees and webs remains open. Some initial results for the case of ordered tree representations are currently available.

[5]Prof. Fu (to whose memory we have dedicated this paper) has authored or co-authored dozens of papers and books in this area. We recommend [40] as an ideal reference for the entire field as it was studied in its infancy.

[6]A discussion of the various dictionary models encountered in syntactic PR follows presently.

(i) $d_S(.,.)$ is a map from $\mathbf{A} \times \mathbf{A} \rightarrow \mathbf{R}^+$ and is the Substitution Map. $d_S(a,b)$ is the distance associated with substituting b for a, $a,b \in \mathbf{A}$. Although not mandatory, $d_S(a,a)$ is generally set to zero.

(ii) $d_i(.)$ is a map from $\mathbf{A} \rightarrow \mathbf{R}^+$ and is called the Insertion Map. The quantity $d_i(a)$ is the distance associated with inserting the symbol $a \in \mathbf{A}$.

(iii) $d_e(.)$ is a map from $\mathbf{A} \rightarrow \mathbf{R}^+$ and is called the Deletion or Erasure Map. The quantity $d_e(a)$ is the distance associated with deleting (or erasing) the symbol $a \in \mathbf{A}$.

If for all $a,b \in \mathbf{A}$ these distances are assigned unit values as below :

$$d_S(a, b) = 1 \quad \text{if } a \neq b$$
$$= 0 \quad \text{if } a = b$$
$$d_i(a) = d_e(a) = 1 \quad \text{for all a,}$$

the inter-string edit distance is called the Levenshtein distance.

More interesting and novel assignment of the distances are the parametric distances recently introduced by Bunke *et al* [3]. In this case, for all $a, b \in \mathbf{A}$ the distances are assigned values as :

$$d_S(a, b) = r \quad \text{if } a \neq b$$
$$= 0 \quad \text{if } a = b$$
$$d_i(a) = d_e(a) = 1 \quad \text{for all a.}$$

The parametric string distance has some amazingly interesting properties derived in [3]. The assignment of 'r' and the application of the inter-string distance in PR has also been alluded to in [3].

If, however, the elementary symbol edit distances are *symbol* dependent, the distance is called the Generalized Levenshtein Distance[7]. The question of how the elementary symbol edit distances can be assigned is relatively open; indeed, they can be parametrically assigned as in [3] or can be related to the inter-symbol confusion probabilities via their negative logarithms as recommended in [18, 33]. The explicit form of the individual edit distances often takes the form :

$$d_S(a, b) = -\ln [\, Pr(a \rightarrow b) / Pr(a \rightarrow a) \,]$$
$$d_e(a) = -\ln [\, Pr(a \text{ is deleted}) / Pr(a \rightarrow a) \,]$$
$$d_i(a) = K_i \cdot d_e(a)$$

where K_i is an empirically determined constant.

The fundamental problem that arises from all the above three assignment strategies is that the final classified string obtained using such edit distances has no probabilistic significance except in some rather simple cases. Furthermore, if D(X, Y) is the edit distance associated with editing X to Y, the latter has no explicit relationship to $Pr(X \rightarrow Y)$ except in a few rather trivial cases.

A little insight into the problem would reveal that the fundamental question which traditional strategies avoid is one of stochastically modeling the structural behaviour of the patterns. Viewed from a reverse engineering (black-box) perspective this question is one of specifying how the

[7]The terminology used in the literature varies. Throughout this paper, we shall categorically uses these terms.

individual patterns from the various classes *could have been generated*, an understanding of which could lead to the designing of optimal classifiers. This is the central problem studied in this paper. Consequently, we shall endeavour to 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 model, $\mathfrak{M}^*$, is the first generalized model of its type[8]. Indeed, we also show that if we compare our model with all other channel models which have the same common underlying garbling philosophy - that transmitted symbols can only be substituted for or deleted, and received symbols are obtained as either a result of transmitted symbols being substituted for or as inserted symbols), our model attains the information theoretic upper bound for the recognition accuracy.

Apart from its straightforward applications in string generation and recognition, like its predecessor [2,37], the model also has potential applications in speech and uni-dimensional signal processing.

I.1 Alphabet and Dictionary Representations

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]. In contour recognition, the primitive symbols are the vectors of the chain code.

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 [6] 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]Our present paper also gives a formal basis for some of the experimental results which we presented in [19].

[11,16-18,21-23,25,27-30,33,35]. 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, avoiding "meaningless" decisions. Finally, the time complexity of the computation involved in the recognition is typically quadratic per word and linear in the size of the dictionary. The quadratic complexity per word can be often decreased if the dictionary is modelled using a trie [18], 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 [4,15,26,36]. 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 [2,5,15,20,24,26, 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.

I.2 Stochastic Channel versus Dictionary Representations

As opposed to the stochastic models given above for dictionaries, in this paper we shall consider the channel as an excited random string generator. This is quite distinct from the philosophy adopted by Fu and his co-authors [40] who initially pioneered the field. 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 [2,11,14,16-19,22,23,27,30,34,35,38,39], and furthermore, these operations can also be used to study problems involving subsequences and supersequences [1,13,14,23,33]. 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 [2,11,16-19,22,23,27,30,34,35,38] and even noisy subsequences [28,29,32]. Viewed from the perspective of edit operations, our model is a "distant" relative of the Viterbi algorithms [8,26,34,37].

From the perspective of the channel, $\mathfrak{M}^*$, 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 number of insertions does not necessarily obey a mixture of geometric distributions. Although not explicitly stated, it is easy to verify that the latter is tacitly assumed to be the distribution for the number of insertions for models such as those introduced in [2] and for the hidden Markov models used in text, character and texture classification [42-44].

Throughout this paper, for the sake of simplicity, we shall use the terms "model" and "generator" interchangeably. 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, $\mathfrak{M}^*$:

(i) $\mathfrak{M}^*$ 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 involved for the various garbling operations in $\mathfrak{M}^*$ can be completely arbitrary. These constitute the parameters of the generator (model) which are not merely "real numbers", but arbitrary distributions.

(iii) The model $\mathfrak{M}^*$ even captures the scenarios in which the probability of a particular word U being transformed into the same word Y repeatedly is 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 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 an optimal, minimum

probability of error pattern classification rule. In a non-Bayesian approach this would be a maximum likelihood pattern classification rule. Observe that both these decision rules would be independent of the model used for the dictionary itself. Also, in both these cases our models actually attains the information theoretic bound for recognition accuracy when compared with all the other models which have the same underlying garbling philosophy.

Before we proceed, we shall first formally 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 $(A \cup \{\lambda\})$. 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 operations, the symbol ξ is introduced. Let X' be any string in $(A \cup \{\xi\})^*$, the set of strings over $(A \cup \{\xi\})$. X' is called an input edit sequence . ξ is distinct from λ, but is used analogously to denote a symbol insertion. 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 $1 \leq i \leq |U'|$, it is not the case that $u'_i = \xi$ and $y'_i = \lambda$.

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

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 as follows :

(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 :

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

Proof : First of all note that $\left|\Gamma(U,Y)\right|$ 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,

$$\#(\text{Possible U}') = \binom{|U|+k}{k}. \qquad (3)$$

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/SYNTHESIS -- THE STRING GENERATION PROCESS

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

First of all we assume that $\mathfrak{M}^*$ 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 the number of insertions is z given that U is the input word. 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. Although, the distributions can be arbitrarily general, for the sake of simplicity, we shall assume that *Z* is independent of U.

The second distribution that $\mathfrak{M}^*$ 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, $\mathfrak{M}^*$, 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 the given symbol $a \in A$ in the input string is mutated by a stochastic substitution or deletion -- in which case 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 string generation process. Let |U| = N. Using the distribution G, the generator randomly[9] determines *Z*, the number of symbols to be inserted. Based on the random choice 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 $U' \in (A \cup \{\xi\})^*$. In this paper we assume that each of the $\binom{N+k}{k}$ possible strings are equally likely.

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. These occurrences of ξ are transformed independently into the individual symbols of the alphabet using the distribution Q. Finally, the non-inserted symbols in U' are now substituted for or deleted using the distribution S.

This defines the model, $\mathfrak{M}^*$, completely. The process followed by the model is formally given as Algorithm GenerateString below.

Algorithm $\mathfrak{M}^*$_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 transform the occurrences of ξ into symbols of A using Q.
4. Randomly independently substitute or delete the non-inserted symbols in U' using S.

END Algorithm $\mathfrak{M}^*$GenerateString

A graphical display of the channel model, $\mathfrak{M}^*$, is shown in Figure I. An example will help clarify the string generating process.

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 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 new string that is to be operated on is thus U' = "uglki". The non-inserted symbols of U' (the 'u', 'l' and 'i') 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 obtained is thus Y'= "ugdkλ". The resultant string Y, which is the final garbled version of U is obtained by removing the occurrences of λ from Y', and is thus Y = "ugdk". ♦♦♦

Let |U| = N and |Y| = M. Then, using the above notation, we can prove the following results.

[9]We assume that the system 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].

Theorem I

If the edit operations occur independently $\Pr[Y|U]$, the probability of receiving Y from $\mathfrak{M}^*$ given that U is transmitted has the form :

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

where, $C_I(U')=U$, $C_O(Y')=Y$, and,

(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)

Furthermore, the probability framework is both functionally complete and consistent.

Proof : Using the notation above, and based on the assumption of independence, we can express the probability of a given input edit sequence U' yielding the output edit sequence Y' as :

$$\Pr[Y'|U'] = \prod_{i=1}^{N+z} p(y'_i|u'_i),$$

where U' contains exactly z occurrences of ξ.

To compute the probability of Y occurring from a particular U' we have to, clearly, sum the above quantity over all values of Y' which when operated on by C_O yield $C_O(Y')=Y$. Thus,

$$\Pr[Y|U'] = \sum_{Y' \text{ s.t. } C_o(Y')=Y} \prod_{i=1}^{N+z} p(y'_i|u'_i) .$$

The expression (7) follows from the fact that the *total* probability is computed from the conditional probability by multiplying $\Pr[Y|U']$ with the probability of U' being the input edit sequence and adding over all permissible values of z. In this case, the multiplying factor is exactly the product of G(z) and $\frac{(N!\ z!)}{((N+z)!)}$ since the former is the probability of z insertions occurring in U', and the latter is the probability of any one edit sequence with z insertions occurring if all the $\binom{N+z}{z}$ possible strings are equally likely.

The functional completeness of $\mathfrak{M}^*$ 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 $\Gamma(U,Y)$. 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 theorem 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. ANALYSIS : COMPUTING P[Y|U] EFFICIENTLY

Now that the "synthesis" aspect of $\mathfrak{M}^*$ has been considered we shall show how the relatively cumbersome expression given by (7) (i.e., by Theorem I) can be computed efficiently.

Consider the problem of $\mathfrak{M}^*$ transforming 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.

$$\begin{aligned} W(i,e,s) &= 0, && \text{if i,e or s} < 0 \\ &= \frac{(s+e+i)!}{i!\ (s+e)!}\ \Pr[Y_{i+s}|U_{e+s}\ ;\ Z=i] && \text{otherwise.} \end{aligned} \tag{15}$$

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).

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

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

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 (15) 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(b|a) 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 term in (24) individually. In every pair in $\Gamma^1_{i,e,s}$, $u'_{rL} = u_r$ and $y'_{qL} = y_q$. Hence,

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

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

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

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

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 follows by noting that h(i) depends ***only*** on i. ♦♦♦

To evaluate $Pr[Y|U]$ we make use of the fact that although this quantity does not seem to have any inherent 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 (which, obviously, requires cubic time and space respectively) 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=Max[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 $\Pr[Y|U]$ itself does not seem to have any recursive properties, the index W(.,.,.), which is closely related to it had the properties stated in Theorem III. However, we shall now present a more efficient[10] process to compute the probability $\Pr[Y|U]$. To do this, we shall take advantage of the 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 utilize this fact as follows.

To optimize space we shall successively evaluate 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 below, merely

[10]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.

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 it.

The pseudo-code for the optimized algorithm follows.

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($\lambda|x_e$)
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($\lambda|\ x_e$)
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($\lambda|\ x_{e+s}$) + 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,0) = 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($\lambda|x_{e+s}$)
+ Wes0(e,s).Q(y_{i+s})

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

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

Remarks :

(i) 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 is trivial. By using techniques analogous to those used in the computation of the Normalized Edit Distance [41] it is easy to do some fine tuning to achieve the computation with a *single* quadratic array. We omit these details here in the interest of brevity.

(ii) A note about the *modus operandus* of the proof of computing Pr[Y|U] is not out of place. The techniques used here are not merely straightforward dynamic programming principles [11-13,18,23, 27,28,29,32,33,35,38] applicable to the scenario when the computational operators are the arithmetic addition and multiplication respectively. 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. Consequently, 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, by proving the recursive properties of the related index, W(i,e,s), the quantity Pr[Y|U] can be evaluated. This makes the proof more interesting and challenging, and 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". Thus, one of the salient features of our scheme is that it demonstrates how dynamic programming can been applied to evaluate quantities involving complex combinatorial expressions and which simultaneously maintain rigid probability consistency constraints.

IV.1 An Information Theoretic Bound

Using the model $\mathfrak{M}^*$ proposed in the previous sections, it is easy to see how optimal syntactic pattern recognition can be obtained. Indeed, if the distributions G, Q and S are known[11] PR can be achieved by evaluating the string U^* which maximizes the probability $\Pr[Y|U]$ over all U in the dictionary. Viewed from a Bayesian perspective this would be equivalent to computing the *a posteriori* probabilities if all the strings are equally likely *a priori*, and thus yield optimal, minimum probability of error pattern classification. In a non-Bayesian approach this represents a maximum likelihood pattern classification scheme.

We shall now conclude this section by proving that the PR obtained by utilizing $\mathfrak{M}^*$ is not only optimal; it also attains the information theoretic upper bound. To show this, we shall resort to arguments analogous to those used in developing bounds for sorting and other computer science operations[12]. Observe that this presupposes that we compare $\mathfrak{M}^*$ with all other channel models which have the same common underlying garbling philosophy.

[11]The inference (estimation) problem of these distributions from the set of transmitted words and their corresponding noisy versions remains open.

[12]In all brevity, we illustrate here how the bound for the complexity of sorting is proven using information theoretic arguments. To sort K elements with only pair-wise comparison computations, one, first of all, recognizes that the total information that has to be analyzed is contained in the set of all possible permutations - which is of cardinality K!. Since a sequence of h pair-wise comparisons can be perceived as a binary tree of height h (with 2^h leaves), one argues that $\text{Log}_2 K!$ comparisons is required to distinguish between the K! permutations. The complexity follows since $\text{Log}_2 K!$ is $O(K \cdot \text{Log}_2 K)$ whenever $K > 4$.

Theorem V

If transmitted symbols can only be substituted for or deleted and received symbols are obtained as either a result of transmitted symbols being substituted for or as inserted symbols, then, for specific distributions G, Q and S, the garbling model $\mathfrak{M}^*$ attains the information theoretic bound for recognition accuracies.

Proof : First of all, as stated in the theorem, we assume that all the models among which the comparison is made, possess the same stochastic garbling distributions, namely, G, Q and S. Let the transmitted string be U (with |U|=N) and as dictated by G, Q and S, let us assume that i insertions, e deletions and s substitutions occur in the transmission. Since transmitted symbols can only be substituted for or deleted, clearly e+s = N.

Now, let $\mathfrak{M}^k$ be the channel model in which k deletion operations are accomplished **before** the insertion operations are effected. Also, let $\mathfrak{J}^{U,k}$ be the set of possible input edit sequences when the transmitted string is U and when it is garbled by the channel $\mathfrak{M}^k$.

We shall first prove that for all $q < r < N$, $\mathfrak{J}^{U,q} \supset \mathfrak{J}^{U,r}$.

Consider the model $\mathfrak{M}^q$. In $\mathfrak{M}^q$, since q deletion operations are accomplished **before** the insertion operations are effected, the remaining e-q deletions are effected **subsequent** to the insertions having had their garbling effect. Since |U| = N, this implies that **before** any insertions are effected the length of the string which is effectively transmitted is reduced by q. Thus the length of the resultant string which serves as the input to the module which introduces insertions is N-q. Now, since i insertions have to be effected, these can be introduced in any one of the possible positions where a ξ could be positioned in the string U', and by the arguments used in Lemma O, the number of input edit sequences permissible for the transformation of U is :

$$\#(\text{Possible U' for } \mathfrak{M}^q) = \binom{N-q+i}{i} .$$

Arguing in a similar fashion for model $\mathfrak{M}^r$, we see that :

$$\#(\text{Possible U' for } \mathfrak{M}^r) = \binom{N-r+i}{i}$$

A simple counting argument reveals that :

$$\#(\text{Possible U' for } \mathfrak{M}^q) > \#(\text{Possible U' for } \mathfrak{M}^r) \text{ whenever } q < r.$$

Besides this, every element in $\mathfrak{J}^{U,r}$ is found in $\mathfrak{J}^{U,q}$, since any input edit sequence in $\mathfrak{J}^{U,r}$ yields a corresponding edit sequence in $\mathfrak{J}^{U,q}$ by merely inserting additional ξ's in q-r new positions which are the positions inhibited by $\mathfrak{M}^r$, and thus, for all $q < r < N$, $\mathfrak{J}^{U,q} \supset \mathfrak{J}^{U,r}$.

Consider now two PR systems developed using models $\mathfrak{M}^q$ and $\mathfrak{M}^r$ respectively. The system which uses $\mathfrak{M}^r$ will not be capable of yielding a recognition accuracy which $\mathfrak{M}^q$ yields, because, it ignores various edit possibilities which are considered by $\mathfrak{M}^q$ whenever $q < r$. Conversely, if the edit operations which generated the noisy string were due to an edit sequence found in the set difference $(\mathfrak{J}^{U,q} - \mathfrak{J}^{U,r})$, no algorithm which uses $\mathfrak{M}^r$ will be capable of correctly recognizing it.

The theorem follows since the model which we have introduced, $\mathfrak{M}^*$, is identical to the model $\mathfrak{M}^0$ (i.e, $\mathfrak{M}^k$ with k=0), since in $\mathfrak{M}^*$ the effect of **all** the insertion operations are considered **before** the fate of any of the transmitted symbols is ascertained. Hence the theorem ! ♦♦♦

V. EXPERIMENTAL RESULTS

To investigate the power of our new model (and its computation) and to demonstrate the accuracy of our new scheme in the original PR problem various experiments were conducted. The results obtained were remarkable. The algorithm was compared with PR results obtained with

(i) Algorithm_LD : A PR scheme which used any traditional editing [1,13,25,27,32,33,38] algorithm and unit inter-symbol costs.

(ii) Algorithm_GLD : A PR scheme which used any traditional editing [1,13,25,27,32,33,38] algorithm using symbol-dependent costs. The symbol-dependent costs are assigned as :

$d_s(a, b) = -\ln [Pr(a \rightarrow b) / Pr(a \rightarrow a)]$

$d_e(a) = -\ln [Pr(a \text{ is deleted}) / Pr(a \rightarrow a)]$

$d_i(a) = K_i \cdot d_e(a)$, where K_i is determined to yield the most conservative distance.

As opposed to recognition obtained using the above distance criteria, to verify our theoretical results, as mentioned above, optimal PR was achieved by evaluating the string U^* which maximized the probability $Pr[Y|U]$ over all U in the dictionary. In a non-Bayesian approach this represents a maximum likelihood pattern classification scheme.

The dictionary consisted of 342 words obtained as a subset of the 1023 most common English words [4] augmented with words used in computer literature. The length of the words was greater than or equal to 7 and the average length of a word was approximately 8.3 characters.

From these, two sets (**SA** and **SB** respectively) of 1026 noisy strings were generated using the method described in Section III. The conditional probability of inserting any character $a \in A$ given that an insertion occurred was assigned the value 1/26; and the probability of deletion was set to be 1/20. The table of probabilities for substitution (typically called the confusion matrix) was based on the proximity of the character keys on a standard QWERTY keyboard and is given in Table I. The statistics associated with the sets **SA** and **SB** are given below in Table II. Notice that the percentage error was intentionally made large to test the algorithms for non-trivial error conditions. A subset of some of the words in SA is given Table III. Observe that some words are very similar even before garbling such as "official" and "officials"; "attention", "station" and "situation". These are words whose noisy versions can themselves easily be mis-recognized. The average percentage number of errors **per word** associated with these two sets was 36 % and 39.75 % respectively.

	SA	SB
Number of insertions	1872 (1.825)	2142 (2.088)
Number of deletions	418 (0.407)	414 (0.404)
Number of substitutions	769 (0.750)	822 (0.801)
Total number of errors	3059 (2.981)	3378 (3.292)
Percentage error	36.00%	39.75%

Table II: Noise statistics of the sets **SA** and **SB**. The figures in brackets are the average number of errors per word.

The three algorithms, Levenshtein Distance (Algorithm_LD), the Generalized Levenshtein Distance (Algorithm_GLD) and our algorithm (Algorithm_OPT_PR), were tested with the sets of 1026 noisy words, **SA** and **SB**. The results obtained in terms of the recognition accuracy for the two sets are tabulated below in Tables IV. Note that our scheme far outperforms the traditional string correction algorithm (eg. 97.66 % instead of 94.93 % in SA). It also outperforms the GLD algorithm (eg. 96.49 % instead of 94.35 % in SB). The reader should observe that, as in all PR problems, it is much harder to increase the recognition accuracy at the higher end of the spectrum. The power of our strategy is obvious !!

Algorithm	Accuracy (SA)	Accuracy (SB)
Algorithm_LD	94.93%	93.76%
Algorithm_GLD	96.00%	94.35%
Algorithm_OPT_PR	97.66%	96.49%

Table IV : The recognition results obtained from the sets **SA** and **SB**.

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. Also, we have only dealt with the concepts of syntactic PR in which the patterns are represented "linearly" as strings. The problem of developing optimal classifiers for PR systems using two-dimensional structures such as trees and webs still remains open. We currently have some initial results for the case of ordered tree representations.

VI. CONCLUSIONS

In this paper we have presented a formal foundation for designing optimal and informatic theoretic syntactic pattern recognizers. We have done this by presenting a new model for noisy channels which permit arbitrarily distributed substitution, deletion and insertion errors. 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 algorithms 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. Apart from the model having straightforward applications in string generation and recognition, we believe that it also has powerful potential applications in speech and uni-dimensional signal processing.

Acknowledgements : We are very indebted to Richard Loke for helping us prepare the final manuscript and for assisting us obtain the experimental results. We presented these results at a few seminars and we are very grateful to the audiences of these seminars for helpful discussions. We would particularly like to thank Prof. Haralick from the University of Washington.

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 H. Bunke, and J. Csirik, Parametric string edit distance and its application to pattern Recognition, *IEEE Trans. Systems, Man and Cybern.*, SMC-25:202-206 (1993).
4. L. Devroye, *Non-Uniform Random Variate Generation*, Springer-Verlag, (1986).
5. L. Devroye, W. Szpankowski and B. Rais, A note on the height of suffix trees, *SIAM J. of Computing*, 21:48-54, (1992).
6. G. Dewey, *Relative Frequency of English Speech Sounds*, Cambridge, MA, Harvard Univ. Press, (1923).
7. R. O. Duda, P.E. Hart. *Pattern Classification and Scene Analysis*. Wiley & Sons, 1973.
8. G.D. Forney, The Viterbi Algorithm, *Proceedings of the IEEE*, Vol. 61. (1973).
9. K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, 1972.
10. 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).
11. P. A. V. Hall and G.R. Dowling, Approximate string matching, *Comput. Surveys*, 12:381-402 (1980).
12. D. S. Hirschberg, Algorithms for longest common subsequence problem, *J. Assoc. Comput. Mach.*, 24:664-675 (1977).
13. D. S. Hirschberg, A linear space algorithm for computing maximal common subsequences, *Comm. Assoc. Comput. Mach.*, 18:341-343 (1975).
14. J. W. Hunt and T. G. Szymanski, A fast algorithm for computing longest common subsequences, *Comm. Assoc. Comput. Mach.*, 20:350-353 (1977).
15. P. Jacquet and W. Szpankowski, Analysis of digital tries with markovian dependencies, *IEEE Trans. Information Theory*, IT-37:1470-1475 (1991).
16. 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).
17. R. L. Kashyap and B. J. Oommen, The noisy substring matching problem, *IEEE Trans. Software Engg.*, SE-9:365-370 (1983).
18. 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).
19. R. L. Kashyap, and B. J. Oommen, String correction using probabilistic methods, *Pattern Recognition Letters*, 147-154 (1984).

20. 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).
21. R. Lowrance and R. A. Wagner, An extension of the string to string correction problem, *J. Assoc. Comput. Mach.*, 22:177-183 (1975).
22. A. Levenshtein, Binary codes capable of correcting deletions, insertions and reversals, *Soviet Phys. Dokl.*, 10:707-710 (1966).
23. W. J. Masek and M. S. Paterson, A faster algorithm computing string edit distances, *J. Comput. System Sci.*, 20:18-31 (1980).
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 (1980).
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).
40. K. S. Fu, *Syntactic Methods in Pattern Recognition*, Academic Press, New York, 1974.
41. B. J. Oommen and K. Zhang, "The Normalized String Editing Problem Revisited". To appear in the *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
42. L. R. Rabiner and B-H. Juang, "An Introduction to Hidden Markov Models", *IEEE Acc. Sound and Sig. Proc.*, pp.4-16 (1986)
43. A. Kundu, Y. He and P. Bahl, "Recognition of the Handwritten Word: First and Second Order Hidden Markov Model Based Approach", *Pattern Recognition*, 22: 283-297 (1989).
44. M. Y. Chen, A. Kundu, S. Srihari "Handwritten Word Recognition Using Morphological Segmentation and Variable Duration - Hidden Markov Model", *Proc. CVPR*, June 1993.

Original word (dictionary)	Noisy word	Total number of errors
accomplishments	accoplsuments	3
administration	sdmlnistratib	5
advance	ewafawdvxsance	7
advantage	taodbivawafxe	8
affairs	kafruvkfnixsrs	9
artillery	vuaegrdtuilllordiery	11
beginning	ssbehsimgjninmmg	10
children	cnuhialren	4
citizens	ciylzens	2
communicated	commuivated	2
community	cmmunrieztd	6
concept	concdxept	2
concepts	concewpts	1
control	fodvntopl	5
cooperation	coeopewryaueipxn	8
decreased	drcfreased	2
defender	cefenrded	3
developed	cdexsvfesulohned	9
development	dbvelrlpmenr	4
efficiency	efsfickiepncy	3
election	elecqtbibobn	4
employed	frmmpwuoycemac	10
enclosed	dnclbosed	2
excellent	excellejt	1
executive	yslxvkrcutivoe	7
existence	eaxidstenfe	3
facilities	fbacmwlifipeab	8
followed	zdslfdxllwkedekuid	14
judgment	juzdgseynyt	4
justice	jusbtipxe	3
necessity	jwscessitc	4
neither	either	1
reached	rlepdached	3
reasons	crezastons	3
simulated	ksimrujtated	4
sitting	psxruttoing	6
strength	mzeckieeotrenxsbth	13
striking	yvysatqrickinwet	10
telling	kmtelsling	3
terminal	teerlmnl	4
themselves	themslvs	2
victory	vbtctlavrdy	7
village	evvillagr	3
without	xvwigobuhnout	8
working	wrqkng	3
yourself	myorself	2

Table III : A subset of the dictionary, some noisy strings and their error characteristics.

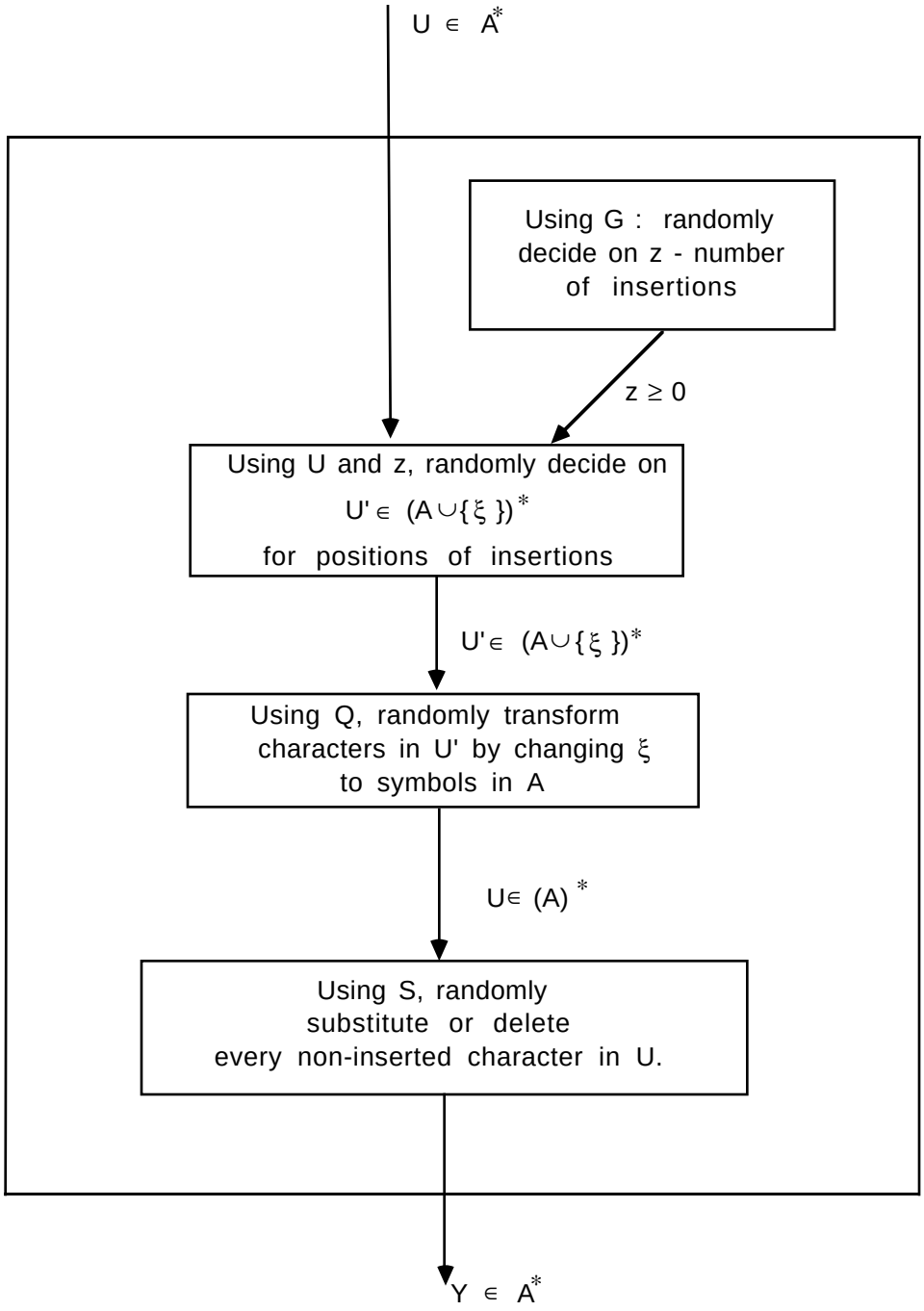

Figure I : A pictorial representation for the model for the channel. The input to the channel is the string U, and the output is the random string Y.

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t
a	867	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	1	20	1
b	1	861	1	1	1	1	14	14	1	1	1	1	1	20	1	1	1	1	1	1
c	1	1	861	14	1	14	1	1	1	1	1	1	1	1	1	1	1	1	1	1
d	1	1	14	835	14	20	1	1	1	1	1	1	1	1	1	1	1	10	20	1
e	1	1	1	14	857	5	1	1	1	1	1	1	1	1	1	1	1	20	14	1
f	1	1	14	20	10	824	20	1	1	1	1	1	1	1	1	1	1	20	1	10
g	1	14	1	1	1	20	835	20	1	1	1	1	1	1	1	1	1	5	1	14
h	1	14	1	1	1	1	20	835	1	20	1	1	1	14	1	1	1	1	1	5
i	1	1	1	1	1	1	1	1	861	10	14	5	1	1	20	1	1	1	1	1
j	1	1	1	1	1	1	1	20	10	835	20	1	14	14	1	1	1	1	1	1
k	1	1	1	1	1	1	1	1	14	20	848	20	14	1	10	1	1	1	1	1
l	1	1	1	1	1	1	1	1	5	1	20	876	1	1	14	14	1	1	1	1
m	1	1	1	1	1	1	1	5	1	14	14	1	876	20	1	1	1	1	1	1
n	1	20	1	1	1	1	5	14	1	14	1	1	20	857	1	1	1	1	1	1
o	1	1	1	1	1	1	1	1	20	1	14	14	1	1	861	20	1	1	1	1
p	1	1	1	1	1	1	1	1	1	1	1	14	1	1	20	893	1	1	1	1
q	14	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	892	1	5	1
r	1	1	1	14	17	14	5	1	1	1	1	1	1	1	1	1	1	863	1	17
s	17	1	1	17	10	1	1	1	1	1	1	1	1	1	1	1	5	1	841	1
t	1	1	1	1	1	14	14	10	1	1	1	1	1	1	1	1	1	17	1	85
u	1	1	1	1	1	1	1	14	17	14	10	1	1	1	1	1	1	1	1	1
v	1	17	17	5	1	14	14	1	1	1	1	1	1	1	1	1	1	1	1	1
w	10	1	1	5	17	1	1	1	1	1	1	1	1	1	1	1	17	1	14	1
x	5	1	17	14	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	1
y	1	1	1	1	1	1	14	14	1	5	1	1	1	1	1	1	1	1	1	17
z	14	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	1

Table I: The "confusion matrix" with the probabilities of substituting a character with another character. The figures in the table are to be multiplied by a factor of 10^{-3}.