RECOGNIZING SOURCES OF RANDOM STRINGS[1]

R.S. Valiveti and B.J. Oommen[2]

SCS-TR-161, JANUARY 1990
(Revised)

[1] Previous version was titled "On the data analysis of random permutations and its application to source recognition.

[2] Both authors partially supported by Natural Science and Engineering Research Council of Canada.

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

Recognizing Sources of Random Strings

R.S. Valiveti and B.J. Oommen *
School Of Computer Science
Carleton University
Ottawa, Canada, K1S 5B6

Abstract

Let us assume that we have a number of independent sources where each source generates random strings of fixed length M, composed of symbols drawn from an alphabet $\mathcal{R}$. Each source generates these random strings according to its own distribution. The problem we consider in this paper is one of identifying the source given a sequence of random strings. Two modes of random string generation are analyzed. In the first mode, arbitrary strings are generated in which the individual symbols can occur many times in the strings. In the second mode the individual symbols occur exactly once in each random string. The latter case corresponds to the situation in which the sources generate random permutations.

In both these cases, the best match to the distribution being used by each source can be obtained by maintaining an exponential number of statistics. This being infeasible, we propose a simple parametrization of the distributions. For arbitrary strings, the simple unigram based model (U-model) has been proposed. For the case of permutations, we have proposed a new model called the S-model and employed it to analyse and/or approximate **unknown** distributions of permutations. The relevant estimation procedures together with the applications to source recognition have been presented.

Considering the fact that the symbolic data is processed, and statistically analyzed, our method clearly presents a unique blend of syntactic and statistical pattern recognition.

Keywords:
Pattern Recognition, Bayesian Decision, Probability distributions, Parametric approximation, Parameter estimation, Random Permutation Generation, Approximation and modeling, Closeness of approximation,

*Both authors partially supported by Natural Sciences and Engineering Research Council Of Canada.

1 Introduction

Let $\mathcal{R}$ be a finite alphabet. We have c sources belonging to the set $\{\omega_1, \omega_2, \ldots, \omega_c\}$, which generate strings composed of the symbols of the alphabet $\mathcal{R}$. Let us assume, for simplicity that these sources generate strings having a fixed length M, which in practice, can be the buffer size. This means that each source generates one of a set of $|\mathcal{R}|^M$ possible strings, at any given time. Additionally each source generates a string from this set according to its *own* distribution. The problem for a receiver is to identify the (possible) source, from a sequence of random strings. We wish to study this problem, referred to as the Source Recognition Problem, in the framework of supervised pattern recognition.

In supervised pattern recognition, the system undergoes a period of training before it can be used for classification. In such a pattern recognition system, in the training phase, the input to the system is a set of training samples. From these samples, features are extracted with the understanding that the information contained *in the features* essentially constitutes the information in the samples. Once the features have been extracted, typically, the stochastic properties of the features are estimated, and these are subsequently used in the testing phase. When an unknown sample is received, the same features are extracted from the sample and using a pattern classification strategy, the sample is assigned to one of c distinct classes.

The technique which we use in the Source Recognition Problem is analogous in principle. However the technique is distinct because even in the training phase, each input is not a single sample but a finite stream of random strings, from a particular source. Since the information presented is essentially symbolic in nature, a syntactic "front-end" sub-system must be used before a pattern recognition strategy can come into effect.

A very obvious approach to attack the above problem exists. Given that "only" $|\mathcal{R}|^M$ strings could be generated by the sources, the brute force approach would suggest that the features to be extracted consist of the $|\mathcal{R}|^M$ estimates per source — with each estimate denoting the probability of occurrence of a specific string. This approach is impractical even if $|\mathcal{R}|$ is as small as 2. The other alternative to feature extraction is to construct a model of the string generation strategy used by the source and then to estimate the parameters of the model. One simple minded model assumes that the source ω_i generated each symbol of the output string *independently*, according to a fixed distribution U_i. This distribution is completely specified by a vector of dimension $|\mathcal{R}|$, typically called the Unigram distribution. More sophisticated models would account for statistical dependence between adjacent symbols. When the dependence is merely specified in terms of the immediately

preceding symbol, we have the familiar bigram model of string generation. Higher order dependencies can easily be considered. Some applications of such a simple minded modeling strategy are found in Section 2.

To explore the idea of statistical dependence further, let us consider an extreme form of dependence analogous to the "without replacement" model of computation. In this mode, once a symbol has appeared in the string, it cannot again appear anywhere in the string. If we assume that $|\mathcal{R}| > M$, then the sources are generating random subsets of size M from the alphabet $\mathcal{R}$. On the other hand, if $|\mathcal{R}| = M$, the sources are generating permutations of the alphabet $\mathcal{R}$. Recognizing the sources in the case of such dependencies will also be studied in the paper.

In order to extract features from a stream of permutations, a very simple method would recommend maintaining the $M!$ frequency counters which estimate the probabilities with which the given source generates the various permutations. Clearly such a strategy is infeasible even when M is as small as 10. Considering the above, our first major deviation is to model each permutation generator in a fashion which requires a linear (as opposed to $M!$) number of parameters. We refer to this model as the S-model, and it is not out of place to mention that these parameters constitute a probability distribution in themselves.

In the training phase, when we are given a stream of permutations coming from a given source, we estimate the parameters of the S-model of this source. These parameters will be referred to as the S-vector or the S-parameters. It will be seen that each stream of permutations yields us a single S-vector, and if we are given a training set consisting of c such independent streams (i.e. one from each source), we can estimate the S-vector for each source. In other words, at the end of the training phase, the system has c estimates for the S-model parameters of these c sources. We refer to these parameters as $\hat{\mathcal{S}}_1, \hat{\mathcal{S}}_2, \hat{\mathcal{S}}_3, \ldots, \hat{\mathcal{S}}_c$, and the techniques involved in computing these estimates will be discussed in detail in this paper.

Coming now to the testing phase, the problem is essentially one in which we associate a stream of permutations with an unknown source, with the intent of minimizing the probability of misclassification. Given this stream, say Q_{test}, the strategy which could be proposed is as follows. By examining Q_{test}, the Maximum Likelihood Estimate of the S-parameters which could have generated it can be computed. Let $\hat{\mathcal{S}}_{test}$ be the vector obtained by this process. The classification can be achieved by comparing $\hat{\mathcal{S}}_{test}$ to each one of $\hat{\mathcal{S}}_1, \hat{\mathcal{S}}_2, \hat{\mathcal{S}}_3, \ldots, \hat{\mathcal{S}}_c$, and the stream Q_{test} can be associated with the source whose S-parameters are closest to $\hat{\mathcal{S}}_{test}$. In this context, one possible measure of closeness is the Eucledean Metric.

Although the above method conforms to that used in traditional pattern recognition systems, we have opted to choose a different route. Viewed from our perspective, the above approach can be quite erroneous because in the testing of one stream of permutations, Q_{test}, the number of permutations presented to the system, (i.e. the size of Q_{test}) could be very small. Thus even though accurate estimates of the S-model parameters of the sources can be obtained in the training phase, the accurate estimation of $\hat{\mathcal{S}}_{test}$ may not be possible. If the number of permutations in Q_{test} is small, the estimate $\hat{\mathcal{S}}_{test}$ is either not obtainable or is grossly inaccurate, because of the singularities that are encountered in the estimation process. Thus such a minimum distance classifier cannot be expected to perform in any reasonably accurate fashion.

As opposed to this, our strategy utilizes the maximum likelihood classifier. Rather than resorting to a parameter estimation procedure in the testing phase, in our strategy, we compute the probability with which a particular source ω_i could have generated Q_{test}. The source ω_j for which this probability is maximized is declared to be the generator of Q_{test}. This computation encounters no singularities and thus the classification achieved is both accurate and fast, even when the size of Q_{test} is small.

The data that we analyze is symbolic and thus, it is clear that a syntactic "front-end" is mandatory. Furthermore, the information contained is not merely in the strings but in the arrangement of the symbols within a string. Thus the structure of the stream of random strings itself contains information of paramount importance. Finally once this piece of information has been gleaned, statistical pattern recognition must come into effect, during the estimation and classification phases. Thus, it is our belief that our method presents a unique blend of syntactic, "structural", and statistical pattern recognition.

In Section 2 we present the unigram model, called the U-model for arbitrary strings generators. An example application involving encrypted messages is also presented in the same section. In Section 3 we shall describe the alternate S-model for permutation generators. Apart from the S-model being directly applicable to the problem on hand, various other applications can be envisioned. These applications are also catalogued, for the sake of completeness, in Section 3. The analytic properties of this model, together with the estimation procedures are described in Section 4. The use of S-model in source recognition is discussed in Section 5. Experimental results involving multiple sources of permutations are also presented there.

2 The U-model for Arbitrary Strings

We have alluded to the fact that a precise modeling of a source of random strings of length M, based on a finite alphabet $\mathcal{R}$ requires $|\mathcal{R}|^M$ estimates. In this section, we propose a very simple but useful model, called the unigram U-model, for such sources. This model uses $|\mathcal{R}|$ parameters instead of the previously indicated number.

U is a vector describing the distribution of the symbols of $\mathcal{R}$ in received strings. It is assumed that each one of the M symbols is generated, independent of the prefix already generated, according to this distribution U. The U-model essentially implies that the elements of $\mathcal{R}$ are generated according to a a multinomial distribution. Using this information, it is possible to compute the Maximum Likelihood Estimate for the vector U for any specific source. It is easy to show that this maximum likelihood estimate coincides with the intuitive frequency based estimate [2].

The U-model as described above simply corresponds to one in which only the unigrams of the symbols (i.e. individual probabilities of occurrence) are maintained. An immediate extension to this model can be suggested. In this, the U-vector need not remain fixed for all positions in a specific generated string, but could vary from position to position in a Markovian fashion. With some insight, this can be seen to exactly correspond to the bigram model of string generation.

We shall now explore the usefulness of the U-model. It is clear that due to its simplicity this model *may* be an unreasonable one, if the sources themselves use the bigram (or in general, the n-gram) probabilities for the generation strategy. In such cases, the U-model based on the *original* alphabet need not be accurate. However, it is conceivable that with regard to some transformed alphabet, a unigram model may be acceptable. To clarify this point consider the case when the sources generate 256 bytes in a single transmission unit, where the original alphabet may consist of the letters 'a' ... 'z' (or the ASCII set). Given the same block of 2048 bits, we can sample selected bit positions of this bit string, to form the primitive symbols of a *new* alphabet. Instead of modeling the unigrams of the symbols of the underlying source alphabet, it may be convenient and even efficient to model the source string using the unigrams of the *transformed* alphabet. This is indeed a justified assumption, as can be seen from the following sample application.

Consider the case when we have c sources $\Omega = \{\omega_1, \omega_2, \ldots, \omega_c\}$ where each source is generating random strings according to the some underlying distribution D, *which need not be a unigram based distribution.* The output of each source is separately encrypted — for example, by using a common encryption scheme, but with each source having its own distinct

"key". This situation is depicted in Figure 1. Following the encryption, we effectively have a new set of c sources $\Omega' = \{\omega'_1, \omega'_2, \ldots, \omega'_c\}$. Thus distinguishing between these sources in Ω', is equivalent to distinguishing between the original sources or alternatively, between the individual encryption keys.

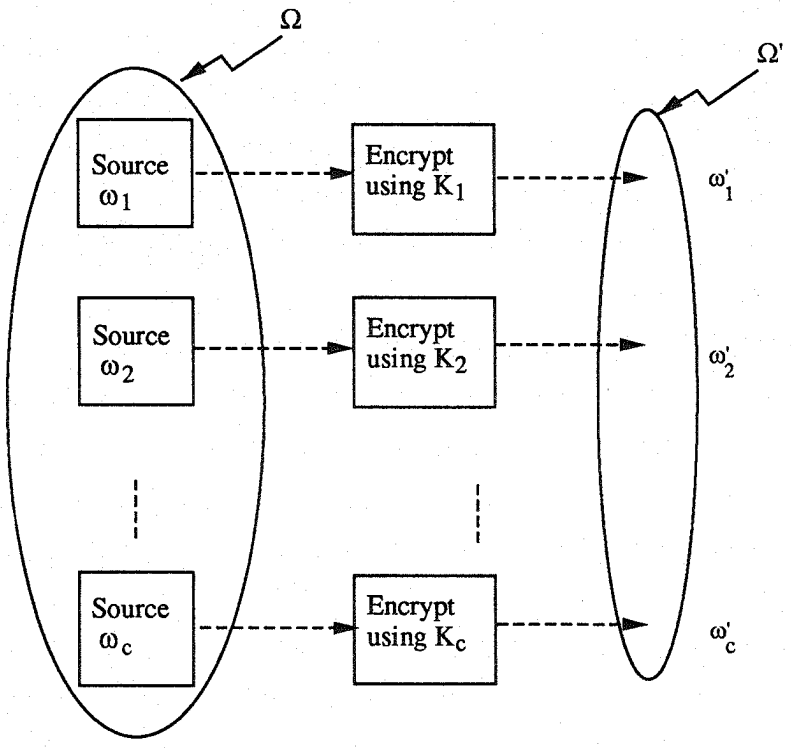

Figure 1: Set up for generating data for source recognition. Every source ω_i generates data based on the English bigram distribution. The data generated by the source ω_i is encrypted using the key K_i to yield the output of the pseudo-source ω'_i.

To justify the use of the U-model in source recognition, we have conducted various experiments involving character text and a well known encryption scheme. In our experiments, the method used by the sources in the set Ω is the string generation technique based on English bigrams, and not on the unigrams alone. The encryption is performed using the Rivest-Shamir-Adleman (RSA) scheme [1]. In this scheme, if P and E denote the plaintext and the encrypted string respectively, the operation of encryption can be described as follows:

$$E = P^e \bmod (p * q)$$

where p, q, and e are parameters of the encryption scheme, and are referred to as the "key". In this setting, the sources of multiple strings are simply obtained by encrypting the plaintext P under different "keys". The problem of source recognition immediately leads to one of distinguishing between the keys used by the sources for their encryption.

The details about the experiments conducted are as follows. Table 1 gives a list of all the encryption keys used in the experiments. It must be noted that only a subset of these keys were used in any single experiment, and this subset depends on the number of sources being distinguished. Specifically, if the number of classes is c, we would use the keys $K_1, K_2, \ldots, K_c$ from Table 1. In each case, the input data was generated as a continuous stream of characters, based on the English *bigrams*, and this stream was sub-divided into units of four bytes length. Each such sequence of the string generated by the source ω_i was encrypted using key K_i, and this stream of data was reckoned to be the encrypted bit string generated by the source ω_i'. To obtain the relevant U-model, we have regarded four consecutive bits in this encrypted sample to represent one symbol of the transformed alphabet. Therefore, the alphabet describing the output of the sources in the set Ω' consists of 16 symbols, namely the hexadecimal digits {0...9, A...F}. Consequently, a unigram model for *this* output stream would comprise of a vector of dimension 16.

In our experiments, for each of the sources, the training phase presented the recognition system with 1000 sample encrypted bit strings. The features extracted from these samples was a single 16-dimensional vector for each source.

In order to test the accuracy of the recognition system, experiments were performed with varying number of classes. For each value of the number of classes, the recognition system was tested with 3200 different bit strings generated by the sources in the set Ω'. Each bit string Q_{test} was composed of only 100 4-byte units as described earlier. The system estimated the unigram vector for Q_{test} and subsequently utilized the Maximum Likelihood Classifier for determining the source of the string. For each value of the number of classes, the keys chosen for encryption are shown in Table 2. Observe that when the number of classes was 8, we obtained an accuracy of 99.93%. When the number of classes increased to 32, the recognition accuracy obtained was 99.56%. The results are indeed remarkable. What is perhaps more surprising is the fact that even though our classifier is not of the hierarchical type, it performs quite well in a 32 class problem.

Thus, even a simple model such as the U-model may be acceptable for source recognition if the underlying strings are arbitrary and are from a finite alphabet.

We now turn our attention to the far more complex case when the strings generated

Key	p	q	e
K_1	31	751	4901
K_2	37	601	14059
K_3	43	409	13981
K_4	47	631	24947
K_5	53	61	71
K_6	53	401	1461
K_7	67	541	10493
K_8	71	349	23509
K_9	73	127	5287
K_{10}	83	179	3145
K_{11}	89	503	2247
K_{12}	97	443	19765
K_{13}	101	281	20063
K_{14}	107	281	4351
K_{15}	113	503	38133
K_{16}	113	257	23789
K_{17}	127	151	15277
K_{18}	131	409	20209
K_{19}	137	389	4817
K_{20}	139	317	14987
K_{21}	149	179	25743
K_{22}	151	307	27467
K_{23}	157	233	27305
K_{24}	163	307	41159
K_{25}	167	229	32755
K_{26}	173	301	5413
K_{27}	197	263	27341
K_{28}	199	263	46285
K_{29}	211	257	51811
K_{30}	229	257	57397
K_{31}	239	281	49369
K_{32}	541	31	211

Table 1: This table specifies the various keys used by the RSA scheme. A source ω_i is characterized by its unique key K_i. The results of source recognition are given in Table 2.

are not arbitrary strings, but are constrained to be permutations. In this case, a model similar to the U-model is applicable, except that the associated pattern recognition phases of estimation and classification become more complicated.

Experiment No.	No Of Classes	Keys Used	No of Missclassifications	Accuracy (in %)
I	8	$\{K_1 \ldots K_8\}$	2	99.937
II	16	$\{K_1 \ldots K_{16}\}$	3	99.906
III	24	$\{K_1 \ldots K_{24}\}$	7	99.781
IV	32	$\{K_1 \ldots K_{32}\}$	14	99.562

Table 2: This table shows the recognition accuracy obtained when the source recognition system was used to distinguish between several keys. For each experiment, 3200 bit strings were classified using the Maximum likelihood Classifier described in Section 2.

3 The S-Model for Permutation Generators

As has been mentioned earlier, the brute force approach to discovering the generation strategy of a stream of permutations is to maintain estimates of probabilities for each of the possible permutations. This is impractical, for it involves maintaining $M!$ estimates.

If we are to arrive at a sensible solution to the above problem we must drift from the notion of estimating all occurrence probabilities. Instead, we must focus on well defined and easily computable parametric approximations to the distributions of all permutations. In doing so, we would have to be careful to keep the number of parameters estimated as low as possible, but still arrive at an acceptable closeness between the underlying distribution and the one dictated by the (simple) model.

The crucial point to realize is that any reasonable **model** of permutation generation will be based on an assumed underlying **generation strategy**. When M is small (typically, $M \leq 6$), any arbitrary distribution $\mathrm{G}(\pi)$ can be specified by means of simple assignment statements. This obviously entails the enumeration of $\mathrm{G}(\pi_i)$ for all $\pi_i \in \Pi$, where Π is the set of all possible permutations. When M is large, such a strategy is infeasible because of the finite precision that can be expected of random number generators. The solution presented in [6] circumvents both these problems. We shall now describe the assumed underlying generation strategy.

Let $\mathcal{R} = \{R_1, R_2, \ldots, R_M\}$ be an ordered set of M distinct elements where $R_i < R_j$ if $i < j$. In the case of sources emitting random permutations, the elements of $\mathcal{R}$ quite simply consist of the individual symbols of the alphabet. When $M \gg 1$, we assume that the underlying data generation strategy can be completely specified by a control vector of dimension M. This control vector $\mathcal{S}$ is an $M \times 1$ probability vector denoted by $[s_1, s_2, \ldots, s_M]^T$ which satisfies:

$$\sum_{i=1}^{M} s_i = 1.$$

Let $\mathcal{V}$ be any subset of $\mathcal{R}$. We define the vector $\mathcal{S}_\mathcal{V}$ as the **conditional** control vector of (normalized probabilities) in which only the quantities corresponding to the members of $\mathcal{V}$ have non-zero values. Thus $\mathcal{S}_\mathcal{V}$ consists of normalized probabilities $[s'_i]$, where:

$$s'_i = \begin{cases} 0 & \text{if } R_i \notin \mathcal{V} \\ \dfrac{s_i}{\sum_{R_i \in \mathcal{V}} s_i} & \text{otherwise} \end{cases} \tag{1}$$

Observe that the vector $\mathcal{S}$ is exactly equivalent to $\mathcal{S}_\mathcal{R}$.

Given the vector parameter $\mathcal{S}$, we assume that the underlying generation strategy is as follows. For the sake of explanation, let $X \in \Pi$ be a randomly generated piece of data

(permutation) of $\mathcal{R}$. We denote X by $[x_1, x_2, \ldots, x_M]$. The generation of a random permutation X clearly consists of randomly assigning a position to each element of $\mathcal{R}$. The S-model proposes that this is done by computing the prefixes of X in sequence. Initially x_1 is randomly assigned an element in $\mathcal{R}$, based on the control vector $\mathcal{S}$(which is the same as $\mathcal{S}_{\mathcal{R}}$). By this we mean that element R_i is chosen with probability s_i. The subsequent problem is simply one of generating a (random) permutation of the $(M-1)$ elements of $\mathcal{R} - \{x_1\}$. The S-model proposes that this generation is done using the control vector $\mathcal{S}_{\mathcal{V}}$ and the process is recursively repeated by successively updating $\mathcal{V}$. For the sake of clarity, we present an algorithmic form of the above (assumed) generation strategy.

Algorithm S-model

Input:
The control vector $\mathcal{S} = [s_1, s_2, \ldots, s_M]^T$,
where the components of $\mathcal{S}$ satisfy $\sum_{i=1}^{M} s_i = 1$.
Output:
A permutation $\{x_1, x_2, \ldots, x_M\}$ of $\mathcal{R}$.
Method:

```
begin
    V:= R;
    for i := 1 to M do
    begin
        Compute the conditional distribution S_V
        according to (1).
        Choose an element x_i according to this distribution.
        V := V - {x_i}
    endfor
end
```

End Algorithm S-model

Program 1: S-model for permutation generation

The next section introduces the notation that we will be using in the rest of the paper.

3.1 Notation

We use the notation $R_u \rightarrow j$ to denote the fact that the element R_u appears at the j^{th} position in the permutation. Furthermore, $R_u, R_v \rightarrow < i, j >$ represents the fact that the elements R_u and R_v appear at the positions i and j in the permutation, respectively. The extensions to this notation, where we are specifying the position of more than two elements

are straightforward and are interpreted in an intuitive fashion.

$R_u \prec R_v$ denotes the event that element R_u precedes element R_v in a given permutation.

We now proceed to state and prove a very important property of the permutations generated by **Algorithm S-model.**

Theorem 1 *Let $\mathcal{S}$ be the (M-dimensional) parameter vector of the S-model. Also let $R_u, R_{v_1}, \ldots, R_{v_K}$ be distinct elements of $\mathcal{R}$. Then the probability that the element R_u precedes each one of the elements of the set $\{R_{v_1}, \ldots, R_{v_K}\}$ in any permutation generated by Algorithm S-model, is given by:*

$$Pr(R_i \prec \{R_{v_1}, \ldots, R_{v_K}\} \mid \mathcal{S}) = \frac{s_u}{s_u + \sum_{j=1}^{K} s_{v_j}} \tag{2}$$

Proof:

We prove this result by induction. Observe that the result stated in Theorem 1 must be proved for all (feasible) values of K and M. Since $\{R_{v_1}, \ldots, R_{v_K}\}$ can at most include all the elements of $\mathcal{R} - \{R_u\}$, it follows that $K \leq M - 1$. A traditional induction on K would merely involve proving that the result is true for all K satisfying $K \leq M - 1$. Although M is a constant in our original setting, it must be treated as a parameter, to render the proof valid. The rationale for considering M as a parameter is as follows. For a given M the result could be proved by induction, without thereby implying that the result would continue to be true if the dimensionality of the problem M is changed. Observe that (2) states the assertion of the theorem independent of the dimension of the set $\mathcal{R}$.

As a consequence of the above reasoning, it appears that a "two-dimensional induction", i.e. an induction on two variables is unavoidable. However, we shall reduce this into a single parameter induction. This is done as follows. For a given M, we shall prove that (2) is valid if it is valid for all values of K and $M = M - 1$. The formal steps of the proof follow.

The theorem is trivially valid if $K = M - 1$. In this case, the denominator of (2) essentially consists of the sum $\sum_{i=1}^{M} s_i$, which is unity. The expression in (2) thus simplifies to s_u — which is indeed the probability that the element R_u appears before each of the other elements of $\mathcal{R}$, i.e. in the first position.

We now proceed by induction on the parameter M. Let us assume that the theorem holds for all values of $K \leq M - 1$, for a specific value M_0 of the parameter M. We will establish that the same property holds for $M = M_0 + 1$ as well.

To establish the basis for the induction, we examine the case when $M = 2$. In this case the only meaningful value of K is 1. Since $K = M - 1$, the result indeed holds for the case when $M = 2$. We now proceed with the induction step.

Let $\mathcal{S} = [s_1, s_2, \ldots, s_M]^T$, where $M = M_0 + 1$. In order to derive the probability that the element R_u precedes each one of the elements in $\{R_{v_1}, \ldots, R_{v_K}\}$, we note that this event happens if:

(i) R_u appears in the first position.

(ii) Any element not contained in the set $\{R_u\} \cup \{R_{v_1}, \ldots, R_{v_K}\}$ appears in the first position and *then* R_u (recursively) precedes $\{R_{v_1}, \ldots, R_{v_K}\}$ in the permutation of the remaining $M - 1$ elements.

Note that the events labelled as (i) and (ii) are mutually exclusive and hence using the notation of Section 3.1, from the law of total probability, we have:

$$\begin{aligned} & Pr(R_u \prec \{R_{v_1}, \ldots, R_{v_K}\} \mid \mathcal{S}) \\ &= Pr(R_u \rightarrow 1) + \sum_{R_l \notin \{R_u\} \cup \{R_{v_1}, \ldots, R_{v_K}\}} Pr(R_l \rightarrow 1) Pr(R_u \prec \{R_{v_1}, \ldots, R_{v_K}\} \mid \mathcal{S}_{\mathcal{R}-\{R_l\}}) \\ &= s_u + \sum_{R_l \notin \{R_u\} \cup \{R_{v_1}, \ldots, R_{v_K}\}} s_l Pr(R_u \prec \{R_{v_1}, \ldots, R_{v_K}\} \mid \mathcal{S}_{\mathcal{R}-\{R_l\}}) \end{aligned} \tag{3}$$

We note that $\mathcal{S}_{\mathcal{R}-\{R_l\}}$ is a control vector of dimension M_0, and can be denoted as $[s'_\beta \mid \beta \in \{1, 2, \ldots, M_0\} - \{l\}]^T$ and hence each of the probabilities in the RHS of equation (3), can be calculated using the induction hypothesis. To illustrate, let us consider the term:

$$Pr(R_u \prec \{R_{v_1}, \ldots, R_{v_K}\} \mid \mathcal{S}_{\mathcal{R}-\{R_l\}}) = \frac{s'_u}{s'_u + \sum_{j=1}^{K} s'_{v_j}}. \tag{4}$$

Notice that $s'_\beta = s_\beta/(1 - s_l)$, since $1 - s_l$ is the normalizing factor for **all** the probabilities in the control vector $\mathcal{S}_{\mathcal{R}-\{R_l\}}$. Therefore:

$$Pr(R_u \prec \{R_{v_1}, \ldots, R_{v_K}\} \mid \mathcal{S}_{\mathcal{R}-\{R_l\}}) = \frac{s_u}{s_u + \sum_{j=1}^{K} s_{v_j}}. \tag{5}$$

Notice that this probability does not dependent on the index l. Combining (5) with (3), we have:

$$\begin{aligned} Pr(R_i \prec \{R_{v_1}, \ldots, R_{v_K}\} \mid \mathcal{S}) &= s_u + \sum_{R_l \notin \{R_u\} \cup \{R_{v_1}, \ldots, R_{v_K}\}} s_l \frac{s_u}{s_u + \sum_{j=1}^{K} s_{v_j}} \\ &= s_u \left(1 + \frac{\sum_{R_l \notin \{R_u\} \cup \{R_{v_1}, \ldots, R_{v_K}\}} s_l}{s_u + \sum_{j=1}^{K} s_{v_j}} \right) \\ &= \frac{s_u}{s_u + \sum_{j=1}^{K} s_{v_j}} \cdot \left(\sum_{l=1}^{M_0+1} s_l \right) \\ &= \frac{s_u}{s_u + \sum_{j=1}^{K} s_{v_j}}. \end{aligned} \tag{6}$$

Let $f(K,M)$ represent the fact that (2) is true for specific values of K and M. Thus, as a consequence of (6) we have,

$$\{\forall K \leq M-1\}\,[f(K,M)\,] \Rightarrow \{\forall K \leq M-1\}\,[f(K,M+1)\,]$$

Since f(M, M+1) is trivially true, we have effectively proved that

$$\{\forall K \leq M-1\}\,[f(K,M)\,] \Rightarrow \{\forall K \leq M\}\,[f(K,M+1)\,]$$

This completes the proof. □

Corollary 1 *In a permutation generated by the S-model, the probability that a element R_u precedes another element R_v is simply given by $s_u/(s_u+s_v)$.*

Proof:

The result is a special case of (2) when $K=1$. This agrees with the result derived in [6]. □

We shall make extensive use of Corollary 1, in deriving the estimates for the control vector S. This aspect is the subject of the next section. However, for the sake of completeness, it is not inappropriate to present a few additional applications of the S-model. These applications are in areas not directly related to the source recognition problem.

3.2 Applications of the S-model

It can be seen that the S-model can be used generate random permutations, with a wide variety of distributions. This for example can be used to derive fast approximate solutions to the classical Travelling Salesman Problem (TSP). This problem belongs to the well-known class of NP-hard problems. However, a very practical approach for solving TSP, based on the idea of "simulated annealing" exists. "Simulated annealing" is essentially a probabilistic global minimum (or maximum) finding procedure. In the case of the TSP, we randomly choose an initial permutation of cities. The procedure to solve this problem maintains information about the shortest route encountered so far. If a different, shorter route is found, the algorithm revises its idea of the best route. The key idea is one of experimenting with several possibilities and eventually converging to the correct minimum (or maximum). Crucial to the simulated annealing family of algorithms is the ability to perturb the current configuration of the system — which in our case would be equivalent to the permutation of cities. These perturbations could be affected by suitably invoking a random permutation generator.

A more specialized application involves the pattern recognition of strings [3]. Consider the problem of a string X_s being transmitted over a noisy channel. If the noisy channel induces substitution, insertion and deletion errors, the received string, say Y can be used in conjunction with the characteristics of the channel to estimate X_s. Typically, this involves the editing of Y to every X in the dictionary, and techniques are available to achieve fast editing. If, on the other hand, the channel also induces permutation errors (i.e. the channel randomly swaps symbols of X_s), the problem is far more complex. This is due the fact that the editing of Y to every X with substitutions, insertions, deletions and transpositions is an NP-complete problem [5]. In this case, we anticipate that the techniques we present, can be used to perform the pattern recognition. Indeed, the strategy we propose is as follows. We first model the noisy channel as a cascading of two sub-mechanisms, the first of which performs a random permutation and the second which garbles the string using substitution, insertion and deletions. Clearly if the parameters of these model can be extracted using the techniques suggested here, the receiver can try to estimate the string X_s by mimicking the channel to obtain an estimate that yields the minimum probability of error. This is analogous to the method presented in [4] for substitution, insertion, and deletion errors. We are currently investigating a feasible solution to this problem.

For a third example, consider a typical operating (or scheduling) system in which M users, $\{R_j \mid 1 \leq j \leq M\}$ submit their jobs according to random priorities $\{z_j(i) \mid 1 \leq j \leq M\}$. Let us assume that the tasks are executed in the decreasing order of their priorities. Thus a typical scheduling sequence consists of a permutation of the set $\{R_j \mid 1 \leq j \leq M\}$. Notice that if the on the i^{th} day the set of priorities are $\{z_j(i) \mid 1 \leq j \leq M\}$, the scheduling sequence for that day will be $\pi(i) \in \Pi$, and one such permutation results from each job submission. Typical questions which the operating system is faced with would involve analyzing the data so as to project the mean position of user R_j in any scheduling sequence. Observe that this is no hypothetical situation, but is commonplace in every scheduling application.

We now focus our attention on the estimation of the parameters of the S-model.

4 Modeling Distributions Using the S-model

In the following analysis, we shall use the notation $\mathbf{X}$ to denote the vector (of dimension M) representing a permutation of $\mathcal{R}$. Let us further assume that P($\mathbf{X}$) denotes the actual (unknown) underlying density function and $P_a(\mathbf{X})$ denotes the approximate distribution arrived at, using the "S-Model" of permutation generation. Then the closeness measure between these distributions can be computed using the well known information theoretic

metric as:

$$I(P, P_a) = \sum_{\mathbf{X}} P(\mathbf{X}) \log \frac{P(\mathbf{X})}{P_a(\mathbf{X})} \tag{7}$$

It is well known that $I(P, P_a) \geq 0$ and that $I(P, P_a) = 0$ only if the approximation is exact.

In the light of this closeness metric, it is clear that we must choose the approximation $P_a(\mathbf{X})$ in such a way that the closeness measure is minimized. Since the approximating distribution is completely specified by the control vector $\mathcal{S}$, our intention of finding the best approximating density function translates to the problem of finding the best estimate for the vector $\mathcal{S}$. In the following discussion we therefore focus on the aspect of estimating the control vector $\mathcal{S}$ subject to this constraint.

At this point it is useful to write down the expression for the probability of generating a specific permutation $\mathbf{X} = [x_1, x_2, \ldots, x_M]$ when the underlying control vector is $\mathcal{S} = [s_1, s_2, \ldots, s_M]^T$. To calculate the above probability, we use the *Chain rule*, and the notation of Section 3.1, to yield:

$$\begin{aligned} Pr(x_1, x_2, \ldots, x_M) &= Pr(x_1 \rightarrow 1) \cdot Pr(x_2 \rightarrow 2 \mid x_1 \rightarrow 1) \cdot \\ &\quad Pr(x_3 \rightarrow 3 \mid x_1, x_2 \rightarrow < 1, 2 >) \cdot \\ &\quad \cdots Pr(x_M \rightarrow M \mid x_1, x_2, \ldots, x_{M-1} \rightarrow < 1, 2, \ldots, M-1 >). \end{aligned} \tag{8}$$

Clearly $Pr(x_1 \rightarrow 1)$ is s_{x_1}. Thus, a specific application of Theorem 1 yields the following expressions for the first few factors in (8):

$$\begin{aligned} Pr(x_2 \rightarrow 2 \mid x_1 \rightarrow 1) &= \frac{s_{x_2}}{1 - s_{x_1}} \\ Pr(x_3 \rightarrow 3 \mid x_1, x_2 \rightarrow < 1, 2 >) &= \frac{s_{x_3}}{1 - s_{x_1} - s_{x_2}} \end{aligned} \tag{9}$$

The generalizations of (9) for indices larger than three is very simple and hence combining (8) and (9) we obtain the expression for the joint probability $Pr(x_1, x_2, \ldots, x_M)$ as:

$$Pr(x_1, x_2, \ldots, x_M) = \prod_{i=1}^{M-1} \frac{s_{x_i}}{1 - \sum_{j=1}^{i-1} s_{x_j}} \tag{10}$$

At this juncture, two methods for the estimation of the control vector can be seen. The first method is based on minimizing the closeness metric $I(P, P_a)$. The second method is the well known Maximum likelihood estimation of parameters [2]. Although superficially these two methods appear to be different, we shall show that they essentially yield the same

estimates for the control vector $\mathcal{S}$. We now proceed to state and prove a theorem concerning the "volume" of "sufficient statistics" to be maintained, in order to attempt calculation of the Maximum Likelihood Estimate.

Theorem 2 *The maximum likelihood estimate of the control vector $\mathcal{S}$ can be obtained only if we maintain an exponential amount of statistics.*

Proof:

Let $X^1, X^2, \ldots, X^N$ denote the N independent samples of permutations of the elements of $\mathcal{R}$= $\{R_1, R_2, \ldots, R_M\}$, where each element of $\mathcal{R}$ is fully identified by its index. We use the notation that α_i^k is the i^{th} element in the k^{th} sample permutation X^k. It is useful to recall that the Maximum likelihood estimate for the control vector $\mathcal{S}$ is that vector which supports the N samples in the strongest manner and thus the likelihood of generating this set of permutations is maximized to yield the best estimate of the parameters.

We can use (10) to calculate the probability of generating any one of the sample permutations presented to us. In particular, the probability of generating the sample permutation X^k can be written from (10) as:

$$Pr(X^k) = \prod_{i=1}^{M-1} \frac{s_{\alpha_i^k}}{1 - \sum_{j=1}^{i-1} s_{\alpha_j^k}}$$

Therefore, based on the fact that the permutations themselves are independently generated[1], the likelihood function $\mathcal{L}$ is given by

$$\begin{aligned} \mathcal{L} &= \prod_{k=1}^{N} Pr(X^k) \\ &= \prod_{k=1}^{N} \prod_{i=1}^{M-1} \left\{ \frac{s_{\alpha_i^k}}{1 - \sum_{j=1}^{i-1} s_{\alpha_j^k}} \right\} \end{aligned} \tag{11}$$

In order to simplify the expression for the likelihood function in (11), we define the following quantities:

$\mathcal{N}_i^j$	Number of times the integer i occurs in position j
$\mathcal{N}_{ij}$	Number of times the pair (i,j) appears in one of the positions (1,2) or (2,1).
$\mathcal{N}_{\alpha_1,\alpha_2,\ldots,\alpha_r}$	Number of times the tuple $(\alpha_1, \alpha_2, \ldots, \alpha_r)$ or any permutation of it appears in the positions $1, 2, \ldots, r$.

[1]Note that this does not imply that the symbols **within** a permutation are independently generated.

We now proceed to simplify the expression in (11). Observe that each of α_i^k is an integer in the range {1, 2, ..., M}. For concreteness, let us consider the term s_1. s_1 occurs in the innermost product of equation (11), if the integer 1 appears in the permutation in any one of the positions $\{1 \ldots M-1\}$. Similar conclusion holds w.r.t. the other parameters $s_2, \ldots, s_M$. Also note that in the denominator of (11), the term $(1 - s_i)$ occurs $\mathcal{N}_i^1$ times; $(1 - s_i - s_j)$ occurs $\mathcal{N}_{ij}$ times and so on. Thus, the logarithm of the likelihood function $\mathcal{L}$ defined in (11) can be written down as:

$$\begin{aligned} \ell &\triangleq \log \mathcal{L} \\ &= \sum_{j=1}^{M-1} \mathcal{N}_i^j \cdot \log s_i - \sum_{i<j} \log(1 - s_i - s_j) \cdot \mathcal{N}_{ij} - \sum_{i<j<k} \log(1 - s_i - s_j - s_k) \cdot \mathcal{N}_{ijk} \\ &\quad \cdots - \sum \log(1 - s_{i_1} - s_{i_2} \cdots - s_{i_{M-2}}) \cdot \mathcal{N}_{i_1, i_2, \ldots, i_{M-2}} \end{aligned} \tag{12}$$

We note that $\sum_{j=1}^{M-1} \mathcal{N}_i^j = N - \mathcal{N}_i^M$, and hence, instead of maintaining $O(M^2)$ counters of the form $\mathcal{N}_i^j$, we can maintain exactly M counters which correspond to the frequencies of each of the integers $1, 2, \ldots, M$ appearing in the last position in the permutation.

It is easy to see that the number of counters to be maintained for $\mathcal{N}_{\alpha_1, \alpha_2, \ldots, \alpha_r}$ is simply given by $\binom{M}{r}$. Hence the total amount of information to be gathered is given by

$$M + \sum_{r=2}^{M-2} \binom{M}{r} = \sum_{r=1}^{M-2} \binom{M}{r} = 2^M - M - 2.$$

This establishes the claim that the number of statistics to maintain is exponential in the parameter M. □

The above result has given an idea of the amount of statistics to be gathered in order to arrive at a Maximum Likelihood estimates for the control vector $\mathcal{S}$. Although this method of arriving at the estimate for the control vector is difficult to implement, it possesses one important characteristic. We will show an important relationship between the maximum likelihood estimate and the estimate which minimizes the closeness of approximation $I(P, P_a)$.

Theorem 3 *Let $\mathcal{E}$be any event with associated probability $Pr(\mathcal{E})$. If $\hat{Pr}(\mathcal{E})$ is the frequency estimate of the probability $Pr(\mathcal{E})$ obtained using the law of large numbers, then the maximum likelihood estimate obtained as per Theorem 2 also minimizes the closeness of approximation $I(P, P_a)$.*

Proof:

Let us now focus our attention to the aspect of minimizing the closeness measure $I(P, P_a)$.

The closeness measure $I(P, P_a)$ is:

$$\begin{aligned} I(P, P_a) &= \sum_{\mathbf{X}\in\Pi} P(X) \log \frac{P(\mathbf{X})}{P_a(\mathbf{X})} \\ &= \sum_{\mathbf{X}\in\Pi} P(\mathbf{X}) \log P(\mathbf{X}) - \sum_{\mathbf{X}\in\Pi} P(\mathbf{X}) \log P_a(\mathbf{X}). \end{aligned} \tag{13}$$

Observe that the first term of (13) is independent of the approximation $P_a(\mathbf{X})$. Hence we shall concentrate on the second term of (13). Utilizing the expression for the probability of generating a specific permutation under the S-model of permutation generation given in (10) the second term of (13) can be simplified as follows:

$$\begin{aligned} &\sum_{\mathbf{X}\in\Pi} P(\mathbf{X}) \log P_a(\mathbf{X}) \\ &= \sum_{\mathbf{X}\in\Pi} P(\mathbf{X}) \log \left(\prod_{i=1}^{M-1} \frac{s_{x_i}}{1 - \sum_{j=1}^{i-1} s_{x_j}} \right) \\ &= \sum_{\mathbf{X}\in\Pi} P(\mathbf{X}) \cdot \left(\sum_{i=1}^{M-1} \log \frac{s_{x_i}}{1 - \sum_{j=1}^{i-1} s_{x_j}} \right) \\ &= \sum_{i=1}^{M} \sum_{j=1}^{M-1} \log s_i \cdot Pr(i \rightarrow j) \\ &\quad - \sum_{i<j} \log(1 - s_i - s_j) \cdot Pr(i, j \rightarrow < 1,2 > \cup i, j \rightarrow < 2,1 >) \\ &\quad - \sum_{r=2}^{M-2} \log \left(1 - \sum_{j=1}^{r} s_{i_j} \right) \cdot \\ &\quad Pr(\text{some perm. of } i_1, i_2, \ldots, i_r \text{ appears in positions } 1, 2, \ldots, r\ ) \end{aligned}$$

Note that the probabilities of the form $Pr(i \rightarrow j)$ are not known *a priori* and hence they must be estimated from the samples. As per the hypothesis, if we use the frequency estimates obtained by the law of large numbers, the estimates for the corresponding probabilities can be written down as:

$$\begin{aligned} Pr(i \rightarrow j) &= \frac{\mathcal{N}_i^j}{N} \\ Pr(i, j \rightarrow < 1,2 > or < 2,1 >) &= \frac{\mathcal{N}_{ij}}{N} \\ &\vdots \end{aligned}$$

With these estimates for the probabilities it is clear that maximizing the term $\sum_{\mathbf{X}} P(\mathbf{X}) \log P_a(\mathbf{X})$ is the same as maximizing the likelihood function derived in (12) in Theorem 2. Hence the theorem. □

Theorem 2 has established that in order to rigorously obtain the Maximum Likelihood Estimate for the control vector $\mathcal{S}$, we are required to maintain $O(2^M)$ statistics. From a practical point of view, this is simply not a viable approach to parameter estimation. We therefore examine the approximate estimates that could be obtained, if one is restricted to "collecting" a substantially smaller amount of statistics. Our next result develops the method to be used for estimating the parameters, if only $O(M^2)$ statistics are available.

Theorem 4 *Let ${}_uN_v$ represent the number of times the element R_u appears before the element R_v, in the N (sample) permutations that were observed*[2]*. If only the frequency counts of the form ${}_uN_v$ are available for all pairs u, and v, then the maximum likelihood estimate for the control vector $\mathcal{S}$ can be obtained as the solution of the simultaneous non-linear equations given below:*

$$\begin{aligned} \sum_{i \neq j} \left\{ \frac{{}_iN_j}{s_i} - \frac{N}{s_i + s_j} \right\} + \lambda &= 0 \qquad \text{for} \quad 1 \leq i \leq M \\ \sum_{i=1}^{M} s_i &= 1. \end{aligned} \tag{14}$$

Proof:

We emphasize that in this part of the proof, we are not attempting to maximize the likelihood of generating the N sample permutations presented to us. Instead we focus on maximizing the likelihood of explaining the frequencies of the events of the form "element R_i precedes element R_j" which are symbolically represented as "$R_i \prec R_j$".

Consider the event in which the element R_i appeared before the element R_j for a total of ${}_iN_j$ times. Since the N sample permutations are *statistically independent*, we can write down the probability of occurrence of this event as:

$$Pr(R_i \prec R_j \text{ occurs } {}_iN_j \text{ times}) = \binom{N}{{}_iN_j} [Pr(R_i \prec R_j)]^{{}_iN_j} [Pr(R_j \prec R_i)]^{{}_jN_i} \tag{15}$$

Note that as a consequence of Corollary 1, we have the following result:

$$\begin{aligned} Pr(R_i \prec R_j) &= \frac{s_i}{s_i + s_j} \\ Pr(R_j \prec R_i) &= \frac{s_j}{s_i + s_j} \end{aligned}$$

[2] ${}_uN_v$ must not be mistaken for the $\mathcal{N}_{ij}$ which was used in the previous theorems. Unlike the former defined here, the latter count refers to the number of times i, j appeared in one of the positions $< 1,2 >$ or $< 2,1 >$.

and hence,

$$\begin{aligned} Pr(R_i \prec R_j \text{ occurs } {}_iN_j \quad \text{times}) &= \binom{N}{{}_iN_j} \left(\frac{s_i}{s_i+s_j}\right)^{{}_iN_j} \left(\frac{s_j}{s_i+s_j}\right)^{{}_jN_i} \\ &= \binom{N}{{}_iN_j} \frac{s_i^{{}_iN_j}\ s_j^{{}_jN_i}}{(s_i+s_j)^N} \end{aligned} \tag{16}$$

The statistics extracted from the data samples consists of counts ${}_iN_j$, for all $i \neq j$, i.e. M(M-1)/2 counters and these are sufficient to evaluate terms of the form (16).

In order to arrive at the best estimates for the control vector $\mathcal{S}$, we have to maximize the quantity $Pr[\cap_{i<j}(R_i \prec R_j \text{ occurs } {}_iN_j \text{ times})]$. Assuming that these events are *statistically independent* [3], we shall consider an approximation to the likelihood function $\mathcal{L}$ (given only the statistics ${}_iN_j$) as shown below:

$$\begin{aligned} \mathcal{L} &\triangleq Pr[\cap_{i<j}(R_i \prec R_j \text{ occurs } {}_iN_j \text{times}] \\ &= \prod_{i<j} Pr[(R_i \prec R_j \text{ occurs } {}_iN_j \text{times}] \\ &= \prod_{i<j} \binom{N}{{}_iN_j} \frac{s_i^{{}_iN_j}\ s_j^{{}_jN_i}}{(s_i+s_j)^N} \qquad \text{from (16)} \end{aligned}$$

Hence the logarithm of the likelihood function, $\ell \triangleq \log \mathcal{L}$, can be written as:

$$\ell = Z + \sum_{i<j} \{{}_iN_j \log s_i + {}_jN_i \log s_j - N \log(s_i+s_j)\} \tag{17}$$

where Z stands for the quantity $\sum_{i<j} \log \binom{N}{{}_iN_j}$.

It is now clear that our aim is to maximize the quantity ℓ subject to the condition that $\sum_{i=1}^{M} s_i = 1$. This is achieved by using *Lagrange Multipliers* [7] in which if λ is the Lagrange multiplier, the auxiliary function is given by (18) below:

$$\sigma = \ell + \lambda(\sum_{i=1}^{M} s_i - 1). \tag{18}$$

In order to obtain the maxima of this auxiliary function σ, we merely differentiate it w.r.t. each of the variables $s_1, s_2, \ldots, s_M$ and equate the result to 0. Differentiating σ w.r.t. the variable s_i and simplifying yields:

$$\frac{\partial \sigma}{\partial s_i} = \sum_{i \neq j} \left(\frac{{}_iN_j}{s_i} - \frac{N}{s_i+s_j}\right) + \lambda = 0.$$

[3] This assumption is strictly not true. See [8] for a counter example.

Therefore, the solution vector $\mathcal{S}$ is obtained by solving the set of equations

$$\sum_{i \neq j} \left(\frac{{}_iN_j}{s_i} - \frac{N}{s_i + s_j} \right) + \lambda = 0, \quad \text{for} \quad 1 \leq i \leq M$$

$$\sum_{i=1}^{M} s_i = 1.$$

Hence the result. □

Remark:

It must be noted that the above set of equations are a non-linear **set** of simultaneous equations involving the s_i's and the Lagrange multiplier λ. Clearly, an analytic solution is not feasible. We chose to use the generalized form of the Newton-Raphson (NR) method for solving these equations. The essential technique is as follows. Let us assume that the set of simultaneous equations is $f_i(\mathbf{X}) = 0$, for $1 \leq i \leq n$, where $\mathbf{X}$ is an n-dimensional vector. If $\mathbf{X_0}$ is an approximate solution, then the update rule for NR iteration is simply:

$$\mathbf{X}_{k+1} = \mathbf{X}_k - \mathbf{J}^{-1}[f_1(\mathbf{X}_k) \cdots f_n(\mathbf{X}_k)]^T$$

where, $\mathbf{J}$ is the well known Jacobian

$$\mathbf{J} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}$$

A first-order estimate for the control vector can be found by examining (14). In our simulations, the initial estimate for NR-iteration was the solution obtained by the solving the (linear) simultaneous equations in (19).

$$\begin{aligned} \frac{{}_1N_2}{s_1} - \frac{N}{s_1 + s_2} &= 0 \\ \frac{{}_1N_3}{s_1} - \frac{N}{s_1 + s_3} &= 0 \\ &\vdots \\ \frac{{}_1N_M}{s_1} - \frac{N}{s_1 + s_M} &= 0 \\ \sum_{i=1}^{M} s_i &= 1 \end{aligned} \tag{19}$$

The reader will recognize that the first $M-1$ of these equations are obtained by examining the equation of the derivative w.r.t. s_1 and individually equating the bracketed terms to 0. An alternative way to look at these equations is to view that we are merely basing our estimates of the control vector $\mathcal{S}$ on the number of times the element R_1 appears before each of the other elements. It should be also clear that we can use another set of linear equations, in which the coefficients are dependent on the counts of some other element R_j preceding the other elements. We can effectively get M initial approximations. These initial estimates will obviously be different.

4.1 Experimental Results

To demonstrate the strength of the procedures used to estimate the control vector, we have performed two types of experiments. These two experiments differ only in the strategy used to generate the sample permutations which are examined by the data analysis procedures.

In the first set of experiments, the model used to generate the permutations is the S-model defined in Section 3. Firstly, the number of elements M in $\mathcal{R}$ is chosen and then a control vector $\mathcal{S}^*$ is chosen randomly. The experiment is conducted as follows. Sample permutations are generated, based on the control vector $\mathcal{S}^*$. With each sample, the statistic of the form ${}_iN_j$ (defined in Theorem 4) are updated. The estimation of the control vector is not attempted after each sample, but rather after a fixed number of samples (this number was set to 200 in the experiments). The NR method was used to estimate the control vector (denoted as $\hat{\mathcal{S}}$). If this method for solving the system of equations did not converge, we resorted to an explicit numerical optimization procedure to optimize the likelihood function given by (17). The method chosen by us, to perform the necessary constrained optimization is the David-Fletcher-Powell algorithm, used in conjunction with the interior penalty function [7]. Clearly, in this case, we are interested in observing the convergence of $\hat{\mathcal{S}}$ to $\mathcal{S}^*$ with time. Figure 2 depicts this convergence for the case when M, the number of elements in each permutation is 6. We use the euclidean distance between the vector $\mathcal{S}^*$ and the estimate $\hat{\mathcal{S}}$ as the measure of closeness between these vectors. Figure 2 shows the variation of the quantity $\mid \mathcal{S}^* - \hat{\mathcal{S}} \mid$ as a function of the number of samples. The results shown in Figure 2 are the ensemble average of 25 experiments (consisting of 10,000 samples each) performed with the same control vector $\mathcal{S}^*$. Observe that in for the case shown in Figure 2, the quantity $\mid \mathcal{S}^* - \hat{\mathcal{S}} \mid$ reduced to about 15% of its initial value after about 10,000 samples. In all cases we have studied, we have observed that $\hat{\mathcal{S}}$ converges to the underlying control vector $\mathcal{S}^*$. It must be noted that the rate of convergence of the quantity $\mid \mathcal{S}^* - \hat{\mathcal{S}} \mid$ is dependent on the

control vector $\mathcal{S}^*$ itself. If some of the components of the control vector are small, some of the counts ${}_iN_j$ are zero until a sufficient number of samples have been generated. This clearly influences the convergence of the estimate $\hat{\mathcal{S}}$ to the underlying vector $\mathcal{S}^*$.

In the second set of experiments, our aim is to judge the effectiveness of the S-model of permutation generation in approximating general distributions. As was done in the case of the earlier set of experiments, we first choose M, the number of elements in $\mathcal{R}$. To avoid specifying a large number of probability masses, the number M was chosen to be a small number (4 in our case). Having fixed the value of the parameter M, we generated a random distribution of the permutations. In other words, we randomly assign probabilities to each of the $M!$ permutations (subject to the condition that their sum is unity). As opposed to the case in which we were using the S-model of permutation generation, in this case we are not so much interested in the convergence of the estimate $\hat{\mathcal{S}}$ to its final value. In this case the experiment consisted of generating 10,000 sample permutations, based on the chosen distribution and updating the counts ${}_iN_j$ with the arrival of each sample. When the incoming samples were exhausted, the estimation of the control vector was performed. Figure 3 (composed of Figures 3a and 3b) shows the results we have obtained in two specific runs. Both the figures show the underlying distribution (bold shading) and the approximated distribution (mosaic shading). Figure 3b shows the more favourable case in which there is good similarity between the underlying distribution and the estimated distribution. We believe that whenever adjacent (in terms of symbolic inversions) permutations have comparable probabilities in the underlying distribution, the S-model yields a good approximation to the underlying distribution.

4.2 Numerical Instability

The solution of (14) to yield the maximum likelihood estimate for the vector $\mathcal{S}$ obviously entails the solution of M simultaneous non-linear equations. Since these equations have to be solved numerically, this solution automatically inherits the numerical instability of the root solving technique utilized. Thus it is no wonder that whenever the underlying problem is ill-conditioned, the root solving technique does not converge.

It must be noted that unlike problems which involve root solving, the coefficients of the non-linear equations are themselves random variables. If in the course of a particular experiment, a certain event did not occur, the corresponding coefficient would be exactly zero. Thus it is not too astonishing to observe that when the number of samples is small, the matrices involved are singular and the underlying problems are extremely ill-conditioned.

In many of the cases, the NR method leads to a divergent solution. In these cases, rather than solve (14), we have opted to solve the underlying optimization problem using the David-Fletcher-Powell algorithm (in conjunction with a penalty function) [7]. We would like to emphasize that our primary contribution is not the root solving or the optimizing algorithm, but the existence and properties of the estimates themselves. Clearly, the strategy of evaluating them is merely one of using the well established numerical tools.

5 Use of S-model in Source Recognition

We now return to the application of the S-model to the original problem we have on hand, i.e. one of distinguishing between several sources of random permutations. As has been explained earlier, our problem is one of supervised pattern classification. In the training phase of our recognition system, we are presented with a stream Q_{train}, together with information on the source which was used to generate this stream. The "feature extraction" process is now one of estimating the S-parameters for this source, using the estimation procedures explained in Section 4. If the number of sources that we need to distinguish is c, the above process can be repeated once for each source. Hence after the "training phase" the recognizer has c estimates $\hat{\mathcal{S}}_1, \hat{\mathcal{S}}_2, \hat{\mathcal{S}}_3, \ldots, \hat{\mathcal{S}}_c$ for the S-vectors of the c classes.

In the usual pattern recognition system, one is also required to characterize the distribution of features within a given class. In our case this means that we ought to determine the distribution of the control vectors $\hat{\mathcal{S}}_i|\omega_i$, for $1 \leq i \leq c$. Since no information about these conditional distributions is available, this step cannot be attempted. The effect of not computing these distributions is that each source ω_i is represented in the feature space by a single point, namely the S-parameter $\hat{\mathcal{S}}_i$.

For the tesing phase, the traditional approach would require that we estimate the control vector $\hat{\mathcal{S}}_{test}$ from the test samples and then classify the given test sample based on some discriminant functions. There are two difficulties when we wish to use the same approach in our source recognition problem:

1. If the number of samples in Q_{test} is small, the estimate for $\hat{\mathcal{S}}_{test}$ may be not obtainable or grossly inaccurate. We encounter the first condition because of singularities in the matrices involved in the estimation process.

2. Any discriminant based method assumes that we can compute the probability $Pr(\hat{\mathcal{S}}_{test} \mid \omega_i)$ for i = 1,2,...,c. These probabilities can be computed using the information about class-conditional distributions. As was mentioned above, the estima-

tion of the class-conditional distributions was completely eliminated by our training procedure.

In our case we have opted not to estimate the control vector $\hat{\mathcal{S}}_{test}$ from the test samples but instead to follow the *ab initio* approach: that is, decide the test sample as belonging to the class ω_i if $Pr(\omega_i \mid Q_{test}) > Pr(\omega_j \mid Q_{test})$ for all $i \neq j$, where Q_{test} stands for the sequence of permutations presented to source recognition system. This decision criterion can be easily rewritten as:

$$\text{decide} \quad \omega_i \quad \text{if} \quad Pr(\omega_i)Pr(Q_{test} \mid \omega_i) > Pr(\omega_j)Pr(Q_{test} \mid \omega_j) \quad \text{for} \quad \text{all} \quad \text{i} \neq \text{j}$$

If we assume that all the classes are equally likely, the above inequality further simplifies to:

$$\text{decide} \quad \omega_i \quad \text{if} \quad Pr(Q_{test} \mid \omega_i) > Pr(Q_{test} \mid \omega_j). \quad \text{for} \quad \text{all} \quad \text{i} \neq \text{j} \tag{20}$$

In effect, this means that the Bayesian classifier can be obtained by essentially utilizing the maximum likelihood rule (or equivalently its logarithmic form). We note that since the individual permutations in the test samples are statistically independent, we can easily compute the likelihood function $Pr(Q_{test} \mid \omega_i)$ as:

$$Pr(Q_{test} \mid \omega_i) = \prod_{q \in Q_{test}} Pr(q \mid \hat{\mathcal{S}}_i) \tag{21}$$

We now present the experimental results performed in connection with the source recognition problem.

5.1 Experimental results

In the experiments which we have performed, the number of elements in a permutation was set to 4, and thus at any instant the recognizer obtains one of twenty-four different permutations. To render the problem non-trivial, we consider a 4-class problem in which we have four independent sources of permutations, all of which are based on the S-model of permutation generation. The control vectors used by the sources (labelled as Classes ω_1 through ω_4) were:

$$\begin{aligned}
\mathcal{S}_1 &= [0.4, 0.3, 0.2, 0.1]^T \\
\mathcal{S}_2 &= [0.1, 0.4, 0.3, 0.2]^T \\
\mathcal{S}_3 &= [0.2, 0.1, 0.4, 0.3]^T \\
\mathcal{S}_4 &= [0.3, 0.2, 0.1, 0.4]^T
\end{aligned}$$

The distributions of these permutations are shown in Figure 4.

During the training phase the source recognition system was presented with 1,000 permutations from each class. This information was used to estimate the control vector for each of the four classes. The recognizer was subsequently tested on 200 test cases where each test case consisted of Z sample permutations. The Bayesian classification rule involves computing the likelihood functions given in (21) based on the (estimated) control vectors for each of the classes. The class which maximized the likelihood function was chosen by the classifier as the source of the permutations.

The experimental results which we have obtained are truly remarkable. After the training phase in which 1,000 samples from each class were used to get the estimates $\{\hat{\mathcal{S}}_i\}$, the number of samples, Z, used for each testing operation was varied. When Z was as large as 500 the estimates $\hat{\mathcal{S}}_{test}$ that we would have obtained would be reasonably close to the estimate $\hat{\mathcal{S}}_i$ of the underlying class. Thus obtaining a high classification accuracy using (20) is not too astonishing. Indeed the accuracy that we obtained was a 100% even though we did not estimate $\hat{\mathcal{S}}_{test}$ but merely invoked the maximum likelihood classifier described by (20). However when the number of samples in a testing operation is decreased, it is clear that the estimate $\hat{\mathcal{S}}_{test}$ is increasingly inaccurate and thus the accuracy of such a classifier would diminish. This phenomenon is indeed what we observed but the accuracy is remarkably high even for very small number of test samples. When only 12 samples per test case were used, the accuracy was 95%. Observe that this meant that in this scenario at least half of the 24 permutations were not even seen **once** by the classifier. This accuracy increased to 98% when 18 test samples were used per testing operation. The accuracy was 100% when the number of test samples was greater than 24. The power of the strategy is obvious.

We have also performed analogous experiments when the number of sources was 4, but the number of elements in each permutation was 6 and 8. For the case when each permutation consisted of 6 elements, the control vectors utilised for the sources (labelled as Classes ω_1 through ω_4) were:

$$\begin{aligned}
\mathcal{S}_1 &= [0.2424, 0.2121, 0.1818, 0.1515, 0.1212, 0.0909]^T \\
\mathcal{S}_2 &= [0.0322, 0.2580, 0.2258, 0.1935, 0.1612, 0.1290]^T \\
\mathcal{S}_3 &= [0.0689, 0.0348, 0.2758, 0.2413, 0.2068, 0.1724]^T \\
\mathcal{S}_4 &= [0.1111, 0.0740, 0.0370, 0.2962, 0.2592, 0.2222]^T
\end{aligned}$$

Once again, the recogniton accuracy was excellent. We noticed that in the case when 10 samples per case were utilised (i.e. only 10 out of 720 permutations were seen by the classifier), the recognition accuracy was an astonishing 99%. The accuracy was 100% when more

than 10 samples per test case were presented to the recognizer. The recognition accuracy for the case of 8 elements per permutation was even better. Even when 10 samples per test case were used, we obtained 100% accuracy.

6 Conclusion

In this paper we have studied the problem of distinguishing between sources that randomly generate strings, composed of the symbols of a finite alphabet. In the case when the model of random string generation is one "with replacement", the strings generated are arbitrary strings. For this scenario we have presented a simple model called the U-model. An application of this recognition strategy in distinguishing between several keys used in encryption is presented and the experimental results demonstrate the power of the scheme.

If the generation of random strings is done "without replacement", we have the case when sources emit random permutations. We have studied the problem of distinguishing between sources that are randomly generating permutations. A brute force approach to pattern recognition would require the maintenance of $M!$ estimates, where M is the number of elements in each permutation. This leads us to the problem of approximating a general distribution of permutations by using parametrized distributions. The parametrized model referred to as the S-model, has been first described, and is based on a control vector of dimension M. Since the components of this vector constitute a probability vector, we only need to estimate $M-1$ parameters.

We have demonstrated that the maximum likelihood estimates for the control vector $\mathcal{S}$ can be obtained if we are prepared to maintain an exponential number of statistics. This approach is almost as impractical as the original problem of maintaining estimates for individual probabilities of occurrence. By maintaining $O(M^2)$ statistics, a modified maximum likelihood estimate can be obtained which is shown to be adequate.

Experimental results indicate that the method for parameter estimation converges to the control vector $\mathcal{S}^*$, if the generation technique is the S-model defined in Section 3. In other cases, the model obtained is both reasonable and elegant.

The effectiveness of the S-model in the proposed source recognition strategy has been experimentally validated. The experimental results involving multiple sources are very significant. The recognition accuracy, even if very small number of permutations are presented to the classifier, is remarkable.

References

[1] D.E. Denning, *"Encryption and Data Security"*, Addison-Wesley, 1983.

[2] R.O. Duda and P.E. Hart, *"Pattern Classification and Scene Analysis"*, Wiley Inter-Science, 1973.

[3] P.A.V. Hall and G.R. Dowling, "Approximate String Matching", *ACM Computing Surveys*, Vol. 12, 1980, pp. 381-402.

[4] R.L. Kashyap and B.J. Oommen, "Spelling Correction Using Probabilistic Methods", *Pattern Recognition Letters*, Vol.2, 1984, pp. 147-154.

[5] R. Lawrence and R.A. Wagner, "An extension of the string to string correction problem", *Journal of the ACM*, Vol.22, 1975, pp. 177-183.

[6] B.J.Oommen and D.T.H. Ng, "Arbitrarily Distributed Random Permutation Generation", *Proceedings of 1989 ACM Comp. Sci. Conf.*, Louisville, KY, Feb 1989, pp. 27-32. Also available as Technical Report SCS-TR-138 from the School Of Computer Science, Carleton University, Ottawa, Canada, K1S 5B6.

[7] S.S. Rao, *"Optimization methods: theory and applications"*, John Wiley, 1980.

[8] R.S. Valiveti, Ph.D. thesis, in preparation.

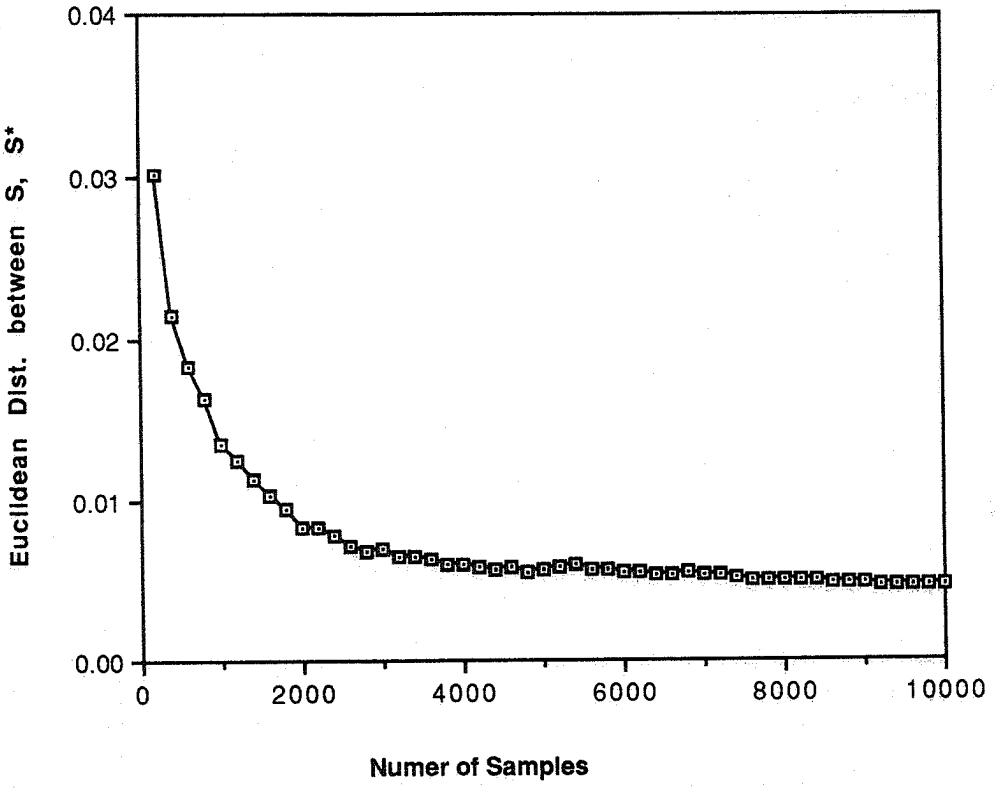

Figure 2: The graphs displays the results obtained by the estimation procedure for the case when the model used to generate the permutations was the S-model. The figure shows the convergence of the estimate of the parameter vector S to the "real" (i.e. underlying vector) S*. The values depicted above are the ensemble average of 25 experiments, each consisting of 10,000 samples.

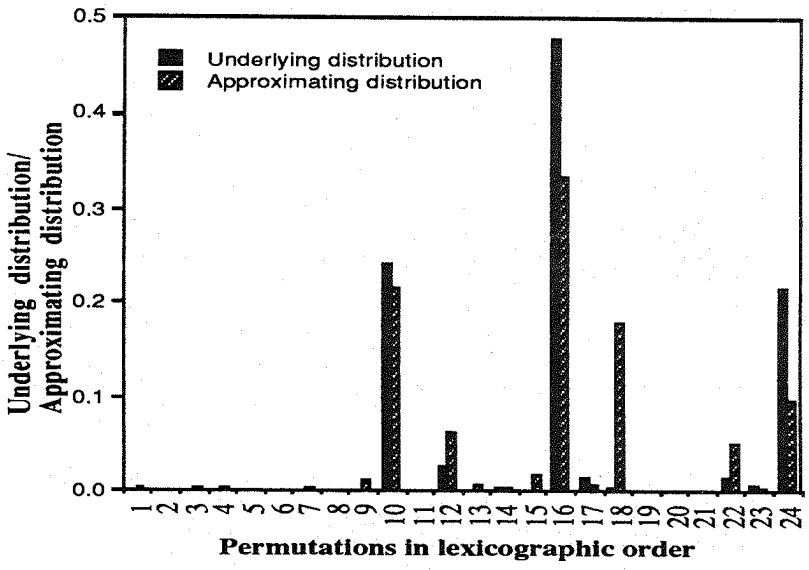

Figure 3a

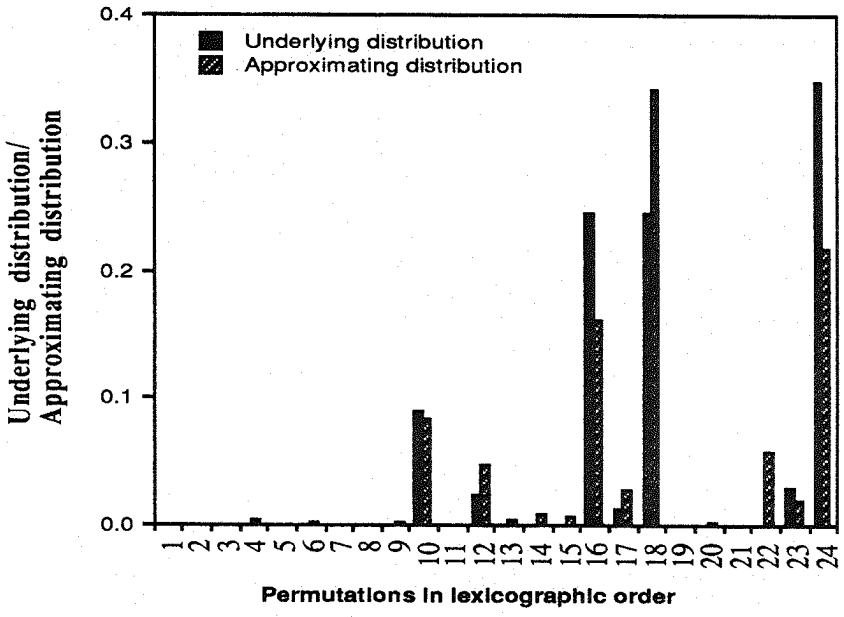

Figure 3b

Figure 3: This figure depicts the results obtained for the case when a general permutation distribution is being approximated with the S-model. Figure 3a and Figure 3b depict two separate experiments to approximate unknown distributions of permutations of 4 elements. Figure 3b is the more favourable case, when the approximation is reasonably close.The graphs show the underlying distribution (shown in bold shading), and the approximating distribution (shown in mosaic shading) obtained after examining 10,000 samples Note that the distributions are displayed in the lexicographic ordering of the permutations.

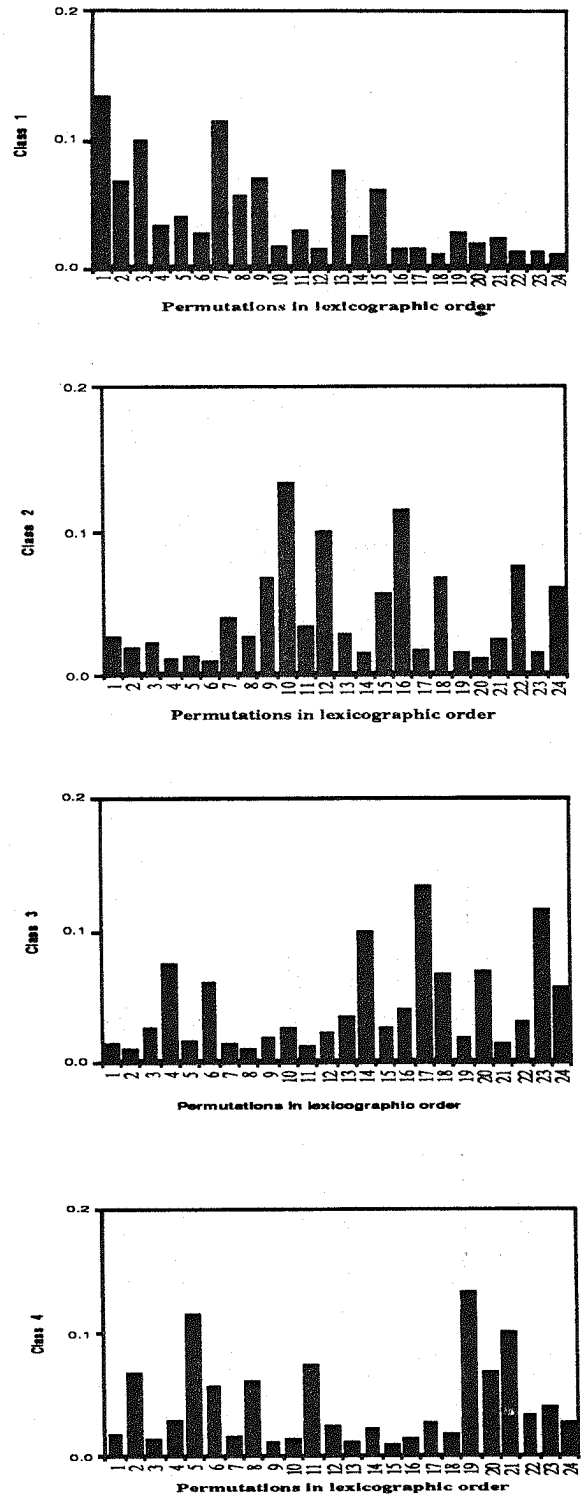

Figure 4: This figure displays the distributions of the permutations generated by 4 sources used in the source recognition experiments. The pattern recognition system is first trained with 1000 samples drawn from each class. Subsequently the recognizer is tested to verify its ability to identify the source of a sequence of permutations. The accuracy obtained is 100% even if only 24 sample permutations are used for each testing operation.

School of Computer Science, Carleton University
Bibliography of Technical Reports

SCS-TR-127 **On the Packet Complexity of Distributed Selection**
A. Negro, N. Santoro and J. Urrutia, November 1987.

SCS-TR-128 **Efficient Support for Object Mutation and Transparent Forwarding**
D.A. Thomas, W.R. LaLonde and J. Duimovich, November 1987.

SCS-TR-129 **Eva: An Event Driven Framework for Building User Interfaces in Smalltalk**
Jeff McAffer and Dave Thomas, November 1987.

SCS-TR-130 **Application Frameworks: Experience with MacApp**
Out of print
John R. Pugh and Cefee Leung, December 1987.
Available in an abridged version in the Proceedings of the Nineteenth ACM SIGSCE Technical Symposium, February 1988, Atlanta, Georgia.

SCS-TR-131 **An Efficient Window Based System Based on Constraints**
Danny Epstein and Wilf R. LaLonde, March 1988.

SCS-TR-132 **Building a Backtracking Facility in Smalltalk Without Kernel Support**
See Third International Conference on OOPSLA, San Diego, Calif., Sept. '88.
Wilf R. LaLonde and Mark Van Gulik, March 1988.

SCS-TR-133 **NARM: The Design of a Neural Robot Arm Controller**
Daryl H. Graf and Wilf R. LaLonde , April 1988.

SCS-TR-134 **Separating a Polyhedron by One Translation from a Set of Obstacles**
Otto Nurmi and Jörg-R. Sack, December 1987.

SCS-TR-135 **An Optimal VLSI Dictionary Machine for Hypercube Architectures**
Frank Dehne and Nicola Santoro, April 1988.

SCS-TR-136 **Optimal Visibility Algorithms for Binary Images on the Hypercube**
Frank Dehne, Quoc T. Pham and Ivan Stojmenovic, April 1988.

SCS-TR-137 **An Efficient Computational Geometry Method for Detecting Dotted Lines in Noisy Images**
F. Dehne and L. Ficocelli, May 1988.

SCS-TR-138 **On Generating Random Permutations with Arbitrary Distributions**
B. J. Oommen and D.T.H. Ng, June 1988.

SCS-TR-139 **The Theory and Application of Uni-Dimensional Random Races With Probabilistic Handicaps**
D.T.H. Ng, B.J. Oommen and E.R. Hansen, June 1988.

SCS-TR-140 **Computing the Configuration Space of a Robot on a Mesh-of-Processors**
F. Dehne, A.-L. Hassenklover and J.-R. Sack, June 1988.

SCS-TR-141 **Graphically Defining Simulation Models of Concurrent Systems**
H. Glenn Brauen and John Neilson, September 1988

SCS-TR-142 **An Algorithm for Distributed Mutual Exclusion on Arbitrary Networks**
H. Glenn Brauen and John E. Neilson, September 1988

SCS-TR-143 to 146 are unavailable.

SCS-TR-147 **On Transparently Modifying Users' Query Distributions**
B.J. Oommen and D.T.H. Ng, November 1988

SCS-TR-148 **An O(N Log N) Algorithm for Computing a Link Center in a Simple Polygon**
H.N. Djidjev, A. Lingas and J.-R. Sack, July 1988
Available in STACS 89, 6th Annual Symposium on Theoretical Aspects of Computer Science, Paderborn, FRG, February 16-18, 1989, Lecture Notes in Computer Science, Springer-Verlag No. 349

SCS-TR-149 **Smallscript: A User Programmable Framework Based on Smalltalk and Postscript**
Kevin Haaland and Dave Thomas, November 1988

SCS-TR-150 **A General Design Methodology for Dictionary Machines**
Frank Dehne and Nicola Santoro, February 1989

SCS-TR-151 **On Doubly Linked List ReOrganizing Heuristics**
D.T.H. Ng and B. John Oommen, February 1989

SCS-TR-152 **Implementing Data Structures on a Hypercube Multiprocessor, and Applications in Parallel Computational Geometry**
Frank Dehne and Andrew Rau-Chaplin, March 1989

SCS-TR-153 **The Use of Chi-Squared Statistics in Determining Dependence Trees**
R.S. Valiveti and B.J. Oommen, March 1989

SCS-TR-154 **Ideal List Organization for Stationary Environments**
B. John Oommen and David T.H. Ng, March 1989

SCS-TR-155 **Hot-Spot Contention in Binary Hypercube Networks**
Sivarama P. Dandamudi and Derek L. Eager, April 89

SCS-TR-156 **Some Issues in Hierarchical Interconnection Network Design**
Sivarama P. Dandamudi and Derek L. Eager, April 1989

SCS-TR-157 **Discretized Pursuit Linear Reward-Inaction Automata**
B.J. Oommen and Joseph K. Lanctot, April 1989

SCS-TR-158 **Parallel Fractional Cascading on a Hypercube Multiprocessor**
Frank Dehne, Afonso Ferreira and Andrew Rau-Chaplin, May 1989

SCS-TR-159 **Epsilon-Optimal Stubborn Learning Mechanisms**
J.P.R. Christensen and B.J. Oommen, June 1989

SCS-TR-160 **Disassembling Two-Dimensional Composite Parts Via Translations**
Doron Nussbaum and Jörg-R. Sack, June 1989

SCR-TR-161 (revised) **Recognizing Sources of Random Strings**
R.S. Valiveti and B.J. Oommen, January 1990
Revised version of SCS-TR-161 "On the Data Analysis of Random Permutations and its Application to Source Recognition", published June 1989

SCS-TR-162 **An Adaptive Learning Solution to the Keyboard Optimization Problem**
B.J. Oommen, R.S. Valiveti and J. Zgierski, October 1989

SCS-TR-163 **Finding a Central Link Segment of a Simple Polygon in O(N Log N) Time**
L.G. Alexandrov, H.N. Djidjev, J.-R. Sack, October 1989

SCS-TR-164 **A Survey of Algorithms for Handling Permutation Groups**
M.D. Atkinson, January 1990

SCS-TR-165 **Key Exchange Using Chebychev Polynomials**
M.D. Atkinson and Vincenzo Acciaro, January 1990

SCS-TR-166 **Efficient Concurrency Control Protocols for B-tree Indexes**
Ekow J. Otoo, January 1990

SCS-TR-167 **A Hierarchical Stochastic Automaton Solution to the Object Partitioning Problem**
B.J. Oommen, January 1990

SCS-TR-168 **Adaptive List Organizing for Non-stationary Query Distributions. Part I: The Move-to-Front Rule**
R.S. Valiveti and B.J. Oommen, January 1990