STRING TAXONOMY USING LEARNING AUTOMATA[+]

B. John Oommen and Edward V. de St. Croix
School of Computer Science
Carleton University
Ottawa ; Canada : K1S 5B6.

ABSTRACT

A typical syntactic pattern recognition (PR) problem involves comparing a noisy string with every element of a dictionary, H. The problem of classification can be greatly simplified if the dictionary is partitioned into a set of sub-dictionaries. In this case, the classification can be hierarchical -- the noisy string is first compared to a representative element of each sub-dictionary and the closest match within the sub-dictionary is subsequently located. Indeed, the entire problem of sub-dividing a set of strings into subsets where each subset contains "similar" strings has been referred to as the "String Taxonomy Problem". To our knowledge there is no reported solution to this problem (see footnote on Page 2). In this paper we shall present a learning-automaton based solution to string taxonomy. The solution utilizes the Object Migrating Automaton (OMA) whose power in clustering objects and images [33,35] has been reported. The power of the scheme for string taxonomy has been demonstrated using random strings and garbled versions of string representations of fragments of macromolecules.

Keywords : String Taxonomy, String Clustering, Dictionary Partitioning, Syntactic Pattern Recognition.

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

I. INTRODUCTION

Syntactic and structural pattern recognition (PR) are distinct from statistical PR because, unlike in the latter, in the former two areas, the processing of the patterns is achieved by representing them symbolically using primitive or elementary symbols. The PR system symbolically models noisy variations of typical samples of the patterns, and these models are utilized in both the training and testing phases of the system.

There are essentially two strategies utilized in statistical pattern recognition. In a non-parametric scheme, the classifier is presented with a set of training samples from each class. Typically, when a testing sample is encountered, the classifier compares the latter with every training sample, and a decision is made based on the training samples which are its closest neighbours. Clearly, this is a computationally expensive strategy. The alternative strategy involves modelling the class conditional densities parametrically. The parameters of the individual densities are then estimated in the learning (training) phase. The testing phase involves utilizing the features of the test sample in a computation which usually uses the estimated parameters of the individual class densities. Thus, though there may be thousands of training samples, the testing phase does not compare the test sample with every one individually. Instead, it "generalizes" the properties of the overall class by examining the features of the individual samples. This "generalization" is achieved by the system learning the class densities, and the "generalized" information is stored in terms of the functional form of the density and the estimated values of the parameters themselves.

The problem in syntactic PR is quite similar except that the solutions are far more complex because there is no known metric which can effectively cluster strings. A typical syntactic PR problem involves comparing a noisy string with every element of a dictionary, H. Analogous to the scenario in statistical PR, the problem of classification can be greatly simplified if the dictionary is partitioned into a set of sub-dictionaries -- analogous to obtaining the various class conditional densities. In this case, the classification can be hierarchical -- the noisy string can be first compared to a representative element of each sub-dictionary and the closest match within the various sub-dictionary can be subsequently located. Indeed, the entire problem of sub-dividing a set of strings into subsets where each subset contains "similar" strings is called the "String Taxonomy Problem".

To our knowledge there is no reported solution to this problem. Indeed, in his plenary talk at CPM 1992, The Third International Symposium on Combinatorial Pattern Matching, in Tucson, Professor Ehrenfeucht from the University of Colorado, a pioneer in this field, spoke elaborately about the problem [8]. He spoke about the complexity issues that shroud this problem and challenged the audience to tackle it[1]. Apart from the other issues that the authors of this paper learned from the talk, it was also clear that a good taxonomical scheme would have to not only

[1]The first author is very grateful to Professor Ehrenfeucht for introducing him to this problem. We regret that there is no published record of his plenary presentation at CPM-1992.

utilize the dissimilarities between the strings as evaluated by an appropriate metric, but additionally incorporate an effective **learning** mechanism which would infer the dissimilarity between a string and a set of strings from the corresponding dissimilarities between the individual strings themselves. The solution presented in this paper attempts to meet that goal.

One of the fields where string taxonomy will be very powerful is in molecular biology. Currently, there is a great deal of research investigating the mutations of molecules such as those seen in RNA sequences. In their simplest forms, these molecules can be viewed as long strings of letters which represent their component bases [4,27,39]. It is well known that these sequences mutate, over time, into different sequences. In order to study these various sequences it is useful to be able to associate them collectively. Hopefully, a good algorithm will process a set of sequences and partition them efficiently so that those which mutated from the same source, group together. This could, in turn, assist a researcher to identify mutated sequences without an *a priori* knowledge of the source molecule. Furthermore, it could also help the researcher to quantify how well a sequence fits into the grouping to which it is assigned.

1.1 Implications of Dictionary Modelling

On formulating the problem we observe that its complexity is closely related to the model for the dictionary. First of all, observe that the first step in this modelling scenario involves specifying the alphabet, which, in most cases, 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 "space" delimiter. In speech applications, the individual symbols are the set of phonemes [2,39,44] and in the recognition of noisy macro-molecules, the individual symbols are the underlying amino-acids [4,27,39].

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 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 [20,21,25,26]. 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 [14-18,31,32,37,39,41]. The advantages of using a

finite dictionary in text recognition applications are many. First of all, the accuracy of the recognition is very high. Secondly, a noisy string is never recognized as a word which is not in the language, and thus, the question of "meaningless" decisions is irrelevant. Finally, the time complexity of the computation involved in the text recognition process is typically quadratic per word and is linear in the size of the dictionary. The complexity per word can often be decreased if the dictionary is modelled using a trie [17], and if the alphabet size is decreased [1, 24, 41,46].

When the dictionary is prohibitively large, problem analysts tackle the problem by modelling the dictionary 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 [5,13,29,38,42]. 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 [2,5,13,20,21,25,26,29, 39-42,44] where the probability of a particular symbol occurring depends on the previous symbol. Essentially, this model is identical to the one which uses 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). Models which utilize the positional bigrams (and their variants) of the language have also been reported (See references in [37,41]).

In this paper, which we believe, is a pioneering work in the field, we shall assume that we are dealing with a finite dictionary, $H = \{X_1,...X_J\}$. We intend to partition H into K equi-sized sub-dictionaries. The problem of partitioning H into unequally sized sub-dictionaries is still open. Also, although the case when H is modelled using a Bernoulli or Markovian model remains open, we believe that this scenario is relatively simpler to tackle than the finite dictionary case primarily because, in these cases, the characteristics of the sub-dictionaries can be learned using traditional *statistical* PR training methodologies. We believe that in these cases the heart of the problem will involve systematic estimation procedures, and we are currently working on characterizing and formulating how these procedures can themselves be formalized.

II. LEARNING AUTOMATA AND OBJECT PARTITIONING

Our solution to the string taxonomy problem involves learning automata. Learning automata have been used to model biological learning systems and also to learn the optimal action which a random environment offers. Learning is achieved by interacting with the environment and

processing its responses to the actions that are chosen. Such automata have various applications such as parameter optimization, statistical decision making and telephone routing [28,33,35,36,43]. An excellent book on the field by Narendra and Thathachar [28] contains a review of the families and applications of learning automata.

The learning process of an automaton can be described as follows: The automaton is offered a set of actions by the environment with which it interacts, and it is constrained to choose one of these actions. When an action is chosen, the automaton is either rewarded or penalized by the environment with a certain probability. A learning automaton is one which learns the optimal action, which is the action that has the minimum penalty probability. Hopefully, the automaton will eventually choose this action more frequently than other actions.

Stochastic learning automata can be classified into two main families : (a) Fixed structure stochastic automata and (b) automata whose structures evolve with time. Examples of the former type are the Tsetlin, Krinsky and Krylov automata [28,36,43]. Although the latter automata are called variable structure stochastic automata because their transition and output matrices are time varying, in practice, they are merely defined in terms of action probability updating rules [28].

A FSSA is a quintuple (α, Φ, β, F, G) where :

(i) $\alpha = \{\alpha_1, ..., \alpha_R\}$ is the set of actions that it must choose from.

(ii) $\Phi = \{\phi_1, ..., \phi_S\}$ is its set of states.

(iii) $\beta = \{0, 1\}$ is its set of inputs where '1' represents a penalty and '0' a reward.

(iv) F is a map from Φ x β to Φ. It defines the transition of the state of the automaton on receiving an input. F may be stochastic.

(v) G is a map from Φ to α, and determines the action taken by the automaton if it is in state ϕ_i. With no loss of generality G is deterministic [28,36,43].

The selected action serves as the input to the environment which outputs a stochastic response $\beta(n)$ at time 'n'. $\beta(n)$ is an element of $\beta = \{0,1\}$ and is the feedback response of the environment to the automaton. The environment penalizes (i.e., $\beta(n) = 1$) the automaton with the penalty c_i, which is action dependent. On the basis of the response $\beta(n)$, the state of the automaton $\phi(n)$ is updated and a new action chosen at (n+1). Note that the $\{c_i\}$ are unknown initially and it is desired that as a result of interaction with the environment the automaton arrives at the action which presents it with the minimum penalty response in an expected sense.

In this paper we propose that the string taxonomy problem be solved by viewing the problem not as a estimation or parameter-based training problem, but instead as one that falls in the domain of object partitioning problems. The goal is not just to find strings in H that match other strings, but to group all similar strings together so that subsequent searches will proceed much faster. Thus, instead of using some classification method which stipulates the membership of the strings into groups, the system adaptively decides the grouping by extracting information about relative

resemblances between the various elements when they are considered in pair-wise comparisons. The algorithm uses previous sub-dictionary patterns to intelligently partition the entire dictionary to obtain a superior partitioning. Furthermore, the solution not only decides the groupings but also quantifies the "closeness of fit" of how well the strings belong to this sub-dictionary.

There are many advantages to this approach. Unlike estimation methods, the finite dictionary can be quite general. Instead, the pairs of strings are individually compared to achieve the learning. Also, the technique is adaptive. Furthermore, unlike heuristic methods [7,30] which can merely impose a user's criterion for closeness between two strings, we can now generalize a closeness criterion to extrapolate whether a string belongs in a potential sub-dictionary. Finally, (and far from being insignificant -- especially when the strings are long and the dictionary is large) there is no human intervention required to decide on a "best" string for each sub-dictionary. The system automatically and adaptively stipulates its own "best" representative for every sub-dictionary.

The strategy utilized in this paper utilizes the philosophy of the Object Migrating Automaton (OMA) that is powerful in equi-partitioning [35,36,47]. In the interest of brevity, we omit the description of the OMA here and refer the reader to [35,36] for its structural details and for a review of the other reported solutions to equi-partitioning. In passing, we would like to mention that the OMA is extremely accurate and fast -- experimentally, it converges to the true solution all the time, and does so with a speed which is an order of magnitude faster than the scheme due to Yu *et. al.* [47] especially when all the objects are initialized to be in the respective boundary states.

III. AUTOMATON-BASED STRING TAXONOMY

Note that we have assumed that $H=\{X_1,...X_J\}$ is to be partitioned into K equi-sized sub-dictionaries. To do this, we first specify how the strings themselves are to be compared. Various numeric and non-numeric measures relating two strings have been reported in the literature. Some of the numeric measures [1,10-12,14-17,19,22-24,27,31,32,37,39,41,45,46] include the Generalized Levenshtein Distance, the Length of their Longest Common Subsequence (LLCS) and the Length of their Shortest Common Supersequence. Indeed, in [14,15] a common basis for all these numerical measures has been specified. Although in this paper we shall quantify the similarity between two strings using a function of their LLCS, by virtue of the results in [14,15] we believe that any of the numeric measures catalogued there will yield comparable results. We define Sim(X,Y), the similarity between X and Y as the normalized LLCS defined as follows:

$$\text{Sim}(X,Y) = \frac{2.\ \text{LLCS}(X,Y)}{|X|+|Y|}$$

For example, if X= "AATGCC" and Y="ATGCA", their LLCS is 4, and Sim(X,Y) is 0.7273.

To make an automaton arrive at an efficient partitioning we shall require it to migrate the pairs of strings between the partitions based on the latter similarity metric. Indeed, we shall require that

the automaton reckon X and Y to be classified together if the Sim(X,Y) is greater than a user-defined threshold, θ. Throughout the first part of this study we have set the threshold θ to be 0.5. In the latter part of the study when we have attempted to hierarchically partition the dictionary into sub-dictionaries and further partition each sub-dictionary into sub-sub-dictionaries, we have set θ to be 0.5 at the first level and to be 0.7 at the "leaf" level. By computing the Sim(X,Y) between each pair of strings and by systematically utilizing a table of similarities the automaton must adaptively learn how to partition H effectively.

3.1 The String Taxonomy Learning Automaton

The learning automaton presented here, called the String Taxonomy Learning Automaton (STLA), utilizes the philosophy of the OMA and assumes that there is an underlying unknown grouping. When the algorithm is initialized (i.e., before the partitioning algorithm is invoked) the elements of H may be randomly scattered among the various sub-dictionaries. Hopefully, as the learning proceeds the STLA will utilize the similarity between the strings intelligently and migrate them so that similar strings are associated together.

We define the String Taxonomy Learning Automaton (STLA) as an 8-tuple as below :

(H , $\{\phi_1, \phi_2, ..., \phi_{KN}\}$, $\{\alpha_1, \alpha_2, ..., \alpha_K\}$, β, **Q**, **G** , **M**, **Z**), where,

(i) $H=\{X_1,...X_J\}$ is the set of strings.

(ii) $\{\phi_1, \phi_2, ..., \phi_{KN}\}$ is the set of states.

(iii) $\{\alpha_1, \alpha_2, ..., \alpha_K\}$ is the set of K actions, each representing a certain sub-dictionary into which the elements of H must fall.

(iv) $\beta = \{0, 1\}$ is its set of inputs where '1' represents a penalty and '0' a reward.

(v) **Q**, the transition function specifies how the strings should move between the various states and is quite involved. It will be explained in detail presently.

(vi) The function **G** partitions the set of states for the sub-dictionaries. For each action α_j, there is a set of states $\{\phi_{(j-1)N+1}, ..., \phi_{jN}\}$, where N is the depth of memory. Thus,

$$\mathbf{G}(\phi_i) = \alpha_j \quad \text{if} \quad (j-1)N + 1 \leq i \leq jN \qquad (1)$$

This means that the string in the automaton chooses α_1 if it is in any of the first N states, it chooses α_2 if it is in any of the states from ϕ_{N+1} to ϕ_{2N}, etc. We assume $\phi_{(j-1)N+1}$ to be the most internal state of action α_j, and ϕ_{jN} to be the boundary state. These are called the states of *MaximumCertainty MinimumCertainty* respectively

(vii) **M** is the set of Similarity Measures, Sim(X,Y) between all pairs in H.

(viii) **Z** is the set specifying the strings deemed to be individually similar. It is stored as a list in which the adjacent elements $<z_k,z_{k+1}>$ (where k is odd) are strings whose similarity index is greater than θ.

As in the case of the OMA, we shall require that all the elements of H move around between the states of the machine, and thus it is distinct from traditional learning automata. Also, if X_i is in action α_j, it signifies that it is in the sub-dictionary whose index is j. Observe too that if the states occupied by the strings are given, the sub-dictionaries can be trivially obtained using (1) . This will thus completely specify the set of sub-dictionaries dictated by the STLA.

Let $\omega_i(n)$ be the index of the state occupied by $X_i \in H$ at the n^{th} time instant. Based on $\{\omega_i(n)\}$ and (1) let us suppose that the STLA decides a current partitioning of H into sub-dictionaries. Using this notation we shall later describe the transition map of the STLA.

First of all, observe that the different states within a given sub-dictionary quantify the measure of certainty that the scheme has for a given string belonging to the sub-dictionary in question. At system start-up all the strings are placed in the boundary state (of *MinimumCertainty*) of their initially randomly chosen sub-dictionaries indicating that the scheme is initially uncertain of the placement of all the strings. As the learning proceeds, similar strings will be rewarded for their being together in the same sub-dictionary and they will thus migrate towards their most internal state of the sub-dictionary -- their corresponding states of *MaximumCertainty*, indicating that the system is more certain that the strings belong there. Likewise other strings will be penalized and are either moved towards their boundary state or to another sub-dictionary, indicating the system's current ambiguity in associating them to the current sub-dictionary.

Initially, the STLA begins its learning process by evaluating the table of similar string pairs **Z** as follows. Consider the strings X_u and X_v. First of all a function which computes the similarity between them is invoked and the result is stored in the array **M**. Whenever the strings X_u and X_v are reckoned similar (i.e., $Sim(X_u, X_v) \geq \theta$), X_u and X_v are appended to **Z**.

The algorithm now moves into its main learning loop. The list **Z** is now traversed repeatedly and consecutive similar elements X_u and X_v are processed. If they are both assigned to the same sub-dictionary, the automaton (and in particular, X_u and X_v) is rewarded. However, if they are both assigned to distinct sub-dictionaries, the automaton is penalized. This mode of penalizing is called the *PenalizeSimilarStrings* mode, because, in this mode, strings which are actually similar are assigned to distinct sub-dictionaries, and the partitioning is therefore to be penalized.

After the complete list **Z** has been processed, the algorithm moves into the second phase of the learning which involves comparing each string to the best representative of its currently assigned sub-dictionary. For each sub-dictionary, this string should be the one which is currently most "certain" of its assignment. Typically, this string is the one which has received the most rewards for being in that sub-dictionary. Since the state occupied by a string represents the confidence of the automaton being in the current partitioning, for each sub-dictionary, we define its representative as the one which is closest to its most internal state. The second phase of the learning proceeds as follows. Every string that is dissimilar to the best representative of its current

sub-dictionary is stochastically penalized by attempting to migrate it to another sub-dictionary -- to the one whose best representative is most similar to the string in question. The stochasticity for the transition will be explained presently. As opposed to the previous mode of penalizing, this mode is called the *PenalizeDissimilarStrings* mode, since the penalizing is caused by dissimilar strings being assigned to the same sub-dictionary.

The cycle then continues to the next iteration where both the algorithm's phases repeat.

We now describe the actual transitions described by **Q** for each of these operations.

(i) Transitions for Rewards

On being rewarded, since X_u and X_v are in the same sub-dictionary, say, α_j, both of them are moved toward the most internal state of that sub-dictionary, $\phi_{(j-1)N+1}$, one step at a time. See Figure I(a).

(ii) Transitions for Penalties : PenalizeSimilarStrings Mode

This is the case encountered when two similar strings, X_u and X_v, are located in distinct sub-dictionaries. Let us assume that X_u and X_v lie in different sub-dictionaries, say α_j and α_m respectively, (i.e. X_u is in state ω_u, where $\omega_u \in \{\phi_{(j-1)N+1},.., \phi_{jN}\}$, and X_v is in state ω_v, where $\omega_v \in \{\phi_{(m-1)N+1}, ..., \phi_{mN}\}$). Then they are moved away from $\phi_{(j-1)N+1}$ and $\phi_{(m-1)N+1}$ as follows:

a) If $\omega_u \neq \phi_{jN}$ and $\omega_v \neq \phi_{mN}$, then move X_u and X_v one state towards ϕ_{jN} and ϕ_{mN} respectively. (Move them towards the boundary states.) See Figure I(b).

b) If at least one of X_u or X_v is in the boundary state of *MinimumCertainty*, (i.e. either $\omega_u = \phi_{jN}$ or $\omega_v = \phi_{mN}$) , then move the string in the boundary state, say X_u, to ϕ_{mN}, the boundary state of α_m. In this case, since this will result in an excess of strings in α_m, one of the strings in α_m other than X_u is moved to ϕ_{jN}, the boundary state of α_j. We choose to move the one closest to ϕ_{mN}. See Figure I(c).

(iii) Transitions for Penalties : PenalizeDissimilarStrings Mode

In the second phase, every string, U, that is dissimilar[2] to the best representative of its current sub-dictionary, say α_j, is penalized stochastically with a probability which is initially set to zero and incremented as the learning continues. This means that initially, the second phase will be seldomly invoked, and as the learning proceeds, this phase will be invoked more frequently. Let us suppose that the string U is in state ω_U. If both U and Y are not in the boundary state, they are merely moved towards the boundary by one state. If, however, U is in the boundary state, the scheme opts to migrate U to another sub-dictionary. In order to achieve this, the algorithm first of all, searches for the best sub-dictionary to which it should be migrated. This is done by searching among the sub-dictionaries for the one whose best representative is most similar to U. Let us

[2]In the experiments conducted, the definition of similarity was slightly modified for the second phase. In the first phase, we reckoned X and Y to be similar if Sim(X,Y) was greater than Θ. In this case, the strings were reckoned to be similar if Sim(X,Y) was greater than or equal to Θ-0.1. This was purely a subjective choice.

suppose that this sub-dictionary is α_{Sw}. The string closest to the boundary state of α_{Sw} is now moved to the sub-dictionary of U and U in turn is migrated to the sub-dictionary α_{Sw}. The analogous migration is done if Y is in the boundary state but not U.

The actual algorithm for the STLA is formally presented in the Appendix.

Note that although the fundamental principles involved in the individual migrations are based on the philosophy used in the OMA (namely, the Tsetlin-like transitions on being rewarded and penalized), the algorithm is completely different. The primary differences are the following :

(i) Unlike the OMA where the migrations are done "on request" (i.e., when a user performs a query), in the STLA the migrations are performed for all similar pairs in **Z**.

(ii) Unlike the OMA, which has no way of penalizing "non-accessed elements" the STLA has a strategy of penalizing them by considering how similar the strings within the same sub-dictionary are. Clearly, this cannot be done in the OMA because, in that case, the system is absolutely dependant on the users' queries. In the present case the system can quantify how fitting a string is for a sub-dictionary, because M is readily available.

(iii) Unlike the OMA, the concept of comparing elements to the best representative of a sub-dictionary has been introduced for the first time in the STLA. In *statistical* PR this can be done because the mean feature for a class can serve as its representative. In this case, although such a mean does not exist, the string closest to the most internal state can be reckoned to be the string that best represents *that* sub-dictionary. This has rendered the second phase of the loop possible -- and permitted us to migrate a dissimilar word from its current sub-dictionary to another in which it is more suitable.

(iv) Finally, the concept of stochastically migrating dissimilar elements is new to the STLA. This has rendered the second phase of the algorithm to be rather irrelevant in the initial stages of the algorithm and to be more frequently invoked once the strings tend to find their rightful places. Of course, this concept cannot be used in the traditional OMA because, in the latter, the question of comparing "dissimilar" elements never occurs. Indeed, in the OMA, whenever the user requests two elements they are assumed to be similar, and thus the objects migrated are fully controlled by the users' query stream.

IV. EXPERIMENTAL RESULTS

The STLA has been rigorously tested and the results that we have received are quite fascinating. The data which was used was obtained from three sources. In the first set of experiments the data consisted of noisy strings obtained from English words. In the second set of experiments, the data was obtained by using long noisy English sentences in which the delimiter information (found in the locations of the spaces) was discarded. The final experiment consisted of a dictionary of mutated noisy substrings of biochemical macromolecules. The results of each of

these experiments is given in the following subsections. Before we describe the details of the experimental results we first present a short description about the noisy string generation process.

4.1 Noisy String Generation

Let us suppose that we have to obtain a noisy version of a string $U \in A^*$, where A is the alphabet under consideration. The generation process assumes the definition of three distributions, G, R and S defined below. G is a distribution over the set of positive integers and defines the number of insertions performed in the mutating process and it satisfies :

$$\sum_{z \geq 0} G(z) = 1.$$

Examples of the distribution G are the Poisson and the Geometric Distributions.

The second distribution required is the distribution R, where the quantity R(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 R has to satisfy the following constraint :

$$\sum_{a \in A} R(a) = 1.$$

Finally, apart from G and R, the generation requires a probability distribution S over A x $(A \cup \{\lambda\})$, where λ is the null symbol. 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 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 $a \in A$:

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

Using the above distributions we now describe the garbling algorithm (the noisy string generation process). Let $|U| = N$. Using the distribution G, we first randomly decide on the number of symbols to be inserted, say, k. The algorithm then determines the position of the insertions among the individual symbols of U. In this case, each of the $(N+k)! / (N!\ k!)$ possible positions are assumed equally likely. The actual symbols of U which are not at the inserted positions are now substituted or deleted using the distribution S. Finally, the individual symbols of the alphabet are inserted using the distribution R at the inserted positions.

The above process has been shown to be stochastically consistent and functionally complete [34] and is to our knowledge the only reported method by which noisy strings containing

substitution, deletion and insertion errors can be generated according to any specified distribution. Since our intention was to rigorously test the STLA for various mutations of strings, noisy strings were generated using this generation scheme and these strings served as the input for the partitioning algorithm.

4.2 Experiment I : Short Noisy English Strings

The first set of experiments involved studying the partitioning ability of the STLA for noisy English strings. Eight sets of strings (a total of eighty) were generated from an initial set of eight root words. The number of insertions permitted was distributed geometrically and the substitutions were generated using a confusion matrix based on the proximity of keys on the typewriter keyboard. Some of the noisy strings generated are :

engineering → { enneeriunjk, jngineeving, qagibmfring, sngdnegering}
psychology → { psycfgholgy, psvholsgy, psychqfogy, psychocogr }
mathematics → { mahematrcs, marhecatics, madhemaics, tathematiqs }

A complete list of the eight strings generated is given in Table I. The set of noisy strings was then specified as the input to the STLA without the latter knowing whence the strings were generated. The eighty strings were randomly assigned to the eight sub-dictionaries and placed at the corresponding boundary states. The STLA was then invoked and after the initial preprocessing which involved evaluating the inter-string similarities, the various strings were migrated as per the reward and penalty transition maps. Table Ia and Ib list the initial and final partitionings respectively. Note that finally, all the eighty strings were correctly partitioned -- without individually comparing each of them to a "template" string as would have been the strategy employed by a traditional syntactic PR environment. The power of the scheme is obvious !!

4.3 Experiment II : Long Strings of English Characters

The second set of experiments involved studying the partitioning ability of the STLA for long strings of English characters. Ten sets of noisy strings were generated from ten original strings of length approximately 50. A typical original source string used was :

"some of the worlds best water skiers come from canada".

Since there is considerable information in the delimiter, space, the latter was removed, yielding the corresponding source string to be :

"someoftheworldsbestwaterskierscomefromcanada"

The strings were then noisily garbled using the above described garbling mechanism, where, as before, the number of insertions permitted was distributed geometrically and the substitutions were generated using a confusion matrix based on the proximity of keys on the typewriter keyboard. A typical noisy string obtained as a result of the garbling was :

"someofwhewcrmdsbestzbersjitrseomefsomcandds"

The set of one hundred noisy strings served as the input to the STLA. The strings were randomly assigned to the ten sub-dictionaries and placed at the corresponding boundary states. The STLA then migrated the strings using its reward and penalty transition maps. Table II shows the final partitioning in which all the hundred strings were correctly partitioned. Again, the power of the scheme is clear especially when we realize that the system is absolutely unaware of the original strings which generated the elements of the dictionary, and thus it did not have any "error-free" fixed string to which it could compare the noisy strings to. Also note that the performance of the STLA is not forfeited by extracting the crucial inter-word delimiter information.

4.4 Experiment III : Taxonomy of Mutated Macromolecules

In the final set of experiments we studied the power of the STLA to partition macromolecules in a hierarchical fashion. Consider the following mutating process. Let us suppose that we started the process with a set of macro-molecules $\{X_1, X_2, .., X_J\}$. Each X_i is randomly mutated to yield a new set of molecules for the "next generation". For the string X_i we refer to the latter set as $\{X_{i1}, X_{i2}, .., X_{iK}\}$. Now, for the subsequent generation, each X_{ij} is further mutated to yield a set of new macromolecules $\{X_{ij1}, X_{ij2}, .., X_{ijM}\}$. The dictionary, H, in this case consists of the entire set of strings,

$H=\{X_{111},X_{112},..,X_{11M},...,X_{1K1},X_{1K2},..,X_{1KM},..X_{ij1},X_{ij2},..,X_{ijM},..,X_{JK1},X_{JK2},..,\ X_{JKM}\}$.

The task of partitioning is now much more complex than what was studied in the earlier two experiments. By allowing a "tree" of STLA to process H, we intend to hierarchically partition H not only in the respective sub-dictionaries, but also to partition each sub-dictionary into the corresponding "sub-sub-dictionaries". Of course, the basic premise for the whole experiment is that the tree of STLA is unaware of the original set of macro-molecules, $\{X_1, X_2, .., X_J\}$, and consequently, the individual machines are constrained to partition them by just comparing noisy strings with other noisy strings.

The data for the experiment was obtained from *The Atlas of Protein Sequence and Structure* [3, page D81]. The task of the STLA at the lowest level (the level closest to the root) was to partition the JKM elements into J sub-dictionaries. At the next level, each of these sub-dictionaries was processed by another STLA whose task was to partition *its* input (which was a sub-dictionary) into K sub-sub-dictionaries.

In this set of experiments the strings used were substrings of the following proteins :

(i) myoglobin from the harbour seal,
(ii) the human hemoglobin gamma chain
(iii) ferradoxin obtained from spinach, and,
(iv) adrenodoxin obtained from bovine.

The composition of these proteins is given in Table IIIa.

These four long strings were first mutated by garbling approximately 25% of the string through the mechanism described earlier. Unlike the previous cases, where we worked with the English alphabet and the typewriter keyboard, in this case the noise generation was based on a subjectively created "random" confusion matrix which caused the individual molecular symbols to be substituted, inserted and deleted. For the first level, the fragment of the string (approximately 25%) which was mutated was randomly chosen. This was repeated for the four randomly chosen quarters, and thus 16 mutated strings were obtained from the original four. At the next level each of these sixteen were further mutated, and in this case, to further accentuate the garbling process, the entire string was rendered noisy. This yielded the total input set of 64 noisy strings.

The set of 64 strings were now classified at the "root" level into four sub-dictionaries using a single STLA. For the initialization stage, they were first randomly distributed into the four sub-dictionaries and assigned positions at the boundary states of these dictionaries. Subsequently, at the "leaf level" four distinct STLA operated in parallel on the sub-dictionaries to further partition them into sub-sub-dictionaries. At this level, the value of θ was set to be 0.7, and thus the STLA asserted that two strings were similar only if their similarity index was greater than or equal to 0.7.

The hierarchy of STLA performed very elegantly. Indeed, in this case, all the strings were correctly partitioned into their respective sub-dictionaries, and the sub-dictionaries were also correctly partitioned. Notice that as a result of this, the scheme could correctly learn the entire mutating pattern of the proteins without any *a priori* information of the molecular compositions of the original "source" proteins.

A subset of 16 of the 64 strings clustered in their sub-sub-dictionaries is given in Table III. Notice that the clustering is achieved without comparing each of the strings to a template, but by merely comparing them between themselves and migrating them using "similar-dissimilar decisions" as dictated by the STLA. The power of the hierarchy of STLA is clear.

4.5 Drawbacks of the STLA

Although the STLA is powerful and, to our knowledge, is a pioneering contribution to the entire area of string taxonomy, it still, unfortunately, has some noticeable drawbacks. The first major disadvantage of the scheme is that it assumes that the dictionary can be equi-partitioned. First of all notice that using techniques similar to those utilized in [35,36,47] this problem can be shown to be NP-Hard. With a little insight it is easy to see that the equi-partitioning constraint translates into the "equally likely" scenario for the *a priori* distributions of the classes traditionally used in statistical PR. The case when the sub-dictionaries are not equally sized, is yet open. If we know the relative sizes of the sub-dictionaries, we believe that the problem is still tractable using ideas similar to the STLA, because, the current size of a sub-dictionary would inform us whether a new

entry would require the migration of another element or not. But if the relative sizes of the sub-dictionaries are themselves unknown, the problem is yet unsolved. We are currently investigating whether the solution to the underlying object partitioning problem [36] can be adapted to this.

The second major drawback of the STLA is that it requires the computation of the pair-wise similarity of all the strings in **H**. This is typical of all "nearest neighbour" type algorithms, and thus usually, cannot be circumvented. However, in this case, since the string in the most internal state of a sub-dictionary can be viewed as its most ideal representative element, a comparison between an element and the various "best representatives" may suffice to achieve an effective taxonomy. We are currently investigating this possibility.

V. CONCLUSIONS

In this paper we have presented, to our knowledge, the first reported solution to the "String Taxonomy Problem" which can be utilized to enhance the capabilities of a typical syntactic PR system. Typically, such a system compares a noisy string with every element of a dictionary, H. The problem of classification can be greatly simplified if the dictionary is partitioned into a set of sub-dictionaries, because, in this case, the classification can be hierarchical -- the noisy string can be first compared to a representative element of each sub-dictionary and the closest match within the sub-dictionary can subsequently located. In its generality, the "String Taxonomy Problem" involves the problem of sub-dividing a set of strings into subsets where each subset contains "similar" strings. In this paper we have presented a learning-automaton based solution to the problem. The solution is the String Taxonomy Learning Automaton (STLA) which has been developed using the same philosophy as that used in the Object Migrating Automaton (OMA) whose power in clustering objects and images [33,35] has been reported. The power of the scheme for string taxonomy has been demonstrated using random strings and garbled versions of string representations of fragments of macromolecules.

Table I : String Taxonomy of Short Strings

Table Ia : List of Strings Prior to Taxonomical Analysis

Sub-dictionary : ω_1

String_index	State	String
0	9	qagibmfring
1	9	comelpfxity
2	9	psychqfogy
3	9	axcliwectur
4	9	sngdnegering
5	9	engineerina
6	9	eyginring
7	9	afgpritaamic
8	9	engitzering
9	9	pmvchomog

Sub-dictionary : ω_2

String_index	State	String
10	19	architecnure
11	19	photograpmy
12	19	cohjvlerity
13	19	engineaarrng
14	19	arkhitezturx
15	19	psgeochelogy
16	19	madhemaics
17	19	guohrafhijjq
18	19	ensineerinq
19	19	psychokogy

Sub-dictionary : ω_3

String_index	State	String
20	29	goraphicsleq
21	29	geogrpphical
22	29	mathematics
23	29	photigraphy
24	29	irchatmcturi
25	29	photogravhy
26	29	marhematics
27	29	arghitecjure
28	29	mayhematias
29	29	psecholody

Sub-dictionary : ω_4

String_index	State	String
30	39	mahematrcs
31	39	algorichroc
32	39	gvograuhicap
33	39	acgwtithmic
34	39	marhecatics
35	39	veograpaical
36	39	muthematzilo
37	39	eogrophicalg
38	39	photography
39	39	guographicah

Sub-dictionary : ω_5

String_index	State	String
40	49	zomplexipy
41	49	gaographieac
42	49	archytemtrre
43	49	klotogrrpur
44	49	abchiteptqrey
45	49	complekity
46	49	architicjtge
47	49	engieeering
48	49	architqcture
49	49	enneeriunjk

Sub-dictionary : ω_6

String_index	State	String
50	59	jngineeving
51	59	psychojojy
52	59	ptotsogrdphy
53	59	photagroihy
54	59	algorithmil
55	59	geoeraphical
56	59	olgorrtmic
57	59	psychocogr
58	59	geogcrgphiccl
59	59	algorivhmiz

Sub-dictionary : ω_7

String_index	State	String
60	69	tomplexahy
61	69	amroritwmic
62	69	nrchitemthre
63	69	psycfgholgy
64	69	uathematics
65	69	argorimid
66	69	komplexity
67	69	phjcolocq
68	69	xhonogradhz
69	69	phytograpsy

Sub-dictionary : ω_8

String_index	State	String
70	79	tathematiqs
71	79	mzsheiatice
72	79	atgorithmic
73	79	yomplexity
74	79	algowithmic
75	79	psvholsgy
76	79	cofplexisy
77	79	phsdtoygapy
78	79	jkgplexgby
79	79	comolpity

Table I : String Taxonomy of Short Strings

Table Ib : List of Strings After Taxonomical Analysis

Sub-dictionary : ω_1

String_index	State	String
70	5	tathematiqs
26	3	marhematics
22	2	mathematics
64	2	uathematics
71	0	mzsheiatice
34	0	marhecatics
30	0	mahematrcs
28	0	mayhematias
16	0	madhemaics
36	0	muthematzilo

Sub-dictionary : ω_2

String_index	State	String
0	11	qagibmfring
5	11	engineerina
18	10	ensineerinq
49	10	enneeriunjk
13	10	engineaarrng
8	10	engitzering
6	10	eyginring
4	10	sngdnegering
47	10	engieeering
50	10	jngineeving

Sub-dictionary : ω_3

String_index	State	String
75	20	psvholsgy
57	20	psychocogr
19	20	psychokogy
9	20	pmvchomog
67	20	phjcolocq
51	20	psychojojy
29	20	psecholody
15	20	psgeochelogy
2	20	psychqfogy
63	20	psycfgholgy

Sub-dictionary : ω_4

String_index	State	String
17	39	guohrafhijjq
20	31	goraphicsleq
21	31	geogrpphical
32	31	gvograuhicap
39	31	guographicah
55	31	geoeraphical
58	31	geogcrgphiccl
35	31	veograpaical
41	30	gaographieac
37	30	eogrophicalg

Sub-dictionary : ω_5

String_index	State	String
46	42	architicjtge
42	40	archytemtrre
27	40	arghitecjure
10	40	architecnure
48	40	architqcture
44	40	abchiteptqrey
14	40	arkhitezturx
3	40	axcliwectur
62	40	nrchitemthre
24	40	irchatmcturi

Sub-dictionary : ω_6

String_index	State	String
72	59	atgorithmic
7	51	afgpritaamic
74	50	algowithmic
59	50	algorivhmiz
56	50	olgorrtmic
54	50	algorithmil
61	50	amroritwmic
33	50	acgwtithmic
31	50	algorichroc
65	50	argorimid

Sub-dictionary : ω_7

String_index	State	String
53	61	photagroihy
77	61	phsdtoygapy
69	60	phytograpsy
38	60	photography
25	60	photogravhy
23	60	photigraphy
11	60	photograpmy
43	60	klotogrrpur
68	60	xhonogradhz
52	60	ptotsogrdphy

Sub-dictionary : ω_8

String_index	State	String
78	70	jkgplexgby
45	70	complekity
79	70	comolpity
60	70	tomplexahy
73	70	yomplexity
12	70	cohjvlerity
1	70	comelpfxity
40	70	zomplexipy
76	70	cofplexisy
66	70	komplexity

Table II : String Taxonomy of Long Strings after Analysis

String_index	Sub-dictionary	String
52	1	someoftheworudsbestwateaskieryxomafromeanada
41	1	shmeofthewrldsbestwaterskierscoxefromcfnada
29	1	somooftheworldsyestoaterskixrocpmefromcanada
28	1	someofwhewcrmdsbestzbersjitrseomefsomcandds
9	1	someofmhwworldsbasxwaherskvmrswfmefrxmcanada
7	1	domeoltjezorldsaebtwatwrsniecscomifrnmctgada
4	1	sosemfthpworywsbestwaterskierscomeffomcansda
72	1	soopofthewdrdsbestwaterstiersmomegrymcanafa
53	1	someoftheworlvsbestwaeqrskierscomefromcanada
73	1	iomeqftheworldsbestwateskiergcomefromdandda
String_index	Sub-dictionary	String
1	2	nhvarinfylifehavqievrbeentonorthwettteruttorvs
0	2	nevgrinyeiselaveieverbeeqtongrthwimttedritopies
81	2	ieverinmylieuhavpieixrbegntonojthwemqterritories
70	2	revhrinmylbfelzveqevdrbefntonovtowesttlrritssves
58	2	neverinmyofehavievzrbeektunortiwesttergitories
84	2	neaerinmylifchavejenvrbeenbonortgwestterrimorpzk
37	2	neveriemylifentleselerbecntonorrhwvstuerritories
27	2	nevfribmyyieehaveuevnrbvntonozthaehtyyrritolkis
18	2	keverinmgcikemayesevecfeentonortzwesttergigorien
93	2	nzverinlylzfehaveiecerbeentonorthwestlerzitories
String_index	Sub-dictionary	String
68	3	aachitecturalstabnwxunsyreajphiknyincdeqzate
63	3	architeczuralstmiwhnspreadthtvlyisadzquate
60	3	arcqitectfralstwpnbhenshreadthinlvisamequhte
57	3	arcjilvcturalstainwhenseroadthislyisadequate
55	3	xmhikcctural staiqwhpnsxreadthjzlyisadeeuate
30	3	architecxuralstainwyenspreadthenayisadequate
22	3	architkctkralstainwzensfrearthjnkynsaoequate
13	3	arcvitectjrrlstazndhenspreudthiwlyisaxequate
90	3	arahiteorjralstainwhenspreadshinlyisadoquate
76	3	architecturmlataindjenspreadthjnlyisadequaqe
String_index	Sub-dictionary	String
80	4	haveyoilvrbefntoazodinwjxghquablgiraffebhbound
94	4	haveyoeveybeenvlazooinwhichquaiogiraffezjbgunx
79	4	haveyogeverjeentoazooinwhichquailgiraffeqglound
38	4	javtyouevegbeznkoazoounwhicxqryiroirafuesaboucd
89	4	naqeyoueverbhewtolzpoinwhichqqiilgiraffesabtund
36	4	hsvgyoulverbeentoazoobnmhlctquailgirrffesacwund
86	4	haveyouepubeentoszooinwhichqjailgirafftsabwund
35	4	eavekoueverbeeytoafainahkchhuaiwfirtffesabound
33	4	haveyoueverbetntoazoxtnwtichquaeltieaffesrbjunz
21	4	haveyoueverseqntoazyjinwhichqdlilgiraffesbound
String_index	Sub-dictionary	String
40	5	seapesostreesalvuseeulforprovicinfshadefoodqed
17	5	peavesqnerzesareuseftlforprovzdixghhadgfvodbed
25	5	lhavesontrmesareusefulforprovidijgxhasefonrbed
71	5	loacesontrqesaredszfulforprovipibgshadefoodcei
44	5	aavesontkeecxrnusetulfvrprovipscgshadefsodbed
51	5	mhavesojtrehaarensefulforrrovidrjgshadefvodbdd
32	5	leavesontrjesarhusefhlforprovidinishadeaoodved
31	5	lepvezctrewkareusefxmforpjovidineshadefondbed
46	5	lejvesonbreesaiehsefulflqprovidinkshaddfoodbed
48	5	leiveswtrhesaveusefulfouprovidingshadnaiodbed

Table II : String Taxonomy of Long Strings after Analysis (Contd)

String_index	Sub-dictionary	String
64	6	atriedanytruqmetqodofcpnwdcoptrokiszeargadowater
14	6	ataieuandtruemethoqofcrowdfntrolrsteargasorwatur
87	6	atriedaxdtruemethokkfcroddiontrdlisheargasorwter
75	6	rkrhedandtrcemethodvhcrowbcontroyisleajasorrater
56	6	utpihdandtruevethtdofcrswdcozsroligteargawowateo
47	6	morimlamdteuebhnhodofcrowdcontnolfvuaargasorwatm
69	6	avrietandoruemethodifcrvwdcoktrylisteargasorwater
24	6	ftzptcandtruelethodofcrowucontrolisaeaugaworwater
20	6	atriedapdfkmgmethouopcroidconbrolisteargwsorwater
74	6	ytjiedandtruemethodofcroddcontrolisteargasorwater
String_index	Sub-dictionary	String
77	7	orienteeringisawtyoflifpforlastuinnsscedjszogu
3	7	orienteerangnsarayoglureporosytfinnsszfmesoogs
26	7	orienkuerqnjipawayoflvfeformostfinnsswedwsnogh
83	7	orienteeringwsawayofliueformostfinnsspedosnoos
97	7	osiewteeringisawgfofliieformtctfinnsswidesnogs
66	7	orienteeringisaiayoflifebormostfinnsswedesgodb
85	7	iuienteqringilawayofqifeformultfinnsswedesnogg
65	7	tricnteurdngifpwayvfljfeformostfinneswebesnogs
54	7	orienteerfngistwayfflzferormostfinnssweyesnogs
39	7	orienteerinucsawayiflifeformostficnssweselnpgs
String_index	Sub-dictionary	String
92	8	thijisatestoverclongstrikgqollengmhmbotfifty
62	8	thidisatestofverylongsaringsoflengthaboutfifty
16	8	rhisioaeestoflerylongstringsoflengthaboctfifay
8	8	vhisiuatestofveryoongstrclusojlengthabvutfifta
6	8	hisqsatustmfverylongrtringsoflezgthabouteifiy
5	8	tnisisatestcfverylongstrinisofledgthaboutjtfti
59	8	thisisatdstofveryfongstringscflengthabeutfifty
78	8	qhisitztestofvesllongstrinjsoflegthaboutfisty
2	8	thisiaatustofverylongstripgsoflenguhhboftfjjty
34	8	khisisatestofverylongsaribmsoflekkgtuhabottfif
String_index	Sub-dictionary	String
49	9	rowmanyskeepcanasleepsheacershearrfashwkpslecpw
45	9	howmanysteepcanbshrerohearersheafipasheepsleeps
43	9	hkwmanyshkepcahasheepsheareyhearifasheepsveeps
67	9	ygwnnysheepcanashvxpshearedszdarifasheepsfeeps
2	9	iormaxxshwepcanasheepstqarqrshearifasheegssoeps
19	9	hozoanysheepyanasheepshekreqshearifasheepsgbeps
11	9	iowmanysheepcabasheepsheasnrlhearmfashmepsleeps
10	9	howjanycteepcanazreexsdeaietshearifasheypsmeeps
2	9	hoemanysheepcanoshnopshlarershearifasheepsleeps
88	9	howmanysheepcvnaszhepshearrruhearifashevpsleeps
String_index	Sub-dictionary	String
95	10	frogstladsahvalamaqkerslbveundetrowksanqmiss
12	10	frogseadsandsplamanderslivqhnderrovksandooss
91	10	frxgstobpzandsalamandersliveunuerrocwsawdxofs
96	10	grwgstoadsandfalamandmwsbiveundoarocksandmoss
82	10	srogqdoadswndsalamddersliveuvdwrujnksandmass
61	10	foogqiodsandyalamxnderbliveunqerrocksandmors
50	10	foogchoedsandsalamanpeksleveunderrocksandmoss
15	10	ajogstoadsabdmalambndorslileunderrocksandvoss
99	10	frogstoadmandsalmmanuersliveunddkroiksnndmugs
98	10	frogstogdsaddsabakafdersbivehndeezocksaydmcss

Table III : Hierarchical String Taxonomy of Biological Macromolecules

Table IIIa : List of the Four Original Protein Sequences

Protein source : harbour seal
Protein Name : myoglobin
Protein Structure :
glsdgewhlvlnvwgkvetdlaghgqevlirlfkshpetlekfdkfkhlkseddmrrsedlrkhgntvltalggilkkkghhe-aelkplaqshatkhkipikylefiseaiihvlhskhpaefgadaqaamkkalelfrndiaakykelgfhg

Protein source : human
Protein Name : hemoglobin gamma chain
Protein Structure :
ghfteedkatitslwgkvnvedaggetlgrllvvypwtqrffdsfgnlssasaimgnpkvkahgkkvltslgdaikhlddlkgt-faqlselhcdklhvdpenfkllgnvlvtvlaihfgkeftpevqaswqkmvtgvasalssryh

Protein source : spinach
Protein Name : ferradoxin
Protein Structure :
aaykvtlvtptgnvefqcpddvyildaaeeegidlpyscragscsscagklktgslnqddqsfldddqidegwvltcaaypvs-dvtiethkeeelta

Protein source : bovine
Protein Name : adrenodoxin
Protein Structure :
sssqdkitvhfinrdgetlttkgkigdslldvvvzbnldidgfgacegtlacstchlifeqhifekleaitneennmbzlldlaygltd-rsrlgcqicltkamdnmdtvrvpdavsda

Table IIIb : A subset of 16 of the 64 Protein Sequences which were partitioned into sub-dictionaries and sub-sub-dictionaries

Sub-sub-dictionary : $\omega_{1,1}$
Source : glsdgewhlvlnvwgkvetdlaghgqevlirlfkshpetlekfdkfkhlkseddmrrsedlrkhgntvltalggilkkkghheaelkplaqshatk
Mutated Strings :
houylesnxrqpwculgvonbqetjfkfaslfkshpetlekfdkfkhlkseddmrrsedlrkhgntvltalggilkkkghheaelkplaqshatk
glsdgewhlvlnvwgkvetdlaghgqevlirwfksryqlaqkhsovkglksesojrrsedlrkhgntvltalggilkkkghheaelkplaqshatk
glsdgewhlvlnvwgkvetdlaghgqevlirlfkshpetlekfdkfkhlkseddmrshdlekheohdptqeigecfkxghheaelkplaqshatk
glsdgewhlvlnvwgkvetdlaghgqevlirlfkshpetlekfdkfkhlkseddmrrsedlrkhgntvltalggilkkkdgheeyzfrewhgsiyc

Sub-sub-dictionary : $\omega_{2,1}$
Source : ghfteedkatitslwgkvnvedaggetlgrllvvypwtqrffdsfgnlssasaimgnpkvkahgkkvltslgdaikhlddlkgtfaqlselhcdklh
Mutated Strings :
dgjjxkovqqktkwopviveragyfgqghqglvypwtqrffdsfgnlssasaimgnpkvkahgkkvltslgdaikhlddlkgtfaqlselhcdklh
ghfteedkatitslwgkvnvedaggetlgrllvgatxdttkfxaqtntsihmangndkvkahgkkvltslgdaikhlddlkgtfaqlselhcdklh
ghfteedkatitslwgkvnvedaggetlgrllvvypwtqrffdsfgnlssasaimgnplvksagtjnltlhpjfbkvgddlkgtfaqlselhcdklh
ghfteedkatitslwgkvnvedaggetlgrllvvypwtqrffdsfgnlssasaimgnpkvkahgkkvltslgdaikhlmfggnfdhsmlhcmbkdlh

Sub-sub-dictionary : $\omega_{3,1}$
Source : aaykvtlvtptgnvefqcpddvyildaaeeegidlpyscragscsscagklktgslnqddqsfldddqidegwvltcaaypvsdvtiethkeeelta
Mutated Strings :
naemckwkftknhrfftwmdvkmldpaeeegidlpyscragscsscagklktgslnqddqsfldddqidegwvltcaaypvsdvtiethkeeelta
aaykvtlvtptgnvefqcpddvyildaojfimqgipksgrevmnthtajklepkshnqddqsfldddqidegwvltcaaypvsdvtiethkeeelta
aaykvtlvtptgnvefqcpddvyildaaeeegidlpyscragscsscagklktgslnfbfjmoexolkypqcwksiwasdpdidvtiethkeeelta
aaykvtlvtptgnvefqcpddvyildaaeeegidlpyscragscsscagklktgslnqddqsfldddqidegwvltcaaypvsdrpckmkzdmtbta

Sub-sub-dictionary : $\omega_{4,1}$
Source : sssqdkitvhfinrdgetlttkgkigdslldvvvzbnldidgfgacegtlacstchlifeqhifekleaitneennmbzlldlaygltdrsrlgcqi
Mutated Strings :
aerbzblrjvferyogiltbqrupgdstlnvvvzbnldidgfgacegtlacstchlifeqhifekleaitneennmbzlldlaygltdrsrlgcqi
sssqdkitvhfinrdgetlttkgkigdslldnlbxqtjyfvghabpvvxlryjtvpyofeqhifekleaitneennmbzlldlaygltdrsrlgcqi
sssqdkitvhfinrdgetlttkgkigdslldvvvzbnldidgfgacegtlacstchlifxfhlfelxejifttcyjbzlldlaygltdrsrlgcqi
sssqdkitvhfinrdgetlttkgkigdslldvvvzbnldidgfgacegtlacstchlifeqhifekleaitneennmketulrauylosnskdcbf

REFERENCES

1. A. V. Aho, D.S. Hirschberg, and J. D. Ullman, Bounds on the complexity of the longest common subsequence problem, *J. Assoc. Comput. Mach.*, 23:1-12 (1976).
2. R. L. Bahl and F. Jelinek, Decoding with channels with insertions, deletions and substitutions with applications to speech recognition, *IEEE Trans. Information Theory*, IT-21:404-411 (1975).
3. L. Devroye, *Non-Uniform Random Variate Generation*, Springer-Verlag, (1986).
4. M. O. Dayhoff, *Atlas of Protein Sequence and Structure*, National Biomedical Res. Found. ,Washington, (1971).
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. Dubes and A. K. Jain, Validity Studies in Clustering Methodologies, *Pattern Recognition*, , 8:235-254 (1976).
8. A. Ehrenfeucht, Plenary Lecture, CPM 1992, The Third International Symposium on *Combinatorial Pattern Matching*, in Tucson, May 1992.
9. G. D. Forney, The Viterbi Algorithm, *Proceedings of the IEEE*, 61-3:268-278 (1973)
10. P. A. V. Hall and G.R. Dowling, Approximate string matching, *Comput. Surveys*, 12:381-402 (1980).
11. D. S. Hirschberg, Algorithms for longest common subsequence problem, *J. Assoc. Comput. Mach.*, 24:664-675 (1977).
12. J. W. Hunt and T. G. Szymanski, A fast algorithm for computing longest common subsequences, *Comm. Assoc. Comput. Mach.*, 20:350-353 (1977).
13. P. Jacquet and W. Szpankowski, Analysis of digital tries with markovian dependencies, *IEEE Trans. Information Theory*, IT-37:1470-1475 (1991).
14. 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).
15. R. L. Kashyap and B. J. Oommen, Similarity measures for sets of strings, *Internat. J. Comput. Math.*, 13:95-104 (1983).
16. R. L. Kashyap and B. J. Oommen, The noisy substring matching problem, *IEEE Trans. Software Engg.*, SE-9:365-370 (1983).
17. 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).
18. R. L. Kashyap, and B. J. Oommen, String correction using probabilistic methods, *Pattern Recognition Letters*, 147-154 (1984).
19. A. Levenshtein, Binary codes capable of correcting deletions, insertions and reversals, *Soviet Phys. Dokl.*, 10:707-710 (1966).
20. S. H. Levine, An adaptive approach to optimal keyboard design for nonvocal communication, *Proc. of the Internat. Conference of Cybernetics and Society*, 334-337 (1985).
21. 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).
22. R. Lowrance and R. A. Wagner, An extension of the string to string correction problem, *J. Assoc. Comput. Mach.*, 22:177-183 (1975).
23. D. Maier, The complexity of some problems on subsequences and supersequences, *J. Assoc. Comput. Mach.*, 25:322-336 (1978).
24. W. J. Masek and M. S. Paterson, A faster algorithm computing string edit distances, *J. Comput. System Sci.*, 20:18-31 (1980).

25. S. L. Minneman, A simplified Touch-Tone telecommunication aid for deaf and hearing impaired individuals, *Proc. of the 8th. Annual RESNA Conference*, Memphis, 209-211 (1985).
26. S. L. Minneman, Keyboard optimization technique to improve output rate of disabled individuals, *Proc. of the 9th. Annual RESNA Conference*, Minneapolis, 402-404 (1986).
27. 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).
28. K. S. Narendra and M. A. L.Thathachar, *Learning Automata : An Introduction*, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1989.
29. D. L. Neuhoff, The Viterbi algorithm as an aid in text recognition, *IEEE Trans. Information Theory*, 222-226 (1975).
30. N. J.Nilsson, *Principles of Artificial Intelligence*, Tioga Publishing, Palo Alto, California, (1980).
31. 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).
32. B. J. Oommen, Recognition of Noisy Subsequences Using Constrained Edit Distances, *IEEE Trans. on Pattern Anal. and Mach. Intel.*, PAMI-9:676-685 (1987).
33. B. J. Oommen and C. Fothergill, Fast Learning Automaton-Based Image Examination and Retrieval, *The Computer Journal*, 36-6:542-553, (1993).
34. B. J. Oommen and R. L. Kashyap, Symbolic Channel Modelling for Noisy Channels which Permit Arbitrary Noise Distributions, *Proc. of the 1993 International Symposium on Computer and Information Sciences*, Istanbul, Turkey, November, 492-499 (1993).
35. B. J. Oommen and D. C. Y. Ma, Deterministic Learning Automata Solutions to the Equipartitioning Problem in *IEEE Transactions on Computers*, 37-1:2-13 (1988).
36. B. J. Oommen and D. C. Y. Ma, Stochastic Automata Solutions to the Object Partitioning Problem, *The Computer Journal*, 35:A105-A120 (1992).
37. J. L. Peterson, Computer programs for detecting and correcting spelling errors, *Comm. Assoc. Comput. Mach.*, 23:676-687 (1980).
38. M. Regnier, A language approach to string searching evaluation, *Proceedings of CPM-92, The Third Annual Symposium on Combinatorial Pattern Matching*, 15-26 (1992).
39. D. Sankoff and J. B. Kruskal, *Time Warps,String Edits and Macromolecules: The Theory and practice of Sequence Comparison*, Addison-Wesley (1983).
40. R. Shinghal, and G. T. Toussaint, Experiments in text recognition with the modified Viterbi algorithm, *IEEE Trans on Pat. Anal. and Mach. Intel.*, 1:184-192 (1979).
41. S. Srihari, *Computer Text Recognition and Error Correction*, IEEE Computer Society Press, (1984).
42. W. Szpankowski, Probabilistic analysis of generalized suffix trees, *Proceedings of CPM-92, The Third Annual Symposium on Combinatorial Pattern Matching*, 1-14 (1992).
43. M. L.Tsetlin, *Automaton Theory and the Modelling of Biological Systems*, New York and London, Academic, (1973).
44. A. J. Viterbi, Error bounds for convolutional codes and an asymptotically optimal decoding algorithm, *IEEE Trans. on Information Theory*, 260-26 (1967).
45. R. A. Wagner and M. J. Fisher, The string to string correction problem, *J. Assoc. Comput. Mach.*, 21:168-173 (1974).
46. C. K. Wong and A. K. Chandra, Bounds for the string editing problem, *J. Assoc. Comput. Mach.*, 23:13-16 (1976).
47. C. T.Yu, M. D.Siu, D. Lam and F.Tai,Adaptive Clustering Schemes: General Framework, in *Proc. of the IEEE COMPSAC Conference*, 81-89 (1981).

APPENDIX
THE STRING TAXONOMY LEARNING AUTOMATON

PROCEDURE STLA_System

Input : The dictionary $H = \{X_1,...X_J\}$, to be partitioned into K sub-dictionaries.
θ, θ_2 parameters which are used to decide whether two strings are reckoned similar.
In our implementation $\theta_2 := \theta - 0.1$.
The increment to the probability parameter $\Delta\mu^*$. In our implementation $\Delta\mu^* := 0.05$.

Output : The system lists the J strings as they appear in the K sub-dictionaries and their associated states.

Notation: (i) ω_i is the state of the string X_i. It is an integer in [1..KN], where, if $(j-1)N + 1 \le \omega_i \le jN$, then string X_i is assigned to the sub-dictionary α_j.

(ii) **Z** is the list of strings whose adjacent elements $<z_k,z_{k+1}>$ (where k is odd) are reckoned to be similar.

Method

Initialize **Z** to be the empty list
Initialize Prob. parameter μ^* to zero
For each $<X_i,X_j>$

$M_{i,j} := \frac{2 \cdot LLCS(X_i,X_j)}{|X_i|+|X_j|}$ (*Build matrix of similarity measures *)

If $M_{i,j} \ge \theta$ **Then** (* Build list of similar string pairs *)
Concatenate X_i and X_j to **Z**
EndIf
EndFor

Randomly initialize ω_i for $1 \le i \le J$, to the boundary states of the sub-dictionaries, each having J/K strings
Initialize pointer to the Head of **Z**
Repeat
For X_i and X_j the next two elements of **Z Do** (* Process similar elements *)
If $((\omega_i \text{ div } N) = (\omega_j \text{ div } N))$ **Then** (*Reward partitioning *)
Reward(X_i,X_j)
Else (* Penalize partitioning *)
PenalizeSimilarStrings(X_i,X_j)
EndIf
EndFor
For all $U \in H$ **Do** (* Entering Phase II *)
Y := Representative string for current sub-dictionary of U
If $Sim(U,Y) < \theta_2$ **Then**
If (Random(0,1) < μ^*) **Then** (* Randomly move U or *)
PenalizeDissimilarStrings(U,Y) (* Y from current class*)
EndIf
EndIf
EndFor
$\mu^* := \mu^* + \Delta\mu^*$ (*Increment prob. parameter*)
Initialize pointer to the Head of **Z**
Until Satisfied

END PROCEDURE STLA_System

PROCEDURE Reward
Input : Indices of strings X_i and X_j to be rewarded.
Output : The new states of X_i and X_j.
Method
If ((ω_i mod N) $\neq$ 1) **Then** (* Move X_i towards the internal state *)
$\omega_i := \omega_i - 1$
EndIf
If ((ω_j mod N) $\neq$ 1) **Then** (* Move X_j towards the internal state *)
$\omega_j := \omega_j - 1$
Endif
END PROCEDURE Reward

PROCEDURE PenalizeSimilarStrings
Input : Indices of strings X_i and X_j to be penalized.
Output : The new states of X_i and X_j.
Method
If (((ω_i mod N) $\neq$ 0) **and** ((ω_j mod N) $\neq$ 0)) **Then** (* Both are in internal states *)
$\omega_i := \omega_i + 1$
$\omega_j := \omega_j + 1$
Else
If (ω_i mod N $\neq$ 0) **Then** (* X_i is in an internal state *)
$\omega_i := \omega_i + 1$ (* Update state of X_i *)
temp := ω_j (* Store the state of X_j *)
$\omega_j := (\omega_i$ DIV N) * N (* Move X_j to same group as X_i *)
t := index of an word in sub-dictionary of X_i
where $X_t \neq X_i$ and is closest to boundary state of ω_i
ω_t := temp (* Move X_t to the old state of X_j *)
Else
If (ω_j mod N) $\neq$ 0) **Then** (* X_i has to be moved *)
$\omega_j := \omega_j + 1$ (* Update state of X_j *)
EndIf
temp := ω_i (* Store the state of X_i *)
$\omega_i := (\omega_j$ DIV N) * N (* Move X_i to same group as X_j *)
t := index of an word in sub-dictionary of X_j
where $X_t \neq X_j$ and is closest to boundary state of ω_j
ω_t := temp (* Move X_t to the old state of X_i *)
EndIf
EndIf
END PROCEDURE PenalizeSimilarStrings

PROCEDURE PenalizeDissimilarStrings
Input : Indices of strings U and Y, the representative string for sub-dictionary chosen by U.
Output : The new states of U and Y. The original state of U is ω_U and of Y is ω_Y.
Method

```
If ((ω_U mod N) ≠ 0 ) and ( ω_Y mod N) ≠ 0 )Then      (* U & Y are in internal states *)
    ω_U := ω_U + 1
    ω_Y := ω_Y + 1
Else                                                  (* U or Y is in a boundary state *)
    If ( ω_U mod N) ≠ 0 ) Then                        (* Y is in a boundary state *)
        ω_U := ω_U + 1
        BestSimilarity := ∞
        For all the sub-dictionaries k other than the one chosen by U Do
            Y_k := Representative string for current sub-dictionary
            If Sim(U,Y_k) < BestSimilarity Then
                BestSimilarity := Sim(U,Y_k)
                BestSubDictionary := k                (*Sub-dictionary k is superior *)
            EndIf
        EndFor
        X_Sw := String Closest to boundary in sub-dictionary BestSubDictionary
        temp := ω_U                                   (* Store the state of U *)
        ω_U := (ω_Sw DIV N) * N                       (*Move U to same group as X_Sw*)
        ω_Sw := temp                                  (*Move X_Sw to old state of U *)
    Else
        If ( ω_Y mod N) ≠ 0 ) Then                    (* U is a boundary state *)
            ω_Y := ω_Y + 1
        EndIf
        BestSimilarity := ∞
        For all the sub-dictionaries k other than the one chosen by U Do
            Y_k := Representative string for current sub-dictionary
            If Sim(Y,Y_k) < BestSimilarity Then
                BestSimilarity := Sim(Y,Y_k)
                BestSubDictionary := k                (*Sub-dictionary k is superior *)
            EndIf
        EndFor
        X_Sw := String Closest to boundary in sub-dictionary BestSubDictionary
        temp := ω_Y                                   (* Store the state of Y *)
        ω_Y := (ω_Sw DIV N) * N                       (*Move Y to same group as X_Sw*)
        ω_Sw := temp                                  (*Move X_Sw to old state of Y *)
    EndIf
EndIf
END PROCEDURE PenalizeDissimilarStrings
```

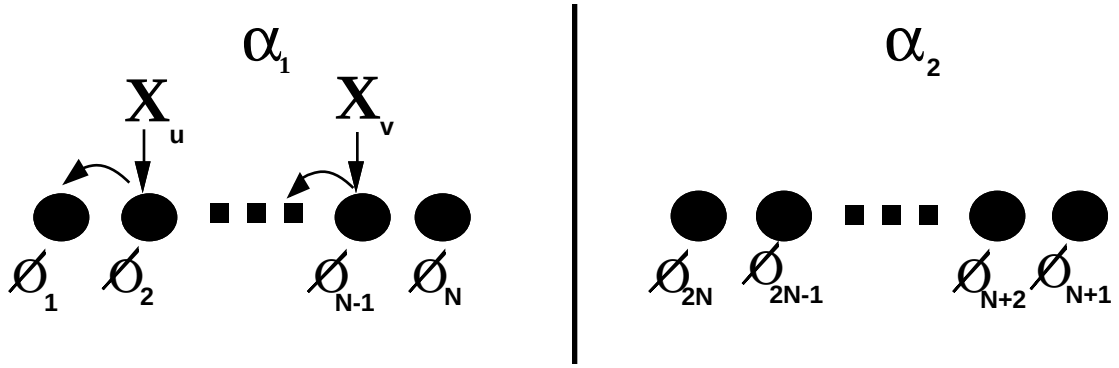

Figure I : Reward transitions for the 2N-State STLA. Here X_u and X_v are similar and located in the same sub-dictionary.

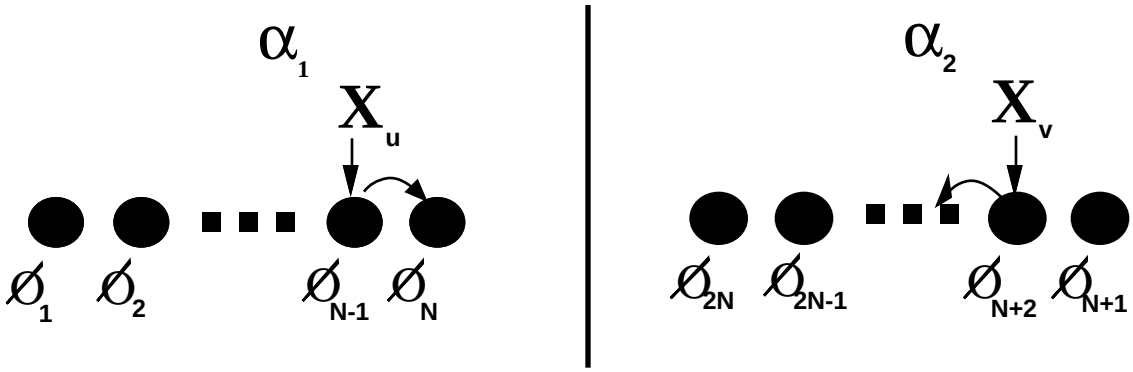

Figure IIa: Penalty transitions for the 2N-State STLA -- *PenalizeSimilarStrings* Mode. X_u and X_v are similar but located in the distinct sub-dictionaries. Neither of them is in a boundary state.

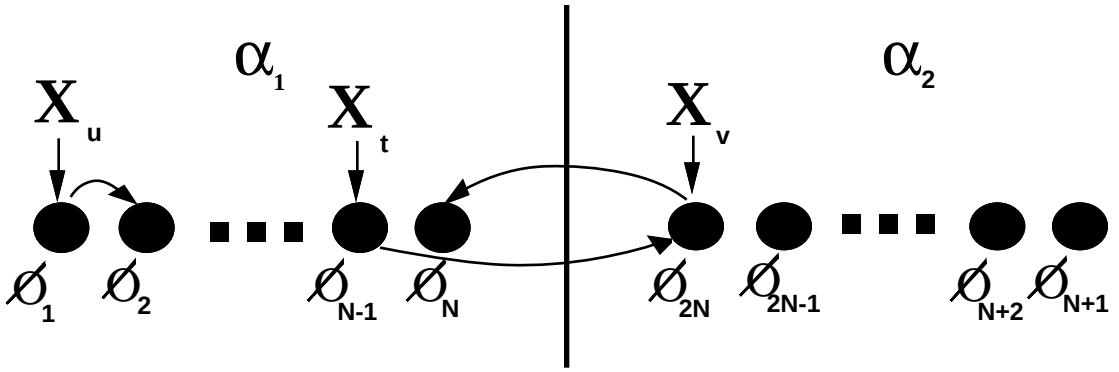

Figure IIb: Penalty transitions for the 2N-State STLA -- *PenalizeSimilarStrings* Mode. Here X_u and X_v are similar but located in the distinct sub-dictionaries. However of them (X_v) is in a boundary state.

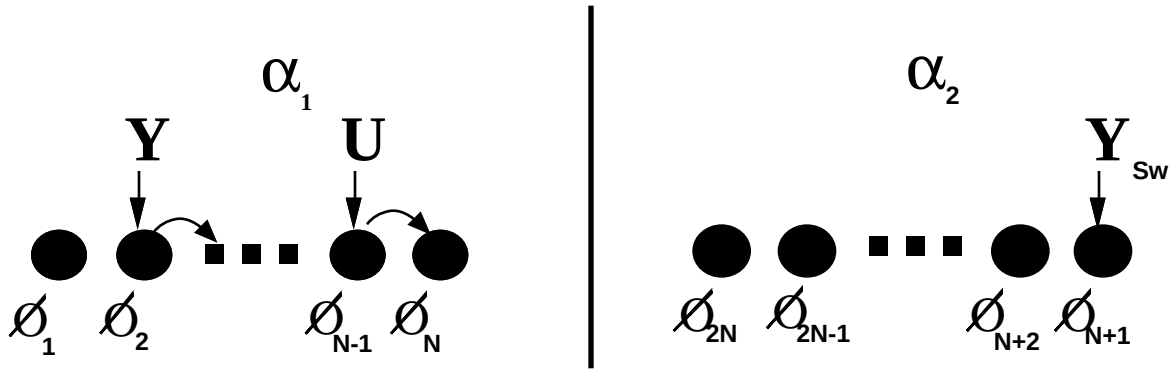

Figure IIIa: Penalty transitions for the 2N-State STLA -- *PenalizeDissimilarStrings* Mode. Here U is dissimilar to Y, the best representative of its current sub-dictionary. Neither of them is in a boundary state.

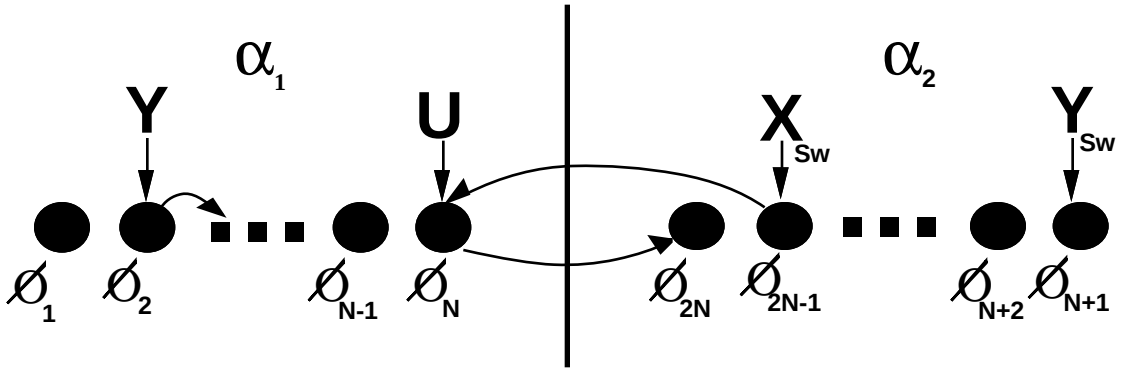

Figure IIIb: Penalty transitions for the 2N-State STLA -- *PenalizeDissimilarStrings* Mode. Here U is dissimilar to Y, the best representative of its current sub-dictionary and is in the boundary state. Y_{Sw} is the best representative of the sub-dictionary to which U should be migrated. X_{Sw}, the closest word here, and U swap sub-dictionaries.