Machine Learning, 25, 117–149 (1996)
© 1996 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

The Power of Amnesia: Learning Probabilistic Automata with Variable Memory Length

DANA RON danar@theory.lcs.mit.edu
Laboratory for Computer Science, MIT, Cambridge, MA 02139

YORAM SINGER singer@research.att.com
AT&T Labs, 600 Mountain Avenue, Murray Hill, NJ 07974

NAFTALI TISHBY tishby@cs.huji.ac.il
Institute of Computer Science, Hebrew University, Jerusalem 91904, Israel

Editor: Thomas Hancock

Abstract. We propose and analyze a distribution learning algorithm for variable memory length Markov processes. These processes can be described by a subclass of probabilistic finite automata which we name *Probabilistic Suffix Automata* (*PSA*). Though hardness results are known for learning distributions generated by general probabilistic automata, we prove that the algorithm we present can efficiently learn distributions generated by PSAs. In particular, we show that for any target PSA, the KL-divergence between the distribution generated by the target and the distribution generated by the hypothesis the learning algorithm outputs, can be made small with high confidence in polynomial time and sample complexity. The learning algorithm is motivated by applications in human-machine interaction. Here we present two applications of the algorithm. In the first one we apply the algorithm in order to construct a model of the English language, and use this model to correct corrupted text. In the second application we construct a simple stochastic model for E.*coli* DNA.

Keywords: Learning distributions, probabilistic automata, Markov models, suffix trees, text correction

1. Introduction

Statistical modeling of complex sequences is a fundamental goal of machine learning due to its wide variety of natural applications. The most noticeable examples of such applications are statistical models in human communication such as natural language, handwriting and speech (Jelinek, 1985, Nadas, 1984), and statistical models of biological sequences such as DNA and proteins (Krogh, Mian & Haussler, 1993).

These kinds of complex sequences clearly do not have any simple underlying statistical source since they are generated by natural sources. However, they typically exhibit the following statistical property, which we refer to as the *short memory* property. If we consider the (empirical) probability distribution on the next symbol given the preceding subsequence of some given length, then there exists a length L (the *memory length*) such that the conditional probability distribution does not change substantially if we condition it on preceding subsequences of length greater than L.

This observation lead Shannon, in his seminal paper (Shannon, 1951), to suggest modeling such sequences by Markov chains of order $L > 1$, where the order is the memory length of the model. Alternatively, such sequences may be modeled by Hidden Markov Models

(HMMs) which are more complex distribution generators and hence may capture additional properties of natural sequences. These statistical models define rich families of sequence distributions and moreover, they give efficient procedures both for generating sequences and for computing their probabilities. However, both models have severe drawbacks. The size of Markov chains grows exponentially with their order, and hence only very low order Markov chains can be considered in practical applications. Such low order Markov chains might be very poor approximators of the relevant sequences. In the case of HMMs, there are known hardness results concerning their learnability which we discuss in Section 1.1.

In this paper we propose a simple stochastic model and describe its learning algorithm. It has been observed that in many natural sequences, the memory length depends on the context and is *not fixed*. The model we suggest is hence a variant of order L Markov chains, in which the order, or equivalently, the memory, is variable. We describe this model using a subclass of Probabilistic Finite Automata (PFA), which we name *Probabilistic Suffix Automata* (PSA).

Each state in a PSA is labeled by a string over an alphabet Σ. The transition function between the states is defined based on these string labels, so that a walk on the underlying graph of the automaton, related to a given sequence, always ends in a state labeled by a suffix of the sequence. The lengths of the strings labeling the states are bounded by some upper bound L, but different states may be labeled by strings of different length, and are viewed as having *varying* memory length. When a PSA generates a sequence, the probability distribution on the next symbol generated is completely defined given the previously generated subsequence of length at most L. Hence, as mentioned above, the probability distributions these automata generate can be equivalently generated by Markov chains of order L, but the description using a PSA may be much more succinct. Since the size of order L markov chains is exponential in L, their estimation requires data length and time exponential in L.

In our learning model we assume that the learning algorithm is given a sample (consisting either of several sample sequences or of a single sample sequence) generated by an unknown target PSA M of some bounded size. The algorithm is required to output a hypothesis machine $\hat{M}$, which is not necessarily a PSA but which has the following properties. $\hat{M}$ can be used both to efficiently generate a distribution which is similar to the one generated by M, and given any sequence s, it can efficiently compute the probability assigned to s by this distribution.

Several measures of the quality of a hypothesis can be considered. Since we are mainly interested in models for statistical classification and pattern recognition, the most natural measure is the Kullback-Leibler (KL) divergence. Our results hold equally well for the variation (L_1) distance and other norms, which are upper bounded by the KL-divergence. Since the KL-divergence between Markov sources grows linearly with the length of the sequence, the appropriate measure is the KL-divergence per symbol. Therefore, we define an ϵ-good hypothesis to be an hypothesis which has at most ϵ KL-divergence per symbol to the target source.

In particular, the hypothesis our algorithm outputs, belongs to a class of probabilistic machines named Probabilistic Suffix Trees (PST). The learning algorithm grows such a suffix tree starting from a single root node, and adaptively adds nodes (strings) for which there is strong evidence in the sample that they significantly affect the prediction properties of the tree.

We show that every distribution generated by a PSA can equivalently be generated by a PST which is not much larger. The converse is not true in general. We can however characterize the family of PSTs for which the converse claim holds, and in general, it is always the case that for every PST there exists a not much larger PFA that generates an equivalent distribution. There are some contexts in which PSAs are preferable, and some in which PSTs are preferable, and therefore we use both representation in the paper. For example, PSAs are more efficient generators of distributions, and since they are probabilistic automata, their well defined state space and transition function can be exploited by dynamic programming algorithms which are used for solving many practical problems. In addition, there is a natural notion of the *stationary distribution* on the states of a PSA which PSTs lack. On the other hand, PSTs sometimes have more succinct representations than the equivalent PSAs, and there is a natural notion of growing them.

Stated formally, our main theoretical result is the following. If both a bound L, on the memory length of the target PSA, and a bound n, on the number of states the target PSA has, are known, then for every given $0 < \epsilon < 1$ and $0 < \delta < 1$, our learning algorithm outputs an ϵ-good hypothesis PST, with confidence $1 - \delta$, in time polynomial in L, n, $|\Sigma|$, $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$. Furthermore, such a hypothesis can be obtained from a *single* sample sequence if the sequence length is also polynomial in a parameter related to the rate in which the target machine converges to its stationary distribution. Despite an intractability result concerning the learnability of distributions generated by Probabilistic Finite Automata (Kearns, et al., 1994), that is described in Section 1.1, our restricted model can be learned in a PAC-like sense efficiently. This has not been shown so far for any of the more popular sequence modeling algorithms.

We present two applications of the learning algorithm. In the first application we apply the algorithm in order to construct a model of the English language, and use this model to correct corrupted text. In the second application we construct a simple stochastic model for E.*coli* DNA. Combined with a learning algorithm for a different subclass of probabilistic automata (Ron, Singer & Tishby, 1995), the algorithm presented here is part of a complete cursive handwriting recognition system (Singer & Tishby, 1995).

1.1. Related Work

The most powerful (and perhaps most popular) model used in modeling natural sequences is the Hidden Markov Model (HMM). A detailed tutorial on the theory of HMMs as well as selected applications in speech recognition is given by Rabiner (Rabiner, 1989). A commonly used procedure for learning an HMM from a given sample is a maximum likelihood parameter estimation procedure that is based on the *Baum-Welch* method (Baume, et al., 1970, Baume, 1972) (which is a special case of the EM (Expectation-Maximization) algorithm (Dempster, Laired & Rubin, 1977)). However, this algorithm is guaranteed to converge only to a *local* maximum, and thus we are not assured that the hypothesis it outputs can serve as a good approximation for the target distribution. One might hope that the problem can be overcome by improving the algorithm used or by finding a new approach. Unfortunately, there is strong evidence that the problem cannot be solved efficiently.

Abe and Warmuth (Abe & Warmuth, 1992) study the problem of *training* HMMs. The HMM training problem is the problem of approximating an arbitrary, unknown source distribution by distributions generated by HMMs. They prove that HMMs are *not* trainable in time polynomial in the alphabet size, unless $RP = NP$. Gillman and Sipser (Gillman & Sipser, 1994) study the problem of *exactly inferring* an (ergodic) HMM over a binary alphabet when the inference algorithm can query a *probability oracle* for the long-term probability of any binary string. They prove that inference is hard: any algorithm for inference must make exponentially many oracle calls. Their method is information theoretic and does not depend on separation assumptions for any complexity classes.

Natural simpler alternatives, which are often used as well, are order L Markov chains (Shannon, 1951, Good, 1969), also known as n-gram models. As noted earlier, the size of an order L Markov chain is exponential in L and hence, if we want to capture more than very short term memory dependencies in the sequences, of substantial length in the sequences, then these models are clearly not practical.

Höffgen (Höffgen, 1993) studies families of distributions related to the ones studied in this paper, but his algorithms depend exponentially and not polynomially on the order, or memory length, of the distributions. Freund *et. al.* (Freund, et al., 1993) point out that their result for learning *typical* deterministic finite automata from random walks without membership queries, can be extended to learning *typical* PFAs. Unfortunately, there is strong evidence indicating that the problem of learning *general* PFAs is hard. Kearns *et. al.* (Kearns, et al., 1994) show that PFAs are not efficiently learnable under the assumption that there is no efficient algorithm for learning noisy parity functions in the PAC model.

The machines used as our hypothesis representation, namely Probabilistic Suffix Trees (PSTs), were introduced (in a slightly different form) in (Rissanen, 1983) and have been used for other tasks such as universal data compression (Rissanen, 1983, Rissanen, 1986, Weinberger, Lempel & Ziv, 1982, Willems, Shtarkov & Tjalkens, 1993). Perhaps the strongest among these results (which has been brought to our attention after the completion of this work) and which is most tightly related to our result is (Willems, Shtarkov & Tjalkens, 1993). This paper describes an efficient sequential procedure for universal data compression for PSTs by using a larger model class. This algorithm can be viewed as a distribution learning algorithm but the hypothesis it produces is not a PST or a PSA and hence cannot be used for many applications. Willems *et. al.* show that their algorithm can be modified to give the minimum description length PST. However, in case the source generating the examples is a PST, they are able to show that this PST convergence only *in the limit* of infinite sequence length to that source.

Vitter and Krishnan (Vitter & Krishnan, 1991, Krishnan & Vitter, 1993) adapt a version of the Ziv-Lempel data compression algorithm (Ziv & Lempel, 1978) to get a page prefetching algorithm, where the sequence of page accesses is assumed to be generated by a PFA. They show that the page fault rate of their algorithm converges to the page fault rate of the best algorithm that has full knowledge of the source. This is true for almost all page access sequences (in the limit of the sequence length). Laird and Saul (Laird & Saul, 1994) describe a prediction algorithm which is similar in spirit to our algorithm and is based on the *Markov tree* or *Directed Acyclic Word Graph* approach which is used for data compression (Blumer, 1990). They do not analyze the correctnes of the algorithm formally, but present several applications of the algorithm.

1.2. Overview of the Paper

The paper is organized as follows. In Section 2 we give basic definitions and notation and describe the families of distributions studied in this paper, namely those generated by PSAs and those generated by PSTs. In Section 4 we discuss the relation between the above two families of distributions. In Section 5 the learning algorithm is described. Some of the proofs regarding the correctness of the learning algorithm are given in Section 6. Finally, we demonstrate the applicability of the algorithm by two illustrative examples in Section 7. In the first example we use our algorithm to learn the structure of natural English text, and use the resulting hypothesis for correcting corrupted text. In the second example we use our algorithm to build a simple stochastic model for E.*coli* DNA. The detailed proofs of the claims presented in Section 4 concerning the relation between PSAs and PSTs are provided in Appendices A and B. The more technical proofs and lemmas regarding the correctness of the learning algorithm are given in Appendix C.

2. Preliminaries

2.1. Basic Definitions and Notations

Let Σ be a finite alphabet. By Σ^* we denote the set of all possible strings over Σ. For any integer N, Σ^N denotes all strings of length N, and $\Sigma^{\leq N}$ denotes the set of all strings with length *at most* N. The empty string is denoted by e. For any string $s = s_1 \dots s_l$, $s_i \in \Sigma$, we use the following notations:

- The longest prefix of s different from s is denoted by $\mathit{prefix}(s) \overset{\text{def}}{=} s_1 s_2 \dots s_{l-1}$.

- The longest suffix of s different from s is denoted by $\mathit{suffix}(s) \overset{\text{def}}{=} s_2 \dots s_{l-1} s_l$.

- The set of all suffixes of s is denoted by $\mathit{Suffix}^*(s) \overset{\text{def}}{=} \{s_i \dots s_l \mid 1 \leq i \leq l\} \cup \{\mathrm{e}\}$. A string s' is a *proper* suffix of s, if it a suffix of s but is not s itself.

- Let s^1 and s^2 be two strings in Σ^*. If s^1 is a suffix of s^2 then we shall say that s^2 is a *suffix extension* of s^1.

- A set of strings S is called a *suffix free* set if $\forall s \in S$, $\mathit{Suffix}^*(s) \cap S = \{s\}$.

2.2. Probabilistic Finite Automata and Prediction Suffix Trees

2.2.1. Probabilistic Finite Automata

A *Probabilistic Finite Automaton (PFA)* M is a 5-tuple $(Q, \Sigma, \tau, \gamma, \pi)$, where Q is a finite set of *states*, Σ is a finite *alphabet*, $\tau : Q \times \Sigma \rightarrow Q$ is the *transition function*, $\gamma : Q \times \Sigma \rightarrow [0,1]$ is the *next symbol probability function*, and $\pi : Q \rightarrow [0,1]$ is the *initial probability distribution* over the starting states. The functions γ and π must satisfy the following conditions: for every $q \in Q$, $\sum_{\sigma\in\Sigma} \gamma(q,\sigma) = 1$, and $\sum_{q\in Q} \pi(q) = 1$. We assume that the transition function τ is defined on all states q and symbols σ for which $\gamma(q,\sigma) > 0$, and on no other state-symbol pairs. τ can be extended to be defined on $Q \times \Sigma^*$ as follows: $\tau(q, s_1s_2 \ldots s_l) = \tau(\tau(q, s_1 \ldots s_{l-1}), s_l) = \tau(\tau(q, \mathit{prefix}(s)), s_l)$.

A PFA M generates strings of infinite length, but we shall always discuss probability distributions induced on prefixes of these strings which have some specified finite length. If P_M is the probability distribution M defines on infinitely long strings, then P_M^N, for any $N \geq 0$, will denote the probability induced on strings of length N. We shall sometimes drop the superscript N, assuming that it is understood from the context. The probability that M generates a string $r = r_1r_2 \ldots r_N$ in Σ^N is

$$P_M^N(r) = \sum_{q^0\in Q} \pi(q^0) \prod_{i=1}^{N} \gamma(q^{i-1}, r_i) \ , \tag{1}$$

where $q^{i+1} = \tau(q^i, r_i)$.

2.2.2. Probabilistic Suffix Automata

We are interested in learning a subclass of PFAs which we name *Probabilistic Suffix Automata* (PSA). These automata have the following property. Each state in a PSA M is labeled by a string of finite length in Σ^*. The set of strings labeling the states is suffix free. For every two states $q^1, q^2 \in Q$ and for every symbol $\sigma \in \Sigma$, if $\tau(q^1,\sigma) = q^2$ and q^1 is labeled by a string s^1, then q^2 is labeled by a string s^2 which is a suffix of $s^1 \cdot \sigma$. In order that τ be well defined on a given set of strings S, not only must the set be suffix free, but it must also have the following property. For every string s in S labeling some state q, and every symbol σ for which $\gamma(q,\sigma) > 0$, there exists a string in S which is a suffix of $s\sigma$. For our convenience, from this point on, if q is a state in Q then q will also denote the string labeling that state.

We assume that the underlying graph of M, defined by Q and $\tau(\cdot,\cdot)$, is *strongly connected*, i.e., for every pair of states q and q' there is a directed path from q to q'. Note that in our definition of PFAs we assumed that the probability associated with each transition (edge in the underlying graph) is non-zero, and hence strong connectivity implies that every state can be reached from every other state with non-zero probability. For simplicity we assume M is *aperiodic*, i.e., that the greatest common divisor of the lengths of the cycles in its underlying graph is 1. These two assumptions ensure us that M is ergodic. Namely, there exists a distribution Π_M on the states such that for every state we may start at, the probability distribution on the state reached after time t as t grows to infinity, converges to Π_M. The probability distribution Π_M is the unique distribution satisfying

$$\Pi_M(q) = \sum_{q' \ s.t. \ \tau(q',\sigma)=q} \Pi_M(q')\gamma(q',\sigma) \ , \tag{2}$$

and is named the *stationary distribution* of M. We ask that for every state q in Q, the initial probability of q, $\pi(q)$, be the stationary probability of q, $\Pi_M(q)$. It should be noted that the assumptions above are needed only when learning from a single sample string and not when learning from many sample strings. However, for sake of brevity we make these requirements in both cases.

For any given $L \geq 0$, the subclass of PSAs in which each state is labeled by a string of length at most L is denoted by L-PSA. An example 2-PSA is depicted in Figure 1. A special case of these automata is the case in which Q includes *all* strings in Σ^L. An example of such a 2-PSA is depicted in Figure 1 as well. These automata can be described as *Markov chains of order L*. The states of the Markov chain are the symbols of the alphabet Σ, and the next state transition probability depends on the last L states (symbols) traversed. Since every L-PSA can be extended to a (possibly much larger) equivalent L-PSA whose states are labeled by all strings in Σ^L, it can always be described as a Markov chain of order L. Alternatively, since the states of an L-PSA might be labeled by only a small subset of $\Sigma^{\leq L}$, and many of the suffixes labeling the states may be much shorter than L, it can be viewed as a Markov chain with *variable order*, or *variable memory*.

Learning Markov chains of order L, i.e., L-PSAs whose states are labeled by *all* Σ^L strings, is straightforward (though it takes time exponential in L). Since the 'identity' of the states (i.e., the strings labeling the states) is known, and since the transition function τ is uniquely defined, learning such automata reduces to approximating the next symbol probability function γ. For the more general case of L-PSAs in which the states are labeled by strings of variable length, the task of an efficient learning algorithm is much more involved since it must reveal the identity of the states as well.

2.2.3. *Prediction Suffix Trees*

Though we are interested in learning PSAs, we choose as our hypothesis class the class of *prediction suffix trees* (PST) defined in this section. We later show (Section 4) that for every PSA there exists an equivalent PST of roughly the same size.

A PST T, over an alphabet Σ, is a tree of degree $|\Sigma|$. Each edge in the tree is labeled by a single symbol in Σ, such that from every internal node there is exactly one edge labeled by each symbol. The nodes of the tree are labeled by pairs (s, γ_s) where s is the string associated with the walk starting from that node and ending in the root of the tree, and $\gamma_s : \Sigma \rightarrow [0, 1]$ is the *next symbol probability function* related with s. We require that for every string s labeling a node in the tree, $\sum_{\sigma\in\Sigma} \gamma_s(\sigma) = 1$.

As in the case of PFAs, a PST T generates strings of infinite length, but we consider the probability distributions induced on finite length prefixes of these strings. The probability that T generates a string $r = r_1 r_2 \ldots r_N$ in Σ^N is

$$P_T^N(r) = \Pi_{i=1}^N \gamma_{s^{i-1}}(r_i) \ , \tag{3}$$

where $s^0 = e$, and for $1 \leq j \leq N-1$, s^j is the string labeling the deepest node reached by taking the walk corresponding to $r_i r_{i-1} \ldots r_1$ starting at the root of T. For example, using the PST depicted in Figure 1, the probability of generating the string 00101, is $0.5 \times 0.5 \times 0.25 \times 0.5 \times 0.75$, and the labels of the nodes that are used for the prediction are $s^0 = e$, $s^1 = 0$, $s^2 = 00$, $s^3 = 1$, $s^4 = 10$. In view of this definition, the requirement that every internal node have *exactly* $|\Sigma|$ sons may be loosened, by allowing the omission of nodes labeled by substrings which are generated by the tree with probability 0.

PSTs therefore generate probability distributions in a similar fashion to PSAs. As in the case of PSAs, symbols are generated sequentially and the probability of generating a symbol depends only on the previously generated substring of some bounded length. In both cases there is a simple procedure for determining this substring, as well as for determining the probability distribution on the next symbol conditioned on the substring. However, there are two (related) differences between PSAs and PSTs. The first is that PSAs generate each symbol simply by traversing a single edge from the current state to the next state, while for each symbol generated by a PST, one must walk down from the root of the tree, possibly traversing L edges. This implies that PSAs are more efficient generators. The second difference is that while in PSAs for each substring (state) and symbol, the next state is well defined, in PSTs this property does not necessarily hold. Namely, given the current generating node of a PST, and the next symbol generated, the next node is not necessarily uniquely defined, but might depend on previously generated symbols which are not included in the string associated with the current node. For example, assume we have a tree whose leaves are: `1,00,010,110` (see Figure B.1 in Appendix B). If `1` is the current generating leaf and it generates `0`, then the next generating leaf is either `010` or `110` depending on the symbol generated just prior to `1`.

PSTs, like PSAs, can always be described as Markov chains of (fixed) finite order, but as in the case of PSAs this description might be exponentially large.

We shall sometimes want to discuss only the structure of a PST and ignore its prediction property. In other words, we will be interested only in the string labels of the nodes and not in the values of $\gamma_s(\cdot)$. We refer to such trees as *suffix trees*. We now introduce two more notations. The set of leaves of a suffix tree T is denoted by $\mathcal{L}(T)$, and for a given string s labeling a node v in T, $T(s)$ denotes the subtree rooted at v.

3. The Learning Model

The learning model described in this paper is motivated by the PAC model for learning boolean concepts from labeled examples and is similar in spirit to that introduced in (Kearns, et al., 1994). We start by defining an ϵ-*good* hypothesis PST with respect to a given PSA.

Definition. Let M be a PSA and let T be a PST. Let P_M and P_T be the two probability distributions they generate respectively. We say that T is an **ϵ-good** hypothesis with respect

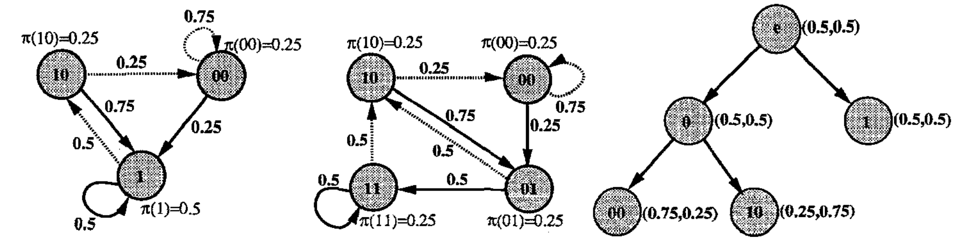

Figure 1. Left: A 2-PSA. The strings labeling the states are the suffixes corresponding to them. Bold edges denote transitions with the symbol '1', and dashed edges denote transitions with '0'. The transition probabilities are depicted on the edges. Middle: A 2-PSA whose states are labeled by *all* strings in $\{0, 1\}^2$. The strings labeling the states are the last two observed symbols before the state was reached, and hence it can be viewed as a representation of a Markov chain of order 2. Right: A prediction suffix tree. The prediction probabilities of the symbols '0' and '1', respectively, are depicted beside the nodes, in parentheses. The three models are equivalent in the sense that they induce the same probability distribution on strings from $\{0, 1\}^*$.

to M, if for every $N > 0$,

$$\frac{1}{N} D_{KL}[P_M^N || P_T^N] \le \epsilon \quad ,$$

where

$$D_{KL}[P_M^N || P_T^N] \stackrel{\text{def}}{=} \sum_{r \in \Sigma^N} P_M^N(r) \log \frac{P_M^N(r)}{P_T^N(r)}$$

is the *Kullback-Leibler* divergence between the two distributions.

In this definition we chose the *Kullback-Leibler* (KL) divergence as a distance measure between distributions. Similar definitions can be considered for other distance measures such as the variation and the quadratic distances. Note that the KL-divergence bounds the variation distance as follows (Cover & Thomas, 1991): $D_{KL}[P_1 || P_2] \ge \frac{1}{2} || P_1 - P_2 ||_1^2$. Since the L_1 norm bounds the L_2 norm, the last bound holds for the quadratic distance as well. Note that the KL-divergence between distributions, generated by finite order markov chains, is proportional to the length of the strings over which the divergence is computed, when this length is longer than the order of the model. Hence, to obtain a measure independent of that length it is necessary to divide the KL-divergence by the length of the strings, N.

A learning algorithm for PSAs is given the maximum length L of the strings labeling the states of the target PSA M, and an upper bound n on the number of states in M. The algorithm is also given a confidence (security) parameter $0 < \delta < 1$ and an approximation parameter $0 < \epsilon < 1$. We analyze the following two learning scenarios. In the first scenario the algorithm has access to a source of sample strings of minimal length $L + 1$, independently generated by M. In the second scenario it is given only a *single* (long) sample string generated by M. In both cases we require that it output a hypothesis PST $\hat{T}$, which with probability at least $1 - \delta$ is an ϵ-good hypothesis with respect to M.

The only drawback to having a PST as our hypothesis instead of a PSA (or more generally a PFA), is that the prediction procedure using a tree is somewhat less efficient (by at most a factor of L). Since no transition function is defined, in order to predict/generate each symbol, we must walk from the root until a leaf is reached. As mentioned earlier, we show in Appendix B that every PST can be transformed into an equivalent PFA which is not much larger. This PFA differs from a PSA only in the way it generates the first L symbols. We also show that if the PST has a certain property (defined in Appendix B), then it can be transformed into an equivalent PSA.

In order to measure the efficiency of the learning algorithm, we separate the case in which the algorithm is given a sample consisting of independently generated sample strings, from the case in which it is given a single sample string. In the first case we say that the learning algorithm is *efficient* if it runs in time polynomial in L, n, $|\Sigma|$, $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$. In order to define efficiency in the latter case we need to take into account an additional property of the model – its mixing or convergence rate. To do this we next discuss another parameter of PSAs (actually, of PFAs in general).

For a given PSA, M, let R_M denote the $n \times n$ stochastic transition matrix defined by $\tau(\cdot,\cdot)$ and $\gamma(\cdot,\cdot)$ when ignoring the transition labels. That is, if s^i and s^j are states in M and the last symbol in s^j is σ, then $R_M(s^i, s^j)$ is $\gamma(s^i, \sigma)$ if $\tau(s^i, \sigma) = s^j$, and 0 otherwise. Hence, R_M is the transition matrix of an ergodic Markov chain.

Let $\tilde{R}_M$ denote the *time reversal* of R_M. That is,

$$\tilde{R}_M(s^i, s^j) = \frac{\Pi_M(s^j) R_M(s^j, s^i)}{\Pi_M(s^i)} \quad ,$$

where Π_M is the stationary probability vector of R_M as defined in Equation (2). Define the *multiplicative reversiblization* U_M of M by $U_M = R_M \tilde{R}_M$. Denote the second largest eigenvalue of U_M by $\lambda_2(U_M)$.

If the learning algorithm receives a single sample string, we allow the length of the string (and hence the running time of the algorithm) to be polynomial not only in L, n, $|\Sigma|$, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$, but also in $1/(1 - \lambda_2(U_M))$. The rationale behind this is roughly the following. In order to succeed in learning a given PSA, we must observe each state whose stationary probability is non-negligible enough times so that the algorithm can identify that the state is significant, and so that the algorithm can compute (approximately) the next symbol probability function. When given several independently generated sample strings, we can easily bound the size of the sample needed by a polynomial in L, n, $|\Sigma|$, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$, using Chernoff bounds. When given one sample string, the given string must be long enough so as to ensure convergence of the probability of visiting a state to the stationary probability. We show that this convergence rate can be bounded using algebraic properties of U_M, namely, its second largest eigenvalue (Fill, 1991).

4. Emulation of PSAs by PSTs

In this section we show that for every PSA there exists an equivalent PST which is not much larger. This allows us to consider the PST equivalent to our target PSA, whenever it is convenient.

THEOREM 1 *For every L-PSA, $M = (Q, \Sigma, \tau, \gamma, \pi)$, there exists an equivalent PST T_M, of maximal depth L and at most $L \cdot |Q|$ nodes.*

Proof: (Sketch) We describe below the construction needed to prove the claim. The complete proof is provided in Appendix A.

Let T_M be the tree whose leaves correspond to the strings in Q. For each leaf s, and for every symbol σ, let $\gamma_s(\sigma) = \gamma(s, \sigma)$. This ensures that for every given string s which is a suffix extension of a leaf in T_M, and for every symbol σ, $P_M(\sigma|s) = P_{T_M}(\sigma|s)$. It remains to define the next symbol probability functions for the internal nodes of T_M. These functions must be defined so that T_M generates all strings related to its nodes with the same probability as M.

For each node s in the tree, let the *weight* of s, denoted by w_s, be $w_s \overset{\text{def}}{=} \sum_{s' \in Q,\, s \in \mathit{Suffix}^*(s')} \pi(s')$. In other words, the weight of a leaf in T_M is the stationary probability of the corresponding state in M; and the weight of an internal node labeled by a string s, equals the sum of the stationary probabilities over all states of which s is a suffix (which also equals the sum of the weights of the leaves in the subtree rooted at the node). Using the weights of the nodes we assign values to the γ_s's of the internal nodes s in the tree in the following manner. For every symbol σ let $\gamma_s(\sigma) = \sum_{s' \in Q,\, s \in \mathit{Suffix}^*(s')} \frac{w_{s'}}{w_s} \gamma(s', \sigma)$. The probability $\gamma_s(\sigma)$, of generating a symbol σ following a string s, *shorter* than any state in M, is thus a weighted average of $\gamma(s', \sigma)$ taken over all states s' which correspond to suffix extensions of s. The weight related with each state in this average, corresponds to its stationary probability. As an example, the probability distribution over the first symbol generated by T_M, is $\sum_{s \in Q} \pi(s)\gamma(s, \cdot)$. This probability distribution is equivalent, by definition, to the probability distribution over the first symbol generated by M.

Finally, if for some internal node in T_M, its next symbol probability function is equivalent to the next symbol probability functions of *all* of its descendants, then we remove all its descendants from the tree. □

An example of the construction described in the proof of Theorem 1 is illustrated in Figure 1. The PST on the right was constructed based on the PSA on the left, and is equivalent to it. Note that the next symbol probabilities related with the leaves and the internal nodes of the tree are as defined in the proof of the theorem.

5. The Learning Algorithm

We start with an overview of the algorithm. Let $M = (Q, \Sigma, \tau, \gamma, \pi)$ be the target L-PSA we would like to learn, and let $|Q| \leq n$. According to Theorem 1, there exists a PST T, of size bounded by $L \cdot |Q|$, which is equivalent to M. We use the sample statistics to define the *empirical probability function*, $\tilde{P}(\cdot)$, and using $\tilde{P}$, we construct a suffix tree, $\bar{T}$, which with high probability is a subtree of T. We define our hypothesis PST, $\hat{T}$, based on $\bar{T}$ and $\tilde{P}$,

The construction of $\bar{T}$ is done as follows. We start with a tree consisting of a single node (labeled by the empty string e) and add nodes which we have reason to believe should be in the tree. A node v labeled by a string s is added as a leaf to $\bar{T}$ if the following holds. The empirical probability of s, $\tilde{P}(s)$, is non-negligble, and for some symbol σ, the empirical probability of observing σ following s, namely $\tilde{P}(\sigma|s)$, differs substantially from the empirical probability of observing σ following $\mathit{suffix}(s)$, namely $\tilde{P}(\sigma|\mathit{suffix}(s))$. Note that $\mathit{suffix}(s)$ is the string labeling the parent node of v. Our decision rule for adding v, is thus dependent on the ratio between $\tilde{P}(\sigma|s)$ and $\tilde{P}(\sigma|\mathit{suffix}(s))$. We add a given node only when this ratio is substantially *greater* than 1. This suffices for our analysis (due to properties of the KL-divergence), and we need not add a node if the ratio is smaller than 1.

Thus, we would like to grow the tree level by level, adding the sons of a given leaf in the tree, only if they exhibit such a behavior in the sample, and stop growing the tree when the above is not true for any leaf. The problem is that the node might belong to the tree even though its next symbol probability function is equivalent to that of its parent node. The leaves of a PST must differ from their parents (or they are redundant) but internal nodes might not have this property. The PST depicted in Figure 1 illustrates this phenomena. In this example, $\gamma_0(\cdot) \equiv \gamma_e(\cdot)$, but both $\gamma_{00}(\cdot)$ and $\gamma_{10}(\cdot)$ differ from $\gamma_0(\cdot)$. Therefore, we must continue testing further potential descendants of the leaves in the tree up to depth L.

As mentioned before, we do not test strings which belong to branches whose empirical count in the sample is small. This way we avoid exponential grow-up in the number of strings tested. A similar type of *branch-and-bound* technique (with various bounding criteria) is applied in many algorithms which use trees as data structures (*cf.* (Kushilevitz & Mansour, 1993)). The set of strings tested at each step, denoted by $\bar{S}$, can be viewed as a kind of potential *frontier* of the growing tree $\bar{T}$, which is of bounded size. After the construction of $\bar{T}$ is completed, we define $\hat{T}$ by adding nodes so that all internal nodes have full degree, and defining the next symbol probability function for each node based on $\tilde{P}$. These probability functions are defined so that for every string s in the tree and for every symbol σ, $\gamma_s(\sigma)$ is bounded from below by γ_{min} which is a parameter that is set subsequently. This is done by using a conventional smoothing technique. Such a bound on $\gamma_s(\sigma)$ is needed in order to bound the KL-divergence between the target distribution and the distribution our hypothesis generates.

The above scheme follows a *top-down* approach since we start with a tree consisting of a single root node and a frontier consisting only of its children, and incrementally grow the suffix tree $\bar{T}$ and the frontier $\bar{S}$. Alternatively, a *bottom-up* procedure can be devised. In a bottom-up procedure we start by putting in $\bar{S}$ *all* strings of length at most L which have significant counts, and setting $\bar{T}$ to be the tree whose nodes correspond to the strings in $\bar{S}$. We then trim $\bar{T}$ starting from its leaves and proceeding up the tree by comparing the prediction probabilities of each node to its parent node as done in the top-down procedure. The two schemes are equivalent and yield the same prediction suffix tree. However, we find the incremental top-down approach somewhat more intuitive, and simpler to implement. Moreover, our top-down procedure can be easily adapted to an online setting which is useful in some practical applications.

Let P denote the probability distribution generated by M. We now formally define the *empirical probability function* $\tilde{P}$, based on a given sample generated by M. For a given string s, $\tilde{P}(s)$ is roughly the relative number of times s appears in the sample, and for any symbol σ, $\tilde{P}(\sigma|s)$ is roughly the relative number of times σ appears after s. We give a more precise definition below.

If the sample consists of one sample string r of length m, then for any string s of length at most L, define $\chi_j(s)$ to be 1 if $r_{j-|s|+1} \ldots r_j = s$ and 0 otherwise. Let

$$\tilde{P}(s) = \frac{1}{m-L} \sum_{j=L}^{m-1} \chi_j(s) \ , \tag{4}$$

and for any symbol σ, let

$$\tilde{P}(\sigma|s) = \frac{\sum_{j=L}^{m-1} \chi_{j+1}(s\sigma)}{\sum_{j=L}^{m-1} \chi_j(s)} \ . \tag{5}$$

If the sample consists of m' sample strings $r^1, \ldots, r^{m'}$, each of length $\ell \geq L+1$, then for any string s of length at most L, define $\chi_j^i(s)$ to be 1 if $r^i_{j-|s|+1} \cdots r^i_j = s$, and 0 otherwise. Let

$$\tilde{P}(s) = \frac{1}{m'(\ell - L)} \sum_{i=1}^{m'} \sum_{j=L}^{\ell-1} \chi_j^i(s) \ , \tag{6}$$

and for any symbol σ, let

$$\tilde{P}(\sigma|s) = \frac{\sum_{i=1}^{m'} \sum_{j=L}^{l-1} \chi_{j+1}(s\sigma)}{\sum_{i=1}^{m'} \sum_{j=L}^{l-1} \chi_j(s)} \ . \tag{7}$$

For simplicity we assume that all the sample strings have the same length and that this length is polynomial in n, L, and Σ. The case in which the sample strings are of different lengths can be treated similarly, and if the strings are too long then we can ignore parts of them.

In the course of the algorithm and in its analysis we refer to several parameters which are all simple functions of ϵ, n, L and $|\Sigma|$, and are set as follows:

$$\begin{aligned}
\epsilon_2 &= \frac{\epsilon}{48L} \ , \\
\gamma_{min} &= \frac{\epsilon_2}{|\Sigma|} = \frac{\epsilon}{48L|\Sigma|} \ , \\
\epsilon_0 &= \frac{\epsilon}{2nL\log(1/\gamma_{min})} = \frac{\epsilon}{2nL\log(48L|\Sigma|/\epsilon)} \ , \\
\epsilon_1 &= \frac{\epsilon_2}{8n\epsilon_0\gamma_{min}} = \frac{L\log(48L|\Sigma|/\epsilon)|\Sigma|}{4\epsilon} \ .
\end{aligned}$$

The size of the sample is set in the analysis of the algorithm.

A pseudo code describing the learning algorithm is given in Figure 2 and an illustrative run of the algorithm is depicted in Figure 3.

Algorithm Learn-PSA

```
1. Initialize $\bar{T}$ and $\bar{S}$: let $\bar{T}$ consist of a single root node (corresponding to e), and let
   $\bar{S} \leftarrow \{\sigma \mid \sigma \in \Sigma$ and $\tilde{P}(\sigma) \geq (1-\epsilon_1)\epsilon_0\}$.
2. While $\bar{S} \neq \emptyset$, pick any $s \in \bar{S}$ and do:
   (A) Remove $s$ from $\bar{S}$;
   (B) If there exists a symbol $\sigma \in \Sigma$ such that
           $\tilde{P}(\sigma|s) \geq (1+\epsilon_2)\gamma_{min}$ and $\tilde{P}(\sigma|s)/\tilde{P}(\sigma|suffix(s)) > 1 + 3\epsilon_2$ ,
       then add to $\bar{T}$ the node corresponding to $s$ and all the nodes on the path from the
       deepest node in $\bar{T}$ that is a suffix of $s$, to $s$;
   (C) If $|s| < L$ then for every $\sigma' \in \Sigma$, if $\tilde{P}(\sigma' \cdot s) \geq (1-\epsilon_1)\epsilon_0$, then add $\sigma' \cdot s$ to $\bar{S}$.
3. Initialize $\hat{T}$ to be $\bar{T}$.
4. Extend $\hat{T}$ by adding all missing sons of internal nodes.
5. For each $s$ labeling a node in $\hat{T}$, let
       $\hat{\gamma}_s(\sigma) = \tilde{P}(\sigma|s')(1 - |\Sigma|\gamma_{min}) + \gamma_{min}$ ,
   where $s'$ is the longest suffix of $s$ in $\bar{T}$.
```

Figure 2. Algorithm **Learn-PSA**

6. Analysis of the Learning Algorithm

In this section we state and prove our main theorem regarding the correctness and efficiency of the learning algorithm *Learn-PSA*, described in Section 5.

THEOREM 2 *For every target PSA M, and for every given security parameter $0 < \delta < 1$, and approximation parameter $0 < \epsilon < 1$, Algorithm* Learn-PSA *outputs a hypothesis PST, $\hat{T}$, such that with probability at least $1 - \delta$:*

1. *$\hat{T}$ is an ϵ-good hypothesis with respect to M.*

2. *The number of nodes in $\hat{T}$ is at most $|\Sigma| \cdot L$ times the number of states in M.*

If the algorithm has access to a source of independently generated sample strings, then its running time is polynomial in $L, n, |\Sigma|, \frac{1}{\epsilon}$ and $\frac{1}{\delta}$. If the algorithm has access to only one sample string, then its running time is polynomial in the same parameters and in $1/(1 - \lambda_2(U_M))$.

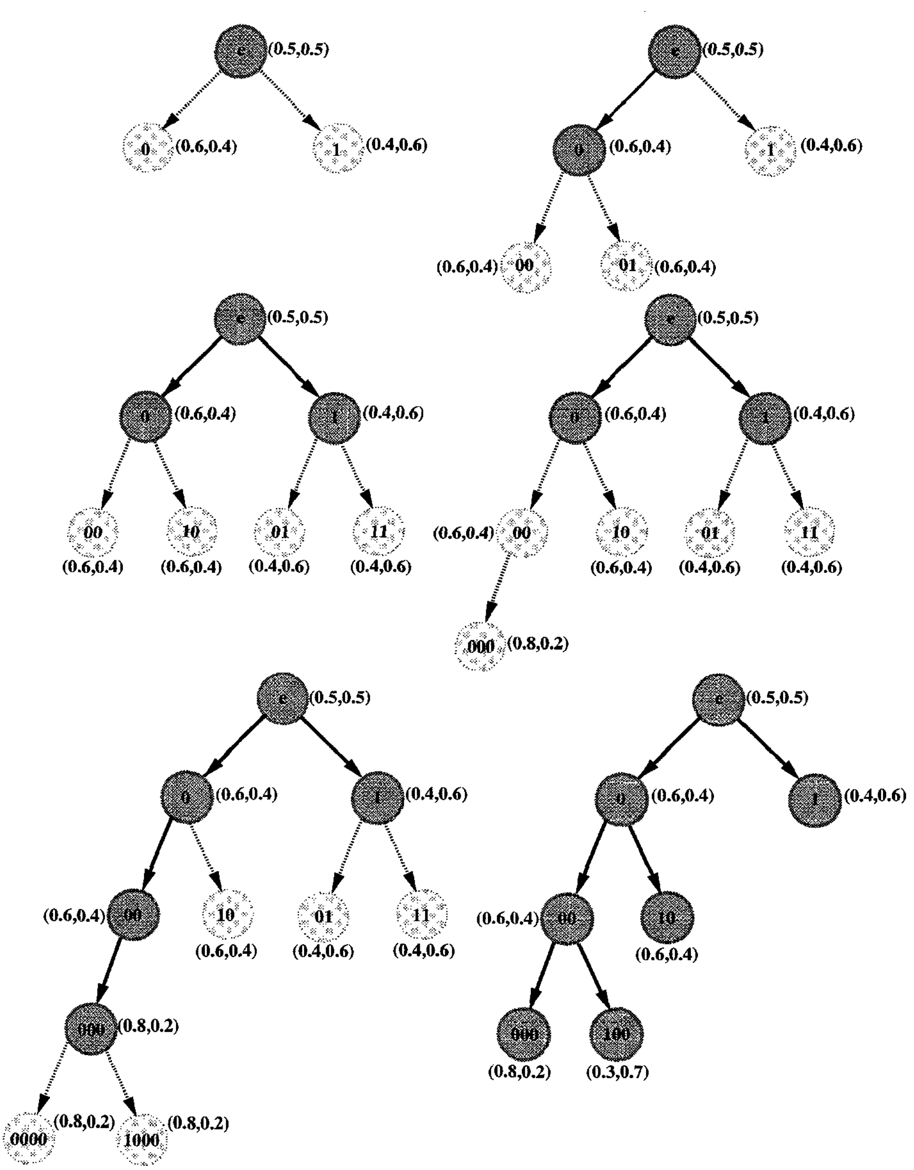

Figure 3. An illustrative run of the learning algorithm. The prediction suffix trees created along the run of the algorithm are depicted from left to right and top to bottom. At each stage of the run, the nodes from $\bar{T}$ are plotted in dark grey while the nodes from $\bar{S}$ are plotted in light grey. The alphabet is binary and the predictions of the next bit are depicted in parenthesis beside each node. The final tree is plotted on the bottom right part and was built by adding to $\bar{T}$ (bottom left) all missing children. Note that the node labeled by 100 was added to the final tree but is not part of any of the intermediate trees. This can happen when the probability of the string 100 is small.

In order to prove the theorem above we first show that with probability $1-\delta$, a large enough sample generated according to M is *typical* to M, where *typical* is defined subsequently. We then assume that our algorithm in fact receives a typical sample and prove Theorem 2 based on this assumption. Roughly speaking, a sample is typical if for every substring generated with non-negligible probability by M, the empirical counts of this substring and of the next symbol given this substring, are not far from the corresponding probabilities defined by M.

Definition. A sample generated according to M is **typical** if for every string $s \in \Sigma^{\leq L}$ the following two properties hold:

1. If $s \in Q$ then $|\tilde{P}(s) - \pi(s)| \leq \epsilon_1 \epsilon_0$;
2. If $\tilde{P}(s) \geq (1-\epsilon_1)\epsilon_0$ then for every $\sigma \in \Sigma$, $|\tilde{P}(\sigma|s) - P(\sigma|s)| \leq \epsilon_2 \gamma_{min}$;

Where ϵ_0, ϵ_1, ϵ_2, and γ_{min} were defined in Section 5.

LEMMA 1

1. *There exists a polynomial m'_0 in L, n, $|\Sigma|$, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$, such that the probability that a sample of $m' \geq m'_0(L, n, |\Sigma|, \frac{1}{\epsilon}, \frac{1}{\delta})$ strings each of length at least $L+1$ generated according to M is typical is at least* $1-\delta$.
2. *There exists a polynomial m_0 in L, n, $|\Sigma|$, $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, and $1/(1-\lambda_2(U_M))$, such that the probability that a single sample string of length $m \geq m_0(L, n, |\Sigma|, \frac{1}{\epsilon}, \frac{1}{\delta}, 1/(1-\lambda_2(U_M)))$ generated according to M is typical is at least* $1-\delta$.

The proof of Lemma 1 is provided in Appendix C.

Let T be the PST equivalent to the target PSA M, as defined in Theorem 1. In the next lemma we prove two claims. In the first claim we show that the prediction properties of our hypothesis PST $\hat{T}$, and of T, are similar. We use this in the proof of the first claim in Theorem 2, when showing that the KL-divergence per symbol between $\hat{T}$ and M is small. In the second claim we give a bound on the size of $\hat{T}$ in terms of T, which implies a similar relation between $\hat{T}$ and M (second claim in Theorem 2).

LEMMA 2 *If Learn-PSA is given a typical sample then:*

1. *For every string s in T, if $P(s) \geq \epsilon_0$ then $\dfrac{\gamma_s(\sigma)}{\hat{\gamma}_{s'}(\sigma)} \leq 1 + \epsilon/2$, where s' is the longest suffix of s corresponding to a node in $\hat{T}$.*
2. $|\hat{T}| \leq (|\Sigma| - 1) \cdot |T|$.

Proof: (Sketch, the complete proofs of both claims are provided in Appendix C.)

In order to prove the first claim, we argue that if the sample is typical, then there cannot exist such strings s and s' which falsify the claim. We prove this by assuming that there exists such a pair, and reaching contradiction. Based on our setting of the parameters ϵ_2

and γ_{min}, we show that for such a pair, s and s', the ratio between $\gamma_s(\sigma)$ and $\gamma_{s'}(\sigma)$ must be bounded from below by $1 + \epsilon/4$. If $s = s'$, then we have already reached a contradiction. If $s \neq s'$, then we can show that the algorithm must add some longer suffix of s to $\bar{T}$, contradicting the assumption that s' is the longest suffix of s corresponding to a node in $\hat{T}$. In order to bound the size of $\hat{T}$, we show that $\bar{T}$ is a subtree of T. This suffices to prove the second claim, since when transforming $\bar{T}$ into $\hat{T}$, we add at most all $|\Sigma| - 1$ siblings of every node in $\bar{T}$. We prove that $\bar{T}$ is a subtree of T, by arguing that in its construction, we did not add any string which does not correspond to a node in T. This follows from the decision rule according to which we add nodes to $\bar{T}$. □

Proof of Theorem 2: According to Lemma 1, with probability at least $1 - \delta$ our algorithm receives a typical sample. Thus according to the second claim in Lemma 2, $|\hat{T}| \leq (|\Sigma| - 1) \cdot |T|$ and since $|T| \leq L \cdot |Q|$, then $|\hat{T}| \leq |\Sigma| \cdot L \cdot |Q|$ and the second claim in the theorem is valid.

Let $r = r_1 r_2 \ldots r_N$, where $r_i \in \Sigma$, and for any prefix $r^{(i)}$ of r, where $r^{(i)} = r_1 \ldots r_i$, let $s[r^{(i)}]$ and $\hat{s}[r^{(i)}]$ denote the strings corresponding to the deepest nodes reached upon taking the walk $r_i \ldots r_1$ on T and $\hat{T}$ respectively. In particular, $s[r^{(0)}] = \hat{s}[r^{(0)}] = \mathrm{e}$. Let $\hat{P}$ denote the probability distribution generated by $\hat{T}$. Then

$$\frac{1}{N} \sum_{r \in \Sigma^N} P(r) \log \frac{P(r)}{\hat{P}(r)} = \tag{8a}$$

$$= \frac{1}{N} \sum_{r \in \Sigma^N} P(r) \cdot \log \frac{\prod_{i=1}^{N} \gamma_{s[r^{(i-1)}]}(r_i)}{\prod_{i=1}^{N} \hat{\gamma}_{\hat{s}[r^{(i-1)}]}(r_i)} \tag{8b}$$

$$= \frac{1}{N} \sum_{r \in \Sigma^N} P(r) \cdot \sum_{i=1}^{N} \log \frac{\gamma_{s[r^{(i-1)}]}(r_i)}{\hat{\gamma}_{\hat{s}[r^{(i-1)}]}(r_i)} \tag{8c}$$

$$= \frac{1}{N} \sum_{i=1}^{N} [\sum_{\substack{r \in \Sigma^N \text{ s.t.} \\ P(s[r^{(i-1)}]) < \epsilon_0}} P(r) \cdot \log \frac{\gamma_{s[r^{(i-1)}]}(r_i)}{\hat{\gamma}_{\hat{s}[r^{(i-1)}]}(r_i)}$$

$$+ \sum_{\substack{r \in \Sigma^N \text{ s.t.} \\ P(s[r^{(i-1)}]) \geq \epsilon_0}} P(r) \cdot \log \frac{\gamma_{s[r^{(i-1)}]}(r_i)}{\hat{\gamma}_{\hat{s}[r^{(i-1)}]}(r_i)}] \ . \tag{8d}$$

For every $1 \leq i \leq N$, the first term in the parenthesis in Equation (8d) can be bounded as follows. For each string r, the worst possible ratio between $\gamma_{s[r^{(i-1)}]}(r_i)$ and $\hat{\gamma}_{\hat{s}[r^{(i-1)}]}(r_i)$, is $1/\gamma_{min}$. The total weight of all strings in the first term equals the total weight of all the nodes in T whose weight is at most ϵ_0, which is at most $nL\epsilon_0$. The first term is thus bounded by $nL\epsilon_0 \log(1/\gamma_{min})$. Based on Lemma 2, the ratio between $\gamma_{s[r^{(i-1)}]}(r_i)$ and $\hat{\gamma}_{\hat{s}[r^{(i-1)}]}(r_i)$ for every string r in the second term in the parenthesis, is at most $1 + \epsilon/2$. Since the total weight of all these strings is bounded by 1, the second term is bounded by $\log(1 + \epsilon/2)$. Combining the above with the value of ϵ_0 (that was set in Section 5 to be $\epsilon/\left(2nL \log(1/\gamma_{min})\right)$), we get that,

$$\frac{1}{N} D_{KL}[P^N || \hat{P}^N] \leq \frac{1}{N} \cdot N \left[n\, L\, \epsilon_0 \log \frac{1}{\gamma_{min}} + \log(1+\epsilon/2) \right] \leq \epsilon \ . \qquad (9)$$

Using a straightforward implementation of the algorithm, we can get a (very rough) upper bound on the running time of the algorithm which is of the order of the square of the size of the sample times L. In this implementation, each time we add a string s to $\bar{S}$ or to $\bar{T}$, we perform a complete pass over the given sample to count the number of occurrences of s in the sample and its next symbol statistics. According to Lemma 1, this bound is polynomial in the relevant parameters, as required in the theorem statement. Using the following more time-efficient, but less space-efficient implementation, we can bound the running time of the algorithm by the size of the sample times L. For each string in $\bar{S}$, and each leaf in $\bar{T}$ we keep a set of pointers to all the occurrences of the string in the sample. For such a string s, if we want to test which of its extensions, σs should we add to $\bar{S}$ or to $\bar{T}$, we need only consider all occurrences of s in the sample (and then distribute them accordingly among the strings added). For each symbol in the sample there is a single pointer, and each pointer corresponds to a single string of length i for every $1 \leq i \leq L$. Thus the running time of the algorithm is of the order of the size of the sample times L.

■

7. Applications

A slightly modified version of our learning algorithm was applied and tested on various problems such as: correcting corrupted text, predicting DNA bases (Ron, Singer & Tishby, 1993), and part-of-speech disambiguation resolving (Schütze & Singer, 1994). We are still exploring other possible applications of the algorithm. Here we demonstrate how the algorithm can be used to correct corrupted text and how to build a simple model for DNA strands.

7.1. Correcting Corrupted Text

In many machine recognition systems such as speech or handwriting recognizers, the recognition scheme is divided into two almost independent stages. In the first stage a low-level model is used to perform a (stochastic) mapping from the observed data (e.g., the acoustic signal in speech recognition applications) into a high level alphabet. If the mapping is accurate then we get a correct sequence over the high level alphabet, which we assume belongs to a corresponding high level language. However, it is very common that errors in the mapping occur, and sequences in the high level language are corrupted. Much of the effort in building recognition systems is devoted to correct the corrupted sequences. In particular, in many optical and handwriting character recognition systems, the last stage employs natural-language analysis techniques to correct the corrupted sequences. This can be done after a good model of the high level language is learned from uncorrupted examples of sequences in the language. We now show how to use PSAs in order to perform such a task.

We applied the learning algorithm to the bible. The alphabet was the english letters and the blank character. We removed Jenesis and it served as a test set. The algorithm was applied to the rest of the books with $L = 30$, and the accuracy parameters (ϵ_i) were of order $O(\sqrt{N})$, where N is the length of the training data. This resulted in a PST having less than 3000 nodes. This PST was transformed into a PSA in order to apply an efficient text correction scheme which is described subsequently. The final automaton constitutes both of states that are of length 2, like `qu' and `xe', and of states which are 8 and 9 symbols long, like `shall be' and `there was'. This indicates that the algorithm really captures the notion of variable memory that is needed in order to have accurate predictions. Building a Markov chain of order L in this case is clearly not practical since it requires $|\Sigma|^L = 27^9 = 7625597484987$ states!

Let $\bar{r} = (r_1, r_2, \ldots, r_t)$ be the observed (corrupted) text. If an estimation of the corrupting noise probability is given, then we can calculate for each state sequence $\bar{q} = (q_0, q_1, q_2, \ldots, q_t)$, $q_i \in Q$, the probability that $\bar{r}$ was created by a walk over the PSA which constitutes of the states $\bar{q}$. For $0 \leq i \leq t$, let X_i be a random variable over Q, where $X_i = q$ denotes the event that the ith state passed was q. For $1 \leq i \leq t$ let Y_i be a random variable over Σ, where $Y_i = \sigma$ denotes the event that the ith symbol observed was σ. For $\bar{q} \in Q^{t+1}$, let $\bar{X} = \bar{q}$ denote the joint event that $X_i = q_i$ for every $0 \leq i \leq t$, and for $\bar{r} \in \Sigma^t$, let $\bar{Y} = \bar{r}$ denote the joint event that $Y_i = r_i$ for every $1 \leq i \leq t$. If we assume that the corrupting noise is i.i.d and is independent of the states that constitute the walk, then the most likely state sequence, $\bar{q}_{ML}$, is

$$\bar{q}_{ML} = \arg\max_{\bar{q}\in Q^{t+1}} P\left(\bar{X} = \bar{q} | \bar{Y} = \bar{r}\right) = \arg\max_{\bar{q}\in Q^{t+1}} P\left(\bar{Y} = \bar{r} | \bar{X} = \bar{q}\right) P(\bar{X} = \bar{q}) \quad (10a)$$

$$= \arg\max_{\bar{q}\in Q^{t+1}} \left\{ \left(\prod_{i=1}^{t} P\left(Y_i = r_i | \bar{X} = \bar{q}\right) \right) \times \left(\pi(q_0) \prod_{i=1}^{t} P\left(X_i = q_i | X_{i-1} = q_{i-1}\right) \right) \right\} \quad (10b)$$

$$= \arg\max_{\bar{q}\in Q^{t}} \left\{ \sum_{i=1}^{t} \log\left(P\left(Y_i = r_i | X_i = q_i\right) + \log(\pi(q_0)) + \sum_{i=1}^{t} \log\left(P\left(X_i = q_i | X_{i-1} = q_{i-1}\right)\right) \right\} , \quad (10c)$$

where for deriving the last Equality (10c) we used the monotonicity of the log function and the fact that the corruption noise is independent of the states. Let the string labeling q_i be $s_1, \ldots, s_l$. Then $P(Y_i = r_i | X_i = q_i)$ is the probability that r_i is an uncorrupted symbol if $r_i = s_l$, and is the probability that the noise process flipped s_l to be r_i otherwise. Note that the sum (10c) can be computed efficiently in a recursive manner. Moreover, the maximization of Equation (10a) can be performed efficiently by using a *dynamic programming* (DP) scheme (Bellman, 1957). This scheme requires $O(|Q| \times t)$ operations. If $|Q|$ is large, then approximation schemes to the optimal DP, such as the *stack decoding* algorithm (Jelinek, 1969) can be employed. Using similar methods it is also possible to correct errors when insertions and deletions of symbols occur as well.

We tested the algorithm by taking a text from Jenesis and corrupting it in two ways. First, we altered every letter (including blanks) with probability 0.2. In the second test we altered every letter with probability 0.1 and we also changed each blank character, in order to test whether the resulting model is powerful enough to cope with non-uniform noise. The result of the correction algorithm for both cases as well as the original and corrupted texts are depicted in Figure 4.

Original Text:

and god called the dry land earth and the gathering together of the waters called he seas and god saw that it was good and god said let the earth bring forth grass the herb yielding seed and the fruit tree yielding fruit after his kind

Corrupted text (1):

and god cavsed the drxjland earth ibd shg gathervng together oj the waters cflled re seas aed god saw thctpit was good ann god said let tae earth bring forth gjasb tse hemb yielpinl peed and thesfruit tree sielxing fzuitnafter his kind

Corrected text (1):

and god caused the dry land earth and she gathering together of the waters called he sees and god saw that it was good and god said let the earth bring forth grass the memb yielding peed and the fruit tree fielding fruit after his kind

Corrupted text (2):

andhgodpcilledjthesdryjlandbeasthcandmthelgatceringhlogetherjfytrezaatersoczlled
xherseasaknddgodbsawwthathitqwasoqoohanwzgodcsaidhletdtheuejrthriringmforth
hbgrasstthexherbyieldingzseedmazdctcybfruitttreeayieldinglfruztbafherihiskind

Corrected text (2):

and god called the dry land earth and the gathering together of the altars called he seasaked god saw that it was took and god said let the earthriring forth grass the herb yielding seed and thy fruit treescielding fruit after his kind

Figure 4. Correcting corrupted text.

We compared the performance of the PSA we constructed to the performance of Markov chains of order 0 – 3. The performance is measured by the negative log-likelihood obtained by the various models on the (uncorrupted) test data, normalized per observation symbol. The negative log-likelihood measures the amount of 'statistical surprise' induced by the model. The results are summarized in Table 1. The first four entries correspond to the Markov chains of order 0 – 3, and the last entry corresponds to the PSA. The order of the PSA is defined to be $\log_{|\Sigma|}(|Q|)$. These empirical results imply that using a PSA of reasonable size, we get a better model of the data than if we had used a much larger full order Markov chain.

Table 1. Comparison of full order Markov chains versus a PSA (a Markov model with variable memory).

	Fixed Order Markov				PSA
Model Order	0	1	2	3	1.84
Number of States	1	27	729	19683	432
Negative Log-Likelihood	0.853	0.681	0.560	0.555	0.456

*7.2. Building A Simple Model for E.*coli *DNA*

The DNA alphabet is composed of four nucleotides denoted by: A, C, T, G. DNA strands are composed of sequences of protein coding genes and fillers between those regions named intergenic regions. Locating the coding genes is necessary, prior to any further DNA analysis. Using manually segmented data of E. *coli* (Rudd, 1993) we built two different PSAs, one for the coding regions and one for the intergenic regions. We disregarded the internal (triplet) structure of the coding genes and the existence of start and stop codons at the beginning and the end of those regions. The models were constructed based on 250 different DNA strands from each type, their lengths ranging from 20 bases to several thousands. The PSAs built are rather small compared to the HMM model described in (Krogh, Mian & Haussler, 1993): the PSA that models the coding regions has 65 states and the PSA that models the intergenic regions has 81 states.

We tested the performance of the models by calculating the log-likelihood of the two models obtained on test data drawn from intergenic regions. In 90% of the cases the log-likelihood obtained by the PSA trained on intergenic regions was higher than the log-likelihood of the PSA trained on the coding regions. Misclassifications (when the log-likelihood obtained by the second model was higher) occurred only for sequences shorter than 100 bases. Moreover, the log-likelihood difference between the models scales linearly with the sequence length where the slope is close to the KL-divergence between the Markov models (which can be computed from the parameters of the two PSAs), as depicted in Figure 5. The main advantage of PSA models is in their simplicity. Moreover, the log-likelihood of a set of substrings of a given strand can be computed in time linear in the number of substrings. The latter property combined with the results mentioned above indicate that the PSA model might be used when performing tasks such as DNA gene locating. However, we should stress that we have done only a preliminary step in this direction and the results obtained in (Krogh, Mian & Haussler, 1993) as part of a complete parsing system are better.

Acknowledgments

We would like to thank Anders Krogh and David Haussler for letting us use their *E. coli* DNA data and for helpful discussions. Special thanks to Kenn Rudd for supplying the E. *coli* sequences used in the DNA experiments. We also would like to thank Ronitt Rubinfeld and Yoav Freund for their helpful comments. Thanks to Lee Giles for providing us with the software for plotting finite state machines. This research has been supported in part by a grant from the Israeli Ministry of Science and Arts and by the Bruno Goldberg endowment

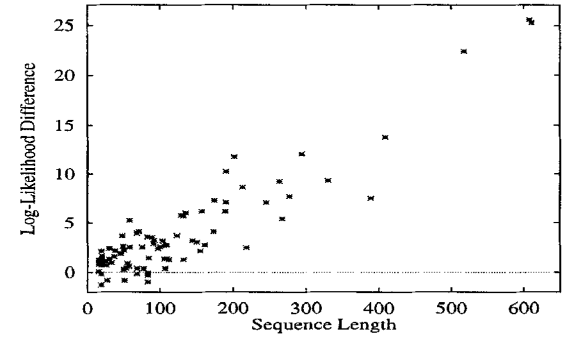

Figure 5. The difference between the log-likelihood induced by a PSA trained on data taken from intergenic regions and a PSA trained on data taken from coding regions. The test data was taken from intergenic regions. In 90% of the cases the likelihood of the first PSA was higher.

fund. Dana Ron would like to thank the support of the Eshkol fellowship. Yoram Singer would like to thank the Clore Foundation for its support.

Appendix A

Proof of Theorem 1

Theorem 1 *For every L-PSA $M = (Q, \Sigma, \tau, \gamma, \pi)$, there exists an equivalent PST T_M, of maximal depth L and at most $L \cdot |Q|$ nodes.*

Proof: Let T_M be the tree whose leaves correspond to the strings in Q (the states of M). For each leaf s, and for every symbol σ, let $\gamma_s(\sigma) = \gamma(s, \sigma)$. This ensures that for every string which is a suffix extension of some leaf in T_M, both M and T_M generate the next symbol with the same probability. The remainder of this proof is hence dedicated to defining the next symbol probability functions for the internal nodes of T_M. These functions must be defined so that T_M generates all strings related to nodes in T_M, with the same probability as M.

For each node s in the tree, let the *weight* of s, denoted by w_s, be defined as follows

$$w_s \stackrel{\text{def}}{=} \sum_{s' \in Q \text{ s.t. } s \in \mathit{Suffix}^*(s')} \pi(s') \tag{A.1}$$

In other words, the weight of a leaf in T_M is the stationary probability of the corresponding state in M; and the weight of an internal node labeled by a string s, equals the sum of the stationary probabilities over all states of which s is a suffix. Note that the weight of any internal node is the sum of the weights of all the leaves in its subtree, and in particular $w_e = 1$. Using the weights of the nodes we assign values to the γ_s's of the internal nodes s in the tree in the following manner. For every symbol σ let

$$\gamma_s(\sigma) = \sum_{s' \in Q \text{ s.t. } s \in \mathit{Suffix}^*(s')} \frac{w_{s'}}{w_s} \gamma(s', \sigma) \ . \tag{A.2}$$

According to the definition of the weights of the nodes, it is clear that for every node s, $\gamma_s(\cdot)$ is in fact a probability function on the next output symbol as required in the definition of prediction suffix trees.

What is the probability that M generates a string s which is a node in T_M (a suffix of a state in Q)? By definition of the transition function of M, for every $s^0 \in Q$, if $s' = \tau(s^0, s)$, then s' must be a suffix extension of s. Thus $P_M(s)$ is the sum over all such s' of the probability of reaching s', when s^0 is chosen according to the initial distribution $\pi(\cdot)$ on the starting states. But if the initial distribution is stationary then at any point the probability of being at state s' is just $\pi(s')$, and

$$P_M(s) \;=\; \sum_{s' \in Q \text{ s.t. } s \in \mathit{Suffix}^*(s')} \pi(s') \;=\; w_s \;. \tag{A.3}$$

We next prove that $P_{T_M}(s)$ equals w_s as well. We do this by showing that for every $s = s_1 \ldots s_l$ in the tree, where $|s| \geq 1$, $w_s = w_{\mathit{prefix}(s)} \gamma_{\mathit{prefix}(s)}(s_l)$. Since $w_e = 1$, it follows from a simple inductive argument that $P_{T_M}(s) = w_s$.

By our definition of PSAs, $\pi(\cdot)$ is such that for every $s \in Q$, $s = s_1 \ldots s_l$,

$$\pi(s) = \sum_{s' \text{ s.t. } \tau(s', s_l) = s} \pi(s') \gamma(s', s_l) \;. \tag{A.4}$$

Hence, if s is a leaf in T_M then

$$\begin{aligned} w_s \;=\; \pi(s) &\overset{(a)}{=} \sum_{s' \in \mathcal{L}(T_M) \text{ s.t. } s \in \mathit{Suffix}^*(s' s_l)} w_{s'} \gamma_{s'}(s_l) \\ &\overset{(b)}{=} \sum_{s' \in \mathcal{L}(T_M(\mathit{prefix}(s)))} w_{s'} \gamma_{s'}(s_l) \\ &\overset{(c)}{=} w_{\mathit{prefix}(s)} \gamma_{\mathit{prefix}(s)}(s_l) \;, \end{aligned} \tag{A.5}$$

where (a) follows by substituting $w_{s'}$ for $\pi(s')$ and $\gamma_{s'}(s_l)$ for $\gamma(s', s_l)$ in Equation (A.4), and by the definition of $\tau(\cdot, \cdot)$; (b) follows from our definition of the structure of prediction suffix trees; and (c) follows from our definition of the weights of internal nodes. Hence, if s is a leaf, $w_s = w_{\mathit{prefix}(s)} \gamma_{\mathit{prefix}(s)}(s_l)$ as required.

If s is an internal node then using the result above and Equation (A.2) we get that

$$\begin{aligned} w_s &= \sum_{s' \in \mathcal{L}(T_M(s))} w_{s'} \\ &= \sum_{s' \in \mathcal{L}(T_M(s))} w_{\mathit{prefix}(s')} \gamma_{\mathit{prefix}(s')}(s_l) \\ &= w_{\mathit{prefix}(s)} \gamma_{\mathit{prefix}(s)}(s_l) \;. \end{aligned} \tag{A.6}$$

It is left to show that the resulting tree is not bigger than L times the number of states in M. The number of leaves in T_M equals the number of states in M, i.e. $|\mathcal{L}(T)| = |Q|$. If every internal node in T_M is of full degree (i.e. the probability T_M generates any string labeling a leaf in the tree is strictly greater than 0) then the number of internal nodes is bounded by $|Q|$ and the total number of nodes is at most $2|Q|$. In particular, the above is true when for every state s in M, and every symbol σ, $\gamma(s, \sigma) > 0$. If this is not the case then we can simply bound the total number of nodes by $L \cdot |Q|$.

■

Appendix B
Emulation of PSTs by PFAs

In this section we show that for every PST there exists an equivalent PFA which is not much larger and which is a slight variant of a PSA. Furthermore, if the PST has a certain property, defined below and denoted by Property$*$, then it can be emulated by a PSA.

Property$*$ For every string s labeling a node in the tree, T,

$$P_T(s) = \sum_{\sigma \in \Sigma} P_T(\sigma s) \ .$$

Before we state our theorem, we observe that Property$*$ implies that for *every* string r,

$$P_T(r) = \sum_{\sigma \in \Sigma} P_T(\sigma r) \tag{B.1}$$

This is true for the following simple reasoning. If r is a node in T, then Equality (B.1) is equivalent to Property$*$. Otherwise let $r = r_1 r_2$, where r_1 is the longest prefix of r which is a leaf in T.

$$\begin{aligned}
P_T(r) &= P_T(r_1) \cdot P_T(r_2|r_1) && \text{(B.2a)}\\
&= \sum_\sigma P_T(\sigma r_1) \cdot P_T(r_2|r_1) && \text{(B.2b)}\\
&= \sum_\sigma P_T(\sigma r_1) \cdot P_T(r_2|\sigma r_1) && \text{(B.2c)}\\
&= \sum_\sigma P_T(\sigma r) \ , && \text{(B.2d)}
\end{aligned}$$

where Equality (B.2c) follows from the definition of PST's.

THEOREM 3 *For every PST, T, of depth L over Σ there exists an equivalent PFA, M_T, with at most $L \cdot |\mathcal{L}(T)|$ states. Furthermore, if Property$*$ holds for T, then it has an equivalent PSA.*

Proof: In the proof of Theorem 1, we were given a PSA M and we defined the equivalent suffix tree T_M to be the tree whose leaves correspond to the states of the automaton. Thus, given a suffix tree T, the natural dual procedure would be to construct a PSA M_T whose states correspond to the leaves of T. The first problem with this construction is that we might not be able to define the transition function τ on all pairs of states and symbols. That is, there might exist a state s and a symbol σ such that there is no state s' which is a suffix of $s\sigma$. The solution is to extend T to a larger tree T' (of which T is a subtree) such that τ is well defined on the leaves of T'. It can easily be verified that the following is an equivalent requirement on T': for each symbol σ, and for every leaf s in T', $s\sigma$ is either a leaf in the subtree $T'(\sigma)$ rooted at σ, or is a suffix extension of a leaf in $T'(\sigma)$. In this case we shall say that T' *covers* each of its children's subtrees. Viewing this in another way, for every leaf s, the longest *prefix* of s must be either a leaf or an internal node in T'. We thus obtain T' by adding nodes to T until the above property holds.

The next symbol probability functions of the nodes in T' are defined as follows. For every node s in $T \cap T'$ and for every $\sigma \in \Sigma$, let $\gamma'_s(\sigma) = \gamma_s(\sigma)$. For each new node $s' = s'_1 \ldots s'_l$ in $T' - T$, let $\gamma'_{s'}(\sigma) = \gamma_s(\sigma)$, where s is the longest suffix of s' in T (i.e. the deepest ancestor of s' in T). The probability distribution generated by T' is hence equivalent to that generated by T. From Equality (B.1) it directly follows that if Property$*$ holds for T, then it holds for T' as well.

Based on T' we now define $M_T = (Q, \Sigma, \tau, \gamma, \pi)$. If Property$*$ holds for T, then we define M_T as follows. Let the states of M_T be the leaves of T' and let the transition function be defined as usual for PSAs (i.e. for every state s and symbol σ, $\tau(s,\sigma)$ is the unique suffix of $s\sigma$.) Note that the number of states in M_T is at most L times the number of leaves in T, as required. This is true since for each original leaf in the tree T, at most $L-1$ prefixes might be added to T'. For each $s \in Q$ and for every $\sigma \in \Sigma$, let $\gamma(s,\sigma) = \gamma'_s(\sigma)$, and let $\pi(s) = P_T(s)$. It should be noted that M_T is not necessarily ergodic. It follows from this construction that for every string r which is a suffix extension of a leaf in T', and every symbol σ, $P_{M_T}(\sigma|r) = P_T(\sigma|r)$. It remains to show that for every string r which is a node in T', $P_{M_T}(r) = P_{T'}(r)$ $(= P_T(r))$. For a state $s \in Q$, let $P^s_{M_T}(r)$ denote the probability that r is generated assuming we start at state s. Then,

$$P_{M_T}(r) = \sum_{s\in Q} \pi(s) P^s_{M_T}(r) \tag{B.3a}$$

$$= \sum_{s\in Q} \pi(s) P_{M_T}(r|s) \tag{B.3b}$$

$$= \sum_{s\in \mathcal{L}(T')} P_{T'}(s) P_{T'}(r|s) \tag{B.3c}$$

$$= \sum_{s\in \mathcal{L}(T')} P_{T'}(sr) \tag{B.3d}$$

$$= P_{T'}(r) \ , \tag{B.3e}$$

where Equality (B.3b) follows from the definition of PSAs, Equality (B.3c) follows from our definition of $\pi(\cdot)$, and Equality (B.3e) follows from a series of applications of Equality (B.1).

If T does not have Property$*$, then we may not be able to define an initial distribution on the states of the PSA M_T such that for every string r which is a node in T', $P_{M_T}(r) = P_{T'}(r)$. We thus define a slight variant of M_T as follows. Let the states of M_T be the leaves of T' and *all their prefixes*, and let $\tau(\cdot,\cdot)$ be defined as follows: for every state s and symbol σ,

$\tau(s,\sigma)$ is the *longest* suffix of $s\sigma$. Thus, M_T has the structure of a *prefix* tree combined with a PSA. If we define $\gamma(\cdot,\cdot)$ as above, and let the empty string, $\mathtt{e}$, be the single starting state (i.e., $\pi(\mathtt{e}) = 1$), then, by definition, M_T is equivalent to T.

An illustration of the constructions described above is given in Figure B.1. ■

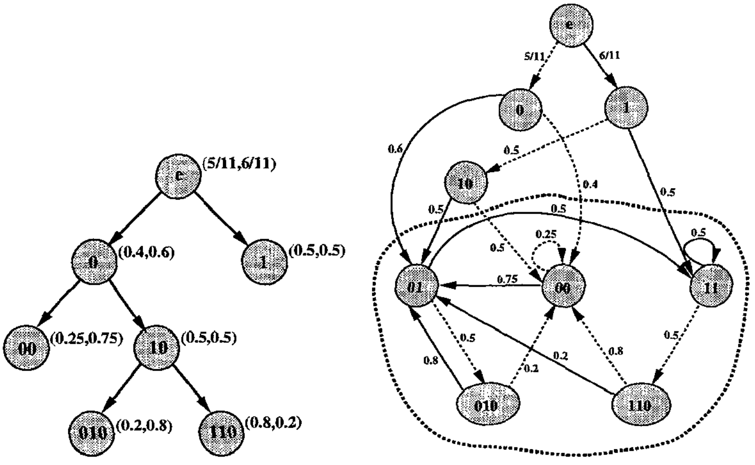

Figure B.1. Left: A Prediction suffix tree. The prediction probabilities of the symbols '0' and '1', respectively, are depicted beside the nodes, in parentheses. Right: The PFA that is equivalent to the PST on the left. Bold edges denote transitions with the symbol '1' and dashed edges denote transitions with '0'. Since Property$*$ holds for the PST, then it actually has an equivalent PSA which is defined by the circled part of the PFA. The initial probability distribution of this PSA is: $\pi(01) = 3/11, \pi(00) = 2/11, \pi(11) = 3/11, \pi(010) = 3/22, \pi(110) = 3/22$. Note that states '11' and '01' in the PSA replaced the node '1' in the tree.

Appendix C

Proofs of Lemma 1 and Lemma 2

Lemma 1

1. *There exists a polynomial* m_0' *in* L, n, $|\Sigma|$, $\frac{1}{\epsilon}$, *and* $\frac{1}{\delta}$, *such that the probability that a sample of* $m' \geq m_0'(L, n, |\Sigma|, \frac{1}{\epsilon}, \frac{1}{\delta})$ *strings each of length at least* $L+1$ *generated according to* M *is typical is at least* $1-\delta$.

2. *There exists a polynomial* m_0 *in* L, n, $|\Sigma|$, $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, *and* $1/(1-\lambda_2(U_M))$, *such that the probability that a single sample string of length* $m \geq m_0(L, n, |\Sigma|, \frac{1}{\epsilon}, \frac{1}{\delta}, 1/(1-\lambda_2(U_M)))$ *generated according to* M *is typical is at least* $1-\delta$.

Proof: Before proving the lemma we would like to recall that the parameters ϵ_0, ϵ_1, ϵ_2, and γ_{min}, are all polynomial functions of $1/\epsilon$, n, L, and $|\Sigma|$, and were defined in Section 5.

Several sample strings We start with obtaining a lower bound for m', so that the first property of a typical sample holds. Since the sample strings are generated independently, we may view $\tilde{P}(s)$, for a given state s, as the average value of m' independent random variables. Each of these variables is in the range $[0, 1]$ and its expected value is $\pi(s)$. Using Hoeffding's inequality we get that if $m' \geq \frac{1}{2\epsilon_1^2\epsilon_0^2} \ln \frac{4n}{\delta}$, then with probability at least $1 - \frac{\delta}{2n}$, $|\tilde{P}(s) - \pi(s)| \leq \epsilon_1\epsilon_0$. The probability that this inequality holds for every state is hence at least $1 - \frac{\delta}{2}$.

We would like to point out that since our only assumptions on the sample strings are that they are generated independently, and that their length is at least $L + 1$, we use only the independence between the different strings when bounding our error. We do not assume anything about the random variables related to $\tilde{P}(s)$ when restricted to any one sample string, other than that their expected value is $\pi(s)$. If the strings are known to be longer, then a more careful analysis can be applied as described subsequently for the case of a single sample string.

We now show that for an appropriate m' the second property holds with probability at least $1 - \frac{\delta}{2}$ as well. Let s be a string in $\Sigma^{\leq L}$. In the following lines, when we refer to *appearances* of s in the sample we mean in the sense defined by $\tilde{P}$. That is, we count only appearances of s which end at the Lth or greater symbol of a sample string. For the ith appearance of s in the sample and for every symbol σ, let $X_i(\sigma|s)$ be a random variable which is 1 if σ appears after the ith appearance of s and 0 otherwise. If s is either a state or a suffix extension of a state, then for every σ, the random variables $\{X_i(\sigma|s)\}$ are independent 0/1 random variables with expected value $P(\sigma|s)$. Let N_s be the total number of times s appears in the sample, and let $N_{min} = \frac{2}{\epsilon_2^2\gamma_{min}^2} \ln \frac{4|\Sigma|n}{\epsilon_0\delta}$. If $N_s \geq N_{min}$, then with probability at least $1 - \frac{\delta\epsilon_0}{2n}$, for every symbol σ, $|\tilde{P}(\sigma|s) - P(\sigma|s)| \leq \frac{1}{2}\epsilon_2\gamma_{min}$. If s is a suffix of several states $s^1, \ldots, s^k$, then for every symbol σ,

$$P(\sigma|s) = \sum_{i=1}^{k} \frac{\pi(s^i)}{P(s)} P(\sigma|s^i) , \tag{C.1}$$

(where $P(s) = \sum_{i=1}^{k} \pi(s^i)$) and

$$\tilde{P}(\sigma|s) = \sum_{i=1}^{k} \frac{\tilde{P}(s^i)}{\tilde{P}(s)} \tilde{P}(\sigma|s^i) . \tag{C.2}$$

Recall that $\epsilon_1 = (\epsilon_2\gamma_{min})/(8n\epsilon_0)$. If:
(1) for every state s^i, $|\tilde{P}(s^i) - \pi(s^i)| \leq \epsilon_1\epsilon_0$;
(2) for each s^i satisfying $\pi(s^i) \geq 2\epsilon_1\epsilon_0$, $|\tilde{P}(\sigma|s^i) - P(\sigma|s^i)| \leq \frac{1}{2}\epsilon_2\gamma_{min}$ for every σ;
then $|\tilde{P}(\sigma|s) - P(\sigma|s)| \leq \epsilon_2\gamma_{min}$, as required.

If the sample has the first property required of a typical sample (i.e., $\forall s \in Q$, $|\tilde{P}(s) - P(s)| \leq \epsilon_1\epsilon_0$), and for every state s such that $\tilde{P}(s) \geq \epsilon_1\epsilon_0$, $N_s \geq N_{min}$, then with probability at least $1 - \frac{\delta}{4}$ the second property of a typical sample holds for all strings which are either states or suffixes of states. If for every string s which is a suffix extension a state such that $\tilde{P}(s) \geq (1 - \epsilon_1)\epsilon_0$, $N_s \geq N_{min}$, then for all such strings the second property holds with probability at least $1 - \frac{\delta}{4}$ as well. Putting together all the bounds above, if $m' \geq \frac{1}{2\epsilon_1^2\epsilon_0^2} \ln \frac{4n}{\delta} + N_{min}/(\epsilon_1\epsilon_0)$, then with probability at least $1 - \delta$ the sample is typical.

A single sample string In this case the analysis is somewhat more involved. We view our sample string generated according to M as a walk on the markov chain described by R_M (defined in Subsection 3). We may assume that the starting state is visible as well since its contribution to $\tilde{P}(\cdot)$ is negligible. We shall need the following theorem from (Fill, 1991) which gives bounds on the convergence rate to the stationary distribution of general ergodic Markov chains. This theorem is partially based on a work by Mihail (Mihail, 1989), who gives bounds on the convergence in terms of combinatorial properties of the chain.

Markov Chain Convergence Theorem (Fill, 1991) *For any state s_0 in the Markov chain R_M, let $R_M^t(s_0, \cdot)$ denote the probability distribution over the states in R_M, after taking a walk of length t starting from state s_0. Then*

$$\left(\sum_{s \in Q} |R_M^t(s_0, s) - \pi(s)|\right)^2 \leq \frac{(\lambda_2(U_M))^t}{\pi(s_0)} \; .$$

First note that by simply applying Markov's inequality, we get that with probability at least $1 - \frac{\delta}{2n}$, $|\tilde{P}(s) - \pi(s)| \leq \epsilon_1\epsilon_0$, for each state s such that $\pi(s) < (\delta\epsilon_1\epsilon_0)/(2n)$. It thus remains to obtain a lower bound on m, so that the same is true for each s such that $\pi(s) \geq (\delta\epsilon_1\epsilon_0)/(2n)$. We do this by bounding the variance of the random variable related with $\tilde{P}(s)$, and applying Chebishev's Inequality.

Let

$$t_0 = \frac{\ln\left(n^3/\, 32\delta^3\epsilon_0^5\epsilon_1^5\right)}{\ln\left(1/\lambda_2(U_M)\right)} \; . \tag{C.3}$$

We next show that for every s satisfying $\pi(s) \geq (\delta\epsilon_1\epsilon_0)/(2n)$, $|R_M^{t_0}(s, s) - \pi(s)| \leq \frac{\delta}{4n}\epsilon_1^2\epsilon_0^2$. By the theorem above and our assumption on $\pi(s)$,

$$\begin{aligned}
\left(R_M^{t_0}(s, s) - \pi(s)\right)^2 &\leq \left(\sum_{s' \in Q} |R_M^{t_0}(s, s') - \pi(s')|\right)^2 && \text{(C.4a)}\\
&\leq \frac{(\lambda_2(U_M))^{t_0}}{\pi(s)} && \text{(C.4b)}\\
&\leq \frac{2n}{\delta\epsilon_0\epsilon_1}(\lambda_2(U_M))^{t_0} && \text{(C.4c)}\\
&= \frac{2n}{\delta\epsilon_0\epsilon_1}e^{-t_0\ln(1/\lambda_2(U_M))} && \text{(C.4d)}\\
&= \frac{\delta^2\epsilon_1^4\epsilon_0^4}{16n^2} \; . && \text{(C.4e)}
\end{aligned}$$

Therefore, $|R_M^t(s, s) - \pi(s)| \leq \frac{\delta}{4n}\epsilon_1^2\epsilon_0^2$.

Intuitively, this means that for every two integers, $t > t_0$, and $i \le t - t_0$, the event that s is the $(i + t_0)$th state passed on a walk of length t, is 'almost independent' of the event that s is the ith state passed on the same walk.

For a given state s, satisfying $\pi(s) \ge (\delta\epsilon_1\epsilon_0)/(2n)$, let X_i be a 0/1 random variable which is 1 *iff* s is the ith state on a walk of length t, and $Y = \sum_{i=1}^{t} X_i$. By our definition of $\tilde{P}$, in the case of a single sample string, $\tilde{P}(s) = Y/t$, where $t = m - L - 1$. Clearly $E(Y/t) = \pi(s)$, and for every i, $Var(X_i) = \pi(s) - \pi^2(s)$. We next bound $Var(Y/t)$.

$$Var\left(\frac{Y}{t}\right) = \frac{1}{t^2}Var\left(\sum_{i=1}^{t} X_i\right) \tag{C.5a}$$

$$= \frac{1}{t^2}\left(\sum_{i,j} E(X_iX_j) - \sum_{i,j} E(X_i)E(X_j)\right) \tag{C.5b}$$

$$= \frac{1}{t^2}\left(\sum_{i,j \text{ s.t. } |i-j|<t_0} E(X_iX_j) + \sum_{i,j \text{ s.t. } |i-j|\ge t_0} E(X_iX_j)\right) - \pi^2(s) \tag{C.5c}$$

$$\le \frac{2t_0}{t}\pi(s) + \frac{\delta}{4n}\epsilon_1^2\epsilon_0^2\pi(s) - \pi^2(s) \ . \tag{C.5d}$$

If we pick t to be greater than $(4nt_0)/(\delta\epsilon_1^2\epsilon_0^2)$, then $Var(Y/t) < \frac{\delta}{2n}\epsilon_1^2\epsilon_0^2$, and using Chebishev's Inequality $Pr[|Y/t - \pi(s)| > \epsilon_1\epsilon_0] < \frac{\delta}{2n}$. The probability the above holds for any s is at most $\frac{\delta}{2}$. The analysis of the second property required of a typical sample is identical to that described in the case of a sample consisting of many strings. ■

Lemma 2 *If Learn-PSA is given a typical sample then:*

1. *For every string s in T, if $P(s) \ge \epsilon_0$ then $\frac{\gamma_s(\sigma)}{\hat{\gamma}_{s'}(\sigma)} \le 1 + \epsilon/2$, where s' is the longest suffix of s corresponding to a node in $\hat{T}$.*

2. $|\hat{T}| \le (|\Sigma| - 1) \cdot |T|$.

Proof:

<u>1st Claim</u> Assume contrary to the claim that there exists a string labeling a node s in T such that $P(s) \ge \epsilon_0$ and for some $\sigma \in \Sigma$

$$\frac{\gamma_s(\sigma)}{\hat{\gamma}_{s'}(\sigma)} > 1 + \epsilon/2, \tag{C.6}$$

where s' is the longest suffix of s in $\hat{T}$. For simplicity of the presentation, let us assume that there is a node labeled by s' in $\bar{T}$. If this is not the case ($suffix(s')$ is an internal node in $\bar{T}$, whose son s' is missing), the analysis is very similar. If $s \equiv s'$ then we easily show below that our counter assumption is false. If s' is a proper suffix of s then we prove the following. If the counter assumption is true, then we added to $\bar{T}$ a (not necessarily proper) suffix of s which is longer than s'. This contradicts the fact that s' is the longest suffix of s in $\hat{T}$.

We first achieve a lower bound on the ratio between the two true next symbol probabilities, $\gamma_s(\sigma)$ and $\gamma_{s'}(\sigma)$. According to our definition of $\hat{\gamma}_{s'}(\cdot)$,

$$\hat{\gamma}_{s'}(\sigma) \geq (1 - |\Sigma|\gamma_{min})\tilde{P}(\sigma|s') \ . \tag{C.7}$$

We analyze separately the case in which $\gamma_{s'}(\sigma) \geq \gamma_{min}$, and the case in which $\gamma_{s'}(\sigma) < \gamma_{min}$. Recall that $\gamma_{min} = \epsilon_2/|\Sigma|$. If $\gamma_{s'}(\sigma) \geq \gamma_{min}$, then

$$\frac{\gamma_s(\sigma)}{\gamma_{s'}(\sigma)} \geq \frac{\gamma_s(\sigma)}{\tilde{P}(\sigma|s')} \cdot (1-\epsilon_2) \tag{C.8a}$$

$$\geq \frac{\gamma_s(\sigma)}{\hat{\gamma}_{s'}(\sigma)} \cdot (1-\epsilon_2)(1-|\Sigma|\gamma_{min}) \tag{C.8b}$$

$$> (1+\frac{\epsilon}{2})(1-\epsilon_2)^2 \ , \tag{C.8c}$$

where Inequality (C.8a) follows from our assumption that the sample is typical, Inequality (C.8b) follows from our definition of $\hat{\gamma}_{s'}(\sigma)$, and Inequality (C.8c) follows from the counter assumption (C.6), and our choice of γ_{min}. Since $\epsilon_2 < \epsilon/12$, and $\epsilon < 1$ then we get that

$$\frac{\gamma_s(\sigma)}{\gamma_{s'}(\sigma)} > 1 + \frac{\epsilon}{4} \ . \tag{C.9}$$

If $\gamma_{s'}(\sigma) < \gamma_{min}$, then $\hat{\gamma}_{s'}(\sigma) \geq \gamma_{s'}(\sigma)$, since $\hat{\gamma}_{s'}(\sigma)$ is defined to be at least γ_{min}. Therefore,

$$\frac{\gamma_s(\sigma)}{\gamma_{s'}(\sigma)} \geq \frac{\gamma_s(\sigma)}{\hat{\gamma}_{s'}(\sigma)} > 1 + \frac{\epsilon}{2} > 1 + \frac{\epsilon}{4} \tag{C.10}$$

as well. If $s \equiv s'$ then the counter assumption (C.6) is evidently false, and we must only address the case in which $s \not\equiv s'$, i.e., s' is a proper suffix of s.

Let $s = s_1 s_2 \ldots s_l$, and let s' be $s_i \ldots s_l$, for some $2 \leq i \leq l$. We now show that if the counter assumption (C.6) is true, then there exists an index $1 \leq j < i$ such that $s_j \ldots s_l$ was added to $\bar{T}$. Let $2 \leq r \leq i$ be the first index for which $\gamma_{s_r \ldots s_l}(\sigma) < (1 + 7\epsilon_2)\gamma_{min}$. If there is no such index then let $r = i$. The reason we need to deal with the prior case is clarified subsequently. In either case, since $\epsilon_2 < \epsilon/48$, and $\epsilon < 1$, then

$$\frac{\gamma_s(\sigma)}{\gamma_{s_r \ldots s_l}(\sigma)} > 1 + \frac{\epsilon}{4}. \tag{C.11}$$

In other words

$$\frac{\gamma_s(\sigma)}{\gamma_{s_2 \ldots s_l}(\sigma)} \cdot \frac{\gamma_{s_2 \ldots s_l}(\sigma)}{\gamma_{s_3 \ldots s_l}(\sigma)} \cdot \ldots \cdot \frac{\gamma_{s_{r-1} \ldots s_l}(\sigma)}{\gamma_{s_r \ldots s_l}(\sigma)} > 1 + \frac{\epsilon}{4} \ . \tag{C.12}$$

This last inequality implies that there must exist an index $1 \leq j \leq i - 1$, for which

$$\frac{\gamma_{s_j \ldots s_l}(\sigma)}{\gamma_{s_{j+1} \ldots s_l}(\sigma)} > 1 + \frac{\epsilon}{8L} \ . \tag{C.13}$$

We next show that Inequality (C.13) implies that $s_j \ldots s_l$ was added to $\bar{T}$. We do this by showing that $s_j \ldots s_l$ was added to $\bar{S}$, that we compared $\tilde{P}(\sigma | s_j \ldots s_l)$ to $\tilde{P}(\sigma | s_{j+1} \ldots s_l)$, and that the ratio between these two values is at least $(1 + 3\epsilon_2)$. Since $P(s) \geq \epsilon_0$ then necessarily

$$\tilde{P}(s_j \ldots s_l) \geq (1 - \epsilon_1)\epsilon_0 \ , \tag{C.14}$$

and $s_j \ldots s_l$ must have been added to $\bar{S}$. Based on our choice of the index r, and since $j < r$,

$$\gamma_{s_j \ldots s_l}(\sigma) \geq (1 + 7\epsilon_2)\gamma_{min}. \tag{C.15}$$

Since we assume that the sample is typical,

$$\tilde{P}(\sigma | s_j \ldots s_l) \geq (1 + 6\epsilon_2)\gamma_{min} > (1 + \epsilon_2)\gamma_{min} \ , \tag{C.16}$$

which means that we must have compared $\tilde{P}(\sigma | s_j \ldots s_l)$ to $\tilde{P}(\sigma | s_{j+1} \ldots s_l)$.

We now separate the case in which $\gamma_{s_{j+1} \ldots s_l}(\sigma) < \gamma_{min}$, from the case in which $\gamma_{s_{j+1} \ldots s_l}(\sigma) \geq \gamma_{min}$. If $\gamma_{s_{j+1} \ldots s_l}(\sigma) < \gamma_{min}$ then

$$\tilde{P}(\sigma | s_{j+1} \ldots s_l) \leq (1 + \epsilon_2)\gamma_{min} \ . \tag{C.17}$$

Therefore,

$$\frac{\tilde{P}(\sigma | s_j \ldots s_l)}{\tilde{P}(\sigma | s_{j+1} \ldots s_l)} \geq \frac{(1 + 6\epsilon_2)\gamma_{min}}{(1 + \epsilon_2)\gamma_{min}} \geq (1 + 3\epsilon_2) \ , \tag{C.18}$$

and $s_j \ldots s_l$ would have been added to $\bar{T}$. On the other hand, if $\gamma_{s_{j+1} \ldots s_l}(\sigma) \geq \gamma_{min}$, the same would hold since

$$\begin{aligned}
\frac{\tilde{P}(\sigma | s_j \ldots s_l)}{\tilde{P}(\sigma | s_{j+1} \ldots s_l)} &\geq \frac{(1 - \epsilon_2)\gamma_{s_j \ldots s_l}(\sigma)}{(1 + \epsilon_2)\gamma_{s_{j+1} \ldots s_l}(\sigma)} && \text{(C.19a)} \\
&> \frac{(1 - \epsilon_2)(1 + \frac{\epsilon}{8L})}{(1 + \epsilon_2)} && \text{(C.19b)} \\
&\geq \frac{(1 - \epsilon_2)(1 + 6\epsilon_2)}{(1 - \epsilon_2)} && \text{(C.19c)} \\
&> 1 + 3\epsilon_2 \ , && \text{(C.19d)}
\end{aligned}$$

where Inequality C.19c follows from our choice of ϵ_2 ($\epsilon_2 = \frac{\epsilon}{48L}$). This contradicts our initial assumption that s' is the longest suffix of s added to $\bar{T}$.

<u>2nd Claim:</u> We prove below that $\bar{T}$ is a subtree of T. The claim then follows directly, since when transforming $\bar{T}$ into $\hat{T}$, we add at most all $|\Sigma| - 1$ siblings of every node in $\bar{T}$.

Therefore it suffices to show that we did not add to $\bar{T}$ any node which is not in T. Assume to the contrary that we add to $\bar{T}$ a node s which is not in T. According to the algorithm, the reason we add s to $\bar{T}$, is that there exists a symbol σ such that $\tilde{P}(\sigma|s) \geq (1+\epsilon_2)\gamma_{min}$, and $\tilde{P}(\sigma|s)/\tilde{P}(\sigma|suffix(s)) > 1+3\epsilon_2$, while both $\tilde{P}(s)$ and $\tilde{P}(suffix(s))$ are greater than $(1-\epsilon_1)\epsilon_0$. If the sample string is typical then

$$P(\sigma|s) \geq \gamma_{min} \ , \ \tilde{P}(\sigma|s) \leq P(\sigma|s) + \epsilon_2\gamma_{min} \leq (1+\epsilon_2)P(\sigma|s) \ , \tag{C.20}$$

and

$$\tilde{P}(\sigma|suffix(s)) \geq P(\sigma|suffix(s)) - \epsilon_2\gamma_{min} \ . \tag{C.21}$$

If $P(\sigma|suffix(s)) \geq \gamma_{min}$ then $\tilde{P}(\sigma|suffix(s)) \geq (1-\epsilon_2)P(\sigma|suffix(s))$, and thus

$$\frac{P(\sigma|s)}{P(\sigma|suffix(s))} \geq \frac{(1-\epsilon_2)}{(1+\epsilon_2)}(1+3\epsilon_2) \ , \tag{C.22}$$

which is greater than 1 since $\epsilon_2 < 1/3$. If $P(\sigma|suffix(s)) < \gamma_{min}$, since $P(\sigma|s) \geq \gamma_{min}$, then $P(\sigma|s)/P(\sigma|suffix(s)) > 1$ as well. In both cases this ratio cannot be greater than 1 if s is not in the tree, contradicting our assumption. ■

References

Abe, N. & Warmuth, M. (1992). On the computational complexity of approximating distributions by probabilistic automata. *Machine Learning*, 9:205–260.

Baum, L.E. (1972). An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov chains. *Inequalities*, 3:1–8.

Baum, L.E., Petrie, T., Soules, G. & Weiss, N. (1970). A maximization technique occuring in the statistical analysis of probabilistic functions of markov chains. *Annals of Mathematical Statistics*, 41(1):164–171.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, 1957.

Blumer, A. (1990). Applications of DAWGs to data compression. In A. Capocelli, editor, *Sequences: Combinatorics, compression, security, and transmition*, pages 303–311. Springer-Verlag.

Cover, T. & Thomas, J. (1991). *Elements of Information Theory*. Wiley.

Dempster, A.P., Laird, N.M. & Rubin, D.B. (1977). Maximum-likelihood from incomplete data via the EM algorithm. *J. Royal Stat. Soc.*, B39:1–38.

Fill, J.A. (1991). Eigenvalue bounds on convergence to stationary for nonreversible Markov chains, with an application to exclusion process. *Annals of Applied Probability*, 1:62–87.

Freund, Y., Kearns, M., Ron, D., Rubinfeld, R., Schapire, R.E. & Sellie, L. (1993). Efficient learning of typical finite automata from random walks. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 315–324.

Gillman, D. & Sipser, M. (1994). inference and minimization of hidden markov chains. In *Proceedings of the Seventh Annual Workshop on Computational Learning Theory*, pages 147–158.

Good, G.I. (1969). Statistics of language: Introduction. In A. R. Meetham and R.A. Hudson, editors, *Encyclopedia of Linguistics, Information and Control*, pages 567–581. Pergamon Press, Oxford, England.

Höffgen, K.-U. (1993). Learning and robust learning of product distributions. In *Proceedings of the Sixth Annual Workshop on Computational Learning Theory*, pages 97–106.

Jelinek, F. (1969). A fast sequential decoding algorithm using a stack. *IBM J. Res. Develop.*, 13:675–685.

Jelinek, F. (1985). Self-organized language modeling for speech recognition. Technical report, IBM T.J. Watson Research Center.

Kearns, M., Mansour, Y., Ron, D., Rubinfeld, R., Schapire, R.E. & Sellie, L. (1994). On the learnability of discrete distributions. In *The 25th Annual ACM Symposium on Theory of Computing*.

Krishnan, P. & Vitter, J. S. (1993). Optimal prediction for prefetching in the worst case. Technical Report CS-1993-26, Duke University.

Krogh, A., Mian, S.I. & Haussler, D. (1993). A hidden markov model that finds genes in E. coli DNA. Technical Report UCSC-CRL-93-16, University of California at Santa-Cruz.

Kushilevitz, E. & Mansour, Y. (1993). Learning decision trees using the Fourier spectrum. *SIAM Journal on Computing*, 22(6):1331–1348.

Laird, P. & Saul, R. (1994). Discrete sequence prediction and its applications. *Machine Learning*, 15:43–68.

Mihail, M. (1989). Conductance and convergence of Markov chains - A combinatorial treatment of expanders. In *Proceedings 30th Annual Conference on Foundations of Computer Science*.

Nadas, A. (1984). Estimation of probabilities in the language model of the IBM speech recognition system. *IEEE Trans. on ASSP*, 32(4):859–861.

Rabiner, L.R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*.

Rissanen, J. (1983). A universal data compression system. *IEEE Trans. Inform. Theory*, 29(5):656–664.

Rissanen, J. (1986). Complexity of strings in the class of Markov sources. *IEEE Trans. Inform. Theory*, 32(4):526–532.

Ron, D., Singer, Y. & Tishby, N. (1993). The power of amnesia. In *Advances in Neural Information Processing Systems*, volume 6. Morgan Kaufmann.

Ron, D., Singer, Y. & Tishby, N. (1995). On the learnability and usage of acyclic probabilistic finite automata. In *Proc. of the 8th Annual Conf. on Computational Learning Theory*.

Rudd, K.E. (1993). Maps, genes, sequences, and computers: An Escherichia coli case study. *ASM News*, 59:335–341.

Schütze, H. & Singer, Y. (1994). Part-of-Speech tagging using a variable memory Markov model. In *Proceedings of ACL 32'nd*.

Shannon, C.E. (1951). Prediction and entropy of printed english. *Bell Sys. Tech. Jour.*, 30(1):50–64.

Singer, Y. & Tishby, N. (1995). An adaptive cursive handwriting recognition system. Technical Report CS-TR-22, Hebrew University.

Vitter, J.S. & Krishnan, P. (1991). Optimal prefetching via data compression. In *Proceedings of the Thirty-Second Annual Symposium on Foundations of Computer Science*, pages 121–130.

Weinberger, M.J., Lempel, A. & Ziv, J. (1982). A sequential algorithm for the universal coding of finite-memory sources. *IEEE Trans. Inform. Theory*, 38:1002–1014.

Willems, F.M.J., Shtarkov, Y.M. & Tjalkens, T.J. (1993). The context tree weighting method: Basic properties. *IEEE Trans. Inform. Theory*. Submitted for publication.

Ziv, J. & Lempel, A. (1978). Compression of individual sequences via variable-rate coding. *IEEE Trans. Inform. Theory*, 24:530–536.

Received September 7, 1994
Accepted May 11, 1995
Final Manuscript August 24, 1995