

Purdue University  
**Purdue e-Pubs**

---

Department of Computer Science Technical  
Reports

Department of Computer Science

---

1999

## Notes on Learning Probabilistic Automata

Alberto Apostolico

Report Number:  
99-028

---

Apostolico, Alberto, "Notes on Learning Probabilistic Automata" (1999). *Department of Computer Science Technical Reports*. Paper 1458.  
<https://docs.lib.purdue.edu/cstech/1458>

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries.  
Please contact [epubs@purdue.edu](mailto:epubs@purdue.edu) for additional information.

**NOTES ON LEARNING PROBABILISTIC AUTOMATA**

**Alberto Apostolico**

**Computer Sciences Department  
Purdue University  
West Lafayette, IN 47907**

**CSD TR #99-028  
September 1999**

# NOTES ON LEARNING PROBABILISTIC AUTOMATA\*

Alberto Apostolico<sup>†</sup>

Purdue University and Università di Padova

Purdue CS TR 99-028

(September 99)

## 1 Introduction

We adopt definitions and notations from [3, 7], with which some familiarity is assumed. We deal with a (possibly singleton) collection  $S$  of strings over a finite alphabet  $\Sigma$ , and use  $\lambda$  to denote the empty string. The *length* of  $S$  is the sum of the lengths of all strings in it and is denoted by  $n$ . With reference to  $S$  and a generic string  $s = s_1s_2\dots s_l$ , the *empirical probability*  $\tilde{P}$  of  $s$  is defined provisionally as the number of times  $s$  occurs in  $S$  divided by the maximum “possible” number of such occurrences. The *conditional empirical probability* of observing the symbol  $\sigma$  soon after the string  $s$  is given by the ratio

$$\tilde{P}(\sigma|s) = \frac{\chi_{s\sigma}}{\chi_{s*}} ,$$

where  $\chi_w$  is the number of occurrences of string  $w$  and  $s*$  is every single-symbol extensions of  $s$  having occurrence in  $S$ . Finally,  $\text{suf}(s)$  denotes  $s_2s_3\dots s_l$ .

In [3, 7], tree-shaped probabilistic automata are built to be used as an aid in the classification of strings. In any such tree, each edge is labeled by a symbol, each node corresponds to a unique string –the one obtained by traveling from that node to the root– and nodes are weighted by a probability vector giving the distribution over the next symbol. In the following,  $\mathcal{T}$  is the tree-shaped automaton,  $\bar{S}$  is the set of strings that we want to check and  $\gamma_s$  is the probability distribution over the next symbol associated with node  $s$ . The construction starts with a tree consisting of just the root node (i.e., the tree associated with the empty word) and adds paths as follows. For each substring  $s$  considered, it is checked whether there is some symbol  $\sigma$  in the alphabet for which the empirical probability of observing it after  $s$  is significant and significantly

---

\*Work supported in part by NSF Grant CCR-9700276 and by the Italian Ministry of Research.

<sup>†</sup>Department of Computer Sciences, Purdue University, Computer Sciences Building, West Lafayette, IN 47907, USA and Dipartimento di Elettronica e Informatica, Università di Padova, Via Gradenigo 6/A, I-35131 Padova, Italy. [axa@cs.purdue.edu](mailto:axa@cs.purdue.edu).

different from the probability of observing it after  $\text{suf}(s)$ . Whenever these conditions hold, the path relative to the substring (and possibly its necessary but currently missing ancestors) are added to the tree.

Given a string, its weighting by a tree is done by scanning the string one letter after the other while assigning a probability to every letter, in succession. The probability of a letter is calculated by walking down the tree in search for the longest suffix that appears in the tree and ends immediately before that letter, and multiplying the corresponding conditional probability. Since, following each input symbol, the search for the deepest node must be resumed from the root, this process cannot be carried out on-line nor in linear-time in the length of the tested sequence.

The following figures reproduce for our convenience the constructions of the trees respectively in [3] and [7]. Here  $L$  is the maximum length for a string to be considered,  $P_{min}$  is the minimum value of the empirical probability in order for the string to be considered,  $r$  measures the prediction difference between the candidate and its father,  $\alpha$  the  $\epsilon$ 's and  $\gamma_{min}$  are suitable constants and part of the input parameters.

We are interested primarily in the asymptotic complexity of the main part (tree construction) of these procedures, and in possible ways to improve it. In this regard, the algorithms may be considered equivalent except for small changes in the treatment of internal nodes and the choice of constants. These details have no substantial bearing on the performance and no consequence on our considerations. Also, for the same reason, we are not interested in the last steps of smoothing probabilities. We see that the body of the algorithms consists of checking all substrings having empirical probability at least  $P_{min}$  and length at most to  $L$ . Although the number of substrings passing these tests may be quite small in practice, still there are in principle  $n - l + 1$  possible different strings for each  $l$ , which would lead to  $\Theta(Ln^2)$  time just to compute and test empirical probabilities ( $nL$  strings in total each requiring at least  $O(n)$  work to test). The discussion that follows shows that in fact overall  $O(n)$  time suffices. Along the way, a possibly more accurate notion of empirical probability is proposed and computed. Finally, we will see that also the structure and use of the automaton can be made more efficient. We assume henceforth that  $S$  contains only one string, which will be denoted by  $x$ .

Our approach must depart considerably from the algorithms of the figures. There, word selection and tree construction go hand in hand in Step B. In our case, even though in the end the two can be re-combined, we decouple these tasks. We concentrate on word selection, hence on the tests of Step B. Essentially, we want to show that all those words can be tested in overall linear time, even if those word lengths may add up to more than linear space. Throughout, we reason to fix the ideas in terms only of the algorithm of Figure 2, since the other one may be handled analogously.

## 2 Computing Empirical Probabilities

Consider for a moment the problem of defining and computing empirical probabilities. This is needed at the beginning and intermediate steps of the algorithms, and also to dynamically update the set  $S$  while keeping its size from "growing exponentially". One problem here is that the notion of empirical probability is not straightforward. Fortunately in the algorithms -in

# PSA Learning Algorithm from [7]

## First Phase

1. Initialize  $\bar{T}$  and  $\bar{S}$ : let  $\bar{T}$  consist of a single node corresponding to  $\lambda$ , and let  $\bar{S} \leftarrow \{\sigma \mid \sigma \in \Sigma \text{ and } \tilde{P}(\sigma) \geq (1 - \epsilon_1)\epsilon_0\}$ .
2. While  $\bar{S} \neq \emptyset$  do

Pick any  $s \in \bar{S}$  and do

- (A) Remove  $s$  from  $\bar{S}$ ;
- (B) if there is 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  $s$ ;

- (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  $\bar{S}$ .

## Second Phase

1. Initialize  $\hat{T}$  to be  $\bar{T}$ .
2. Extend  $\hat{T}$  by adding all missing sons of internal nodes
3. 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 1:

## PSA Learning Algorithm from [3]

1. Initialize  $\bar{T}$  and  $\bar{S}$ : let  $\bar{T}$  consist of a single node corresponding to  $e$ , and let  $\bar{S} \leftarrow \{\sigma | \sigma \in \Sigma \text{ and } \tilde{P}(\sigma) \geq P_{min}\}$ .
2. Building the Skeleton: While  $\bar{S} \neq \emptyset$ , pick any  $s \in \bar{S}$  and do

- (A) Remove  $s$  from  $\bar{S}$ ;
- (B) if there is a symbol  $\sigma \in \Sigma$  such that

$$\tilde{P}(\sigma|s) \geq (1 + \alpha)\gamma_{min},$$

and

$$(1) \frac{\tilde{P}(\sigma|s)}{\tilde{P}(\sigma|suffix(s))} \geq r \text{ or } (2) \frac{\tilde{P}(\sigma|s)}{\tilde{P}(\sigma|suffix(s))} \leq 1/r,$$

then add to  $\bar{T}$  the node corresponding to  $s$  and all the nodes on the path to  $s$  from the deepest node in  $\bar{T}$  that is a suffix of  $s$ ;

- (C) If  $|s| < L$  then for every  $\sigma' \in \Sigma$ , if

$$\tilde{P}(\sigma' \cdot s) \geq P_{min},$$

then add  $\sigma' \cdot s$  to  $\bar{S}$ .

3. Smoothing the prediction probabilities: For each  $s$  labeling a node in  $\bar{T}$ , let  $\tilde{\gamma}_s(\sigma) = \tilde{P}(\sigma|s)(1 - |\Sigma|\gamma_{min}) + \gamma_{min}$

Figure 2:

so far as the construction of the automaton goes— we are interested primarily in *conditional* probabilities which turn out to be less controversial.

One ingredient in the computation of empirical probabilities is the count of occurrences of a string in another string or set of strings. We concentrate on this problem first. Since there can be  $O(n^2)$  distinct substrings in a string of  $n$  symbols, a table storing the number of occurrences of all substrings of the string might take up in principle  $\Theta(n^2)$  space. However, it is well known that linear time and space suffice to build an index suitable to return, for any string  $w$ , its  $\chi_w$  count in  $x$ . Here we need to analyze this fact a little closer. We begin by recalling an important “left-context” property, which is conveniently adapted from [4].

Given two words  $x$  and  $y$ , the *start-set* of  $y$  in  $x$  is the set of *occurrences* of  $y$  in  $x$ , i.e.,  $\text{pos}_x(y) = \{i : y = x_i \dots x_j\}$  for some  $i$  and  $j$ ,  $1 \leq i \leq j \leq n$ . Two strings  $y$  and  $z$  are equivalent on  $x$  if  $\text{pos}_x(y) = \text{pos}_x(z)$ . The equivalence relation instituted in this way is denoted by  $\equiv_x$  and partitions the set of all strings over  $\Sigma$  into equivalence classes. Recall that the *index* of an equivalence relation is the number of equivalence classes in it.

**Fact 2.1** The index  $k$  of the equivalence relation  $\equiv_x$  obeys  $k < 2n$ .

In the example of the string *abaababaabaababaababa*, for instance,  $\{ab, aba\}$  forms one such  $C_i$ -class and so does  $\{abaa, abaab, abaaba\}$ . Fact 2.1 suggests that we might only need to compute empirical probabilities for  $O(n)$  substrings in a string with  $n$  symbols. The following considerations make this statement more precise and in fact give one possible proofs of it.

With  $\$$  a symbol not in  $\Sigma$ , the *suffix tree*  $T_x$  associated with  $x$  is the digital search tree that collects the first  $n$  suffixes of  $x\$$ . In the *compact* representation of  $T_x$  (see Figure 3), chains of unary nodes are collapsed into single arcs, and every arc of  $T_x$  is labeled with a substring of  $x\$$ . A pair of pointers (e.g., starting position, length) to a common copy of  $x$  can be used for each arc label, whence the overall space taken by  $T_x$  is  $O(n)$ . In  $T_x$ , the  $i$ th suffix  $\text{suf}_i$  of  $x\$$  ( $i = 1, 2, \dots, n$ ) is described by the concatenation of the labels on the unique path of  $T_x$  that leads from the root to leaf  $i$ . Similarly, any vertex  $\alpha$  of  $T_x$  distinct from the root describes a subword  $w(\alpha)$  of  $x$  in a natural way: vertex  $\alpha$  is called the *proper locus* of  $w(\alpha)$ . In the compact  $T_x$ , the *locus* of  $w$  is the unique vertex of  $T_x$  such that  $w$  is a prefix of  $w(\alpha)$  and  $w(\text{Father}(\alpha))$  is a proper prefix of  $w$ .

An algorithm for the construction of the expanded  $T_x$  is readily organized: We start with an empty tree and add to it the suffixes of  $x\$$  one at a time. Conceptually, the insertion of suffix  $\text{suf}_i$  ( $i = 1, 2, \dots, n + 1$ ) consists of two phases. In the first phase, we search for  $\text{suf}_i$  in  $T_{i-1}$ . Note that the presence of  $\$$  guarantees that every suffix will end in a distinct leaf. Therefore, this search will end with failure sooner or later. At that point, though, we will have identified the longest prefix of  $\text{suf}_i$  that has a locus in  $T_{i-1}$ . Let  $\text{head}_i$  be this prefix and  $\alpha$  the locus of  $\text{head}_i$ . We can write  $\text{suf}_i = \text{head}_i \cdot \text{tail}_i$  with  $\text{tail}_i$  nonempty. In the second phase, we need to add to  $T_{i-1}$  a path leaving node  $\alpha$  and labeled  $\text{tail}_i$ . This achieves the transformation of  $T_{i-1}$  into  $T_i$ .

Clever constructions such as in [6, 8, 9] are available that build the tree in time  $O(n \log |\Sigma|)$  and linear space. One key element in such constructions is offered by the following easy fact.

**Fact 2.2** If  $w = av$ ,  $a \in \Sigma$ , has a proper locus in  $T_x$ , then so does  $v$ .

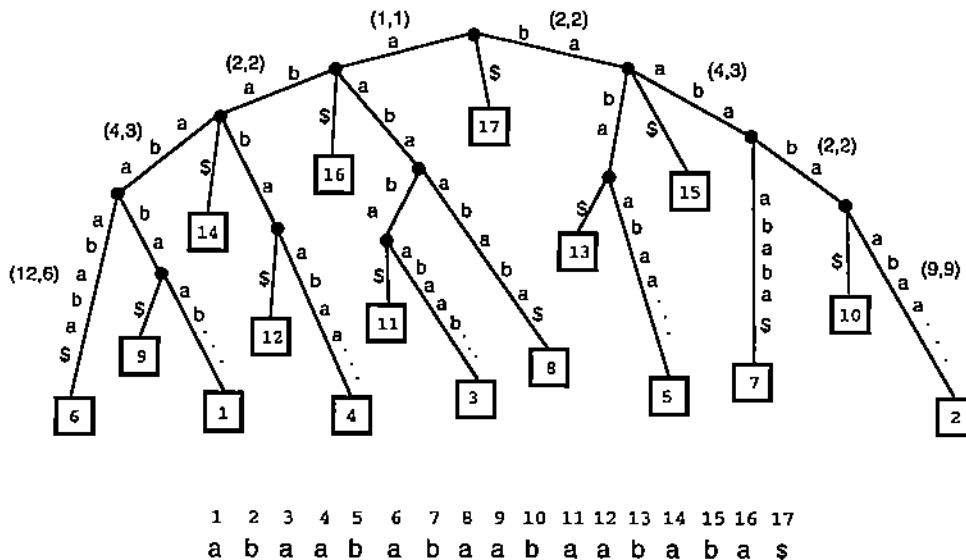


Figure 3: A suffix tree in compact form, shown with only the outermost arcs labeled by pointers

To exploit this fact, *suffix links* are maintained in the tree that lead from the locus of each string  $av$  to the locus of its suffix  $v$ . Here we are interested in Fact 2.2 only for future reference. Having built the tree, some simple additional manipulations make it possible to count and locate the distinct (possibly overlapping) instances of any pattern  $w$  in  $x$  in  $O(|w|)$  steps. For instance, listing all occurrences of  $w$  in  $x$  is done in time proportional to  $|w|$  plus the total number of such occurrences, by reaching the locus of  $w$  and then visiting the subtree of  $T_x$  rooted at that locus. Alternatively, a trivial bottom-up computation on  $T_x$  can weight each node of  $T_x$  with the number of leaves in the subtree rooted at that node. This weighted version serves then as a statistical index for  $x$ , in the sense that, for any  $w$ , we can find the count  $\chi_w$  of  $w$  in  $x$  in  $O(|w|)$  time. Note that the counter associated with the locus of a string reports its correct count even when the string terminates in the middle of an arc. This is, indeed, nothing but a re-statement and a proof of Fact 2.1. From now on, we assume that a tree with weighted nodes has been produced and is available for our constructions.

We are now ready to consider more carefully the notion of empirical probability. One way to define the empirical probability of  $w$  in  $x$  is to take the ratio of the count of the number  $\chi_w$  to  $|x| - |w| + 1$ , where the latter is interpreted as the maximum number of possible starting positions for  $w$  in  $x$ . For  $w$  and  $v$  much shorter than  $x$  we have that the difference between  $|x| - |w| + 1$  and  $|x| - |wv| + 1$  is negligible, which means that the probabilities computed in this way and relative to words that end in the middle of an arc do not change, i.e., we only need to compute those associated with strings that end at a node of the compact  $T_x$ .

This notion of empirical probability, however, assumes that every position of  $x$  compatible with  $w$  length-wise is an equally likely candidate. This is not the case in general, since the maximum number of possible occurrences of one string within another string depends crucially on the compatibility of self-overlaps. For example, the pattern  $aba$  could occur at most once every two positions in *any* text,  $abaab$  once every four, etc. Compatible self-overlaps for a string  $z$  depend on the structure of the periods of  $z$ . We review these notions briefly.



```

procedure maxborder ( y )
  begin
    bord[0] ← -1; r ← -1;
    for m = 1 to h do
      while r ≥ 0 and yr+1 ≠ ym do
        r ← bord[r];
      endwhile
      r = r + 1; bord[m] = r
    endfor
  end

```

Figure 4: Computing the longest borders for all prefixes of  $y$

A string  $z$  has a *period*  $w$  if  $z$  is a prefix of  $w^k$  for some integer  $k$ . Alternatively, a string  $w$  is a period of a string  $z$  if  $z = w^l v$  and  $v$  is a possibly empty prefix of  $w$ . Often when this causes no confusion, we will use the word “period” also to refer to the length or *size*  $|w|$  of a period  $w$  of  $z$ . A string may have several periods. The shortest period (or period length) of a string  $z$  is called *the period* of  $z$ . Clearly, a string is always a period of itself. This period is called the trivial period.

A germane notion is that of a border. We say that a non-empty string  $w$  is a *border* of a string  $z$  if  $z$  starts and ends with an occurrence of  $w$ . That is,  $z = uw$  and  $z = wv$  for some possibly empty strings  $u$  and  $v$ . Clearly, a string is always a border of itself. This border is called the trivial border. It is not difficult to see that two consecutive occurrences of a word may overlap only if their distance equals one of the periods of  $w$ . Hence we have

**Fact 2.3** The maximum possible number of occurrences of a string  $w$  into another string  $x$  is equal to  $(|x| - |w| + 1)/|u|$ , where  $u$  is the smallest period of  $w$ .

Fact 2.3 tells us that in order to compute the empirical probabilities of, say, all prefixes of a string we need to know the borders or periods of all those prefixes. In fact, we can manage to carry out *all* the updates relative to the set of prefixes of a same string in *overall* linear time, thus in amortized constant time per update. This possibility rests on a simple adaptation of a classical implement of fast string searching, that computes the longest borders (and corresponding periods) of all prefixes of a string in overall linear time and space. We report one such construction in Figure 4 below, for the convenience of the reader, but refer for details and proofs of linearity to discussions of “failure functions” and related constructs such as found in, e.g., [5].

The construction of Fig. 4 may be applied, in particular, to each suffix  $su f_i$  of a string  $x$  while that suffix is being inserted as part of the direct tree construction. This would result in an annotated version of  $T_x$  in overall quadratic time and space in the worst case. Note that, unlike in the case of empirical probabilities previously considered, the period –and thus also the empirical probabilities of Fact 2.3– may change along an arc of  $T_x$ , so that we may need to compute explicitly all  $\Theta(n^2)$  of them. However, if we were interested in such probabilities

only at the nodes of the tree, then these could still be computed in overall linear time. The key to this latter fact is to run a suitably adapted version of `maxborder` walking on suffix links “backward”, i.e., traversing them in their reverse direction, beginning at the root of  $T_x$  and then going deeper and deeper into the tree. One way to visualize this process is as follows. Imagine first the “co-tree” of  $T_x$  formed by the reversed suffix links: we can visit such a structure depth first and simultaneously run a procedure much similar to `maxborder` to assign periods to all nodes of  $T_x$ . Correctness rests on the fact that for any word  $w$  the periods of  $w$  and  $w^r$  coincide. We shall see shortly that in situations of interest to us we can limit computation to the nodes of  $T_x$ .

**Fact 2.4** The set of empirical probabilities of all (short) words of  $x$  that have a proper locus in  $T_x$  can be computed in linear time and space.

### 3 Conditional Probabilities and Ratios Thereof

Consider now conditional empirical probabilities, which were defined as the ratio between the observed occurrences of  $s\sigma$  to the occurrences of  $s*$ . The first thing to observe is that the value of this ratio persists along each arc of the tree, i.e.,

$$\tilde{P}(\sigma|s) = \chi_s / \chi_{s\sigma} = 1$$

for any word  $s$  ending in the middle of an arc of  $T_x$  and followed there by a symbol  $\sigma$ . Therefore, we know that every such word passes the first test under  $(B)$ , while continuation of  $s$  by any other symbol would have zero probability and thus fail. These words  $s$  have then some sort of an obvious implicit vector and need not be tested or considered explicitly. On the other hand, whenever both  $s$  and  $\text{suffix}(s)$  end in the middle of an arc, the ratio

$$\frac{\tilde{P}(\sigma|s)}{\tilde{P}(\sigma|\text{suffix}(s))} = \frac{1}{1} = 1 .$$

Since  $r > 1$ , then neither  $r$  nor  $1/r$  may be equal to 1, so that no such word passes either part of the second test under  $B$ . The fate of any such word with respect to inclusion in the tree depends thus on that of its shortest extension with a proper locus in it. The cases where both  $s$  and  $\text{suffix}(s)$  have a proper locus in  $T_x$  are easy to handle, since there is only  $O(n)$  of them and the corresponding tests take linear time overall. By Fact 2.2, it is not possible that  $s$  has a proper locus while  $\text{suffix}(s)$  does not. Therefore, we are left with those cases where a proper locus exists for  $\text{suffix}(s)$  but not for  $s$ . There are still only  $O(n)$  such cases of course, but the problem here is that in these cases we do not find a suffix link to traverse backward from the locus of  $\text{suffix}(s)$  to that of  $s$ . Fortunately, this is easy to fix, but towards this end we need first to define reverse suffix links more formally.

Let  $\nu'$  be the locus of string  $s'$ . We define  $\text{rsuf}(\nu', \sigma)$  as the node  $\nu$  which is the locus of the shortest extension of  $as'$  having a proper locus in  $T_x$ . I.e., node  $\nu$  is such that  $w(\nu) = sz$  with  $s = \sigma s'$  and  $z$  as short as possible if  $z \neq \lambda$ . If  $\sigma s'$  has no occurrence in  $x$  then  $\text{rsuf}(\nu', \sigma)$  is not defined.

Clearly,  $\text{rsuf}$  coincides with the reverse of the suffix link whenever the latter is defined. When no such original suffix link is defined while  $\sigma s'$  occurs in  $x$ , then  $\text{rsuf}$  takes us to the locus of the shortest word in the form  $\sigma s'z$  with the property that all occurrences of  $s$  in  $x$  occur precisely within the context of  $\sigma s'z$  beginning at the second position of this string. In other words, we have  $\text{pos}_x(\sigma s') = \text{pos}_x(\sigma s'z)$ . In these cases, the locus of  $\sigma s'z$  may be treated as a surrogate locus of that of  $\sigma s'$ , since the  $\chi$ -values and ratios thereof would be the same for both.

Setting  $\text{rsuf}$ 's is an easy linear post-processing of  $T_x$  (details will follow). In conclusion, attaching empirical conditional probabilities only to the branching nodes of  $T_x$  suffices. As there are  $O(n)$  such nodes, and the alphabet is finite, the collection of all conditional probability vectors for *all* subwords of  $x$  takes only linear space.

**Fact 3.1** The set of empirical conditional probabilities of all (short) words of a string  $x$  over a finite alphabet can be computed in linear time and space. The conditions under Step B can be tested implicitly for all such words in overall linear time and space.

## 4 PSAs and MPPMs

At this point we can informally summarize as follows the procedure by which words are selected for inclusion in our automaton.

1. Build a compact  $T_x$  for  $x$ , weight its ( $O(n)$ , branching) nodes with the  $\chi$ -counts and empirical probabilities;
2. Determine the words to be included in  $\bar{S}$  and thus in the final automaton. For this, consider the nodes of  $T_x$  bottom up and run the tests of Step B on these nodes. Prune the tree in correspondence with nodes failing the tests (to be detailed). By the above Facts, we know that pruning will have to occur in correspondence with nodes, and not along an arc.

The pruned version of the tree resulting from this treatment represents a Trie-shaped automaton capable of performing much the same function as a PSA and yet having the structure of a Multiple Pattern Matching Machine (MPPM) [1, 5]. The import of this is that, on such a machine, the paths corresponding to the substring undergoing test are traversed from the root to the leaves. The structure of MPPMs, with failure-function links etc., ensures that while the input string is scanned symbol after symbol, we are always at the node of the MPPM that corresponds to the longest suffix of the input having a node in the MPPM. Also by the structure of MPPMs, running a string through it takes always overall linear time in the length of the string.

**Acknowledgements** I am indebted to Gill Bejerano and Golan Yona for many helpful comments.

## References

- [1] Aho, A.V., J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass. (1974).

- [2] Apostolico, A., and Z. Galil (Eds.), *Pattern Matching Algorithms*, Oxford University Press, New York (1997).
- [3] Bejerano, G. and G. Yona, Modeling Protein families using probabilistic suffix trees. *Proceedings of RECOMB99* (S. Istrail, P. Pevzner and M. Waterman, eds.), 15–24, Lyon, France, ACM Press (April 1999).
- [4] Blumer, A., J. Blumer, A. Ehrenfeucht, D. Haussler, M.T. Chen and J. Seiferas, The Smallest Automaton Recognizing the Subwords of a Text, *Theoretical Computer Science* , **40**, 31-55 (1985).
- [5] Crochemore, M., and W. Rytter, *Text Algorithms*, Oxford University Press, New York (1994).
- [6] McCreight, E.M., A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23(2):262–272, April 1976.
- [7] Ron, D., Y. Singer and N. Tishby, The power of amnesia: learning probabilistic automata with variable memory length. *Machine Learning*, 25:117–150 (1996).
- [8] Ukkonen, E., On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.
- [9] Weiner, P., Linear pattern matching algorithm. In *Proceedings of the 14th Annual IEEE Symposium on Switching and Automata Theory*, pages 1–11, Washington, DC, 1973.