, , 1–38 ()
© Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Learning Fallible Deterministic Finite Automata

DANA RON danar@cs.huji.ac.il
Computer Science Institute, Hebrew University, Jerusalem 91904, Israel

RONITT RUBINFELD ronitt@cs.cornell.edu
Computer Science Department, Cornell University, Ithaca, NY 14853, U.S.A.

Editor: Sally A. Goldman

Keywords: PAC Learning under the Uniform Distribution, Persistent Errors, Fallible Expert, Deterministic Finite Automata

1. Introduction

Suppose a scientist is given a comprehensive set of data that has been collected, and is asked to come up with a simple explanation of it. Such situations might include trying to explain data collected by a space mission, data from a national survey, weather pattern information recorded over the last 50 years, or many observations of a doctor doing medical diagnosis. This task is made more difficult by the fact that there may be a large error rate in the data collection process, and if there is no additional independent source of data, the scientist can not easily determine the errors. Ideally, since the error rate of the data collection process may be unacceptable, the explanation should allow the scientist to *correct* most of the errors in the data.

We view this as the problem of learning a concept from a fallible expert. The expert answers all queries about the concept, but often gives incorrect answers. We consider an expert that errs on each input with a fixed probability, independently of whether it errs on the other inputs. We assume though that the expert is *persistent*, i.e., if queried more than once on the same input, it will always return the same answer.

The goal of the learner is to construct a hypothesis algorithm that will not only concisely hold the correct knowledge of the expert, but will actually surpass the expert by using the structural properties of the concept in order to correct the erroneous data.

Specifically, we consider the problem in which the true target concept is a Deterministic Finite Automaton (DFA). Angluin and Laird [3] propose to explore the effect of noise in the case of queries, and specifically ask if Angluin's algorithm [2] for learning DFAs in the error-free case can be modified to handle errors both in the answers to the queries and in the random examples. We answer this question by presenting a polynomial time algorithm for learning fallible DFAs under the uniform distribution on inputs. The algorithm may ask membership queries, and works in the presence of uniformly and independently distributed errors as long as the error probability is bounded away from 1/2. The result can be extended to the following cases. (1) The expert's errors are distributed only k-wise independently for $k = \Omega(1)$; (2) The expert's error probability depends on the length of the input string; (3) The target automaton has more than 2 possible outputs.

Our techniques for solving this problem include a method for partitioning strings into classes, which are intended to correspond to states in the hypothesis automaton. This partitioning is done according to the strings' behavior on large sets of suffixes. In particular, strings reaching the same state in the correct automaton will be in the same class. Using additional properties of the partition, we show how to correct an arbitrarily large fraction of the expert's errors and thus receive a more refined labeled partition on which we base the construction of our hypothesis automaton. Parts of our algorithm rely on a version of Angluin's algorithm [2] for learning finite automata in the error-free case. This version is presented preceding the description of our algorithm.

2. Related Results

In the error free case of learning DFAs, Kearns and Valiant [24] use the prediction preserving reductions of Pitt and Warmuth [28] to show that under cryptographic assumptions, the problem of predicting the class of DFAs is hard when only given access to random examples. Angluin [1] shows that (exact) learning using only membership queries is also hard. She describes a family of automata that cannot be identified in less than exponential time when the learner can only observe the behavior of the machine on inputs of the learner's own choosing.

However, as mentioned earlier, Angluin [2] describes an algorithm for learning DFAs given access both to random examples and to membership queries. Rivest and Schapire [30], [31], [33] present algorithms for inferring DFAs from input/output behavior in the absence of a means of resetting the machine to a start state. Freund et al. [12] present efficient algorithms for learning typical DFAs from random walks without membership queries, both when the learner is provided with the means of resetting the machine and when it is not.

Several results have been obtained for learning in the presence of errors in the Probably Approximately Correct model introduced by Valiant [38]. These include results for learning in the presence of malicious and random noise in classification and attributes [39], [3], [25], [22], [20], [34], [35], [37], [36]. In recent work Kearns [21] identifies and formalizes a sufficient condition on learning algorithms in Valiant's model that permits the immediate derivation of noise-tolerant learning algorithms. He introduces a new model of *learning from statistical queries* and shows that any class efficiently learnable from statistical queries is also learnable with random classification noise in the random examples.

There are fewer results when generalizing PAC learning to learning with membership queries. Sakakibara [32] shows that if for each query there is some independent probability to receive an incorrect answer, and these errors are not persistent then queries can be repeated until the confidence in the correct answer is high enough. Therefore, existing learning algorithms can be modified and then used in this model of random noise.

Dean *et. al.* [11] study Rivest and Schapire's model [30] for learning DFAs when the learner has no means of resetting the machine and thus can be viewed as a robot learning an environment. They investigate the case in which the output of each state may be erroneous with some fixed probability. Since the learner cannot reset the target machine, this problem is in general a harder problem than the one studied in this paper, and the authors solve it only under the assumption that the learner is either given a distinguishing sequence or can generate one efficiently with high probability. It is known (and there are very simple examples illustrating it) that not every automaton has a distinguishing sequence. Moreover, even if the target automaton is known to have a distinguishing sequence, then there is not necessarily an efficient procedure for finding one such sequence.[1] Thus the solution described in [11] is not applicable in the general case, even when the learner does have means of resetting the machine (as our learner does).

Goldman, Kearns, and Schapire [14] consider a model of persistent noise in membership queries which is very similar to the one used in this paper. They present algorithms for exactly identifying different circuits under fixed distributions, and show that their algorithms can be modified to handle large rates of randomly chosen, though persistent, misclassification noise in the queries. Angluin and Slonim [7] consider a more benign model of incomplete membership queries in which with some probability the teacher may answer "I don't know". For more work in this model see [15].

3. Preliminaries

Let A be the deterministic finite state automaton we would like to learn. As usual, A is a 5-tuple $(Q, \Sigma, \tau, q_0, F)$ where Q is a finite set of n *states*, Σ is a finite *alphabet*, $\tau : Q \times \Sigma \rightarrow Q$ is the *transition function*, $q_0 \in Q$ is the *starting* state, and $F \subseteq Q$ is the set of *accepting* states. The transition function, τ, can be extended to be defined on $Q \times \Sigma^*$ in the usual manner. The *label* of a state q is 1 if $q \in F$ and 0 otherwise. The *label* given by A to a string $u \in \Sigma^*$ is defined as the label of the state reached by u, i.e., the label of $\tau(q_0, u)$, and is denoted by $\bar{A}(u)$. Unless stated otherwise, all strings referred to are over the alphabet Σ.

Let D_L be the distribution which is uniform on strings over Σ of length at most L. Both L and n_b, an upper bound on n, the number of states, are given to the learning algorithm. We assume $L = \Omega(\log_{|\Sigma|} n_b)$. The algorithm can generate random strings distributed according to D_L and may make membership queries. For every newly queried string u, independently, and with probability η, the expert's answer, $\mathcal{E}(u)$, received for that string differs from $\bar{A}(u)$. Any additional query on the same string is answered consistently. The error probability, η, is bounded away from one half, so that $\eta \leq 1/2 - \gamma_b$, for some positive quantity, γ_b, which is given to the algorithm.

We now define what an ϵ-*good* hypothesis is and what a *good learning algorithm* is.

Definition. Given two automata A_1 and A_2, we say that A_1 is an ϵ-good hypothesis with respect to A_2 (and distribution D) if $Pr_D[\bar{A}_1(u) \neq \bar{A}_2(u)] \leq \epsilon$. Otherwise A_1 is an ϵ-bad hypothesis.

Definition. We say that Algorithm $\mathcal{A}$ is a good learning algorithm for fallible DFAs if for every approximation parameter $0 < \epsilon \leq 1$, success parameter $0 < \delta \leq 1$ and error probability $0 \leq \eta \leq 1/2 - \gamma_b$, with probability at least $1 - \delta$, after asking a number of membership queries which is polynomial in $n_b, \frac{1}{\gamma_b}, |\Sigma|, L, \frac{1}{\epsilon}$ and $\frac{1}{\delta}$, and after performing a polynomial amount of computation, it outputs a hypothesis automaton A' such that A' is an ϵ-good hypothesis with respect to A and D_L.

The following are additional definitions which are used in the paper.

Definition. Let U_1 and U_2 be two sets of strings. Then $U_1 \circ U_2 \stackrel{\rm def}{=} \{u_1 \cdot u_2 | u_1 \in U_1, u_2 \in U_2\}$.

Definition. We say that two automata A_1 and A_2 *agree* on a string u, if $\bar{A}_1(u) = \bar{A}_2(u)$. Otherwise they *differ* on u.

Definition. Let u_1, u_2 and u_3 be strings.

- The correct label of u_1 is $\bar{A}(u_1)$, and the observed label is $\mathcal{E}(u_1)$.
- The correct behavior of u_1 on (the suffix) u_3 is $\bar{A}(u_1 \cdot u_3)$ while the observed behavior is $\mathcal{E}(u_1 \cdot u_3)$.
- We say that u_1 and u_2 truly differ on (the suffix) u_3 if $\bar{A}(u_1 \cdot u_3) \neq \bar{A}(u_2 \cdot u_3)$. Otherwise they truly behave the same on u_3.
- If $\mathcal{E}(u_1 \cdot u_3) \neq \mathcal{E}(u_2 \cdot u_3)$ we say there is an observed difference between u_1 and u_2 on u_3.

We also need the following. For $M > 0$, let $X_1, X_2, ...X_M$ be M independent 0/1 random variables were $Pr[X_i = 1] = p_i$, and $0 < p_i < 1$. Let $p = \sum_i p_i/M$. Then we have the following two inequalities. The first inequality (additive form) is usually credited to Hoeffding [16] and the second inequality (multiplicative form) is usually credited to Chernoff [10]. The versions below were taken from [6].

references. are they ok?
references. are they ok?

INEQUALITY 1 *For* $0 < \alpha \leq 1$,

$$Pr[\frac{\sum_{i=1}^{M} X_i}{M} - p > \alpha] < e^{-2\alpha^2 M}$$

and

$$Pr[p - \frac{\sum_{i=1}^{M} X_i}{M} > \alpha] < e^{-2\alpha^2 M}$$

INEQUALITY 2 *For* $0 < \alpha \leq 1$,

$$Pr[\frac{\sum_{i=1}^{M} X_i}{M} > (1+\alpha)p] < e^{-\frac{1}{3}\alpha^2 pM}$$

and

$$Pr[\frac{\sum_{i=1}^{M} X_i}{M} < (1-\alpha)p] < e^{-\frac{1}{2}\alpha^2 pM}$$

4. Learning automata from an infallible expert

In this section we give a version of Angluin's algorithm for PAC learning deterministic finite automata, given access to random labeled examples (distributed according to an arbitrary distribution), and membership queries [2]. We assume all examples and queried strings are labeled correctly. In this version the learning algorithm is given an upper bound n_b on n, the number of states in the target automaton.

We assume the learning algorithm is given access to a source of example strings of maximum length L over a known alphabet Σ. These examples are labeled according to the unknown target automaton A, i.e., for each example the learner is told if A accepts (label 1) or rejects (label 0) that string. These examples are distributed according to a fixed but unknown distribution D. The learner may also ask if specific strings are accepted or rejected by A. The learner is given a bound n_b on the number of states n of A, a confidence parameter $0 < \delta \leq 1$ and an approximation parameter $0 < \epsilon \leq 1$. With probability at least $1 - \delta$ after time polynomial in n_b, L, $|\Sigma|$, $\frac{1}{\delta}$ and $\frac{1}{\epsilon}$ it must output a hypothesis automaton A' such that $Pr_D(\bar{A}'(x) \neq \bar{A}(x)) \leq \epsilon$.

By Occam's Razor Cardinality Lemma [8], in order to output such a hypothesis with probability at least $1 - \delta$, it suffices to find an automaton A' with n' states (where $n' = poly(n_b)$) which *agrees* with A on a set of sample strings of size at least $\frac{1}{\epsilon}(\ln N_{DFA}(n', |\Sigma|) + \ln \frac{1}{\delta})$, where $N_{DFA}(n', |\Sigma|)$ is the number of automata with n' states on an alphabet of size $|\Sigma|$. Since $N_{DFA}(n', |\Sigma|) = 2^{poly(n_b, \Sigma)}$ the sample size is polynomial in the relevant parameters.

The following is a high level description of how we construct such a (consistent) hypothesis automaton A'. Given a sample generated according to D, we partition the set of all sample strings and their prefixes (including the empty string and the strings themselves) into disjoint classes having two simple properties. The first property we require the partition have is that all strings which belong to the same class have the same 0/1 label. We then relate each state in A' to one such class, and let the starting state correspond to the class including the empty string, and the accepting states correspond to classes whose strings are labeled by 1. Since we ask that A' agree with A on all strings in the sample, we would like to define A''s transition function so that all strings in the same class reach the same corresponding state in A'. In order to be able to define a transition function having this property, the partition should also have an additional *consistency* property that is defined precisely in Lemma 1 below.

We would like to point out to the reader who is familiar with Angluin's algorithm, that we remove the third *closure* requirement on the partition (which guarantees that the transition function can be fully defined), and replace it by adding a special *sink state* whose exact usage is described in the proof of Lemma 1. This can be done since our algorithm is a PAC learning algorithm and not an *exact* learning algorithm as Angluin's original algorithm is.

In the next lemma, we formally define the properties of the partition we seek, and show how to define the hypothesis automaton A' based on a given partition having these properties. We later describe precisely how to construct such a partition. Note that in particular, a partition in which strings belong to the same class exactly when they reach the same state in A, has the properties defined in the lemma.

Let $R = \{r_1, r_2, \ldots, r_N\}$ be the set of all prefixes of a given set of m sample strings (including the empty string and the sample strings themselves). For $\sigma \in \Sigma$ let the *σ-successor* of a string r be $r\cdot\sigma$. Then we have the following lemma.

LEMMA 1 *Let $\mathcal{P} = \{C_0, C_1, \ldots, C_{k-1}\}$ be a partition of R into k classes having the following properties:*

1. Labeling*: All strings in each class are labeled the same by A, i.e., $\forall i$ s.t. $0 \leq i \leq k-1$, $\forall r_1, r_2 \in C_i$, $\bar{A}(r_1) = \bar{A}(r_2)$.*
2. Consistency*: For every class C_i and for every symbol $\sigma \in \Sigma$, all σ-successors of the strings in C_i which are in R belong to the same class, i.e., $\forall i$ s.t. $0 \leq i \leq k-1$, $\forall \sigma \in \Sigma$, $\forall r_1, r_2 \in C_i$, if $r_1\cdot\sigma, r_2\cdot\sigma \in R$ and $r_1\cdot\sigma \in C_j$, then $r_2\cdot\sigma \in C_j$.*

Then we can define an automaton A' with $k+1$ states which agrees with A on all the sample strings.

Proof: We define the following automaton $A' = (Q', \Sigma, \tau', q'_0, F')$.

- $Q' = \{C_0, \ldots, C_{k-1}\} \cup \{q_{sink}\}$ where q_{sink} is called the *sink state.*
- $q'_0 = C_i$ such that λ (the empty string) $\in C_i$. Without loss of generality $\lambda \in C_0$.
- The transition function τ': For every class C_i and for every symbol σ, if there exists a string $r \in C_i$ such that $r\cdot\sigma$ is in R and belongs to the class C_j, then $\tau'(C_i, \sigma) = C_j$. Note that in this case $\tau'(C_i, \sigma)$ is uniquely defined due to the consistency property of the partition. If there is no such string r in C_i, then $\tau'(C_i, \sigma) = q_{sink}$. $\tau'(q_{sink}, \sigma) = q_{sink}$ for every symbol σ. Note that if there is no class C_i and symbol σ such that $\tau'(C_i, \sigma) = q_{sink}$, then there is no path in the underlying graph of A' from C_0 to q_{sink}, and q_{sink} is redundant.
- $F' = \{C_i \mid \text{all strings in } C_i \text{ are labeled } 1\}$

By this definition, given any string in the sample, the state corresponding to the class the string belongs to is an accepting state *iff* the string is labeled 1. Hence, in order to prove that A' agrees with A on all sample strings, we show that for every string $r \in R$, if $r \in C_j$ then $\tau'(C_0, r) = C_j$. Note that in particular this means that no string in the sample reaches the sink state and hence the sink state's sole purpose is to allow τ' to be fully defined. We prove the above claim by induction on the length of r. Let $C(r)$ denote the class r belongs to. For $\mid r \mid= 0 : \lambda \in C_0$ and $\tau'(C_0, \lambda) = C(\lambda \cdot \lambda) = C_0$. Assuming the induction hypothesis is true for all r such that $\mid r \mid< l$, we prove it for $\mid r \mid= l$. Let $r = r' \cdot \sigma$, $\sigma \in \Sigma$. Since the set of strings R is prefix closed, $r' \in R$. Since $\mid r' \mid< l$, by induction if $r' \in C_i$ then $\tau'(C_0, r') = C_i$. But according to the definition of τ' and the consistency property of the partition, $\tau'(C_i, \sigma) = C_j$ *iff* the σ successors of *all* strings in C_i belong to C_j, r being one of them. Since the sample strings are a subset of R, we are done. ■

In order to partition the strings and their prefixes into classes which fulfill the above requirements, we construct what Angluin calls an *Observation Table*, denoted by T. The rows of the observation table are labeled by the (prefix closed) set of strings R, and the columns are labeled by a suffix-closed set of strings S. Initially S includes only the empty string λ, and in the course of the construction we add additional strings. For $r \in R$, $s \in S$, the value of the entry in the table related to row r and column s, $T(r, s)$, is $\bar{A}(r \cdot s)$. Let $row(r)$ be the row in the table labeled by r. Then, at each stage of the construction , we can define a partition of R into classes in the following manner: two prefix strings $r_i, r_j \in R$ belong to the same class *iff* $row(r_i) = row(r_j)$.

By this definition and since $\lambda \in S$, all strings which belong to the same class have the same label, and hence such a partition has the labeling property required in Lemma 1. The consistency requirement on the partition translates into the following consistency requirement on the table. For every r_i and r_j in R, and for every $\sigma \in \Sigma$, if $row(r_i) = row(r_j)$ and both $r_i{\cdot}\sigma$ and $r_j{\cdot}\sigma$ are in R, then $row(r_i{\cdot}\sigma)$ must equal $row(r_j \cdot \sigma)$. If the table T is consistent then so is the partition defined according to T. Thus, as mentioned above, we start by initializing S to be $\{\lambda\}$ and filling in this first column. Iteratively, and until the table is consistent, we do the following. If there exist r_i and r_j in R and $\sigma \in \Sigma$ such that $row(r_i) = row(r_j)$, both $r_i{\cdot}\sigma$ and $r_j{\cdot}\sigma$ are in R, and there exists a suffix s in S such that $T(r_i{\cdot}\sigma, s) \neq T(r_j{\cdot}\sigma, s)$, then we add $\sigma \cdot s$ to S and query on all new entries. The pseudo-code for this procedure appears in Procedure *Partition-Sample* (Figure 1).

Two issues we have not discussed yet are the size of A' and the running time of the algorithm. As mentioned in the beginning of this section, we need a bound on the size of A' so that we can apply Occam's Razor Cardinality Lemma. Clearly, the number of classes in a partition induced by T in any iteration of the algorithm is at most n. Otherwise there would be two strings r_i and r_j in R which reach the same state in A, but for which there exists a string s such that $\bar{A}(r_i \cdot s) \neq \bar{A}(r_j \cdot s)$. Therefore the number of states in A' is at most $n + 1 \leq n_b + 1$. But this also means that the size of S is less than n, since each suffix added to S refines the partition. Thus the algorithm is polynomial in the relevant parameters, as required.[2]

```
Procedure Partition-Sample()

Initialization:
    let $m = \frac{1}{\epsilon}(\ln N_{DFA}(n_b + 1, |\Sigma|) + \ln \frac{1}{\delta})$
    let $R = \{r_1, r_2, \ldots, r_N\}$ be the set of all prefixes of $m$ randomly generated
        sample strings
    $S \leftarrow \{\lambda\}$
    query all strings in $R$ to fill in the first column (labeled by $\lambda$) in $T$
while table is not consistent:
    $\forall r_i, r_j \in R$ s.t. $\forall s \in S$ $T(r_i, s) = T(r_j, s)$
        if $\exists \sigma \in \Sigma$ s.t. $[r_i \cdot \sigma, r_j \cdot \sigma \in R$ and $\exists s \in S$, s.t. $T(r_i \cdot \sigma, s) \neq T(r_j \cdot \sigma, s)]$ then do
            $S \leftarrow S \cup \{\sigma \cdot s\}$
            query all strings in $R \circ \{\sigma \cdot s\}$ to fill in new column (labeled by $\sigma \cdot s$) in $T$
    { else table is consistent }
```

Figure 1. Procedure *Partition-Sample* (Error-free Case)

Let us summarize this section. We started with the following trivial observation. Given a bound n_b on the number of states of the target automaton, the number of automata with that size is $2^{poly(n_b, |\Sigma|)}$, and hence we may apply Occam's Razor Cardinality Lemma. We then show that if we can partition a given set of sample strings and all their prefixes into k disjoint classes which have the properties defined in Lemma 2, then we can define an automaton with $k + 1$ states that agrees with the target automaton on all the strings in the sample. We conclude by describing how to efficiently construct such a partition with at most $k = n_b$ classes.

5. Overview of the Learning Algorithm

We start with a short overview of the learning algorithm described in Section 6. The final goal of the algorithm is to reach a partition (of a large set of sample strings and their prefixes) which has similar properties to those defined in Lemma 1. Based on this partition we construct our hypothesis automaton. The partition achieved is *consistent* (as defined in Lemma 1), but it has a slightly modified version of the *labeling* property (defined in the same lemma). Namely, we relate a 0/1 label with each class, and show that the true label of most strings is the same as the label of their class.

Consequently, the hypothesis automaton constructed based on this partition agrees with the target automaton A on all but a small fraction (no more than $\epsilon/2$) of the sample strings. The number of classes in the partition and hence the number of states in the hypothesis is bounded by $\theta \ln m$ where θ is a polynomial in $n_b, \frac{1}{\gamma_b}, |\Sigma|, L, \frac{1}{\epsilon}$ and $\ln \frac{1}{\delta}$, and m is the size of the sample. We then (in Section 7) use an Occam's Razor-like claim to prove that the hypothesis automaton is an ϵ-good hypothesis with respect to A. In Subsection 6.2.1 and in Section 8 we describe two example runs of the algorithm.

We present the algorithm stage by stage, and show that each stage can be completed successfully with high probability. The stages of the algorithm are as follows.

1. We compute an estimate of the expert's error probability, η (Subsection 6.1).

2. We generate a set of sample strings according to D_L, and partition all sample strings and their prefixes into disjoint classes, according to their (and some additional strings') *observed behavior* on a large set of suffixes of length logarithmic in n_b (Subsection 6.2). With high probability this initial partition is consistent, and the number of classes in the partition is at most n. A refinement of these classes will correspond to the states in the hypothesis automaton.

3. We further refine the initial partition, and label the classes of the resulting (final) partition. We show that the final partition is consistent and that with high probabilty the correct label of most sample strings is the same as the label of the class they belong to. We determine the labels of the classes in the final partition using the following property of the initial partition. In the initial partition, strings which are in the same class *truly behave the same* on most suffixes among those they were tested on in the previous stage.

6. The Learning Algorithm

In the previous section we stated that the final goal of our algorithm is to reach a partition of a given set of sample strings and their prefixes which has similar properties to those defined in Lemma 1, and based on this partition construct our hypothesis automaton. We shall now be more precise with respect to the properties of the partition and the constructed automaton.

As before let $R = \{r_1, r_2, \ldots, r_N\}$ be the set of all prefixes of m given sample strings (including the empty string and the sample strings themselves).

LEMMA 2 *Let $\mathcal{P} = \{C_0, C_1, \ldots, C_{k-1}\}$ be a partition of R into k classes each labeled 0 or 1 having the following properties:*

1. Labeling*: All but at most $\epsilon/2$ of the* sample *strings have the same label according to A as the label of their class.*

2. Consistency*: For every class C_i and for every symbol $\sigma \in \Sigma$, all σ-successors of the strings in C_i which are in R belong to the same class.*

Then we can define an automaton A' with $k+1$ states which agrees with A on all but at most $\epsilon/2$ of the sample strings.

Proof: A' is defined as in Lemma 1, only its accepting states correspond to classes labeled 1. The sample strings on which A and A' differ are exactly those whose label according to A differs from the label of their class, and their fraction is bounded by $\epsilon/2$. ■

Before we embark upon a detailed description of how we reach a partition having the properties defined in Lemma 2, we add the following definitions. The first is based on terms defined in Section 3.

Definition. Let u_1 and u_2 be strings and let V be a set of (suffix) strings. The **true difference rate** of u_1 and u_2 on U is the fraction of strings in V on which u_1 and u_2 truly differ. Their **observed difference rate** is the fraction of strings on which there is an observed difference.

If δ is the learning success parameter then $\delta' \stackrel{\text{def}}{=} \delta/5$. At each stage in the algorithm we bound the probability our algorithm has erred in that stage by δ'. Our total error probability is bounded by δ. Our errors have two independent sources: errors caused by our interaction with a fallible (as opposed to infallible) expert, and errors due to our generation of a random sample. Most of our probabilistic claims concern the first source, and it is self-evident which (two) claims deal with the latter. In the various stages of the algorithm we refer to several parameters, namely m, l_1, and l_2. Their values are set below.

Changes in Pars (will recheck)

$$m = \frac{2^{14}|\Sigma|^2 n_b^8}{\epsilon^4\gamma_b^6}\cdot\ln^3\frac{2^{14}|\Sigma|^2 n_b^8 L^2}{\epsilon^4\gamma_b^6\delta}, \qquad (1)$$

$$l_1 = \left\lceil \frac{1}{\ln|\Sigma|}\cdot\ln\left[\frac{2^7 n_b^4}{\epsilon^2\gamma_b^4}\cdot\ln\frac{10(n_b^2+1)}{\delta}\right]\right\rceil \quad \text{and} \qquad (2)$$

$$l_2 = \left\lceil \frac{1}{\ln|\Sigma|}\cdot\ln\left[\frac{2^7 n_b^6}{\epsilon^2\gamma_b^4}\cdot\ln\frac{20m^2L^2|\Sigma|^2}{\delta}\right]\right\rceil. \qquad (3)$$

6.1. Estimating the expert's error-probability

In this section we compute an estimate of the expert's error probability. Since the learning algorithm is only given an upper bound, $1/2-\gamma_b$, on the error rate of the expert, η, it needs to compute a more exact approximation of η. This approximation is used in later stages of the algorithm.

The basic idea is the following. If two strings reach the same state in A, then any observed difference in their behavior on any set of suffixes is due only to erroneous answers given by the expert. If two strings reach different states then the observed difference in their behavior is due to the expert's errors and any differences in their correct behavior on those suffixes. We show that for every pair of strings, and for any set of suffixes V, if both strings reach the same state then their expected observed difference rate on V is a simple function of the expert's error probability, namely $2\eta(1-\eta)$, and if they reach different states, it is bounded below by this function. Since there are at most n_b states, given more than n_b strings, at least two must reach the same state.

Using the fact that the errors are independently distributed, we estimate the expert's error probability by looking at the minimum observed difference rate between all pairs of strings (among those chosen) on a large set of suffixes. We assume that the pair of strings which gives the minimal value reach the same state, and calculate the error probability that would generate such an observed difference rate. The above is described precisely in Procedure *Estimate-Error* appearing in Figure 2.

```
Procedure Estimate-Error()
let $W = \{w_1, ..., w_{n_b+1}\}$ be any (arbitrary) set of $n_b + 1$ strings
let $V_1$ be all strings of length $l_1$ over $\Sigma$
query the expert on all strings in $W \circ V_1$
for each pair $w_i \neq w_j$ in $W$, compute their observed difference rate on $V_1$:
    let $\Delta_{ij} \leftarrow \sum_{v \in V_1} (\mathcal{E}(w_i \cdot v) \oplus \mathcal{E}(w_j \cdot v))/|V_1|$
let $\Delta = \min_{i,j} \Delta_{ij}$
if $\Delta > 1/2$ then halt and output error
let $\tilde{\eta}$ be the solution to $\Delta = 2\tilde{\eta}(1-\tilde{\eta})$ such that $\tilde{\eta} \leq 1/2$
```

Figure 2. Procedure *Estimate-Error*

In the following lemma we claim that with high probability Δ is a good estimate of $2\eta(1-\eta)$ and $\tilde{\eta}$ is a good estimate of η.

LEMMA 3 *Let* $\rho = \sqrt{\frac{1}{2}|\Sigma|^{-l_1} \ln 2(n_b+1)^2/\delta'}$. *Then*

1. $Pr[|\Delta - 2\eta(1-\eta)| > \rho] < \delta'$.

2. *If* $|\Delta - 2\eta(1-\eta)| \leq \rho$ *then* $|\tilde{\eta} - \eta| \leq \rho/(2\gamma_b)$.

In order to prove Lemma 3 we need the following two observations.

OBSERVATION 1 *For any given pair of* different *strings* u_1 *and* u_2, *and for any given (suffix) string* v:

1. *If* $\bar{A}(u_1 \cdot v) = \bar{A}(u_2 \cdot v)$, *then* $Pr[\mathcal{E}(u_1 \cdot v) \neq \mathcal{E}(u_1 \cdot v)] = 2\eta(1-\eta)$.

2. *If* $\bar{A}(u_1 \cdot v) \neq \bar{A}(u_2 \cdot v)$, *then* $Pr[\mathcal{E}(u_1 \cdot v) \neq \mathcal{E}(u_1 \cdot v)] = (1-\eta)^2 + \eta^2$.

Hence, if V *is any given set of (suffix) strings, and the fraction of strings in* V *on which* u_1 *and* u_2 *truly differ is* β, *then their expected observed difference rate on* V *is*

$$\begin{aligned} & (1-\beta)\cdot[2\eta(1-\eta)] \;+\; \beta\cdot[(1-\eta)^2+\eta^2] & (4) \\ =\; & 2\eta(1-\eta) \;+\; \beta(1-2\eta)^2. & (5) \end{aligned}$$

OBSERVATION 2 *If u_1 and u_2 are two* different *strings, and V is a set of (suffix) strings* all of the same length, *then for every two suffixes $v_i, v_j \in V$, for k and $l \in \{1,2\}$, $u_k \cdot v_i \neq u_l \cdot v_j$ unless both $k = l$ and $v_i = v_j$. Based on the above and the independence of the noise, for any $v_i \in V$, the event that $\mathcal{E}(u_1 \cdot v_i) \neq \mathcal{E}(u_2 \cdot v_i)$ is independent of the event that $\mathcal{E}(u_1 \cdot v_j) \neq \mathcal{E}(u_2 \cdot v_j)$, for all $j \neq i$.*

Proof of Lemma 3: *1st Claim*: According to Observation 1, for any pair of (different) strings w_i and w_j in W, if w_i and w_j reach the same state in A, then for every string v in V_1, the probability that $\mathcal{E}(w_i \cdot v)$ differs from $\mathcal{E}(w_j \cdot v)$ is $2\eta(1-\eta)$. Thus, according to Inequality 1 and Observation 2

$$Pr[\Delta_{ij} - 2\eta(1-\eta) > \rho] \; < \; e^{-2\rho^2|V_1|} \tag{6}$$

$$= \; e^{-|\Sigma|^{-l_1} \ln \frac{2(n_b+1)^2}{\delta'} |V_1|} \tag{7}$$

$$= \; \frac{\delta'}{2(n_b+1)^2}. \tag{8}$$

Similarly

$$Pr[2\eta(1-\eta) - \Delta_{ij} > \rho] \; < \; \frac{\delta'}{2(n_b+1)^2}. \tag{9}$$

If w_i, w_j reach different states, then for each suffix string v in V the probability that a difference is observed between w_i and w_j on v is at least $2\eta(1-\eta)$. Thus $Pr[2\eta(1-\eta) - \Delta_{ij} > \rho] < \delta'/2(n_b+1)^2$.

We now bound separately the probability that Δ is an overestimate of $2\eta(1-\eta)$, and the probability that it is an underestimate of $2\eta(1-\eta)$. What is the probability that $\Delta > 2\eta(1-\eta) + \rho$? Because Δ was set to be the minimum value of all Δ_{ij}s, this event occurs only if for *all* i, j, $\Delta_{ij} > 2\eta(1-\eta) + \rho$. Since $(n_b + 1) \geq n + 1$, there are at least two strings w_k and w_l that reach the same state in A and hence

$$Pr[\Delta - 2\eta(1-\eta) > \rho] \; \leq \; Pr[\Delta_{kl} - 2\eta(1-\eta) > \rho] \tag{10}$$

$$< \; \frac{\delta'}{2(n_b+1)^2} \; < \; \frac{\delta'}{2}. \tag{11}$$

In order to underestimate $2\eta(1-\eta)$, it suffices that for one pair of strings the observed difference rate is too small. Since there are less than $(n_b+1)^2$ such pairs,

$$Pr[2\eta(1-\eta) - \Delta > \rho] \; = \; Pr[\exists i, j \text{ s.t. } 2\eta(1-\eta) - \Delta_{ij} > \rho] \tag{12}$$

$$< \; (n_b+1)^2 \cdot \frac{\delta'}{2(n_b+1)^2} \; = \; \frac{\delta'}{2}, \tag{13}$$

and we have proved the first claim.

2nd Claim: Assume in contradiction that $|\tilde{\eta} - \eta| > \rho/(2\gamma_b)$. If $\tilde{\eta} > \eta + \rho/(2\gamma_b)$ then since $\tilde{\eta}$ was defined to be at most $1/2$ and $2\tilde{\eta}(1-\tilde{\eta})$ is an increasing function in the range between 0 and $1/2$,

$$\Delta \; = \; 2\tilde{\eta}(1-\tilde{\eta}) \tag{14}$$

$$\begin{aligned} &> 2(\eta + \frac{\rho}{2\gamma_b})(1 - \eta - \frac{\rho}{2\gamma_b}) && (15) \\ &= 2\eta(1-\eta) + (1-2\eta)\frac{\rho}{\gamma_b} - \frac{\rho^2}{2\gamma_b^2}. && (16) \end{aligned}$$

Since $\eta \leq 1/2 - \gamma_b$ we get that

$$\Delta > 2\eta(1-\eta) + 2\rho - \frac{\rho^2}{2\gamma_b^2}. \qquad (17)$$

It is easily verified by substituting the value of l_1 in the definition of ρ that $\rho < 2\gamma_b^2$ and thus $\Delta > 2\eta(1-\eta) + \rho$ contradicting the assumption in the statement of the lemma.

If $\tilde{\eta} < \eta - \rho/(2\gamma_b)$ then

$$\begin{aligned} \Delta &= 2\tilde{\eta}(1-\tilde{\eta}) && (18) \\ &< 2(\eta - \frac{\rho}{2\gamma_b})(1 - \eta + \frac{\rho}{2\gamma_b}) && (19) \\ &= 2\eta(1-\eta) - (1-2\eta)\frac{\rho}{\gamma_b} - \frac{\rho^2}{2\gamma_b^2} && (20) \\ &\leq 2\eta(1-\eta) - 2\rho - \frac{\rho^2}{2\gamma_b^2} && (21) \\ &< 2\eta(1-\eta) - \rho, && (22) \end{aligned}$$

which again contradicts the assumption. ■

In the following stages of our exposition we assume that in fact Δ estimates $2\eta(1-\eta)$ within an additive factor of ρ, and that $\tilde{\eta}$ estimates η within an additive factor of $\rho/(2\gamma_b)$. The probability that this is not true is taken into account in the final analysis.

6.2. Initial partitioning by subsequent behavior

In the second stage of the algorithm, described in this subsection, we make our first step towards reaching a partition which has the properties defined in Lemma 2. By the end of this stage we are able to define (with high probability) an initial *consistent* partition $\mathcal{P}_{int}$ of a set of sample strings and their prefixes into at most n classes. Each class might include strings which reach different states in the target automaton A, but strings which reach the same state are not separated into different classes. We show that $\mathcal{P}_{int}$ has an additional property which is used in the next stage of the algorithm when the partition is further refined.

In the partitioning algorithm for the error-free case (described in Section 4), the set of sample strings and their prefixes, R, is first partitioned according to the labels of the strings (their behavior on the empty suffix). If all strings have the same label then we have a consistent partition composed of a single class. Otherwise, starting from a partition composed of two classes (a '1' class and a '0' class), we try and reach consistency by further refining the partition. Whenever an inconsistency is detected, i.e., there are two strings r_i and r_j in R which belong to the same class, but there exists a symbol σ such that $r_i{\cdot}\sigma$ and $r_j{\cdot}\sigma$ differ on some suffix s and hence belong to different classes, then we have *evidence* that r_i and r_j should belong to different classes. By adding the suffix $\sigma{\cdot}s$ to S and querying all strings in $R\circ\{\sigma{\cdot}s\}$ to fill in the new column in the Observation Table T, we automatically refine the partition.

As noted in Section 4, the difference in behavior between r_i and r_j on $\sigma \cdot s$ is evidence that the two strings reach different states in A, and thus in this process we *never* separate strings which reach the same state into different classes (though strings which reach different states might belong to the same class). In the presence of errors however, a difference in the observed behavior between two strings on a specific suffix, and in particular on the empty suffix (their observed labels), does not necessarily mean that they reach different states. Hence we must find a different procedure to differentiate between strings that reach different states, and then show how to use this procedure in our quest for a consistent partition.

In the previous section we observed (Observation 1), that for any pair of strings and for any set of suffixes V, if both strings reach the same state in A then their expected observed difference rate on V is $2\eta(1-\eta)$, and if they reach different states, it is bounded below by this value. The larger the true difference rate between the strings on the set of suffixes is, the larger the expected observed difference is. Thus, since we have a good estimate, Δ, of $2\eta(1-\eta)$, if the set of suffixes, V, is large enough, then with high probability we are able to differentiate between strings which reach states in A whose true difference rate on V is substantial. This idea is applied in the most basic building block of our algorithm, described in Function *Strings-Test* (Figure 6). This function is given as input two strings, and it returns different if there is a substantial observed difference rate between the two strings on a predefined set of equal length strings V_2, and similar otherwise.

Begin change to graphs (practically till analysis)

As a consequence, given a set of strings U, we can define an undirected graph $G(U)$, called a *similarity* graph. The nodes of $G(U)$ are the strings in U, and there is an edge between every pair of nodes (strings) for which Function *Strings-Test* returns similar. We show that $G(U)$ has the following properties (with high probability):

1. Strings in U that reach the same state in A are in the same connected component in $G(U)$.
2. For each connected component ϕ in $G(U)$, the fraction of strings v in V_2 for which there exist two strings u and u' which belong to ϕ but for which $\bar{A}(u{\cdot}v) \neq \bar{A}(u'{\cdot}v)$, is small.

We refer to the these properties as the *first* and the *second properties of similarity graphs*. Given a similarity graph $G(U)$ having these properties, and a new string $u \notin U$, we can add u to the graph by putting an edge between u and all strings $u' \in U$ such that *Strings-Test*$(u, u') =$ similar. We show that with high probability the resulting graph $G(U \cup \{u\})$ has both properties of similarity graphs. We next discuss the type of Observation Table constructed in this stage, and describe how similarity graphs are used in its construction.

In the error-free case, the algorithm (Procedure *Partition-Sample*) constructs a data structure in the form of an Observation Table T. In this stage we construct (in Procedure *Partition-Erroneous-Sample-1* – Figure 3) a similar table structure T_1. As in the error-free case, the rows of the table are labeled by the prefix closed set R of all sample strings and their prefixes, and the columns are labeled by a suffix closed set of strings S. Initially S includes only the empty string λ, and in the course of the construction we add additional strings. The difference between T and T_1 is that the entrees of T are 0/1 valued, where for $r \in R$ and $s \in S$, $T(r,s) = \bar{A}(r \cdot s)$, while the entrees in T_1 are names of connected components in the current similarity graph $G(R \circ S)$. The entry $T(r,s)$ is the name of the connected component which $r \cdot s$ belongs to in $G(R \circ S)$, denoted by $\phi_{G(R \circ S)}(r \cdot s)$. Equivalently to the error-free case, if $row(r)$ is the row in T_1 labeled by r, then, at each stage of the construction, we can define a partition $\mathcal{P}$ of R into classes in the following manner: two strings $r_1, r_2 \in R$ belong to the same class in $\mathcal{P}$ *iff* $row(r_1) = row(r_2)$. T_1 and the corresponding partition $\mathcal{P}$ are consistent, *iff*, for every r_1 and r_2 in R, and for every $\sigma \in \Sigma$, if $row(r_1) = row(r_2)$ and both $r_1 \cdot \sigma$ and $r_2 \cdot \sigma$ are in R, then $row(r_1 \cdot \sigma)$ equals $row(r_2 \cdot \sigma)$.

Thus, in order to achieve a consistent partition, we begin by calling Procedure *Intialize-Graph* (Figure 4) which constructs the graph $G(R)$. This procedure simply starts with a similarity graph $G(\{r_1\})$ consisting of a single node $r_1 \in R$, and adds all other strings in R to the graph by calling Procedure *Update-Graph* (Figure 5) on each new string. For every $r \in R$ we let $T_1(r, \lambda) = \phi_{G(R)}(r)$. At this stage we have a similarity graph which is defined on R, but it shall be extended to be defined on the growing superset of R, namely $R \circ S$.

Iteratively, and until the table is consistent, we do the following. If there exist two strings r_i and r_j in R and a symbol $\sigma \in \Sigma$ such that $row(r_i) = row(r_j)$, both $r_i \cdot \sigma$ and $r_j \cdot \sigma$ are in R, and there exists a suffix s in S such that $T(r_i \cdot \sigma, s) \neq T(r_j \cdot \sigma, s)$, then we add $\sigma \cdot s$ to S and fill in the new column in T_1 by determining the connected component in the similarity graph of every string in $R \circ \{\sigma \cdot s\}$. If a string u in $R \circ \{\sigma \cdot s\}$ was in $R \circ S$ before $\sigma \cdot s$ was added to S, then its connected component is known. Otherwise, we add u to the graph and simply put an edge between u and every other node u' in the graph such that *Strings-Test*$(u, u') =$ similar. This is done by calling Procedure *Update-Graph* on u. If u adds a new (single node) connected component to the graph, or if it is added to a single existing connected component, then we just fill in the new entry with the name of this component. If it causes several different connected components in the graph to be merged into one connected component, then we need to update T_1, so that all appearances of the old components are changed into the new one.

If strings that reach the same state in A always belong to the same connected component, then the number of times components are merged is at most n, and the total number of columns in T_1 is at most n^2. If at any stage the number of classes in the partition defined according to T_1 is larger than n_b, or the number of columns in T_1 exceeds n_b^2, then we know we have erred and we halt. Assuming that Function *Strings-Test* always returns similar when called on pairs of strings that reach the same state in A, strings that reach the same state in A always belong to the same connected component, and the similarity graphs defined by the algorithm always have the first property of similarity graphs. However, pairs of strings for which *Strings-Test* returns different since the observed difference rates between the two strings on the set of suffixes V_2 is substantial, might also belong to the same connected component due to merging of components. Nonetheless, we show that these mergings of components do not greatly affect the second property of similarity graphs.

```
Procedure Partition-Erroneous-Sample-1()

Initialization:
  let $R = \{r_1, r_2, \ldots, r_N\}$ be the set of all prefixes of $m$ sample strings
     generated according to $D_L$
  $S \leftarrow \{\lambda\}$
  call Initialize-Graph() to construct $G(R)$
  fill in the first column of $T_1$ according to $G(R)$:
     for every $r \in R$, $T_1(r, \lambda) \leftarrow \phi_{G(R)}(r)$
  if the number of connected components in $G(R)$ is larger than $n_b$ then
     halt and output error.
while table is not consistent:
  $\forall r_i, r_j \in R$ s.t. $\forall s \in S$ $T(r_i, s) = T(r_j, s)$
     if $\exists \sigma \in \Sigma$ s.t. $[r_i \cdot \sigma, r_j \cdot \sigma \in R$ and $\exists s \in S$, s.t. $T(r_i \cdot \sigma, s) \neq T(r_j \cdot \sigma, s)]$ then do
        $S \leftarrow S \cup \{\sigma \cdot s\}$
        for every $r \in R$
              call Update-Graph($r \cdot \sigma \cdot s$) and let $G$ be the current similarity graph
              if any connected components were merged then
                 update respective entries in $T_1$
              $T_1(r, \sigma \cdot s) \leftarrow \phi_G(r \cdot \sigma \cdot s)$
        if the number of classes in the partition defined according to $T_1$
        is larger than $n_b$, or if $|S| > n_b^2$ then
           halt and output error
  { else table is consistent }
```

Figure 3. Procedure *Partition-Erroneous-Sample-1* (Initial Partition)

revised def. For ease of the analysis, we define

$$\beta_{max} \stackrel{\rm def}{=} (1 - 2\eta)^{-2} \left[\sqrt{2|\Sigma|^{-l_2}(2n_b^2 \ln |\Sigma| + \ln \frac{4N^2}{\delta'})} + 2\rho \right], \tag{23}$$

```
Procedure Initialize-Graph()

initialize the similarity graph to be the single node graph $G(\{r_1\})$
$U \leftarrow \{r_1\}$ ($U$ is the set of strings the similarity graph is defined on)
for $i = 2$ to $N$ do
    call Update-Graph($r_i$) to add $r_i$ to similarity graph
    $U \leftarrow U \cup \{r_i\}$
```

Figure 4. Function *Initialize-Graph*

```
Procedure Update-Graph($u$)

if $u \notin U$ then do
    $\mathsf{sim}(u) \leftarrow \{u' \mid u' \in U,$ Strings-Test($u$,$u'$) $=$ similar $\}$
    add $u$ to similarity graph and
    put an edge between $u$ and every $u' \in \mathsf{sim}(u)$
    $U \leftarrow U \cup \{u\}$
{ else $u$ is already in the similarity graph }
```

Figure 5. Procedure *Update-Graph*

```
Function Strings-Test($u_1, u_2$)

let $V_2$ be the set of all strings of length $l_2$ (over $\Sigma$)
let $\alpha_1 \leftarrow \sqrt{\frac{1}{2}|\Sigma|^{-l_2}(2n_b^2 \ln|\Sigma| + \ln\frac{4N^2}{\delta'})} + \rho$
query the expert on all strings (not previously queried) in
    $\{u_1\} \circ V_2$ and $\{u_2\} \circ V_2$
let $\Delta_{u_1,u_2} \leftarrow \sum_{v \in V_2}[\mathcal{E}(u_1 \cdot v) \oplus \mathcal{E}(u_2 \cdot v)]/|V_2|$
if $\Delta_{u_1,u_2} > \Delta + \alpha_1$ then return different
else return similar
```

Figure 6. Function *Strings-Test*

where ρ is defined in Lemma 3.

LEMMA 4 *Procedure* Partition-Erroneous-Sample-1 *always terminates, and with probability at least* $1-\delta'$*, the partition* $\mathcal{P}_{int}$ *defined according to* T_1 *upon termination, has the following properties:*

1. $\mathcal{P}_{int}$ *is consistent (as defined in Lemma 2).*
2. *Strings that reach the same state in* A *belong to the same class in* $\mathcal{P}_{int}$*;*
3. *For each class* C *in* $\mathcal{P}_{int}$*, the fraction of suffixes* v *in* V_2 *on which there exist any two strings in* C *that truly differ on* v *is at most* $n \cdot \beta_{max}$ *(where* β_{max} *is defined in Equation 23).*

We start by proving a simple claim regarding the correctness of *Strings-Test*. Let us first define what we mean when we say that the function is *correct*.

We say that *Strings-Test* is *correct* with respect to a pair of strings u_1 and u_2 it is called on if the following holds:

1. If u_1 and u_2 reach the same state in A, then *Strings-Test*(u_1,u_2) returns similar;
2. If u_1 and u_2 reach different states in A and the fraction of suffixes in V_2 on which they truly differ is larger than β_{max} then *Strings-Test*(u_1,u_2) returns different;

Otherwise it is *incorrect*. If u_1 and u_2 reach different states in A and the fraction of suffixes in V_2 on which they truly differ is at most β_{max} then the function is correct both if it returns similar and if it returns different.

new calculations only

LEMMA 5 *For* any *given pair of strings* u_1 *and* u_2*, the probability* Strings-Test *is* correct *with respect to* u_1 *and* u_2 *is at least* $1-\delta'/(4N^2|\Sigma|^{2n_b^2})$*.*

Proof: If u_1 and u_2 reach the same state in A, then as observed in Observation 1, for every string v in V_2, the probability that $\mathcal{E}(u_1 \cdot v)$ differs from $\mathcal{E}(u_2 \cdot v)$, is $2\eta(1-\eta)$. Recall that Δ is the estimate of $2\eta(1-\eta)$, and according to our assumption $\Delta \geq 2\eta(1-\eta)-\rho$. Thus based on Inequality 1 and Observation 2

$$\begin{aligned}
&Pr[\textit{Strings-Test}(u_1,u_2) = \text{different}] \\
&\quad = Pr[\Delta_{u_1,u_2} > \Delta + \alpha_1] && (24)\\
&\quad \leq Pr[\Delta_{u_1,u_2} - 2\eta(1-\eta) > \alpha_1 - \rho] && (25)\\
&\quad < e^{-2(\alpha_1-\rho)^2|V_2|} && (26)\\
&\quad = \delta'/(4N^2|\Sigma|^{2n_b^2}), && (27)
\end{aligned}$$

and hence with probability at least $1-\delta'/(4N^2|\Sigma|^{2n_b^2})$, *Strings-Test*$(u_1,u_2)$ returns similar;

If u_1 and u_2 reach different states in A and the fraction of suffixes on which they truly differ in V_2 is greater than β_{max}, then according to Observation 1, the expected observed difference rate between u_1 and u_2 on V_2 is greater than $2\eta(1-\eta)+\beta_{max}(1-2\eta)^2$. Therefore

$$\begin{aligned} & Pr[\textit{Strings-Test}(u_1,u_2) \;=\; \textsf{similar}] \\ = \;& Pr[\Delta_{u_1,u_2} \le \Delta + \alpha_1] && (28)\\ \le \;& Pr[E(\Delta_{u_1,u_2}) - \Delta_{u_1,u_2} > \beta_{max}(1-2\eta)^2 - \alpha_1 - \rho] && (29)\\ < \;& \delta'/(4N^2|\Sigma|^{2n_b^2}), && (30)\end{aligned}$$

and hence with probability at least $1-\delta'/(4N^2|\Sigma|^{2n_b^2})$, *Strings-Test*$(u_1,u_2)$ returns different. ■

Proof of Lemma 4: Procedure *Partition-Erroneous-Sample-1* terminates either when the table is consistent, or when the number of classes in the partition defined by T_1 is larger than n_b, or when the number of suffixes in S is larger than n_b^2. Since each time inconsistency is detected we add a new suffix to S, if the procedure does not terminate due to the first reason mentioned above, it must terminate due to the third reason, and hence it always terminates.

In order to prove that with probability at least $1-\delta'$, $\mathcal{P}_{int}$ has the properties defined in the lemma, we show that if Function *Strings-Test* is correct with respect to every pair of strings it is called on, then $\mathcal{P}_{int}$ must have these properties. We would have liked to bound the probability that *Strings-Test* is correct with respect to every pair of strings it is called on, simply by the number of pairs of strings it is called on, times the bound given in Lemma 5 on the probability it errs on one pair. However, since the pairs of strings *Strings-Test* is called on are not all chosen prior to receiving any of the experts labels, but rather are chosen dynamically, where the choice of a new pair depends on previous answers given by the expert, we may not use this simple bound. Instead, we need to consider all possible pairs of strings *Strings-Test* may be called on, given our choice of R. Let $\bar{S}$ be the set of all strings over Σ of length at most n_b^2. Since the size of S does not exceed n_b^2, and since every suffix added to S is at most one symbol longer than the longest suffix already in S, all strings in S have length at most n_b^2. Hence, S is always a subset of $\bar{S}$. Let $\mathcal{D}(R) \overset{\text{def}}{=} \{R\circ\bar{S}\}\times\{R\circ\bar{S}\}$. Then the set of pairs of strings which *Strings-Test* is called on is always a subset of $\mathcal{D}(R)$. Since the size of $\mathcal{D}(R)$ is at most $(N\cdot 2\cdot|\Sigma|^{n_b^2})^2$, and by applying Lemma 5, the probability that *Strings-Test* is correct on all pairs in $\mathcal{D}(R)$, (which are all possible pairs it may be called on given R), is at least $1-\delta'$.

change in analysis

end of prob anal, but back to graphs

From now on we assume that *Strings-Test* is correct with respect to all pairs of strings it is called on. We refer to this assumption in the next steps of this proof as the *correctness assumption*. Based on the correctness assumption we now prove that $\mathcal{P}_{int}$ has all three properties defined in the lemma.

2nd Property: Based on the correctness assumption, and the construction of the similarity graphs, strings which reach the same state always belong to the same connected component. Since two strings r_i and r_j in R belong to different classes in $\mathcal{P}_{int}$ only if for some suffix s in S, $r_i\cdot s$ and $r_j\cdot s$ belong to different components (and thus reach different states), then the following must also be true. At any stage in the procedure, strings in R that reach the same state, belong to the same class (in the partition define according to T_1 at that stage). Thus, in particular, $\mathcal{P}_{int}$ has the second property defined in the lemma.

1st Property: In order to prove that $\mathcal{P}_{int}$ is consistent we must show that under the correctness assumption, Procedure *Partition-Erroneous-Sample-1* does not terminate with an error message, which means that it must terminate "naturally" when T_1, and hence $\mathcal{P}_{int}$, are consistent. We have just shown in the previous paragraph that the number of classes in the partition defined in any stage of the procedure, is at most n, which is bounded by n_b. Hence the procedure does not halt due to the number of classes being larger than n_b. It remains to show that the number of suffixes added to S is no larger than n_b^2.

Each time connected components are merged, the number of components decreases by at least 1. Since the number of components at any stage can be no larger than n, components are merged at most $n-1$ times. (If all strings belong to the same component then we necessarily have a consistent partition). Every time a suffix is added and no components are merged, then the partition is refined, and the number of classes grows by at least 1. Hence, between every two merges of components we can add at most $n-1$ suffixes to S, and the total number of suffixes in S is bounded by n_b^2.

3rd Property: We prove that this property holds after every call to *Update-Graph*, and for each set of strings that belong to the same connected component at that stage. Since strings in R which belong to the same class belong to the same connected component, the claim follows.

For each connected component ϕ we define the following undirected graph G_A^ϕ. The nodes in G_A^ϕ are states in A reached by strings belonging to ϕ. We put an edge between two states q and q' *iff* there is an edge in ϕ between some pair of strings u and u' such that u reaches q in A and u' reaches q'. Since ϕ is a connected component, G_A^ϕ must be connected as well.

Given one such graph G_A^ϕ, we look at any arbitrary spanning tree of the graph. For each edge (q_1, q_2) in the tree, let $D(q_1, q_2)$ be the subset of strings in V_2 on which q_1 and q_2 truly differ. Under the correctness assumption, $|D(q_1, q_2)| \leq \beta_{max} \cdot |V_2|$. Let u and u' be two strings in ϕ which reach q and q', respectively. Let $q = q_1, q_2, \ldots, q_l = q'$, be a path in the tree between q and q'. Then all the suffixes in V_2 on which u and u' truly differ belong to the $\bigcup_{i=1}^{l-1} D(q_i, q_{i+1})$. Hence, all the suffixes v in V_2 such that there exist any two strings in ϕ that truly differ on v must belong to the union of $D(q_i, q_j)$ over all edges (q_i, q_j) in the spanning tree of G_A^ϕ. Since the number of nodes in G_A^ϕ is at most n, so are the number of edges in any spanning tree of the graph, and the claim follows. ■

We assume from now until the final analysis that $\mathcal{P}_{int}$ has the properties defined in Lemma 4.

6.2.1. Two Examples

In this subsection we begin to describe two example runs of our algorithm. We complete this description in Section 8 after we present the next and final stage of the algorithm.

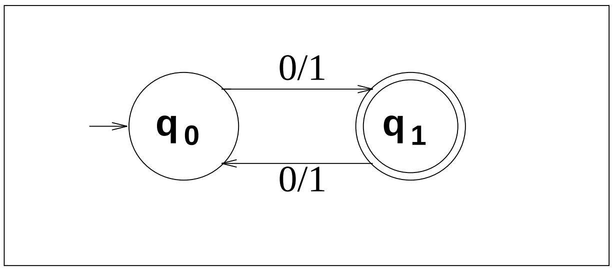

Figure 7. First example target automaton. q_1 is the single accepting state.

In the first example, the target automaton is a two state automaton over the alphabet $\{0,1\}$, which accepts all strings of odd length (and rejects all strings of even length). It is depicted in Figure 7. We now describe what happens in the initial partitioning. For every pair of strings r_i and r_j in R such that r_i reaches q_0 and r_j reaches q_1, r_i and r_j differ on *all* strings in V_2. Hence, with high probability, all strings in R that reach q_0 initially are in the same connected component in $G(R)$, and all strings that reach q_1 are in a different connected component. After filling in the first column in T_1 labeled by λ, we already have a consistent partition into two classes C_0 and C_1, where all strings in C_0 reach q_0 and all strings in C_1 reach q_1. Since the classes in this partition exactly correspond to the states of the target automaton, there exists a labeling of the classes that has the labeling property defined in Lemma 2. However, in the next section we first further refine the partition before labeling the classes.

In the second example, the target automaton is a five state automaton over the alphabet $\{0,1\}$, which accepts all strings of length 2 modulo 3 which end either with the symbols 00 or with the symbols 11 (and rejects all other strings). It is depicted in Figure 8.

Assume that the length of l_2 is 0 modulo 3 (the other two cases are very similar). Then for every pair of strings r_i and r_j in R such that r_i reaches one of q_0, q_1 or q_2 and r_j reaches either q_3 or q_4, r_i and r_j truly differ on exactly half of the strings in V_2 (all those that end either with a 00 or with a 11). For every pair of strings r_i and r_j such that r_i reaches one of q_0, q_1 or q_2, and r_j reaches a different state among these three states, r_i and r_j truly behave the same on all strings in V_2. The same is true for every pair of strings that reach either q_3 or q_4. Hence, with high probability, all strings in R that reach one of q_0, q_1 or q_2 are initially in one connected component in $G(R)$, and all strings that reach q_3 or q_4 are in a different connected component. After filling in the first column in T_1 labeled by λ with the

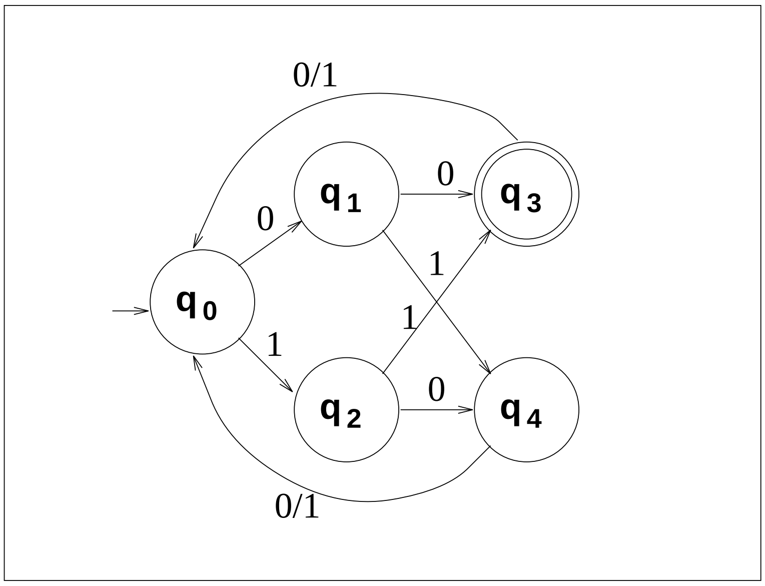

Figure 8. Second example target automaton. q_3 is the single accepting state.

names of these two components, we shall observe the following inconsistency. If r_i is a string that reaches q_0, and r_j is a string that reaches either q_1 or q_2, and if for $\sigma = 0/1$, both $r_i \cdot \sigma$, and $r_j \cdot \sigma$ are in R, then $r_i \cdot \sigma$ and $r_j \cdot \sigma$ are in different connected components (since $r_i \cdot \sigma$ reaches either q_1 or q_2, and $r_j \cdot \sigma$ reaches either q_3 or q_4). After resolving this inconsistency by adding σ to S, the table is consistent, and we have three classes. Let us denote these classes by C_0, $C_{1/2}$ and $C_{3/4}$, where strings in C_0 reach q_0, strings in $C_{1/2}$ reach either q_1 or q_2, and strings in $C_{3/4}$ reach either q_3 or q_4. It is clear that there is no labeling of these classes that has the labeling property defined in Lemma 2, unless there are either very few strings in R that reach q_3, or very few that reach q_4. In the next section we show how this partition is further refined, and how the new classes are labeled.

6.3. Final partitioning by correction

We reach this stage with an initial partition $\mathcal{P}_{int}$ that has with high probability the properties defined in Lemma 4. In this section we continue refining $\mathcal{P}_{int}$. We then label the classes in the final partition, $\mathcal{P}_{fnl}$, so that with high probability the labeled partition has the labeling property defined in Lemma 2. Namely, for most sample strings, the label of their class is their correct label. We give an upper bound on the number of classes in $\mathcal{P}_{fnl}$, so that in Section 7 we can use an Occam's Razor type of argument in order to prove that with high probability the hypothesis automaton defined based on $\mathcal{P}_{fnl}$ is an ϵ-good hypothesis. The resulting automaton might be much larger than the minimal equivalent automaton and so we apply an algorithm for minimizing DFAs [18], [27], [17], and find the smallest equivalent automaton.

The final partition is defined in the following simple manner. For any given string $r \in R$, let $r = r_p \cdot r_s$ where $|r_s| = l_2$. Let the *prefix class* of r, denoted by $C_p(r)$ be the class r_p belongs to in $\mathcal{P}_{int}$. Then

$$\begin{aligned} \mathcal{P}_{fnl} \quad &\stackrel{\text{def}}{=} \quad \{\, \{r \mid C_p(r) = C,\ r_s = s\} \mid C \in \mathcal{P}_{int},\ |s| = l_2\} \\ &\quad \cup\ \{\{r\} \mid |r| < l_2\}\,. \end{aligned} \tag{31}$$

$\mathcal{P}_{fnl}$ is a refinement of $\mathcal{P}_{int}$ since all strings that have the same prefix class and the same suffix (of length l_2) must belong to the same class in the initial partition. For each class $C \in \mathcal{P}_{int}$, and for every string s of length l_2, let (C, s) denote the class in $\mathcal{P}_{fnl}$ which consists of all strings in R whose prefix class is C, and whose suffix of length l_2 is s. There are at most $n_b \cdot |\Sigma|^{l_2}$ classes of this kind, and at most $|\Sigma|^{l_2}$ *singleton* classes each consisting of a single string of length less than l_2. The size of the final partition is hence at most $(n_b+1)|\Sigma|^{l_2}$, and thus grows only logarithmically with the sample size m. We later show that since the initial partition is consistent, so is this final partition.

The classes in $\mathcal{P}_{fnl}$ are labeled by calling Procedure *Label-Classes* (Figure 9). For each class (C, s) the procedure labels the class by the majority observed label of the strings in $C \circ \{s\}$. If all strings in C truly behave the same on the suffix s, and if C is of substantial size, then with high probability the majority observed label is the true label of all strings in the class (C, s) (which is equivalent to $\{C \circ \{s\}\} \cap R$). In this case we say that (C, s) is a *good* class. Based on the assumption that $\mathcal{P}_{int}$ has the third property defined in Lemma 4, we show that the fraction of sample strings whose correct label differs from the label of their class, is small, and hence $\mathcal{P}_{fnl}$ has the labeling property defined in Lemma 2. The singleton classes are all labeled by a default value 0, since we do not have a reliable way of determining their correct labels. This is an arbitrary choice and any other labeling of these classes would not alter our analysis. In particular, there are some special cases where a different labeling would give a better bound on the number of states in the hypothesis automaton. We return to this issue at the end of this subsection.

The initial partition thus serves two purposes. It is used as a basis for the final partition, and it is used to compute the *correct* labels of most sample strings.

LEMMA 6 *1. With probability at least* $1-2\delta'$, *The fraction of sample strings whose correct label differs from the label computed for their class by Procedure* Label-Classes *is at most* $\epsilon/2$.

2. $\mathcal{P}_{fnl}$ *is always consistent, and* $|\mathcal{P}_{fnl}| \leq (n_b+1)|\Sigma|^{l_2}$.

Our main efforts are directed towards proving the first claim of Lemma 6. In order to do this we need to bound the fraction of sample strings for which the label computed by *Label-Classes* for their class is incorrect with non-negligible probability.

```
Procedure Label-Classes()

for each class $(C, s) \in \mathcal{P}_{fnl}$
    let the label of $(C, s)$ be $maj(\mathcal{E}(r \cdot s) \mid r \in C)$.
for each class $\{r_i\} \in \mathcal{P}_{fnl}$ (where $|r_i| < l_2$)
    let the label of $\{r_i\}$ be 0.
```

Figure 9. Procedure *Label-Classes*

We first formally define the notions of *good* and *bad* classes mentioned previously.

Definition. We say that a class $(C, s) \in \mathcal{P}_{fnl}$ is good if all strings in C truly behave the same on s. Otherwise it is bad.

We know (Lemma 4) that with high probability, for every class C in $\mathcal{P}_{int}$, the fraction of suffixes s for which (C, s) is bad, is small. We would like to prove that with high probability the sample chosen is such that most sample strings of length at least l_2 belong to good classes in $\mathcal{P}_{fnl}$. To do so we prove that with high probability *no* string of length l_2 is a suffix of too large a fraction of the sample strings. Assuming this is the case, then in particular all strings s for which there exists a class $C \in \mathcal{P}_{int}$, such that (C, s) is bad can not be suffixes of too large a fraction of the sample strings, and only this small fraction of the sample strings belong to bad classes.

LEMMA 7 *With probability at least* $1 - \delta'$, *there is* no *string of length* l_2 *which is a suffix of more than a fraction of* $2 \cdot |\Sigma|^{-l_2}$ *of the sample strings.*

Proof: Since the sample strings are uniformly distributed, for every given suffix of length l_2, the expected fraction of sample strings having that suffix is $|\Sigma|^{-l_2}$. Applying Inequality 2, we get that for every given suffix of length l_2, the probability that there are more than $2m \cdot |\Sigma|^{-l_2}$ strings with that suffix (i.e., twice the expected number) is less than $e^{-\frac{1}{3}|\Sigma|^{-l_2}m}$. The probability this occurs for any suffix of length l_2 is less than $|\Sigma|^{l_2} e^{-\frac{1}{3}|\Sigma|^{-l_2}m}$, which is less than δ' since $m > 3|\Sigma|^{l_2} \ln(|\Sigma|^{l_2}/\delta')$. ∎

There are two more types of classes in the final partition for which we cannot claim *Label-Classes* is reliable in their labeling (even though they are good) and which we deal with in the proof of Lemma 6: the singleton classes and the classes (C, s) for which $|C|$ is small. In order to prove that with high probability Procedure *Label-Classes* correctly labels all classes (C, s) which are good and for which $|C|$ is not too small, we need the following simple claim.

LEMMA 8 *Let B be any set of strings which have the same correct label $\ell \in \{0,1\}$, and let $0 < \delta'' < 1$. If $|B| \geq \frac{1}{2}\gamma_b^{-2}(\ln 1/\delta'')$, then with probability at least $1-\delta''$, the majority observed label of the strings in B is ℓ.*

Proof: Since the expected value of the observed majority label is $1-\eta$,

$$Pr[\text{ majority observed label is wrong }] \;<\; e^{-2(\frac{1}{2}-\eta)^2|B|} \;\leq\; e^{-2\gamma_b^2|B|} \;\leq\; \delta''.$$

■

We are now ready to prove Lemma 6.

Proof of Lemma 6:

1st Claim: As mentioned previously, there are three kinds of sample strings for which the label of their class in $\mathcal{P}_{fnl}$ might differ from their correct label:

1. Strings shorter than l_2.
2. Strings which belong to *bad* classes.
3. Strings which belong to *good* class (C,s) but for which the majority value of the observed labels in $C\circ\{s\}$ is incorrect.

There are at most $|\Sigma|^{l_2} \ll (\epsilon/8)\cdot m$ of the first kind.

We next turn to the second kind of mislabeled strings. Based on our assumption that $\mathcal{P}_{int}$ has the third property defined in Lemma 4, we know that for each class $C \in \mathcal{P}_{int}$, the fraction of strings $s \in V_2$, such that $(C,s) \in \mathcal{P}_{fnl}$ is bad, is at most $n{\cdot}\beta_{max}$ (where β_{max} is defined in Equation 23). There are at most n classes in $\mathcal{P}_{int}$, and hence the fraction of strings s in V_2 such that there exists any class $C \in \mathcal{P}_{int}$ for which (C,s) is bad is at most $n^2{\cdot}\beta_{max}$. Applying Lemma 7, we get that with probability at least $1-\delta'$ the fraction of mislabeled sample strings of the second kind is at most

$$2n^2{\cdot}\beta_{max} \;=\; 2n^2(1-2\eta)^{-2}\left[\sqrt{2|\Sigma|^{-l_2}(2n_b^2\ln|\Sigma| + \ln\frac{4N^2}{\delta'})} + 2\rho\right]. \qquad (32)$$

Bounding $(1-2\eta)$ from below by $2\gamma_b$, and substituting the values of l_2 and ρ in Equation 32 we get that

$$\begin{aligned} 2n^2{\cdot}\beta_{max} \;&\leq\; n_b^2\gamma_b^{-2}/2 \\ &\times \left[\quad \sqrt{\frac{\epsilon^2\gamma_b^4}{2^6 n_b^6 \ln\frac{20m^2L^2|\Sigma|^2}{\delta}}\cdot(2n_b^2\ln|\Sigma| + \ln\frac{4m^2L^2}{\delta'})}\right. \\ &\qquad + \left. 2\sqrt{\frac{\epsilon^2\gamma_b^4}{2^8 n_b^4 \ln\frac{10(n_b^2+1)}{\delta}}\cdot\ln 2(n_b+1)^2/\delta'} \quad\right] \\ &<\; \frac{\epsilon}{8} \end{aligned} \qquad (33)$$

It remains to bound the fraction of mislabeled sample strings of the third kind. We show that with probability at least $1-\delta'$ there are less than $(\epsilon/4)m$ mislabeled sample strings of this kind. It follows that with probability at least $1-2\delta'$, the fraction of mislabeled sample strings of any one of the three types mentioned above is at most $\epsilon/2$.

Let (C,s) be a good class, and let $|C| \geq \alpha_2$, where

$$\alpha_2 \stackrel{\rm def}{=} \frac{1}{2}\gamma_b^{-2}(n_b \ln 2 + \ln \frac{|\Sigma|^{l_2}}{\delta'}). \tag{34}$$

Based on Lemma 8, the probability that the majority observed label of the strings in $C\circ\{s\}$ is not their correct (common) label, is less than $\delta'/(2^{n_b}|\Sigma|^{l_2})$. For a given class $C \in \mathcal{P}_{int}$, the number of (nonempty) classes (C,s) is at most $|\Sigma|^{l_2}$. The initial partition into classes (which induces a partition into prefix classes) is not chosen independently from the expert's (correct and incorrect) labels of the strings in $R\circ V_2$, but is rather defined based on the knowledge of these labels. Hence, we must consider all possible prefix classes of strings in R. We assume that the initial partition has the properties defined in Lemma 4, and specifically that it has the second property defined in the lemma, namely that all strings that reach the same state in A belong to the same class in the initial partition. Since there are at most 2^{n_b} subsets of the states in A, and for each such subset there is a set of strings in R that reach the states in that subset, there are at most 2^{n_b} possible prefix classes. Therefore, the probability that for all possible prefix classes C of size at least α_2, and for all possible suffixes s such that (C,s) is a good class, the majority observed label of the strings in $C\circ\{s\}$ is the correct label, is at least $1-\delta'$.

Therefore, with probability at least $1-\delta'$, all mislabeled strings of this (third) kind are strings which belong to classes (C,s) such that $|C| < \alpha_2$. For each such C, the number of sample strings which belong to (C,s) for any s, is at most $\alpha_2|\Sigma|^{l_2}$. There are less than n_b such classes, and hence the number of mislabeled strings of this kind is less than

$$\begin{aligned}
n_b\alpha_2|\Sigma|^{l_2} \quad &\leq \quad \frac{1}{2}n_b\gamma_b^{-2}\left(n_b \ln 2 + \ln(\frac{2^7 n_b^6|\Sigma|}{\epsilon^2\gamma_b^4\delta'}\cdot \ln\frac{20m^2L^2|\Sigma|^2}{\delta}\right) \\
&\qquad \times \; \frac{2^7 n_b^6|\Sigma|}{\epsilon^2\gamma_b^4}\cdot\ln\frac{20m^2L^2|\Sigma|^2}{\delta} \qquad (35)\\
&< \quad \frac{2^6 n_b^8|\Sigma|}{\epsilon^2\gamma_b^6}\cdot\left(\ln\frac{2^9 n_b^6}{\epsilon^2\gamma_b^4\delta'} + \ln\ln\frac{20m^2L^2|\Sigma|}{\delta}\right) \\
&\qquad \times \; \ln\frac{20m^2L^2|\Sigma|}{\delta} \qquad (36)\\
&< \quad \epsilon/4m \qquad (37)
\end{aligned}$$

2nd Claim: The bound on the size of $\mathcal{P}_{fnl}$ follows directly from its definition. It remains to show that it is a consistent partition. Let (C,s) be any class in $\mathcal{P}_{fnl}$ where $s = s_1\ldots s_{l_2}$, let σ be a symbol in Σ, and let $r_1 = r_{1p}\cdot s$ and $r_2 = r_{2p}\cdot s$ be two strings which belong to (C,s) such that both $r_1\cdot\sigma$ and $r_2\cdot\sigma$ are in R. Since r_1

and r_2 both have the same suffix of length l_2, so do $r_1 \cdot \sigma$ and $r_2 \cdot \sigma$. Since r_1 and r_2 have the same prefix class C in $\mathcal{P}_{int}$ (i.e., r_{1p} and r_{2p} belong to the same class in $\mathcal{P}_{int}$), and $\mathcal{P}_{int}$ in consistent, $r_1 \cdot \sigma$ and $r_2 \cdot \sigma$ must belong to the same prefix class as well (since $r_{1p} \cdot s_1$ and $r_{2p} \cdot s_1$ must belong to the same class). It follows that $r_1 \cdot \sigma$ and $r_2 \cdot \sigma$ belong to the same class in $\mathcal{P}_{fnl}$. ■

In Lemma 6 we give an upper bound on the number of classes in the partition which is considerably larger than the number of states in the target automaton. As mentioned previously, we can try and minimize the automaton defined based on this partition. If all classes (C, s) are good and all classes (including the singletons) are correctly labeled, and if we do not need to add the sink class, then this minimization results in an automaton of size at most n. Though we do not have a general way to avoid errors resulting from the existence of bad classes or of small prefix classes, we can sometimes avoid errors when labeling the singleton classes.

As we have mentioned in our discussion of Procedure *Label-Classes* (prior to the analysis above), our choice of labeling by 0 all singleton classes, is arbitrary, and any other labeling will do. We next describe a case in which a different labeling is more advantageous.

Assume that $\mathcal{P}_{fnl}$ consists only of good classes (C, s) for which $|C| \geq \alpha_2$. In particular, this may be the case when $\mathcal{P}_{int}$ exactly corresponds to the target automaton in the sense that no two strings which belong to the same class in the initial partition reach different states, and for each state there exists a string that reaches it. If the target automaton is such that: (1) there is non-negligible probability of passing each state in a random walk of length L; (2) every two states either differ on a non-negligible fraction of strings of length l_2, or reach such a pair of states (that differ on a non-negligible fraction of strings of length l_2) on a walk corresponding to some string s; then with high probability $\mathcal{P}_{fnl}$ has the properties mentioned above. The first example described in Subsection 6.2.1 is of this type.

Suppose that when labeling the classes (C, s) we notice that for a class $C' \in \mathcal{P}_{int}$, all classes (C, s) which include strings that belong to C' are labeled the same. Then we label all singleton classes which include strings that reach C' with the same label. If the initial partition in fact corresponds to the target automaton, then this labeling is correct with high probability. We would also like to note that in the preliminary version of this paper [29] we added an additional stage to the algorithm (between the initial and final partitioning) which treated this special case in a different manner.

7. Putting it all together

We have shown how to achieve with high probability a labeled partition of a given set of sample strings and their prefixes that is consistent and for which the fraction of sample strings whose label according to A differs from the label of their class is at most $\epsilon/2$. We have also shown that the number of classes in this partitioning is at most $\theta \ln m$ where θ is a polynomial in n_b, $\frac{1}{\gamma_b}$, $|\Sigma|$, L, $\frac{1}{\epsilon}$ and $\ln \frac{1}{\delta}$. Hence, we can apply Lemma 2 and construct a hypothesis automaton with $\theta \ln m$ states which agrees with A on all but $\epsilon/2$ of the sample strings. Adding up the probabilities our algorithm errs in each of its stages and using the following Occam's Razor-like lemma, we prove that with probability at least $1-\delta$, our hypothesis automaton is an ϵ-good hypothesis with respect to A and D_L.

LEMMA 9 *Let θ be a polynomial in n_b, $\frac{1}{\gamma_b}$, $|\Sigma|$, L, $\frac{1}{\epsilon}$, and $\ln\frac{1}{\delta'}$, and let $\theta' = \max(2|\Sigma|\theta\log_2\theta, \ln\frac{1}{\delta'})$. Given $m \geq \frac{64\theta'}{\epsilon^2}(\ln\frac{64\theta'}{\epsilon^2})^3$ strings chosen according to D_L, if an automaton of size at most $\theta\ln m$ disagrees with A on no more than $\epsilon/2$ of the sample strings, then the probability that it is an ϵ-bad hypothesis with respect to A is at most δ'.*

Proof: Let A' be an automaton of size at most $\theta\ln m$ which is an ϵ-bad hypothesis with respect to A. Given a random sample of size m labeled according to A, the expected number of strings on which A' disagrees with A is at least ϵm. According to Inequality 1, the probability that A' disagrees with A on no more than $\frac{\epsilon}{2}m$ of the m random strings, is at most $e^{-2(\epsilon/2)^2 m}$. Since the number of automata which are ϵ-bad hypotheses with respect to A is at most

$$\begin{aligned} N_{DFA}(\theta\ln m, |\Sigma|) - 1 &< 2^{2|\Sigma|\theta\ln m\log_2(\theta\ln m)} \\ &< 2^{\theta'(\ln m)^2}, \end{aligned}$$

the probability we found such an automaton which disagrees with A on no more than $\frac{\epsilon}{2}$ of the sample strings is at most $2^{\theta'(\ln m)^2}e^{-\frac{1}{2}\epsilon^2 m}$. But for $m \geq \frac{64\theta'}{\epsilon^2}(\ln\frac{64\theta'}{\epsilon^2})^3$

$$\begin{aligned} \frac{m}{(\ln m)^2} &\geq \frac{\frac{64\theta'}{\epsilon^2}(\ln\frac{64\theta'}{\epsilon^2})^3}{(\ln\frac{64\theta'}{\epsilon^2} + 3\ln\ln\frac{64\theta'}{\epsilon^2})^2} \qquad (38) \\ &\geq \frac{\frac{64\theta'}{\epsilon^2}(\ln\frac{64\theta'}{\epsilon^2})^3}{16(\ln\frac{64\theta'}{\epsilon^2})^2} \\ &> \frac{4\theta'}{\epsilon^2}. \end{aligned}$$

Thus

$$\theta'(\ln m)^2 < \frac{1}{4}\epsilon^2 m. \qquad (39)$$

And so

$$2^{\theta'(\ln m)^2}e^{-\frac{1}{2}\epsilon^2 m} < e^{-\frac{1}{4}\epsilon^2 m} < e^{-16\theta'} < \delta'. \qquad (40)$$

■

It is easily verified that the algorithm we have described is polynomial in n_b, $\frac{1}{\gamma_b}$, $|\Sigma|$, L, $\frac{1}{\epsilon}$ and $\ln\frac{1}{\delta}$. Consequently we have proven our main theorem:

THEOREM 1 *The learning algorithm described in this paper is a* good learning algorithm *for fallible DFAs*

8. Examples Revisited

We now return to the example runs presented in Subsection 6.2.1 and see how the algorithm completes these runs.

We start with the first example. Remember that following the initial partitioning we have two classes - C_0 includes the strings in R that reach the state q_0 and C_1 includes the strings that reach q_1. For each class C_i ($i \in \{0,1\}$), all strings in the classes (C_i, s) ($|s| = l_2$) in $\mathcal{P}_{fnl}$, and in general, all strings in the sets $C_i \circ \{s\}$ have the same correct label since they belong to the same class in $\mathcal{P}_{int}$. Assuming that both C_0 and C_1 are larger than α_2,[3] all these classes are labeled correctly.

If all singleton classes are labeled 0, then the automaton based on this partition (depicted in Figure 10) has the following form. It consists of a complete binary tree of depth $l_2 - 1$ whose states are all rejecting states, and whose root, λ is the starting state of the automaton. All transition from the leaves of this tree are to the states (C_0, s) which are all rejecting states. All transition from this layer are to the layer of states (C_1, s), which are all accepting states, and which traverse back to the first layer. The minimized automaton is depicted in Figure 11.

Note that if we use the modified version of *Label-Classes* as described in the end of Subsection 6.3 for labeling the singleton classes, then we can label these classes correctly as well. The (minimized) hypothesis automaton which is based on $\mathcal{P}_{fnl}$ is equivalent to the target automaton.

Also note that in this example (when the modified version is not used), the following modification can be employed as well. Since all strings of a given length reach the same state, and since the first l_2 states do not belong to the strongly connected component of the underlying graph, and hence only the short strings (which we already "gave up" on their correct labeling) reach them, we can remove the first l_2 states from the hypothesis. The new starting state is chosen so that the longer strings reach the states corresponding to their class in the final partition, and are therefore labeled the same by the modified hypothesis as by the original hypothesis. The modified hypothesis is then equivalent to the target automaton.

We now return to the second example. Remember that in this example, strings that reach the same class in $\mathcal{P}_{int}$ do not necessarily have the same correct label. Specifically, part of the strings in the class $C_{3/4}$ reach an accepting state (q_3) in the target automaton, and part of the strings reach a rejecting state (q_4). Note though, that all strings in this class that end either with a 00 or with a 11 reach q_4, while all other strings reach q_3. Based on our assumption that the length of l_2 is 0 modulo 3 (the other two cases are very similar), the prefix class of all strings in $C_{3/4}$ is $C_{3/4}$. For every s, $|s| = l_2$ of the form $s'00$ or $s'11$, all strings in $C_{3/4} \circ \{s\}$ have the same correct label 1, and for every s of the form $s'01$ or $s'10$, all strings in $C_{3/4} \circ \{s\}$ have the same correct label 0. Assuming $|C_{3/4}|$ is larger than α_2, all these classes are labeled correctly. Similarly, if both C_0 and $C_{1/2}$ are larger than α_2, then all classes (C_0, s) $(C_{1/2}, s)$, and (for every s) are labeled correctly by 0.

If all singleton classes are labeled 0, then the automaton based on this partition will have the form depicted in Figure 12. Its minimized version is depicted in

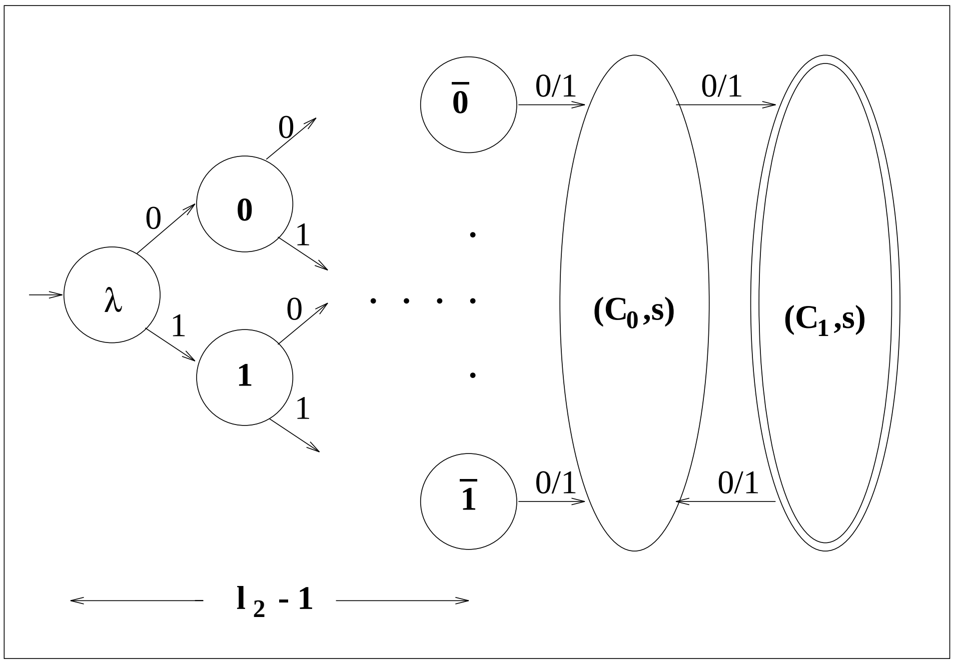

Figure 10. Hypothesis automaton for the first example.

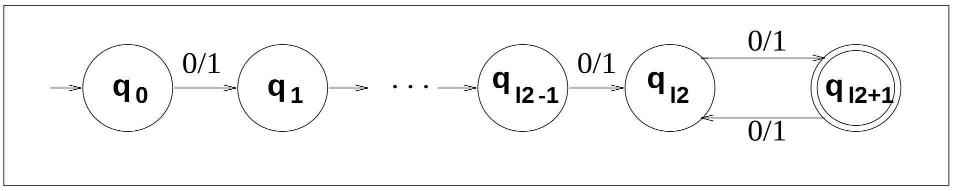

Figure 11. Hypothesis automaton for the first example (minimized version).

Figure 13. In this case the modification of *Label-Classes* mentioned in the first example cannot help label the singleton classes correctly. However, equivalently to the first example, we can remove the states that do not belong to the strongly connected component and choose the new starting state accordingly.

9. Extensions and Further Research

As mentioned in the Introduction, our result can be extended to the following cases:

1. The expert's errors are not completely independent but rather are distributed only k-wise independently for $k = O(1)$.

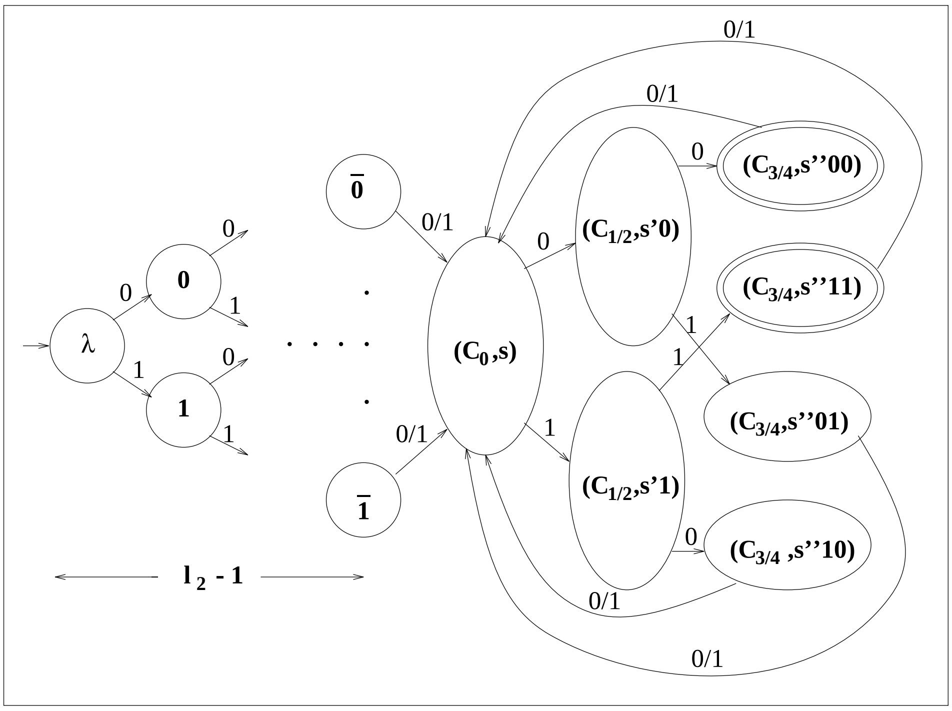

Figure 12. Hypothesis automaton for second example.

2. The expert's error probability is dependent on the length of the input string.

3. The target automaton has more than two possible outputs.

We first describe in short the changes that should be made to the algorithm in each of the cases above and then discuss briefly some other possible extensions and further research directions.

9.1. k-wise Independence of the Expert's Error Probability

In this case, the algorithm need not be altered, only the size of the sample and the sizes l_1 and l_2 of the length of the suffixes on which the sample strings behavior is tested need to be changed. This is due to the fact that we cannot use Inequality 1 when bounding the error probability in different stages of the algorithm, since Inequality 1 is only valid under the assumption that the random variables are independent. Instead, we can use the following inequality that is derived from the high moment inequality of which Tchebychev's inequality is a special case. Let $X_1, X_2, ...X_M$ be M k-wise independent 0/1 random variables where $Pr[X_i = 1] = p_i$, and $0 < p_i < 1$. Let $p = \sum_i p_i/M$.

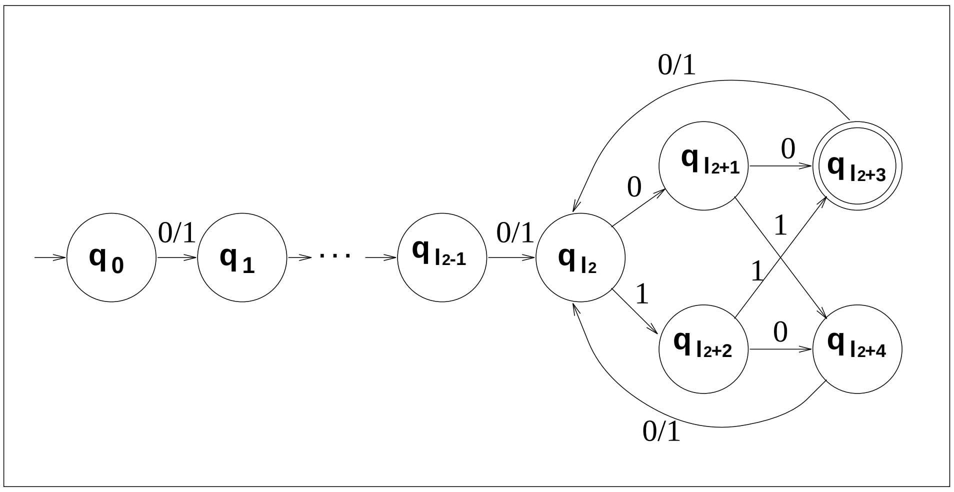

Figure 13. Hypothesis automaton for second example (minimized version).

Inequality 3 *For $0 < \alpha \leq 1$:*

$$Pr[|\frac{\sum_{i=1}^{M} X_i}{M} - p| > \alpha] < \frac{k^k}{M^{\frac{k}{2}} \cdot \alpha^k}$$

If all p_i's are equal then we can get a slightly better bound in which the above expression is multiplied by $p^{\frac{k}{2}}$.

In order that our claims regarding the upper bounds on the error probabilities in all the stages of the algorithm remain true, we must enlarge the size of our sample, m, and the size of the sets V_1 and V_2 on which we test the behavior of the sample strings. It can be shown that the size of these sets remains polynomial in the relevant parameters of the problem, and that the size of the hypothesis automaton grows like m^β for $\beta < 1$. Therefore the learning algorithm remains a *good learning algorithm* for fallible DFAs.

9.2. The Expert's Error Probability is Dependent on the Length of the Input Strings

In this case we assume that for every length $l \geq 0$, the expert errs with probability $\eta_l \leq 1/2 - \gamma_b$ on strings of length l. We use the same technique presented in Section 6.1, for estimating each η_l, only now we compute the corresponding estimate, $\Delta(l)$, of $2\eta_l(1-\eta_l)$, for every $l_1 \leq l \leq L + {l_2}^4$. This is done by picking a set W_l of $n_b + 1$ strings all of the *same length* $l - l_1$ and letting $\Delta(l)$ be the outcome of Function *Estimate-Error* (Figure 2) when executed on the set W_l (and, as before, on the set of all suffixes of length l_1). When bounding the error probability of the revised algorithm, we must take into account that we want that with high probability *all* estimates $\Delta(l)$ be approximately correct.

The fact that the error probability might differ for different string lengths must be taken into account in the following places:

1. The statements in Observation 1 should now be: For any given pair of different strings u_1 and u_2, and for any given (suffix) string v:

   (A) If $\bar{A}(u_1 \cdot v) = \bar{A}(u_2 \cdot v)$, then
   $$Pr[\mathcal{E}(u_1 \cdot v) \neq \mathcal{E}(u_1 \cdot v)] \;=\; (1-\eta_{|u_i|+|v|})\eta_{|u_j|+|v|} + (1-\eta_{|u_j|+|v|})\eta_{|u_i|+|v|}.$$
   (B) If $\bar{A}(u_1 \cdot v) \neq \bar{A}(u_2 \cdot v)$, then
   $$Pr[\mathcal{E}(u_1 \cdot v) \neq \mathcal{E}(u_1 \cdot v)] \;=\; (1-\eta_{|u_i|+|v|})(1-\eta_{|u_j|+|v|}) + \eta_{|u_i|+|v|}\eta_{|u_j|+|v|}.$$

   Hence, if V is a set of (suffix) strings, all of length l, and the fraction of strings in V on which u_1 and u_2 truly differ is q, then their expected observed difference rate on V is
   $$\eta_{|u_1|+l} + \eta_{|u_2|+l} - 2\eta_{|u_1|+l}\eta_{|t_j|+l} + q(1-2\eta_{|u_1|+l})(1-2\eta_{|u_2|+l}).$$
   In the special case where $|u_1| = |u_2| = l'$ we of course get the same result as in the original version of Lemma 1, where η is exchanged by $\eta_{l'+l}$.

   Thus there is still a gap between the expected value of the observed difference in behavior in the case where two strings reach the same state and in the case they do not reach the same state. When exploiting this gap in the process of the initial partitioning (specifically in Function *Strings-Test* appearing in Figure 6), we must take into account that the expected value of the observed difference rate between two strings depends on their lengths, and use the correct expression.

2. In general, as mentioned in specific cases above, the value of almost all parameters (the size of the sample m, the lengths, l_1 and l_2, of the suffix strings on which we test the sample strings, the value of α_1 in Function *Strings-Test*, etc.) must be revised so that the total error probability of the algorithm is bounded by δ.

9.3. Multiple Outputs

Assume that the target automaton has more than two possible outputs, and let the output alphabet be denoted by Π. Assume also that the error process is such that for every (newly) queried string u, independently, and with probability η, the expert's answer, $\mathcal{E}(u)$, received for that string, is chosen uniformly from $\Pi - \{\mathcal{E}(u)\}$. We claim that if we slightly modify some of the parameters of our algorithm, then it remains a good learning algorithm in this case.

There are several places in the algorithm and its analysis where the fact that $|\Pi| > 2$ has to be taken into account: in Observation 1 which implies slight changes in Procedure *Estimate Error*, Lemma 3 and Lemma 5; and in Lemma 8.

It is very easy to verify that Lemma 8 remains correct. Actually, for $|\Pi| > 2$, it suffices that Procedure *Label Classes* label the classes according to their *most common* observed label. For a given set of strings which have the same correct label, the probability that the most common observed label is incorrect decreases very rapidly when $|\Pi|$ increases.

Observation 1 is generalized as follows:

OBSERVATION 3 *For any given pair of* different *strings u_1 and u_2, and for any given (suffix) string v:*

1. *If $\bar{A}(u_1 \cdot v) = \bar{A}(u_2 \cdot v)$, then $Pr[\mathcal{E}(u_1 \cdot v) \neq \mathcal{E}(u_1 \cdot v)] = 2\eta(1-\eta)+\eta^2(1-1/(|\Pi|-1))$.*
2. *If $\bar{A}(u_1 \cdot v) \neq \bar{A}(u_2 \cdot v)$, then $Pr[\mathcal{E}(u_1 \cdot v) \neq \mathcal{E}(u_1 \cdot v)] = (1-\eta)^2 + 2\eta(1-\eta)(1-1/(|\Pi|-1)) + \eta^2(1-(|\Pi|-2)/(|\Pi|-1))$.*

It is not hard to verify that if V is any given set of (suffix) strings, and the fraction of strings in V on which u_1 and u_2 truly differ is β, then their expected observed difference rate on V is

$$2\eta(1-\eta) \;+\; \eta^2(1-1/(|\Pi|-1) \;+\; \beta \cdot \Psi(\eta, |\Pi|), \tag{41}$$

where $\Psi(\eta, |\Pi|)$ is at least $(1-2\eta)^2$ (for $|\Pi| \geq 2$), which was the gap we had when $|\Pi| = 2$. Given this observation, we can slightly modify Procedure *Estimate-Error* in order to compute a good estimate, Δ, of $2\eta(1-\eta)+\eta^2(1-1/(|\Pi|-1)$, and extract from it a good estimate of η. Based on the gap mentioned above, and using these estimates, we can apply Function *Strings-Test*, as in the case of $|\Pi| = 2$, in order to differentiate between strings that reach states in A whose true difference rate on V is substantial. In the analysis of the correctness of *Strings-Test*, described in Lemma 5, we need only take into account the change in the definition of Δ.

9.4. Additional Extensions

Our main assumption in this work is that the error probability of the expert is fixed. As mentioned in the previous subsection, we can deal with the special case in which the error probability is dependent on the length of the input string. The general problem (in which for every string u the expert has a (possibly different) error probability $\eta(u)$) seems hard. It might be argued that the natural problem in this case is to learn the corresponding probabilistic concept [23]. What we would like to know is if there are other (reasonable) special cases for which our algorithm can be adapted. For example, can the problem be solved if the error probability of the expert depends on the state reached by the input string?

Another generalization of our algorithm is to modify it to work under additional distributions other than the uniform distribution. It is unreasonable to expect to find an algorithm that works under *any* input distribution. For example assume that all the weight of the distribution is on one string, or even that it is equally divided among n strings each reaching a different state. However, it may be possible that our algorithm can be modified to work for other “natural” distributions.[5]

One additional point is the question of the practicality of the algorithm. Though the algorithm is polynomial in the relevant parameters of the problem, there is still much to be desired in terms of the exponents in this polynomial. We feel that a more careful (though perhaps more complicated) analysis might yield better bounds. We would like to point out that we were able to improve these bounds with respect to the ones presented in the preliminary version of this paper [29].

9.5. Further Research

An interesting direction for further research is to try and modify additional PAC learning algorithms that use membership queries to the case in which the queries might be answered erroneously. Such algorithms exist for learning monotone DNF [38], read-once Boolean functions [5], Horn Clauses [4], among others.

A more general direction that can be pursued is to study the possible relationship between learning from fallible experts and the area of self-correcting [9], [26]. A simple observation is that any family of functions that has both a known learning algorithm (with or without membership queries), and a self corrector, has a learning algorithm with membership queries that works when queries might be answered erroneously. The idea is that the self corrector serves as a correcting “filter” between the expert and the learner. The learner both ignores the expert’s labels on the sample strings, and does not address any queries directly to the expert. Instead, it always queries the corrector.

A simple example of an application of the above observation is a learning algorithm (using membership queries) for noisy parity functions. This algorithm is composed of an algorithm for learning parity functions [13], [19] by solving a system of linear equations over the field of integers modulo 2, and a self-correcting algorithm [9], [26] for the same family of functions. We do not know of any other self-correcting algorithm that has been directly applied to a related learning problem, but the possibility exists that techniques used in one field may be useful in the other.

Acknowledgments

We would like to thank Naftali Tishby, Oded Goldreich, Mike Luby, Michael Kearns, Nati Linial and Joan Feigenbaum for comments and helpful discussions. We would also like to thank the referees of this paper for their careful reading and very helpful comments.

Notes

1. [11] actually mention that the adversary argument in [1] showing that DFAs cannot be learned using membership queries only can be modified so that the target DFAs all have distinguishing sequences. This implies that even in the error-free case one cannot hope to find a distinguishing sequence in polynomial time by exploration only. This still leaves open the following question. Perhaps it is possible to define and efficiently find a weaker version of a distinguishing sequence which suffices in the case that the learner does have access to randomly labeled examples in addition to membership queries, and is allowed to be only approximately correct with respect to the input distribution.
2. Note that the final partition can be viewed as a partition into *effective equivalence classes* in the following sense: two strings r_i and r_j belong to the same effective equivalence class if we do not find evidence in the sample (and in the answers to our queries) that they differ on any suffix. Since we do not know of any string s such that $\bar{A}(r_i s) \neq \bar{A}(r_j s)$, we assume that they reach the same state in A.
3. If they are not, this is because the sample is not typical. The probability such an event occurs is taken into account in Lemma 9.
4. The values of l_1 and l_2 must be changed accordingly but their usage is the same as in the original version of the algorithm
5. Note that if the input distribution is "almost uniform" in the sense that it has the property that the probability of every string is within a polynomial multiplicative factor, p, of its uniform probability, then the only modification needed in order for any uniform distribution learning algorithm to succeed under this distribution is to run it with a smaller approximation parameter, ϵ/p.

References

1. D. Angluin. (1981). A note on the number of queries needed to identify regular languages. *Information and Control, 51*, 76–87.
2. D. Angluin. (1987). Learning regular sets from queries and counterexamples. *Information and Computation, 75* 87–106.
3. D. Angluin and P. Laird. (1988). Learning from noisy examples. *Machine Learning, 2*, 343–370.
4. D. Angluin, M. Frazier and L. Pitt. (1992). Learning conjunctions of Horn Clauses. *Machine Learning, 9*, 147–164.
5. D. Angluin, L. Hellerstein and M. Karpinski. (1993). Learning read-once formulas with queries. *Journal of the Association for Computing Machinery, 48*, 185–210.
6. N. Alon and J. H. Spencer. (1991). *The Probabilistic Method*. Wiley Interscience.
7. D. Angluin and D. Slonim. (1994). Randomly fallible teachers: Learning monotone DNF with an incomplete membership oracle. *Machine Learning, 14*, 7–26.
8. A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Occam's Razor. (1987). *Information Processing Letters, 27*, 377–380.
9. M. Blum, M. Luby, and R. Rubinfeld. (1993). Self-Testing/Correcting with applications to numerical problems. *Journal of Computer and System Sciences, 47*, 549–595.
10. H. Chernoff. (1952). A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of the Mathematical Statistics, 23*, 493–507.
11. T. Dean, D. Angluin, K. Basye, S. Engelson, L. Kaelbling, E. Kokkevis, and O. Maron. (1992). Inferring finite automata with stochastic output functions and an application to map learning. *Proceedings of the 10th National Conference on Artificial Intelligence* (pp. 208–214).

12. Y. Freund, M. J. Kearns, D. Ron, R. Rubinfeld, R. E. Schapire, and L. Sellie. (1993). Efficient learning of typical finite automata from random walks. *Proceedings of the 25th Annual ACM Symposium on Theory of Computing* (pp. 315–324). San-Diego, CA: The Association for Computing Machinery.
13. Paul Fischer and Hans Ulrich Simon. (1992) On learning ring-sum-expansions. *SIAM Journal on Computing, 21*, 181–192.
14. S. A. Goldman, M. J. Kearns, and R. E. Schapire. (1993). Exact identification of read-once formulas using fixed points of amplification functions. *SIAM Journal on Computing, 22*, 705–726.
15. S. A. Goldman and H. D. Mathias. (1992). Learning k-term DNF formulas with an incomplete membership oracle. *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory* (pp. 85–92). Pittsburgh, PA: The Association for Computing Machinery.
16. W. Hoeffding. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association, 58*, 13–30.
17. J. E. Hopcroft. (1971). An $n \log n$ algorithm for minimizing the states in a finite automaton. *The Theory of Machines and Computations*, 189–196. Academic Press, New-York.
18. D. A. Huffman. (1954). The synthesis of sequential switching circuits. *J. Franklin Institute, 257*, 161–190, 275–303.
19. D. Helmbold, R. Sloan, and M. K. Warmuth. (1992). Learning integer lattices. *SIAM Journal on Computing, 21*, 240–266.
20. M. Kearns. (1990). *The Computational Complexity of Machine Learning.* MIT Press.
21. M. Kearns. (1993). Efficient noise-tolerant learning from statistical queries. *Proceedings of the 25th Annual ACM Symposium on Theory of Computing* (pp. 392–401). San-Diego: The Association for Computing Machinery.
22. M. Kearns and M. Li. (1993). Learning in the presence of malicious errors. *Siam Journal on Computing, 22*, 807–837.
23. M. Kearns and R. Schapire. (1990) Efficient distribution-free learning of probabilistic concepts. *Proceedings of the 31st Annual Symposium on Foundations of Computer Science* (pp. 382–391). To appear in *JCSS.*
24. M. Kearns and L. Valiant. (1994). Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the Association for Computing Machinery, 41*, 67–95.
25. P. Laird. (1988). *Learning From Good Data and Bad*. Kluwer Academic Publishers.
26. R. Lipton. (1991). New directions in testing. *Distributed Computing and Cryptography, DIMACS Series in Discrete Math and Theoretical Computer Science, American Mathematical Society, 2*, 191–202.
27. E. F. Moore. (1956). Gedanken experiments on sequential machines. *Automata Studies*, 129–153. Princeton Univ. Press, Princeton, N.J.
28. L. Pitt and M. Warmuth. (1990) Prediction-preserving reducibility. *Journal of Computer and System Sciences, 41*, 430–467.
29. D. Ron and R. Rubinfeld. (1993). Learning fallible finite state automata. *Proceedings of the 6th Annual ACM Workshop on Computational Learning Theory*, (pp. 218–227). Santa-Cruz, CA: The Association for Computing Machinery.
30. R. Rivest and R. Schapire. (1987). Diversity-based inference of finite automata. *Proceedings of the 28th IEEE Symposium on Foundations of Computer Science*, (pp. 78–87). To appear in *JACM.*
31. R. Rivest and R. Schapire. (1993). Inference of finite automata using homing sequences. *Information and Computation, 103*, 299–347.
32. Y. Sakakibara. (1991). On learning from queries and counterexamples in the presence of noise. *Information Processing Letters, 37*, 279–284.
33. R. E. Schapire. (1991). *The Design and Analysis of Efficient Learning Algorithms.* MIT Press.

34. R. H. Sloan. (1988). Types of noise in data for concept learning. *Proceedings of the 1988 Workshop on Computational Learning Theory* (pp. 91–96). Santa-Cruz, CA: The Association for Computing Machinery.
35. R. H. Sloan. (1989). *Computational Learning Theory: New Models and Algorithms.* Doctoral dissertation, Department of Computer Science, Massachusetts Institute of Technology, Cambridge, MA. Issued as MIT/LCS/TR-448.
36. Y. Sakakibara and Rani Siromoney. (1992). A noise model on learning sets of strings. *Proceedings of the 5th Annual Workshop on Computational Learning Theory* (pp. 295–302). Pittsburgh, PA: The Association for Computing Machinery.
37. G. Shackelford and D. Volper. (1988). Learning k-DNF with noise in the attributes. *Proceedings of the 1988 Workshop on Computational Learning Theory* (pp. 97–103). Santa-Cruz, CA: The Association for Computing Machinery.
38. L. G. Valiant. (1984). A theory of the learnable. *Communications of the ACM, 27*, 1134–1142.
39. L. G. Valiant. (1985) Learning disjunctions of conjunctions. *Proceedings of the 9th International Joint Conference on Artificial Intelligence* (pp. 560–566). Los Angeles, CA: Morgan Kaufmann.