A Polynomial Time Incremental Algorithm for Learning DFA

Rajesh Parekh[1], Codrin Nichitiu[2], and Vasant Honavar[3]

[1] Allstate Research and Planning Center
321 Middlefield Road, Menlo Park CA 94025, USA
rpare@allstate.com
[2] Ecole Normale Superieure de Lyon
46 Allee d'Italie, 69364 Lyon Cedex 07, France
Codrin.Nichitiu@ens-lyon.fr
[3] Department of Computer Science, Iowa State University
Ames IA 50011, USA
honavar@cs.iastate.edu
WWW home page: http://www.cs.iastate.edu/~honavar/aigroup.html

Abstract. We present an efficient incremental algorithm for learning deterministic finite state automata (DFA) from labeled examples and membership queries. This algorithm is an extension of Angluin's *ID* procedure to an incremental framework. The learning algorithm is intermittently provided with labeled examples and has access to a knowledgeable teacher capable of answering membership queries. The learner constructs an initial hypothesis from the given set of labeled examples and the teacher's responses to membership queries. If an additional example observed by the learner is inconsistent with the current hypothesis then the hypothesis is modified minimally to make it consistent with the new example. The update procedure ensures that the modified hypothesis is consistent with all examples observed thus far. The algorithm is guaranteed to converge to a minimum state DFA corresponding to the target when the set of examples observed by the learner includes a *live complete* set. We prove the convergence of this algorithm and analyze its time and space complexities.

1 Introduction

Grammar Inference [BF72,FB75,MQ86,Lan95] is an important machine learning problem with several applications in pattern recognition, language acquisition, bioinformatics, and intelligent agent design. It is defined as the process of learning an unknown grammar from a given finite set of labeled examples. *Regular grammars* describe languages that can be generated (and recognized) by deterministic finite state automata (DFA). Since regular grammars represent a widely used subset of formal language grammars, considerable research has focused on regular grammar inference (or equivalently, identification of the corresponding DFA). However, given a finite set of positive examples and a finite (possibly empty) set of negative examples the problem of learning a minimum state DFA equivalent to the unknown target is *NP*-hard [Gol78]. The learner's task can be simplified by requiring that the set of examples provided meet certain desired criteria (like *structural completeness* [PC78,PH96] or *characteristic sample* [OG92]), or by providing the learner with access to sources of additional information, like a knowledgeable teacher who responds to queries generated by the learner. Angluin's *ID* algorithm learns the target DFA given a *live-complete* sample and a knowledgeable teacher to answer *membership* queries posed by the learner [Ang81]. The interested reader is referred to [MQ86,Pit89,Lan95,PH98] for recent surveys of different approaches to grammar inference.

In many practical learning scenarios, a live-complete sample may not be available to the learner at the outset. Instead, a sequence of labeled examples is provided intermittently and the learner is required to construct an approximation of the target DFA based on the examples and possibly the queries answered by the teacher. In such scenarios, an online or incremental model of learning that is guaranteed to eventually converge to the target DFA in the limit is of interest. Particularly, in the case of intelligent autonomous agents, incremental learning offers an attractive framework for characterizing the behavior of the agents [CM96]. Against this background, we present a provably correct, polynomial time, incremental, interactive algorithm for learning the target DFA from *labeled examples* and *membership queries*. The proposed algorithm *IID* extends the *ID* algorithm to an incremental setting.

2 Preliminaries

Let Σ be a finite set of symbols called the *alphabet*, Σ^* be the set of strings over Σ, α, β, γ be strings in Σ^*, and $|\alpha|$ be the length of the string α. λ is a special string called the *null* string and has length 0. Given a string $\alpha = \beta\gamma$, β is the *prefix* of α and γ is the *suffix* of α. Let $Pref(\alpha)$ denote the set of all prefixes of α. Further if $S \subseteq \Sigma^*$ then $Pref(S) = \cup_{\alpha \in S} Pref(\alpha)$. Given two sets S_1 and S_2, the *set difference* is denoted by $S_1 \backslash S_2$ and the *symmetric difference* is denoted by $S_1 \oplus S_2$.

A *deterministic* finite state automaton (DFA) is a quintuple $A = (Q,\delta,\Sigma,q_0,F)$ where, Q is a finite set of states, Σ is the finite alphabet, $q_0 \in Q$ is the start state, $F \subseteq Q$ is the set of accepting states, and δ is the transition function: $Q \times \Sigma \longrightarrow Q$. A state $d_0 \in Q$ such that $\forall a \in \Sigma$, $\delta(d_0, a) = d_0$ is called a *dead* state. If there exists a state $q \in Q$ such that $\delta(q, a)$ is not defined for some $a \in \Sigma$ then the transition function is said to be incompletely specified. It may be fully specified by adding transitions of the form $\delta(q, a) = d_0$ when $\delta(q, a)$ is undefined. The extension of δ to handle input strings is denoted by δ^* and is defined as follows: $\delta^*(q, \lambda) = q$ and $\delta^*(q, a\alpha) = \delta^*(\delta(q, a), \alpha)$ for $q \in Q$, $a \in \Sigma$ and $\alpha \in \Sigma^*$. The set of all strings accepted by A is its language, $L(A)$. $L(A) = \{\alpha | \delta^*(q_0, \alpha) \in F\}$. The language accepted by a DFA is called a *regular language*.

DFA are represented using state transition diagrams. The start state q_0 is indicated by the symbol > attached to it. Accepting states are denoted using concentric circles. The state transition $\delta(q_i, a) = q_j$ for any letter $a \in \Sigma$ is depicted by an arrow labeled by the letter a from the state q_i to the state q_j. Fig. 1 shows the state transition diagram for a sample DFA.

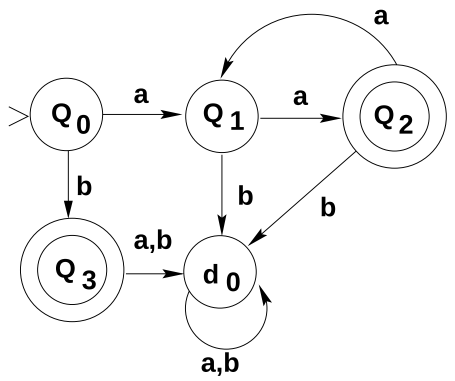

Fig. 1. Deterministic Finite State Automaton.

A labeled example $(\alpha,\ c(\alpha))$ for A is such that $\alpha \in \Sigma^*$ and $c(\alpha) = +$ if $\alpha \in L(A)$ (i.e., α is a positive example) or $c(\alpha) = -$ if $\alpha \notin L(A)$ (i.e., α is a negative example). Thus, $(a, -)$, $(b, +)$, $(aa, +)$, $(aaab, -)$, and $(aaaa, +)$ are labeled examples for the DFA of Fig. 1. S^+ will be used to denote a set of positive examples of A i.e., $S^+ \subseteq L(A)$. Similarly, S^- will be used to denote a set of negative examples of A i.e., $S^- \subseteq \Sigma^* \backslash L(A)$. A sample S will be defined as $S = S^+ \cup S^-$. A is consistent with a *sample* S if it accepts all the positive examples (i.e., all examples in S^+) and rejects all negative examples (i.e., all examples in S^-).

Given any DFA $A^{'}$, there exists a minimum state DFA (also called the *canonical DFA*) A, such that $L(A) = L(A^{'})$. Without loss of generality, we will assume that the target DFA being learned is a canonical DFA. It can be shown that any canonical DFA has at most one dead state.

A state q_i of a DFA A is *live* if there exist strings α and β such that $\alpha\beta \in L(A)$, $\delta^*(q_0, \alpha) = q_i$, and $\delta^*(q_i, \beta) \in F$. A state that is not live is called *dead*. As stated earlier, a canonical DFA can have at most one dead state and we use d_0 to denote this dead state. Given A, a finite set of strings P is said to be *live complete* if for every live state q_i of A there exists a string $\alpha \in P$ such that $\delta^*(q_0, \alpha) = q_i$ [Ang81]. For example, $P = \{\lambda, a, b, aa\}$ is a live complete set for the DFA in Fig. 1. Any superset of a live complete set is also live complete. In order to have a representation for the start state of A we assume that the string λ is part of any live complete set. The set $P^{'} = P \cup \{d_0\}$ represents all the states of A. To account for the state transitions, define a function $f : P^{'} \times \Sigma \longrightarrow \Sigma^* \cup \{d_0\}$ as follows: $f(d_0, a) = d_0$ and $f(\alpha, a) = \alpha a$. Note that $f(\alpha, a)$ denotes the state reached upon reading an input letter $a \in \Sigma$ from the state represented by the string $\alpha \in \Sigma^*$. We will let $T^{'} = P^{'} \cup \{f(\alpha, a) | (\alpha, a) \in P^{'} \times \Sigma\}$ and $T = T^{'} \backslash \{d_0\}$.

Thus, given the live complete set $P = \{\lambda, a, b, aa\}$ corresponding to the DFA in Fig. 1 we obtain the set $T = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab\}$.

3 The *ID* Algorithm

The *ID* algorithm for inference of the target DFA, its correctness proof, and complexity analysis are described in detail in [Ang81]. To keep the discussion that follows self-contained, we review *ID* briefly in this section.

Given a live-complete set P corresponding to the target canonical DFA A, *ID* constructs a partition of the set $T^{'}$ (constructed from P as described above) such that elements of $T^{'}$ that represents the same state of A are grouped together in the same block of the partition. In the process a set of distinguishing strings V is constructed such that no two distinct states of A have the same behavior on all the strings in V. When the set V has i elements, define function $E_i : T^{'} \longrightarrow 2^V$ as follows $E_i(d_0) = \phi$ and $E_i(\alpha) = \{v_j | v_j \in V, 0 \leq j < i, \alpha v_j \in L(A)\}$. Fig. 2 describes the algorithm in detail. Step 1 performs the initialization. The set T_0 represents a trivial partition with all elements belonging to a single block. The first distinguishing suffix v_0 considered is λ. The function E_0 which is computed in step 2 partitions $T^{'}$ into two blocks, representing the accepting and non-accepting states of A respectively. Step 3 refines the individual blocks of the partition of $T^{'}$ based on the behavior of the elements on the distinguishing suffixes $v_0, v_1, \ldots, v_i$. Intuitively, if two elements of $T^{'}$, say α and β, have the same behavior on the current set V (i.e., $E_i(\alpha) = E_i(\beta)$) then α and β appear to represent the same state of the target DFA. However, if the transitions out of the states represented by $E_i(\alpha)$ and $E_i(\beta)$ on some letter of the alphabet lead to different states (i.e., $E_i(f(\alpha, a)) \neq E_i(f(\beta, a))$ for some $a \in \Sigma$) then clearly, α and β cannot correspond to the same state in the target DFA. A distinguishing suffix v_{i+1} is constructed to refine the partition of $T^{'}$ such that α and β appear in separate blocks of the partition. Step 3 terminates when the set V contains a distinguishing suffix for each pair of elements in $T^{'}$ that represent non-equivalent states of A. Step 4, finally constructs the hypothesis DFA M.

3.1 Example

We now demonstrate the use of *ID* to learn the target DFA in Fig. 1. $P = \{\lambda, a, b, aa\}$ is a live complete set and $T^{'} = \{d_0, \lambda, a, b, aa, ab, ba, bb, aaa, aab\}$. Table 1 shows the computation of $E_i(\alpha)$ for the strings $\alpha \in T^{'}$. The leftmost column lists the elements α of the set $T^{'}$. Each successive column represents the function E_i corresponding to the string v_i (listed in the second row of the table).

Note that the DFA returned by the procedure is exactly the DFA in Fig. 1. Angluin [Ang81] has shown that number of membership queries needed is $O(|\Sigma| \cdot N \cdot |P|)$. Thus, *ID* runs in time polynomial in $|\Sigma|$, N, and $|P|$.

Algorithm: ID

Input: A live complete set P and a teacher to answer membership queries.
Output: A DFA M equivalent to the target DFA A.

begin
1) // *Perform initialization*
$i = 0$, $v_i = \lambda$, $V = \{\lambda\}$, $T = P \cup \{f(\alpha, b) \mid (\alpha, b) \in P \times \Sigma\}$,
and $T^{'} = T \cup \{d_0\}$
2) // *Construct function* E_0 *for* $v_0 = \lambda$
$E_0(d_0) = \phi$
$\forall \alpha \in T$ pose the membership query "$\alpha \in L(A)$?"
if the teacher's response is *yes*
then $E_0(\alpha) = \{\lambda\}$
else $E_0(\alpha) = \phi$
end if
3) // *Refine the partition of the set* $T^{'}$
while ($\exists \alpha, \beta \in P^{'}$ and $b \in \Sigma$ such that
$E_i(\alpha) = E_i(\beta)$ but $E_i(f(\alpha, b)) \neq E_i(f(\beta, b))$)
do
Let $\gamma \in E_i(f(\alpha, b)) \oplus E_i(f(\beta, b))$
$v_{i+1} = b\gamma$
$V = V \cup \{v_{i+1}\}$ and $i = i + 1$
$\forall \alpha \in T$ pose the membership query "$\alpha v_i \in L(A)$?"
if the teacher's response is *yes*
then $E_i(\alpha) = E_{i-1}(\alpha) \cup \{v_i\}$
else $E_i(\alpha) = E_{i-1}(\alpha)$
end if
end while
4) // *Construct the representation of the DFA* M
The states of M are the sets $E_i(\alpha)$, where $\alpha \in T$
The initial state q_0 is the set $E_i(\lambda)$
The accepting states are the sets $E_i(\alpha)$ where $\alpha \in T$ and $\lambda \in E_i(\alpha)$
The transitions of M are defined as follows:
$\forall \alpha \in P^{'}$
if $E_i(\alpha) = \phi$
then add self loops on the state $E_i(\alpha)$ for all $b \in \Sigma$
else $\forall b \in \Sigma$ set the transition $\delta(E_i(\alpha), b) = E_i(f(\alpha, b))$
end if
end

Fig. 2. Algorithm *ID*.

Table 1. Execution of ID.

i	0	1	2	3
v_i	λ	b	a	aa
$E(d_0)$	ϕ	ϕ	ϕ	ϕ
$E(\lambda)$	ϕ	$\{b\}$	$\{b\}$	$\{b, aa\}$
$E(a)$	ϕ	ϕ	$\{a\}$	$\{a\}$
$E(b)$	$\{\lambda\}$	$\{\lambda\}$	$\{\lambda\}$	$\{\lambda\}$
$E(aa)$	$\{\lambda\}$	$\{\lambda\}$	$\{\lambda\}$	$\{\lambda, aa\}$
$E(ab)$	ϕ	ϕ	ϕ	ϕ
$E(ba)$	ϕ	ϕ	ϕ	ϕ
$E(bb)$	ϕ	ϕ	ϕ	ϕ
$E(aaa)$	ϕ	ϕ	$\{a\}$	$\{a\}$
$E(aab)$	ϕ	ϕ	ϕ	ϕ

4 *IID* – An Incremental Extension of *ID*

We now present an incremental version of the *ID* algorithm. As stated earlier, this algorithm does not require that the live complete set of examples be available to the learner at the start. Instead the learner is intermittently presented with labeled examples. The learner constructs a model of the target DFA based on the examples it has seen and gradually refines it as new examples become available. Our learning model assumes the availability of a teacher to answer membership queries. Let M_t denote the DFA that corresponds to the learner's current model after observing t examples. Initially, M_0 is a *null* automaton with only one state (the dead state) and it rejects every string in $\Sigma^\star$. Clearly, every negative example encountered by the learner at this point is consistent with M_0. Without loss of generality we assume that the first example seen by the learner is a positive example. When the first positive example is seen M_0 is modified to accept the positive example. For each additional labeled example (α) that is observed, it is determined whether α is consistent with M_t. If α is consistent with M_t then $M_{t+1} = M_t$. Otherwise M_t is suitably modified such that α is consistent with the resulting DFA, M_{t+1}. A detailed description of the algorithm appears in Fig. 3.

4.1 Example

We now demonstrate how the incremental algorithm learns the target DFA of Fig. 1. The learner starts with a model M_0 equivalent to the *null* DFA accepting no strings. Suppose the example $(b, +)$ is encountered. The following actions are taken. $P_0 = \{\lambda, b\}$ and $P_0^{'} = \{d_0, \lambda, b\}$, $T_0 = \{\lambda, a, b, ba, bb\}$, and $T_0^{'} = \{d_0, \lambda, a, b, ba, bb\}$. The computation of the functions E_i is shown in Table 2. At this point the learner constructs a model M_1 of the target DFA (Fig. 4).

Suppose the next example observed by the learner is $(a, -)$. Since, M_1 correctly rejects a, $M_2 = M_1$ and the learner waits for additional examples. Let $(aa, +)$ be the next observed example. Since $aa \notin L(M_2)$ the learner takes the following steps to update M_2. $P_1 = \{\lambda, a, b, aa\}$, $P_1^{'} = \{d_0, \lambda, a, b, aa\}$, $T_1 =$

Algorithm: IID

Input: A stream of labeled examples and a teacher to answer membership queries.
Output: A DFA M_t consistent with all t examples observed by the learner.

begin
1) // *Perform initialization*
   $i = 0,\ k = 0,\ t = 0,\ P_k = \phi,\ T_k = \phi,\ V = \phi$
   Initialize M_t to the *null DFA*
2) // Process the first positive example
   Wait for a positive example $(\alpha, +)$
   $P_0 = Pref(\alpha)$ and $P_0^{'} = P_0 \cup \{d_0\}$
   $T_0 = P_0 \cup \{f(\alpha, b) | (\alpha, b) \in P_0 \times \Sigma\}$ and $T_0^{'} = T_0 \cup \{d_0\}$
   $v_0 = \lambda$ and $V = \{v_0\}$
   $E_0(d_0) = \phi$
   $\forall \alpha \in T_0$ pose the membership query "$\alpha \in L(A)$?"
      if the teacher's response is *yes*
      then $E_0(\alpha) = \{\lambda\}$
      else $E_0(\alpha) = \phi$
      end if
3) // *Refine the partition of the set* $T_k^{'}$ (as described in step 3 of Fig. 2)
4) // *Construct the current representation* M_t *of the target DFA* (as described in step 4 of Fig. 2)
5) // *Process a new labeled example*
   Wait for a new example $(\alpha, c(\alpha))$
   if α is consistent with M_t
   then
      $M_{t+1} = M_t$
      $t = t + 1$
      goto step 5
   else
      $P_{k+1} = P_k \cup Pref(\alpha)$ and $P_{k+1}^{'} = P_{k+1} \cup \{d_0\}$
      $T_{k+1} = T_k \cup Pref(\alpha) \cup \{f(\alpha, b) | (\alpha, b) \in (P_{k+1} \backslash P_k) \times \Sigma\}$
      and $T_{k+1}^{'} = T_{k+1} \cup \{d_0\}$
      $\forall \alpha \in T_{k+1} \backslash T_k$ fill in the values of $E_i(\alpha)$ using membership queries:
         $E_i(\alpha) = \{v_j | 0 \leq j < i, \alpha v_j \in L(A)\}$
      $k = k + 1$
      $t = t + 1$
      goto step 3
   end if
end

Fig. 3. Algorithm *IID*.

Table 2. Execution of *IID* ($k = 0$).

i	0	1
v_i	λ	b
$E(d_0)$	ϕ	ϕ
$E(\lambda)$	ϕ	$\{b\}$
$E(a)$	ϕ	ϕ
$E(b)$	$\{\lambda\}$	$\{\lambda\}$
$E(ba)$	ϕ	ϕ
$E(bb)$	ϕ	ϕ

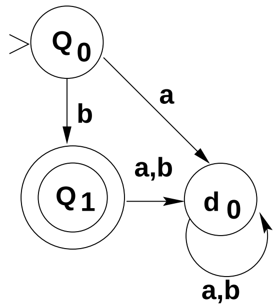

Fig. 4. Model M_1 of the Target DFA.

$\{\lambda, a, b, aa, ab, ba, bb, aaa, aab\}$, and $T_1^{'} = \{d_0, \lambda, a, b, aa, ab, ba, bb, aaa, aab\}$. The function E_1 is extended to cover the new elements belonging to $T_1 \backslash T_0$. The resulting computation of the various E_i's is depicted in Table 3.

The revised model of the target DFA (M_3) is exactly the DFA we are trying to learn (Fig. 1). Note also that at this time the set P_1 is live complete with respect to the target DFA.

4.2 Correctness Proof

The correctness of *IID* is a direct consequence of the following two theorems.

Theorem 1. *IID converges to a canonical representation of the target DFA when the set P_k includes a live complete set for the target as a subset.*

Proof. Consider an execution of *ID* given a live complete set P_l. First we demonstrate that the execution of *ID* can be made to track that of *IID* in that the set V generated during the execution of both the algorithms is the same and hence $\forall \alpha \in T_l$ the values $E_i(\alpha)$ are the same. We prove this claim by induction.

Base Case:

Both *ID* and *IID* start with $v_0 = \lambda$. At $k = 0$, *IID* has the set $P_0 \subseteq P_l$ available to it. Clearly, for all strings $\alpha, \beta \in P_0$ such that $E_0(\alpha) = E_0(\beta)$ but $E_0(f(\alpha, b)) \neq E_0(f(\beta, b))$ in the case of *IID* it is also the case that the same strings $\alpha, \beta \in P_l$ for *ID* such that $E_0(\alpha) = E_0(\beta)$ but $E_0(f(\alpha, b)) \neq E_0(f(\beta, b))$.

Table 3. Execution of *IID* ($k = 1$).

i	1	2	3
v_i	b	a	aa
$E(d_0)$	ϕ	ϕ	ϕ
$E(\lambda)$	$\{b\}$	$\{b\}$	$\{b, aa\}$
$E(a)$	ϕ	$\{a\}$	$\{a\}$
$E(b)$	$\{\lambda\}$	$\{\lambda\}$	$\{\lambda\}$
$E(ba)$	ϕ	ϕ	ϕ
$E(bb)$	ϕ	ϕ	ϕ
$E(aa)$	$\{\lambda\}$	$\{\lambda\}$	$\{\lambda, aa\}$
$E(ab)$	ϕ	ϕ	ϕ
$E(aaa)$	ϕ	$\{a\}$	$\{a\}$
$E(aab)$	ϕ	ϕ	ϕ

Assume that one such pair α, β is selected by both *ID* and *IID*. The string $\gamma \in E_0(f(\alpha, b)) \oplus E_0(f(\beta, b))$ can only be λ. Thus, the string $v_1 = b\gamma$ is the same for both the executions.

Induction Hypothesis:
Assume that after observing t examples, at some value of k $(0 \le k < l)$, when $P_k \subseteq P_l$ is available to *IID*, the sequence of strings $v_0, v_1, \ldots, v_i$ and $\forall \alpha \in T_k$ the values $E_i(\alpha)$ are the same for the executions of both *ID* and *IID*.

Induction Proof:
We now show that the same string v_{i+1} is the generated by both *ID* and *IID*. Following the reasoning presented in the base case, and given the induction hypothesis, we can state that for all strings $\alpha, \beta \in P_k$ such that $E_i(\alpha) = E_i(\beta)$ but $E_i(f(\alpha, b)) \ne E_i(f(\beta, b))$ in the case of *IID* it is also the case that the same strings $\alpha, \beta \in P_l$ for *ID* such that $E_i(\alpha) = E_i(\beta)$ but $E_i(f(\alpha, b)) \ne E_i(f(\beta, b))$.

Assume that one such pair α, β is selected by both executions. By the induction hypothesis $E_i(f(\alpha, b)) \oplus E_i(f(\beta, b))$ is identical for both. Thus, given that the same string $\gamma \in E_i(f(\alpha, b)) \oplus E_i(f(\beta, b))$ is selected, the string $v_{i+1} = b\gamma$ is identical for both executions. When the live complete set P_l is available to *IID*, $\forall \alpha \in T_l$ the values of $E_i(\alpha)$ are exactly the same as the corresponding values of $E_i(\alpha)$ for *ID*.

Given a live complete set of examples, *ID* outputs a canonical representation of the target DFA A [Ang81]. From above we know that at $k = l$ the current model (M_t) of the target automaton maintained by *IID* is identical to one arrived at by *ID*. Thus, we have proved that *IID* converges to a canonical representation of the target DFA. □

Theorem 2. *At any time during the execution of* IID*, all the t examples observed by the learner are consistent with M_t, the current representation of the target.*

Proof. Consider an example α that is not consistent with M_t. *IID* modifies M_t and constructs M_{t+1} a new representation of the target. From step 5 in Fig. 3 we see that $\alpha \in P_{k+1}$ and hence $\alpha \in T_{k+1}$. E_i is extended to all elements of $T_{k+1} \backslash T_k$. Thus, $\lambda \in E_i(\alpha)$ if α is a positive example of A and $\lambda \notin E_i(\alpha)$ if α is a negative example of A. M_{t+1} is constructed from E_j for some $j \geq i$. Since, $E_i(\alpha) \subseteq E_j(\alpha)$ it is clear that α will be accepted (rejected) by M_{t+1} if it a positive (negative) example of A. We now show that all strings μ that were consistent with M_t are also consistent with M_{t+1}.

The set $\{E_i(\beta) \mid \forall \beta \in T_k\}$ represents the set of states of M_t as shown in step 4 of the algorithm (see Fig. 3). Thus, for any string $\mu \in \Sigma^*$ and a state q of M_t there is a corresponding string $\beta \in T_k$ such that $\delta^*(q_0, \mu) = \delta^*(q_0, \beta) = q$. We say that μ is consistent with M_t if either μ is a positive example of A and $\lambda \in E_i(\beta)$ or μ is a negative example of A and $\lambda \notin E_i(\beta)$. Now assume that the algorithm observes a string α that is not consistent with M_t. T_k is modified to T_{k+1} and the function E_i is extended to the elements of $T_{k+1} \backslash T_k$. The algorithm then proceeds to refine the partition of T_{k+1} by generating the distinguishing suffixes $v_{i+1}, v_{i+2}, \ldots, v_j$ and constructing the functions $E_{i+1}, E_{i+2}, \ldots, E_j$. Consider that there exists a string $\gamma \in T_{k+1}$ such that $E_l(\beta) = E_l(\gamma)$ but $E_{l+1}(\beta) \neq E_{l+1}(\gamma)$ for some l where $i \leq l \leq j-1$. Clearly, $E_{l+1}(\beta) \oplus E_{l+1}(\gamma) = v_{l+1}$. Further, $v_{l+1} \neq \lambda$ because $v_0 = \lambda$ is already chosen as the first distinguishing suffix. Thus, $\lambda \notin E_{l+1}(\beta) \oplus E_{l+1}(\gamma)$. The string μ that originally corresponded to the state represented by β would now correspond either to the state represented by β or to the state represented by γ. Further λ will either belong to both $E_{l+1}(\beta)$ and $E_{l+1}(\gamma)$ or to neither depending on whether λ was a member of both $E_l(\beta)$ and $E_l(\gamma)$ or not. Continuing with the argument we can see that there is some string $\kappa \in T_{k+1}$ where $\kappa = \beta$ or $E_j(\beta) \oplus E_j(\kappa) \subseteq \{v_{i+1}, v_{i+2}, \ldots, v_j\}$ such that μ corresponds to κ in that $\delta^*(q_0, \mu) = \delta^*(q_0, \kappa) = q$ for some state q in M_{t+1}. Further, since $\lambda \in E_j(\kappa)$ *iff* $\lambda \in E_j(\beta)$ or equivalently $\lambda \in E_j(\kappa)$ *iff* $\lambda \in E_i(\beta)$ we see that μ is consistent with M_{t+1}. This proves that all strings that were consistent with M_t continue to be consistent with M_{t+1}. □

4.3 Complexity Analysis

Assume that at some $k = l$ the set P_l includes a live complete set for the target DFA A as a subset. From the correctness proof of the algorithm, the current representation of the target M_t is equivalent to the target A.

The size of T_l is at most $|\Sigma| \cdot |P_l| + 1$. Also, the size of V is no more than N (the number of states of A). Thus, the total number of membership queries posed by the learner is $O(|\Sigma| \cdot |P_l| \cdot N)$. Searching for a pair of strings α, β to distinguish two states in the current representation of the target takes time that is $O(T_l^2)$. Thus, the incremental algorithm runs in time polynomial in N, $|\Sigma|$, and $|P_l|$. Since the size of T_l is at most $|\Sigma| \cdot |P_l| + 1$ and the size of V is no more than N, the space complexity of the algorithm is $O(|\Sigma| \cdot |P_l| \cdot N)$.

5 Discussion

Incremental or online learning algorithms play an important role in situations where all the training examples are not available to the learner at the start. We have proposed an incremental version of Angluin's *ID* algorithm for identifying the target DFA from a set of labeled examples and membership queries. The algorithm is guaranteed to converge to the target DFA and has polynomial time and space complexities.

Angluin's L^* algorithm for learning the target DFA is based on *membership* and *equivalence* queries [Ang87]. The equivalence queries can be replaced by a polynomial number of calls to an oracle that supplies labeled examples to give an efficient PAC algorithm for learning DFA. The *IID* algorithm differs from L^* in the following respects: *IID* is guaranteed to converge to the target DFA using only labeled examples and membership queries whereas L^* makes use of equivalence queries in addition to labeled examples and membership queries to guarantee convergence. In contrast, the PAC version of L^* guarantees that with very high probability the DFA output by the algorithm would make very low error (when compared to the unknown target).

Parekh and Honavar's algorithm for regular inference [PH96] searches a lattice of FSA generated by successive state mergings of a *prefix tree automaton* (PTA) for a set of positive examples of the target grammar. The lattice is represented compactly as a version space and is searched using membership queries. In the incremental framework they assume that the positive examples (needed to construct a *structurally complete* set) are provided intermittently. The algorithm augments the version space as needed in response to these new positive examples. Assuming that the examples are provided in increasing order by length, convergence to the target FSA is guaranteed when a structurally complete set of positive examples has been processed. Though provably correct, this algorithm has practical limitations because the size of the lattice grows exponentially with the number of states of the PTA.

The incremental version of the *regular positive and negative inference* (RPNI) algorithm [OG92] for regular grammar inference [Dup96] is also based on the idea of a lattice of partitions of the states of a PTA for a set of positive examples. It uses information from a set of negative examples to guide the ordered search through the lattice. It requires storage of all the examples seen by the learner to ensure that each time the representation of the target is modified, it stays consistent with all examples seen previously. The algorithm runs in time that is polynomial in the sum of lengths of the positive and negative examples and is guaranteed to converge to the target DFA when the set of examples seen by the learner include a *characteristic sample* (see [OG92]) for the target automaton as a subset.

Porat and Feldman's incremental algorithm for learning automata uses a *complete ordered sample* [PF91]. A complete ordered sample includes all the strings in Σ^* in strict lexicographic order. The algorithm maintains a current hypothesis which is updated upon seeing a counter example and is guaranteed to converge in the limit provided the examples appear in strict lexicographic order. The algorithm needs only a finite working storage. However, it is based on an ordered presentation of examples and requires a consistency check with all the previous examples when the hypothesis is modified.

Our framework for incrementally learning a target DFA from labeled examples does not require storage of all the examples. Only those examples that are inconsistent with the current representation of the target are required to be stored (implicitly) by the learner. The algorithm does not require any specific ordering of the labeled examples. Furthermore, the incremental modification of learner's representation of the target DFA is guaranteed to be consistent with all the examples processed by the learner at any stage during learning and no explicit consistency check is needed. The reader should note that like the *ID* algorithm, this incremental version also avails of a knowledgeable teacher capable of answering membership queries.

The learner's reliance on the teacher to provide accurate responses to membership queries poses a potential limitation in applications where a reliable teacher is not available. We are exploring the possibility of learning in an environment where the learner does not have access to a teacher. The algorithms due to Dupont [Dup96] and Porat & Feldman [PF91] operate in this framework. Some open problems include whether the limitations of these algorithms (e.g., need for storage of all the previously seen examples and complete lexicographic ordering of examples) can be overcome without sacrificing efficiency and guaranteed convergence to the target. Porat and Feldman proved a strong negative result stating that there exists no algorithm which when operating with finite working storage can incrementally learn the target DFA from an arbitrary presentation [PF91]. In this context, it is of interest to explore alternative models of learning that: relax the convergence criterion (for example, allow PAC style approximate learning of the target within a given error bound); provide for some additional hints to the learning algorithm (like a bound on the number of states of the target DFA); include a helpful teacher that carefully guides the learner perhaps by providing *simple* examples first (for example see [PH97]).

Acknowledgements

The authors are grateful to the Department of Computer Science at Iowa State University (where a major portion of this research was undertaken) for the research and computing facilities made available to them. Rajesh Parekh is thankful to the Allstate Research and Planning Center for the research support provided to him. Codrin Nichitiu's visit to Iowa State University during the summer of 1996 was made possible by the financial support from Ecole Normale Superieure de Lyon. Vasant Honavar's research was partially supported by the National Science Foundation (through grants IRI-9409580 and IRI-9643299) and the John Deere Foundation.

References

[Ang81] D. Angluin. A note on the number of queries needed to identify regular languages. *Information and Control*, 51:76–87, 1981.

[Ang87] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.

[BF72] A. Biermann and J. Feldman. A survey of results in grammatical inference. In S. Watanabe, editor, *Frontiers of Pattern Recognition*, Academic Press, pages 31–54, 1972.

[CM96] D. Carmel and S. Markovitch. Learning models of intelligent agents. In *Proceedings of the AAAI-96 (vol. 1)*, AAAI Press/MIT Press, pages 62 – 67, 1996.

[Dup96] P. Dupont. Incremental regular inference. In L. Miclet and C. Higuera, editors, *Proceedings of the Third ICGI-96*, Montpellier, France, *Lecture Notes in Artificial Intelligence 1147*, Springer-Verlag, pages 222–237, 1996.

[FB75] K. S. Fu and T. L. Booth. Grammatical inference: Introduction and survey (part 1). *IEEE Transactions on Systems, Man and Cybernetics*, 5:85–111, 1975.

[Gol78] E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.

[Lan95] P. Langley. *Elements of Machine Learning*. Morgan Kauffman, Palo Alto, CA, 1995.

[MQ86] L. Miclet and J. Quinqueton. Learning from examples in sequences and grammatical inference. In G. Ferrate *et al*, editors, *Syntactic and Structural Pattern Recognition*, NATO ASI Series Vol. F45, pages 153–171, 1986.

[OG92] J. Oncina and P. García. Inferring regular languages in polynomial update time. In N. Pérez *et al*, editors, *Pattern Recognition and Image Analysis*, World Scientific, pages 49–61, 1992.

[PC78] T. Pao and J. Carr. A solution of the syntactic induction-inference problem for regular languages. *Computer Languages*, 3:53–64, 1978.

[PF91] S. Porat and J. Feldman. Learning automata from ordered examples. *Machine Learning*, 7:109–138, 1991.

[PH96] R. G. Parekh and V. G. Honavar. An incremental interactive algorithm for regular grammar inference. In L. Miclet and C. Higuera, editors, *Proceedings of the Third ICGI-96*, Montpellier, France, *Lecture Notes in Artificial Intelligence 1147*, Springer-Verlag, pages 238–250, 1996.

[PH97] R. G. Parekh and V. G. Honavar. Learning dfa from simple examples. In *Proceedings of the Eighth International Workshop on Algorithmic Learning Theory (ALT'97)*, Sendai, Japan, *Lecture Notes in Artificial Intelligence 1316*, Springer-Verlag, pages 116–131, 1997. Also presented at the *Workshop on Grammar Inference, Automata Induction, and Language Acquisition* (ICML'97), Nashville, TN. July 12, 1997.

[PH98] R. G. Parekh and V. G. Honavar. Grammar inference, automata induction, and language acquisition. In R. Dale, H. Moisl, and H. Somers, editors, *Handbook of Natural Language Processing*. Marcel Dekker, 1998. (To appear).

[Pit89] L. Pitt. Inductive inference, dfas and computational complexity. In *Analogical and Inductive Inference, Lecture Notes in Artificial Intelligence 397*, Springer-Verlag, pages 18–44, 1989.