Triangle: Language, Literature, Computation, n. 8, 2012
Publicacions Universitat Rovira i Virgili · ISSN: 2013-939X
https://revistes.urv.cat/index.php/triangle

A Survey of State Merging Strategies for DFA Identification in the Limit

Cristina Tîrnăucă

Research Group on Mathematical Linguistics
Universitat Rovira i Virgili
Tarragona, Spain
E-mail: `cristina.bibire@estudiants.urv.cat`

Summary. Identification of deterministic finite automata (DFAs) has an extensive history, both in passive learning and in active learning. Intractability results by Gold [5] and Angluin [1] show that finding the smallest automaton consistent with a set of accepted and rejected strings is NP-complete. Nevertheless, a lot of work has been done on learning DFAs from examples within specific heuristics, starting with Trakhtenbrot and Barzdin's algorithm [15], rediscovered and applied to the discipline of grammatical inference by Gold [5]. Many other algorithms have been developed, the convergence of most of which is based on characteristic sets: RPNI (Regular Positive and Negative Inference) by J. Oncina and P. García [11, 12], Traxbar by K. Lang [8], EDSM (Evidence Driven State Merging), Windowed EDSM and Blue-Fringe EDSM by K. Lang, B. Pearlmutter and R. Price [9], SAGE (Self-Adaptive Greedy Estimate) by H. Juillé [7], etc. This paper provides a comprehensive study of the most important state merging strategies developed so far.

1 Introduction

The problem of DFA identification from examples was first mentioned in a paper by Gold [4] back in 1967, when he also introduced the notion of learning formal languages. Motivated by observing how children acquire their first language, he suggested that learning is an infinite process of guessing of grammars that does not terminate in finite steps but only converges in the limit.

In 1973, Trakhtenbrot and Barzdin described a polynomial time algorithm (henceforth denoted TB) for constructing the smallest DFA consistent with a completely labeled training set (a set that contains all the words up to a certain length).

Five years later Gold rediscovered the TB algorithm and applied it to the discipline of grammatical inference (uniformly complete samples are not required). He also specified the way to obtain indistinguishable states using the so called state characterization matrices. If the data set does not contain the characteristic set mentioned above the algorithm guarantees the consistency at the cost of outputting the prefix tree acceptor (PTA) of the positive sample.

In 1992 Oncina and Garcia proposed the RPNI (Regular Positive and Negative Inference) algorithm [12], and in the same year Lang described the TB algorithm and generalized it to produce a (not necessarily minimum) DFA consistent with a sparsely labeled tree [8]. The algorithm (Traxbar) can deal with incomplete data sets as well as complete data sets.

All the algorithms mentioned above are data-dependent (also called data-driven) and they do not take into account any evidence present in the sample. Since 1997, several evidence-driven algorithms have been proposed. The main contribution to the field in this direction is due to the Abbadingo One contest which took place in 1997. The competition was held by Kevin J. Lang and Barak A. Pearlmutter and presented the challenge of predicting, with 99% accuracy, the labels that an unseen finite state automaton would assign to test data given training data consisting of positive and negative examples. There were two winners: Robert Price, for solving the 60,000-string, 506-state problem and Hugues Juillé, for solving the 1,521-string, 65-state problem.

2 Preliminaries

In this paper we use the standard definitions and notations of formal language theory. The reader is referred to [6, 10] for further information about this domain. Let Σ be a finite set of symbols called *alphabet* and let Σ^* be the set of strings over Σ. A *language* L over Σ is a subset of Σ^*. The elements of L are called *words*. Let u, v, w be strings in Σ^* and $|w|$ be the length of the string w. λ is a special string called the *empty* string and has length 0. Given a string $w = uv$, u is a *prefix* of w and v is a *suffix* of w. We define:

$Pr(L) = \{u \in \Sigma^* \mid \exists v \in \Sigma^* \text{ such that } uv \in L\}$,
$Suf(L) = \{v \in \Sigma^* \mid \exists u \in \Sigma^* \text{ such that } uv \in L\}$,
$L_u = \{v \in \Sigma^* \mid uv \in L\}$.

2.1 Finite automata

A *deterministic finite automaton* is a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, Σ is a finite alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and δ is a partial function that maps $Q \times \Sigma$ to Q. The transition function δ can be extended to strings by doing $\delta(q, \lambda) = q$ and $\delta(q, ua) = \delta(\delta(q, u), a)$, $\forall q \in Q$, $\forall u \in \Sigma^*$, $\forall a \in \Sigma$. A word u is accepted by A if $\delta(q_0, u) \in F$. The set of words accepted by A is denoted by $L(A)$.

A *non-deterministic finite automaton* (NFA) is defined like a DFA with the only difference that that transition function is a mapping from $Q \times \Sigma$ to 2^Q. In general, a *finite state automaton* (FSA) refers to either a DFA or an NFA.

A finite set S_+ is called a *positive sample* for the language L if $S_+ \subseteq L$. Analogous, a *negative sample* for the language L is a finite set S_- such that $S_- \subseteq \Sigma^* \setminus L$. A completely labeled data set includes all example strings up to a given length.

We say that an automaton is *consistent* with a sample if it accepts all positive examples and rejects all negative examples. A set is said to be *structurally complete* with respect to a DFA A if it covers each transition of A and uses each final state of A.

2.2 Quotient automaton

For any set S, a partition π is a set of pairwise disjoint nonempty subsets of S whose union is S. Let s denote an element of S and let $B(s, \pi)$ denote the unique element, or *block* of π containing s. Given two partitions π_i and π_j, π_i is *finer* than π_j if every block of π_j is a union of one or several blocks of π_i. We denote this by $\pi_i \preceq \pi_j$.

Let $A = (Q, \Sigma, \delta, q_0, F)$ be an FSA. The *quotient automaton* $A/\pi = (Q', \Sigma, \delta', B(q_0, \pi), F)$ is defined by:

- $Q' = Q/\pi = \{B(q, \pi) | q \in Q\}$,
- $F' = \{B \in Q' | B \cap F \neq \emptyset\}$,
- $\delta'(B, a) = \{B' \in Q' \mid \exists q \in B, q' \in B' \text{ such that } q' \in \delta(q, a)\}$ for all $B \in Q'$, $a \in \Sigma$.

The states of Q belonging to the same block B of the partition π are said to be *merged* together. The set of all derived automata obtained by systematically merging the states of A represents a *lattice* of FSA [13]. Given a canonical DFA A and a set S_+ that is structurally complete with respect to A, the lattice derived from $PTA(S_+)$ is guaranteed to contain A [2].

2.3 Prefix tree acceptor - augmented prefix tree acceptor

Given a set S_+, let $PTA(S_+)$ denote the *prefix tree acceptor* for S_+. $PTA(S_+)$ is a DFA that contains a path from the start state to an accepting state for each string in S_+ modulo common prefixes. Clearly, $L(PTA(S_+)) = S_+$.

More formally, $PTA(S_+) = (Q, \Sigma, \delta, q_0, F)$ where:

- $Q = Pr(S_+)$,
- $\Sigma =$ the alphabet of S_+,
- $\delta(u, a) = ua$, for all $u, ua \in Q$
- $q_0 = \begin{cases} \lambda, \text{ if } S_+ \neq \emptyset \\ \emptyset, \text{ otherwise.} \end{cases}$
- $F = S_+$.

An *augmented prefix tree acceptor* (APTA) with respect to S_+ and S_-, denoted $APTA(S_+, S_-)$, is defined as a 6-tuple $(Q, \Sigma, \delta, q_0, F_+, F_-)$ where:

- $Q = Pr(S_+ \cup S_-)$,
- $\Sigma =$ the alphabet of $S_+ \cup S_-$,
- $\delta(u, a) = ua$ for all u, $ua \in Q$,
- $q_0 = \begin{cases} \lambda, \text{ if } (S_+ \cup S_-) \neq \emptyset \\ \emptyset, \text{ otherwise.} \end{cases}$
- $F_+ = S_+$
- $F_- = S_-$

Example

Consider sets $S_+ = \{0, 1, 010, 011\}$ and $S_- = \{01, 11\}$, then the $PTA(S_+)$ and the $APTA(S_+, S_-)$ are illustrated in Figure 1.

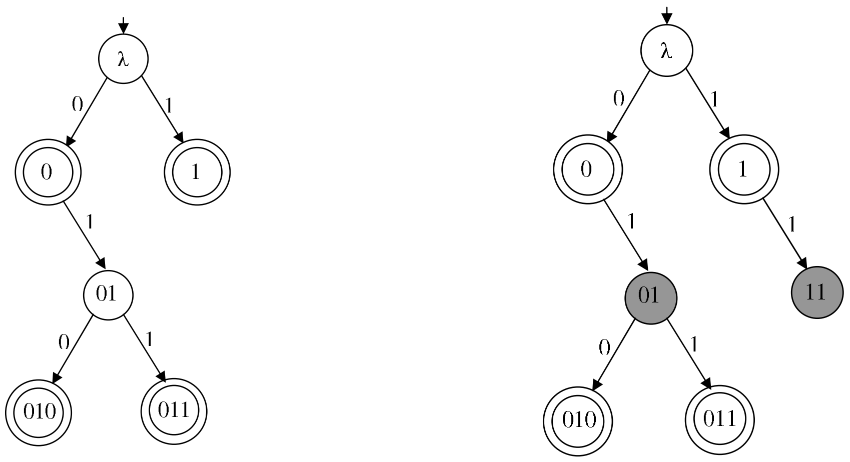

Fig. 1. $PTA(S_+)$ and $APTA(S_+, S_+)$

3 Grammatical Inference

Grammatical inference is known as one of the most attractive paradigms of scientific learning. The goal of any *inference algorithm* is roughly to discover a grammar that generates a given set of sample sentences. The learning model that was first introduced (and also the most used) is *learning in the limit*. In this setting, the learner has access to either a growing sequence of positive examples (*learning from text*), or both positive and negative information (*learning from informant*), and has to output his hypotheses. After some finite time, the guesses must converge to the correct language.

Gold [4] shows that given a positive presentation one cannot identify the class of regular languages, and that any recursively enumerable class is identifiable using a complete presentation (positive and negative data).

Learning paradigms seem not to be applicable to human learning:

- Gold's identification in the limit framework has been criticized as children seem to learn natural language without negative examples;
- All learning paradigms assume a known representation class;
- Some learnability results are based on enumeration.

The problem of minimum automaton identification from incompletely labeled training data has been proved to be NP-complete [5]. However, the average case is tractable [8].

4 Algorithms for Learning DFA

Below we present the most important algorithms for DFA identification from examples.

4.1 The Trakhtenbrot and Barzdin algorithm

The algorithm proposed by Trakhtenbrot and Barzdin [15] produces the canonical DFA for any language, from a complete data set, in polynomial time. Perhaps the biggest advantage of this algorithm is its simplicity. Furthermore, it deals with data sets of various sizes in a very short time frame. Unfortunately, it also has a disadvantage since the algorithm merges compatible nodes in breadth-first order despite evidence or clues present within the training data. In other words, the attempted merge order is predetermined, and very little search of the problem space is necessary to determine the next merge pair.

It is important to note that the advantages and disadvantages outlined above apply not only to the algorithm proposed by Trakhtenbrot and Barzdin [15] for complete data sets but also for the modified version of this algorithm for incomplete training data.

Given an APTA $A = (Q, \Sigma, \delta, q_0, F_+, F_-)$, we say that two states p and q are *distinguishable* in A if there exists a word u in Σ^* such that $(\delta(p,u) \in F_+$ and $\delta(q,u) \in F_-)$ or $(\delta(p,u) \in F_-$ and $\delta(q,u) \in F_+)$. Otherwise, p and q are *not distinguishable* in A. For a detailed description of the procedure *distinguishable*(p,q,A), the reader is referred to [3].

U is a set of *unique* nodes; that is, nodes that are pairwise distinguishable. The algorithm starts by adding the root of the APTA to the list of unique nodes. Then, it visits each proceeding node q of the APTA in breadth-first order, compares the subtree rooted at q with the sub-tree rooted at each node in the unique nodes list. If q is pairwise distinguishable from each node from U, it appends q to the end of the list. Otherwise, it disconnects q from the APTA.

An upper bound on the running time of the algorithm is mn^2, where m is the total number of nodes in the initial APTA and n the total number of states in the final hypothesis (more details in [9]).

The TB Algorithm is described bellow:

Algorithm 1 TB Algorithm

```
TB(APTA($S_+, S_-$))
$A$:=APTA($S_+, S_-$);
$U := \{\lambda\}$;
While $p$ visits each proceeding node of $A$ in breadth-first order
   dist:=true;
   While ($q$ in $U$) and (dist)
      dist:=distinguishable($p$, $q$, $A$);
   End While;
   If dist then append $p$ to $U$
            else disconnect $p$ from $A$;
End While;
```

4.2 Gold's algorithm

The algorithm proposed by Gold [2] is based on the so called state characterization matrix.

A *state characterization matrix* over an alphabet Σ is a triple (S, E, T) where S, E are finite subsets of Σ^* and $T : (S \cup S\Sigma)E \to \{0, 1, \uparrow\}$. The elements of S are called *states*, and those of E are called *experiments*. The function T is defined using the sets S_+ and S_- as follows. For all $u \in S \cup S\Sigma$ and $v \in E$,

$$T(uv) = \begin{cases} 1, \text{ if } uv \in S_+ \\ 0, \text{ if } uv \in S_- \\ \uparrow, \text{ otherwise.} \end{cases}$$

Every element u of $S \cup S\Sigma$ defines a row which will be called *row(u)*. Given $u, v \in S \cup S\Sigma$, we say that *row(u)* is *obviously different* from *row(v)*, and we write *row(u)* $\not\simeq$ *row(v)*, if there exists an experiment $e \in E$ such that $T(ue), T(ve) \in \{0, 1\}$ and $T(ue) \neq T(ve)$.

A state characterization matrix is called *closed* if none of the rows in $S\Sigma - S$ is obviously different from the rows in S.

Gold's algorithm was initially established using Mealy machines. Here we use Moore machines. Doing it this way, the comparisons between algorithms can be seen more clearly.

A *Moore machine* is a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, \Phi)$, where Σ (resp. Γ) is the input (resp. output) alphabet, δ is a partial function that maps $Q \times \Sigma$ in Q and Φ is a function that maps Q in Γ called output function. The behavior of M is given by the partial function $t_M : \Sigma^* \Rightarrow \Gamma$ defined by $t_M(u) = \Phi(\delta(q_0, u))$, for every u in Σ^* such that $\delta(q_0, u)$ is defined.

Given two finite sets of words, S_+ and S_-, we define the *prefix Moore machine* $PTM(S_+, S_-)$ as the Moore machine having $\Gamma = \{0, 1, \uparrow\}$, $Q = Pr(S_+ \cup S_-)$, $q_0 = \lambda$ and $\delta(u, a) = ua$ if $u, ua \in Q$ and $a \in \Sigma$. For every state u the value of the output function associated to u is 1, 0 or $\uparrow$ (undefined) depending whether u belongs to S_+, S_- or it is in the complementary set of $S_+ \cup S_-$.

There are exactly two places where the algorithm may be nondeterministic. The first one is when there are several rows from $S\Sigma - S$ that can be moved to S. The second is when we are building the output automaton and there are several obviously different rows (states) where the transition can be assigned. One solution that can be adopted for both situations is to choose the smallest row in lexicographic order.

For a better understanding of Gold's algorithm the reader is referred to [3] in which it is described as a procedure of merging states in the prefix Moore machine of the sample.

4.3 RPNI algorithm

The regular positive and negative inference (RPNI) algorithm proposed by Oncina and Garcia [12] is a polynomial time algorithm that identifies a target DFA, given the sample $S = S_+ \cup S_-$. It was shown that if the sample includes a characteristic set then the algorithm is guaranteed to return a canonical representation of the target DFA [12].

In order to present the algorithm we need some definitions and notations.

- the set $S_p(L)$ of short prefixes of L is
  $S_p(L) = \{u \in Pr(L) \mid \forall v \in \Sigma^* \text{ such that } L_u = L_v, u \leq v\}$

```
**Algorithm 2** Gold's Algorithm
**Gold**($S_+$,$S_-$)
$S := \{\lambda\}$; $E := Suf(\Sigma^{-1}(S_+ \cup S_-))$;
Build the table ($S$, $E$, $T$);
**While** there exists $s' \in (S\Sigma - S)$ s.t. $row(s') \ncong row(s), \forall s \in S$
    Choose any $s'$;
    $S := S \cup \{s'\}$;
    Update ($S$, $E$, $T$);
**End While**;
$Q := S$; $q_0 := \lambda$;
**For** all $s \in S$
    $\Phi(s) := T(s)$;
    **For** all $a \in \Sigma$
        **If** $sa \in S$ **then** $\delta(s,a) := sa$
                **else** $\delta(s,a) :=$ any $s' \in S$ s.t. **not**$(row(sa) \ncong row(s'))$;
    **End For**;
**End For**;
$M := (Q, \Sigma, \{0,1,\uparrow\}, \delta, q_0, \Phi)$;
**If** $M$ is consistent with $(S_+, S_-)$ **then** Return($M$)
                                **else** Return ($PTM(S_+, S_-)$).
```

- the kernel $N(L)$ of L is
  $N(L) = \{\lambda\} \cup \{ua \in Pr(L) \,|\, u \in S_p(L), a \in \Sigma\}$

A sample $S = S_+ \cup S_-$ is said to be *characteristic* with respect to a regular language L (with the canonical DFA A) if it satisfies the following two conditions:

1. $N(L) \cap L \subseteq Pr(S_+)$,
2. $\forall u \in S_p(L), v \in N(L)$, if $L_u \neq L_v$ then $\exists w \in \Sigma^*$ such that ($uw \in S_+$ and $vw \in S_-$) or ($uw \in S_-$ and $vw \in S_+$).

Intuitively, condition 1 implies structural completeness with respect to A and condition 2 implies that for any distinct states of A there is a suffix w that correctly distinguishes them.

Notice that:

- if you add more strings to a characteristic sample it remains characteristic,
- there can be many different characteristic samples.

The RPNI algorithm is described below:

The convergence of the RPNI algorithm relies on the fact that sooner or later, the set of labeled examples seen by the learner will include a characteristic set.

If the stream of examples provided to the learner is drawn according to a simple distribution, the characteristic set would be made available relatively early (during

```
Algorithm 3 RPNI Algorithm
RPNI (PTA($S_+$), $S_-$)
$A := PTA(S_+)$;
$K := \{q_0\}$; $Fr := \{\delta(q_0, a) | a \in \Sigma^*\}$;
While $Fr \neq \emptyset$
    choose $q$ from $Fr$;
    If $\exists p \in K$ such that $L(\text{dmerge}(A, p, q)) \cap S_- = \emptyset$
        then $A := dmerge$(A,p,q)
        else $K := K \cup \{p\}$;
    $Fr := \{\delta(q, a) | q \in K\} - K$;
End While.
```

learning) with a sufficiently high probability, so the algorithm will converge quickly to the desired target.

RPNI is an optimistic algorithm: at any step two states are compared and the question is: can they be merged? No positive evidence can be produced; merging will take place each time that such a merge does not produce inconsistency. Of course an early mistake can have disastrous effects and a breadth first exploration of the lattice is likely to be better.

4.4 Traxbar algorithm

A variation of the Trakhtenbrot and Barzdin algorithm (Traxbar) was implemented by Lang [8] in order to show that random DFAs can be approximately learned from sparse uniform examples.

The modifications made to the algorithm were needed to maintain consistency with incomplete training sets. For instance, unlabeled nodes and missing transitions in the APTA needed to be considered.

The simple extensions added to the Trakhtenbrot and Barzdin algorithm are summarized as follows.

If node p is to be merged with node q then:

- labels of labeled nodes in the sub-tree rooted at p must be copied over their respective unlabeled nodes in the sub-tree rooted at q;
- transitions in any of the nodes in the sub-tree rooted at p that do not exist in their respective node in the sub-tree rooted at q must be spliced in.

An important observation is that the definition of distinguishable states does not change. However, because the sample is not complete, we do not know for all the states whether they are accepting or rejecting.

As a result of these changes, the Traxbar algorithm will produce a (not necessarily minimum size) DFA that is consistent with the training set.

```
**Algorithm 4** Traxbar Algorithm
Traxbar(APTA($S_+, S_-$))
$A$:=APTA($S_+, S_-$);
$U := \{\lambda\}$;
While $p$ visits each proceeding node of $A$ in breadth-first order
   dist:=true;
   While ($q$ in $U$) and (dist)
      dist:=distinguishable($p$, $q$, $A$);
   End While;
   If dist then append $p$ to $U$
            else $A := $ merge($p, q, A$);
End While;
```

Implementing this label copying process correctly requires careful attention to details, but the conceptually important thing is that the resulting merger of different parts of the training set increases its effective density and constrains succeeding choices of which state to merge. This can be good or bad depending on whether the algorithm's greedy initial state merging choices are correct. If they are not, the resulting merger of unrelated sets of labels can cause the training set to look random and lead to an explosion in the size of the hypothesis. Conversely, if the initial choices are correct there can be a snowballing of constraints leading to a highly accurate hypothesis. Because the algorithm's initial choices are so important they should be based on as much evidence as possible.

4.5 EDSM algorithm

Price won the Abbadingo One Learning Competition by using an evidence-driven state merging (EDSM) algorithm. Essentially, he realized that an effective way of choosing which pair of nodes to merge next within the APTA would simply involve selecting the pair of nodes whose subtrees share the most similar labels.

A post-competition version of the EDSM algorithm as described by Lang, Pearlmutter and Price [9] is included below.

The score is calculated by assigning one point for each overlapping label node within the subtrees rooted at the nodes considered for merging. If the two nodes are distinguishable, the score is $-\infty$. No merging is possible when all the remaining pairs of nodes are pairwise distinguishable.

The general idea of the EDSM approach is to avoid bad merges by selecting the pair of nodes within the APTA which has the highest score. It is expected that the scoring will indicate the correctness of each merge, since on average, a merge that survives more label comparisons is more likely to be correct [9].

```
**Algorithm 5** EDSM Algorithm
EDSM(APTA($S_+, S_-$))
$A$:=APTA($S_+, S_-$);
For all pairs $(p, q)$ in $Q$
  compute $score(p, q)$;
End For;
Repeat
    Find $p, q$ in $Q$ such that $score(p, q)$ is maximum and positive;
    $A := \text{merge}(p, q, A)$;
until no merge is possible.
```

Unfortunately, the difficulty of detecting bad merge choices increases as the density of the training data decreases. Since the number of labeled nodes decreases within the APTA as the training data becomes more sparse, the idea of selecting merge pairs based on the highest score proves less effective.

This explains why the EDSM approach did well with large automata but not as well with low density problems. Considering every potential merge pair at each stage of the inference process is computationally expensive.

4.6 Windowed EDSM algorithm

To improve the running time of the EDSM algorithm, one possibility is to merge only those nodes that lie within a fixed sized window from the root node of the APTA. The recommended size of the window is twice the number of states in the target DFA. This might be a problem when the size of the target DFA is not known. However, a simple solution is to execute the algorithm several times while gradually increasing the window size. Unfortunately, this approach also has a drawback since there is no way of knowing when to stop increasing the window size. The Windowed EDSM algorithm is described below.

As expected, the running time of the W-EDSM algorithm is much better than that of EDSM. The improvement in the running time is due to the reduction of the search space at each merge step of the algorithm. Of course, this can harm the performance of the algorithm in the relatively rare case in which high scoring merges involving deep nodes may be excluded from the window. For instance, the ideal algorithm would consider all possible merge pairs, and select for merging those pairs of nodes that score highest. Since such an algorithm is computationally expensive, only a subset of possible merge pairs are to be considered.

We denoted by n the number of states of the target DFA. Q is the set of states of the APTA A and the score is computed in the same way as in the *EDSM* algorithm.

Algorithm 6 W-EDSM Algorithm

```
**W-EDSM**(APTA($S_+, S_-$))
$A$:=APTA($S_+, S_-$);
$W := \{\lambda\}$;
$winsize := 2 * n$;
**Repeat**
   **While** ($size(W) \leq winsize$) and ($W \neq Q$)
         find the next node $q$ in breadth-first order;
         add $q$ to $W$;
   **End While**;
   $max := -1$;
   **For** all $p, q$ in $W$ do
         compute $score(p, q)$;
         **If** $score(p, q) > max$ **then** $max := score(p, q)$;
                                             $p_{max} := p$ ;
                                             $q_{max} := q$ ;
   **End For**;
   **If** ($max > -1$) **then** $A :=$ merge($p_{max}, q_{max}, A$)
                      **else** $winsize := 2 * winsize$;
**until** ($W = Q$).
```

It is conjectured that a tight upper bound on the running time of the W-EDSM algorithm is closer to $m^3 n$ than to $m^4 n$ where m is the number of nodes in the APTA and n is the number of states in the final hypothesis [9].

4.7 Blue-fringe algorithm

An alternative windowing method to that used by the W-EDSM algorithm is also described by Lang, Pearlmutter and Price [9]. It uses a red and blue coloring scheme to provide a simple but effective way of choosing the pool of merge candidates at each merge level in the search. The Blue-fringe windowing method helps the implementation of the algorithm and improves on its running time.

Similar to the W-EDSM algorithm, Blue-fringe EDSM places a restriction on the merge order. For example, the algorithm always starts with the root node colored red and its children blue resulting in a maximum of two possible merge pairs to choose from at the start.

Considering the sparseness of some of the data sets, one would assume that the pool of possible merge pairs would be greatest at the start and then gradually decrease to save on the running time. All the evidence in the training data would be considered at the start, which helps to make the correct decisions in the initial stage of the algorithm. This is important since changing the label of a node after it has

```
**Algorithm 7** Blue-fringe Algorithm
**Blue-fringe**(APTA($S_+, S_-$))
$A$:=APTA($S_+, S_-$);
$Red := \{\lambda\}$;
$Blue := \emptyset$;
**Repeat**
    **For** all $p$ in $Red$ do
        **For** all sons $q$ of the node $p$ do
            **If** $q$ not in $Red$ **then** add $q$ to $Blue$;
        **End For**;
    **End For**;
    $max := -1$;
    **For** all ($p$ in $Red$) and all ($q$ in $Blue$) do
        compute $score(p, q)$;
        **If** $score(p, q) > max$ **then** $max := score(p, q)$;
                                         $p_{max} := p$ ;
                                         $q_{max} := q$ ;
    **End For**;
    **If** there exist ($q$ in $Blue$) such that for all $p$ in $Red$ $score(p, q) = -\infty$
        **then** add $q$ to $Red$;
             remove $q$ from $Blue$;
        **else** $A :=$ merge($p_{max}, q_{max}, A$);
**until** ($Blue = \emptyset$) and ($max = -1$).
```

been labeled as a result of a merge is not possible within this algorithm. Instead, as the algorithm progresses, the number of red and blue nodes increases, resulting in a large number of possible merge choices.

Despite the restriction in the merge order and the reduction in merge choices at each merge level within the search tree, Blue-fringe EDSM is very effective and its inference capabilities are comparable with those of W-EDSM.

The score is computed in the same way as in the *EDSM* algorithm. We should add that when the algorithm promotes the blue node which is distinguishable from each red node, it chooses the shallowest one.

The upper bound on the running time of the Blue-fringe algorithm is mn^3 where m is the total number of nodes and n is the total number of states in the initial APTA and final hypothesis, respectively [9]. It is important to note that this is of an order of magnitude greater than the Traxbar algorithm.

4.8 SAGE algorithm

The inference engine used by Hugues Juillé is vastly different from the algorithms discussed thus far. Actually, Juillé and Pollack were the first to use random sampling techniques on search trees as a heuristic to control the search. The idea of using a tree to visualize the search space is very practical.

The algorithm is based on a Self-Adaptive Greedy Estimate search procedure (SAGE). Each iteration of the search procedure consists of two phases: a *construction phase* and a *competition phase*.

It is in the construction phase that the list of alternatives or merge choices is determined. All the alternatives in the list have the same depth within the search tree. Each member of a population of processing elements is then assigned one alternative from the list. Each processing element then scores its assigned alternative by randomly choosing a path down the search tree until a leaf node is reached or a valid solution is encountered. Next, the competition phase kicks in.

The scores assigned to each alternative in the search tree are then used to guide the search. The meta-level heuristic determines whether to continue with the next level of search. If so, each processing element is randomly assigned one of the children of its assigned alternative. The search ends when no new node can be expanded upon.

To avoid an exhaustive search of the problem space only the first set of initial merges are explored. These are thought of as the most critical merge choices since each merge places constraints on future merges.

5 Concluding Remarks

We have revised the Trakhtenbrot and Barzdin (TB), Gold, RPNI and Lang algorithms. As can be seen in [3], the first two are in fact the same, while the first description that Lang provides of the TB algorithm agrees with it only when the sample is uniformly complete. The extension he gives to obtain consistent hypotheses is in fact the RPNI algorithm. The evidence driven state merging technique gives better results on large and sparse data sets, mainly because we avoid doing “bad mergings", based on the evidence we have. A totally different approach is the algorithm introduced by Hugues Juillé, in which random sampling techniques are used.

Our main contribution consists of presenting these algorithms in the same framework, which makes the comparison between them much easier and offers a solid base for those who are in the beginning of their research career in Grammatical Inference in general, and state merging strategies for identification in the limit of DFA, in particular.

Acknowledgements

This work was possible thanks to the FPU Fellowships AP2004-6968 from the Spanish Ministry of Education and Science.

References

1. Angluin, D. (1978). On the complexity of minimum inference of regular sets. *Information and Control*, 39(3): 337–350.
2. Dupont, P., L. Miclet and E. Vidal (1994). What is the search space of the regular inference?. In R.C. Carrasco and J. Oncina (eds.), *Proceedings of the* 2nd *International Colloquium on Grammatical Inference (ICGI '94)*, pp. 25-37. Berlin: Springer.
3. García, P., A. Cano and J. Ruiz (2000). A comparative study of two algorithms for automata identification. In A.L Oliveira (ed.), *Proceedings of the* 5th *International Colloquium on Grammatical Inference (ICGI '00)*, pp. 115-126. Berlin: Springer.
4. Gold, E.M. (1967). Language identification in the limit. *Information and Control*, 10(5):447-474.
5. Gold, E.M. (1978). Complexity of automaton identification from given data. *Information and Control*, 37(3): 302–320.
6. Hopcroft, J. and J. Ullman (1979). *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley.
7. Juillé, H. and J. B. Pollack (1998). A stochastic search approach to grammar induction. In V. Honavar and G. Slutski (eds.), *Proceedings of the* 4th *International Colloquium on Grammatical Inference (ICGI '98)*, pp. 126–137. Berlin: Springer.
8. Lang, K.J. (1992). Random DFA's can be approximately learned from sparse uniform examples. In *Proceedings of the* 5th *Annual Workshop on Computational Learning Theory (COLT '92)*, pp. 45-52. ACM Press.
9. Lang, K.J., B.A. Pearlmutter and R.A. Price (1998). Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In V. Honavar and G. Slutski (eds.), *Proceedings of the* 4th *International Colloquium on Grammatical Inference (ICGI '98)*, pp. 1–12. Berlin: Springer.
10. Martín-Vide, C., V. Mitrana and Gh. Păun (eds.) (2004). *Formal Languages and Applications.* Berlin: Springer.
11. Oncina, J. and P. García (1991). Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis*, pp. 49-61. World Scientific Publishing.
12. Oncina, J. and P. García (1992). *A polynomial algorithm to infer regular languages*, pp. 49-61. World Scientific.

13. Pao, T. and J. Carr (1978). A solution of the syntactic induction-inference problem for regular languages. *Computer Languages*, 3: 53-64.
14. Parekh, R. and V.G. Honavar (1997). Learning DFA from simple examples. In M. Li and A. Maruoka (eds.), *Proceedings of the* 8th *International Conference on Algorithmic Learning Theory (ALT '97)*, pp. 116–131. London: Springer.
15. Trakhtenbrot, B. and Y. Barzdin (1973). Finite Automata: Behavior and Synthesis. Amsterdam: North-Holland Publishing.