

A comparative study of two algorithms for automata identification.

P. García, A. Cano and J. Ruiz

Depto. de Sistemas Informáticos y Computación.
Universidad Politécnica de Valencia. Valencia (Spain).

pgarcia@dsic.upv.es

acano@dsic.upv.es

jruiz@dsic.upv.es

Abstract. We describe in this paper the experiments done to compare two algorithms that identify the family of regular languages in the limit, the algorithm of Trakhenbrot and Barzdin/Gold by one hand and the *RPNI*/Lang algorithm by the other. As a previous step, for a better comparison, we formulate the algorithm of Gold as a merging states in the prefix tree acceptor scheme.

1 Introduction

Finite automata identification from samples of their behavior is an old problem in automata theory and is a central topic in the discipline known as Grammatical Inference, connected to the field of Pattern Recognition.

This problem can formally established as a decision problem as follows: Given an integer n and two disjoint sets of words D_+ and D_- over a finite alphabet, do there exist a deterministic finite state automaton (*DFA*) consistent with D_+ and D_- and having a number of states less than or equal to n ?

Gold [2] has proved that this problem is *NP*-complete. Nevertheless, besides the (general) enumeration algorithm, if the sets D_+ and D_- are somehow representative, there are two algorithms that solve the considered problem in deterministic polynomial time.

The first one (1973) is from Trakhenbrot and Barzdin [6], and will be denoted as *TB*. The authors describe it as a contraction procedure in a finite tree that represents a uniformly complete data set (a set that contains all the words up to a certain length). If the given data set comprises a certain characteristic set of an unknown regular language, the algorithm finds the smallest *DFA* that recognizes the language. The characteristic input set required for *TB* algorithm to converge is of polynomial size in relation with the number of states of the canonical acceptor of the target language.

In 1978, Gold [2] rediscovers *TB* algorithm and applies it to the discipline of grammatical inference (uniformly complete samples are not required). He also specifies the way to obtain indistinguishable states using the so called *state characterization matrices*. If the input 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. A complete proof of the equivalence of *TB* and Gold algorithms would enlarge this paper too much.

By other hand, in 1992, Oncina and García [5] propose the *RPNI* (Regular Positive and Negative Inference) algorithm for inference of regular languages from complete presentation. In the same year, Lang [4] describes *TB* algorithm (in a way that, if one doesn't use a complete uniform sample as input it doesn't necessarily coincides with *TB*) and modifies it to guarantee the consistency in the case of a non complete input sample, obtaining this way an algorithm that behaves exactly in the same way as *RPNI* does.

The aim of the present paper is to show the experimental results done in order to compare the behavior of both algorithms (*TB*/Gold by one hand and *RPNI*/Lang by the other). As a previous step, we formulate both algorithms in a homogeneous way. *RPNI* algorithm is based on merging states in the prefix tree acceptor of the sample, while Gold used the -so called- *state characterization matrices*, but as both algorithms are somehow based on Nerode theorem one might expect a similar behavior.

Experiments done show that *RPNI*/Lang algorithm raises convergence more rapidly than *TB*/Gold algorithm does.

2 Definitions and notation.

We suppose that the reader is familiar with the rudiments of formal languages. For further details the reader is referred to [3]. Let Σ be a finite alphabet and let Σ^* be the free monoid generated by Σ with concatenation as the internal law and λ as neutral element. A *language* L over Σ is a subset of Σ^* . The elements of L are called *words*. Given $x \in \Sigma^*$, if $x = uv$ with $u, v \in \Sigma^*$, then u (resp. v) is called *prefix* (resp. *suffix*) of x . $\text{Pr}(L)$ (resp. $\text{Suf}(L)$) denotes the set of prefixes (suffixes) of L . The *right quotient* of a language L by a word u is denoted by $u^{-1}L$, that is, $u^{-1}L = \{v \in \Sigma^* : uv \in L\}$. The right quotient of two languages is $L_1^{-1}L_2 = \{v \in \Sigma^* : \exists u \in L_1, uv \in L_2\}$.

A deterministic finite automaton (*DFA*) is a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$ where Q is the (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 which can be extended to words by doing $\delta(q, \lambda) = q$ and $\delta(q, xa) = \delta(\delta(q, x), a), \forall q \in Q, \forall x \in \Sigma^*, \forall a \in \Sigma$. A word x is accepted by A if $\delta(q_0, x) \in F$. The set of words accepted by A is denoted by $L(A)$.

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 as $t_M(x) = \Phi(\delta(q_0, x))$, for every $x \in \Sigma^*$ such that $\delta(q_0, x)$ is defined.

Given two finite sets of words D_+ and D_- , we define the (D_+, D_-) -*prefix Moore machine* ($PTM(D_+, D_-)$) as the Moore machine having $\Gamma = \{0, 1, \uparrow\}$,

$Q = \text{Pr}(D_+ \cup D_-)$, $q_0 = \lambda$ 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 D_+ , to D_- or to the complementary set of $D_+ \cup D_-$.

A Moore machine $M = (Q, \Sigma, \{0, 1, \uparrow\}, \delta, q_0, \Phi)$ is *consistent* with (D_+, D_-) if $\forall x \in D_+$ we have $\Phi(x) = 1$ and $\forall x \in D_-$ we have $\Phi(x) = 0$.

3 A Description of the algorithms to be compared.

3.1 TB/Gold algorithm.

We are going to see the algorithm proposed by Gold [2], 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 $T : (S \cup S\Sigma)E \rightarrow \{0, 1, \uparrow\}$. The elements of S are called *states*, and those of E are called *experiments*.

The data $(D_+$ and $D_-)$ contained in a state characterization matrix is:

$$\begin{cases} x \in D_+ & \text{if } \exists u \in (S \cup S\Sigma) \exists v \in E, x = uv \wedge T(uv) = 1 \\ x \in D_- & \text{if } \exists u \in (S \cup S\Sigma) \exists v \in E, x = uv \wedge T(uv) = 0 \end{cases}$$

For the rest of the words we set $T(uv) = \uparrow$. Every element u of $(S \cup S\Sigma)$ defines a row which will be called $\text{row}(u)$. Given $u, v \in (S \cup S\Sigma)$, we say that $\text{row}(u)$ is *obviously different* from $\text{row}(v)$, and we write $\text{row}(u) \not\cong \text{row}(v)$, 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 neither row belonging to $S\Sigma - S$ is obviously different from all the rows of S .

Gold algorithm [2] was initially established using Mealey machines. Here we use Moore machines to present it, as they are, to our opinion, the best way to present *RPNI* algorithm also (Moore machines associate the outputs to the states). Doing it this way the comparisons between both algorithms can be seen in a more clearly way.

Remark. The choice that we make here of the set E does not appear as necessary in [2], where it is only established that E must be *suffix-complete* ($\forall x \in E, \text{Suf}(x) \subseteq E$) and that it must be large enough to contain every word belonging to $D_+ \cup D_-$.

The algorithm begins by setting $S = \{\lambda\}$ and E as the right quotient of the set of suffixes of the sample by the alphabet. We have to maintain two sets of words (each word defines a row), the set S and the set $S\Sigma - S$. At every step we have to move from the set $S\Sigma - S$ to the set S one of the rows which is obviously different from the rows of S . We have then to update the set $S\Sigma - S$ and this operation has to be done until neither word can be moved from $S\Sigma - S$ (we say then that the matrix is *closed*). The output of the algorithm is an automaton obtained from the state characterization matrix. If this automaton is not consistent with the data, the algorithm outputs the Moore machine $PTM(D_+, D_-)$.

Algorithm 1 Gold's Algorithm

Gold(D_+, D_-)
 $S = \{\lambda\}$; $E = \text{Suf}(\Sigma^{-1}(D_+ \cup D_-))$;
 Build the table (S, E, T) ;
While there exists $s' \in (S\Sigma - S) : \text{row}(s') \not\subseteq \text{row}(s), \forall s \in S$
 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 $\delta(s, a) = \text{any } s' \in S : \text{Not}(\text{row}(sa) \not\subseteq \text{row}(s'))$ (**)
 End For all
End For all;
 $M = (Q, \Sigma, \{0, 1, \uparrow\}, \delta, q_0, \Phi)$;
If M is consistent with (D_+, D_-) **Then** Return(M)
Else Return($PTM(D_+, D_-)$)
End

There are exactly two places where the algorithm may be non deterministic. The first one¹ 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 assigned². The solution we adopted for both situations is to choose the smallest row in lexicographic order.

Example 1. Let $D_+ = \{abb, bb, bba, bbb, babb\}$ and $D_- = \{\lambda, a, ba, aba, bab\}$. The initial state characterization matrix is:

E	abb bb b λ ba a ab
S	λ 1 1 \uparrow 0 0 0 \uparrow
$S\Sigma - S$	a \uparrow 1 \uparrow 0 0 \uparrow \uparrow
	b 1 1 1 \uparrow 1 0 0

¹ See the part marked with (*) in the algorithm Gold(D_+, D_-).

² See the part marked with (**) in the algorithm Gold(D_+, D_-).

Applying the algorithm until the matrix is closed we obtain:

E		abb	bb	b	λ	ba	a	ab
λ		1	1	\uparrow	0	0	0	\uparrow
S	b	1	1	1	\uparrow	1	0	0
	bb	\uparrow	\uparrow	1	1	\uparrow	1	\uparrow
$S\Sigma - S$	a	\uparrow	1	\uparrow	0	0	\uparrow	\uparrow
	ba	\uparrow	1	0	0	\uparrow	\uparrow	\uparrow
	bba	\uparrow	\uparrow	\uparrow	1	\uparrow	\uparrow	\uparrow
	bbb	\uparrow	\uparrow	\uparrow	1	\uparrow	\uparrow	\uparrow

The Moore machine resulting from running the algorithm with the above example is depicted in figure 1 (a). The corresponding automaton used for pattern recognition is shown in figure 1(b). In this automaton the states in which the output is undefined are considered as non final.

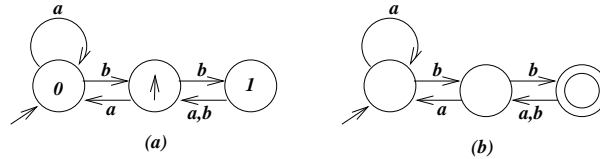


Fig. 1. (a) Moore machine output by Gold algorithm on input positive and negative samples of example 1. (b) Automaton inferred under the same conditions for classification tasks.

3.2 The algorithm *RPNI/Lang*.

The algorithm *RPNI/Lang* [6] has been described and widely used in pattern recognition tasks, see for example [4] and [5]. It is also somehow based on Nerode theorem. The algorithm merges states on the prefix Moore machine of the positive and negative samples and outputs an automaton which is consistent with those samples. After trying to merge two states it has to group together the necessary ones to make a deterministic automaton. If any of the groupings gives an automaton which is not consistent with the data, it has to forget the current merging and try another one. An upper bound of the running time of this algorithm is mn^2 , where m is the size of the initial prefix tree and n is the size of the final automaton ([4]). A detailed and revised description of the algorithm can be found in [1].

4 A merging states version of *TB/Gold* algorithm.

Differences between Gold and *RPNI* algorithms start at the very beginning, so, in order to compare them it can be useful to re-write one of the algorithms in

a similar way as the other is written. We will present *TB/Gold* algorithm as a merging states in the $PTM(D_+, D_-)$ procedure.

Given a Moore machine $M = (Q, \Sigma, \{0, 1, \uparrow\}, \delta, q_0, \Phi)$, we say that $p, q \in Q$ are *distinguishable* in M if there exists a word $x \in \Sigma^*$ with $\Phi(\delta(p, x)), \Phi(\delta(q, x)) \in \{0, 1\}$ and $\Phi(\delta(p, x)) \neq \Phi(\delta(q, x))$. On the opposite way we say that p and q are non distinguishable in M . We next describe the procedure *distinguishable* and the *TB/Gold* algorithm.

In the procedure *distinguishable* we make use of the function *compatible* which assigns to a couple of states the value *False* if one represents a positive sample and the other a negative one, and the value *True* in the rest of the cases. We also make use of the functions defined for lists, *First* which returns the first element of a list and *Rest* which returns the list without the first element.

Algorithm 2 function *distinguishable*: given $p, q \in Q$, returns *True* if $\exists x \in \Sigma^*$: $\Phi(\delta(p, x)), \Phi(\delta(q, x)) \in \{0, 1\}$ and $\Phi(\delta(p, x)) \neq \Phi(\delta(q, x))$

```

distinguishable( $u, v, PTM(D_+, D_-)$ )
list =  $\{u, v\}$ ;
While list  $\neq \emptyset$  do
  ( $p, q$ ) = First(list); list = Rest(list);
  If (Not(compatible( $p, q$ ))) Then Return(True)
  //  $p, q \in Q$  are not compatible if  $\Phi(p), \Phi(q) \in \{0, 1\} \wedge \Phi(p) \neq \Phi(q)$  .//
  Else For every  $a \in \Sigma$ 
    If  $\exists \delta(p, a) \wedge \exists \delta(q, a)$  Then
      Append ( $\delta(p, a), \delta(q, a)$ ) to list
    EndIf
  EndIf
End While
Return(False)
End.

```

Example 2. Let $D_+ = \{bbbb\}$ and $D_- = \{b\}$ as in the above example. The function *distinguishable*($\{\lambda\}, \{b\}, PTM(D_+, D_-)$) returns the result *False*.

If one wishes that this algorithm returns complete automata as the original Gold algorithm does, we only have to send the lacking transitions to the initial state.

With this new approach to Gold algorithm, some differences related to the behavior of both *-RPNI* and Gold- algorithms can be seen. The main one is that at every step Gold only uses the prefix tree acceptor and the states to be merged, so he does not make use of some information that *RPNI* uses, as this last algorithm considers all the merging done at every step as correct.

By other hand, Gold algorithm returns a hypothesis of size less than or equal to the size of the target automaton, as the number of rows of S is bounded by the number of states of the target automaton, which determines the number of equivalence classes in the Nerode equivalence.

Algorithm 3 TB/Gold algorithm as a merging states in the $PTM(D_+, D_-)$ procedure

TB/Gold (D_+, D_-)
 $M_0 = PTM(D_+, D_-)$; // $M_0 = (Q_0, \Sigma, \{0, 1, \uparrow\}, \delta, q_0, \Phi_0)$ //
 $S = \{\lambda\}$;
While there exists $s' \in S\Sigma - S$ such that $\forall s \in S, distinguishable(s, s', M_0)$
 choose s' ;
 $S = S \cup \{s'\}$;
End While;
 $Q = S; q_0 = \lambda$;
For $s \in S$
 $\Phi(s) = \Phi_0(s)$;
 For $a \in \Sigma$
 If $sa \in S$ **Then** $\delta(s, a) = sa$
 Else $\delta(s, a) = \text{any } s'$ such that **Not**($distinguishable(sa, s', M_0)$);
 End For
End For;
 $M = (Q, \Sigma, \{0, 1, \uparrow\}, \delta, q_0, \Phi)$;
If M is consistent with (D_+, D_-) **Return**(M)
Else **Return**(M_0);
End.

Less important is the fact that Gold algorithm always returns complete automata, while this is not necessarily true in case of $RPNI$ algorithm.

4.1 An example of the merging states scheme of Gold algorithm.

Figure 2 represents the (D_+, D_-) -Prefix Moore machine with $D_+ = \{aa, aaa, aaab, aaba, aabba, ab, abab, abb, abba, abbb\}$ and $D_- = \{\lambda, a, aab, b, bb\}$. The states to be merged at every step, and the evolution of S and $S\Sigma$ can be seen in the following table.

S	$S\Sigma$	list
$\{\lambda\}$	$\{a, b\}$	
$\{\lambda, a\}$	$\{b, aa, ab\}$	$\{(b, \lambda)\}$
$\{\lambda, a, aa\}$	$\{b, ab, aaa, aab\}$	
$\{\lambda, a, aa, ab\}$	$\{b, aaa, aab, aba, abb\}$	$\{(b, \lambda), (aaa, ab), (aab, a), (aba, a), (abb, ab)\}$

According to the algorithm, the states of the resulting automaton are $\{\lambda, a, aa, ab\}$. The transition function of the automaton is $\delta(\lambda, a) = a, \delta(a, b) = \lambda, \delta(a, a) = aa, \delta(a, b) = ab, \delta(aa, b) = a, \delta(ab, b) = ab$ and $\delta(ab, a) = a$. The set of final states is $F = \{aa, ab\}$. The automaton is depicted in figure 3.

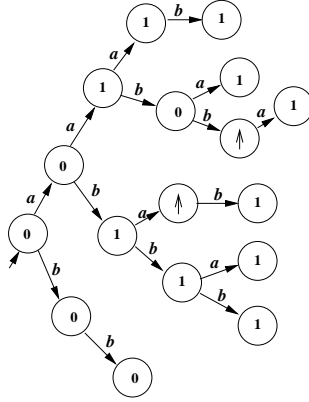


Fig. 2. (D_+, D_-) -Prefix Moore machine with $D_+ = \{aa, aaa, aaab, aaba, aabba, ab, abab, abb, abba, abbb\}$ and $D_- = \{\lambda, a, aab, b, bb\}$.

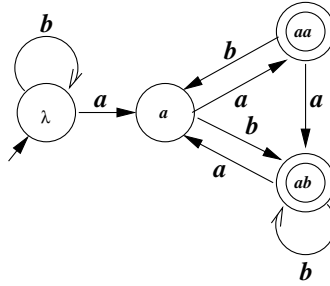


Fig. 3. Output automaton of the execution of the merging states scheme of Gold algorithm for the given sample.

5 Experimental results.

We have compared the new version of *TB*/Gold algorithm (merging in the prefix tree acceptor) and *RPNI*/Lang algorithm.

Description of the experiments:

- We work with minimal automata having 8, 16, 32, 64 and 128 states, the alphabet is $\Sigma = \{a, b\}$. We obtain them beginning with larger automata, we then minimize them and discard the automata which do not have the required size. This method is inspired in [4], Although Lang permits some flexibility in the number of states.
- For the learning process we use randomly generated strings of length less than or equal to 21 over Σ^* . The number of them is shown in the figures that describe the results of the experiments.

- The comparison of the automata is done using all the words of length less than or equal to 15 not used in the learning process. When Gold algorithm obtains an automaton which is not consistent with the sample we use it instead of the prefix tree acceptor of the sample. Other way, as we use for testing different words as for learning, the error rate for that automaton would be 100%. We recall that the objective of the experiment is to measure how close the hypothesis and the target automata are.
- We have done 1000 experiments for each different size of the automata.

The following figures show the mean (and one of them the typical deviation) of the error rate (percentage of words not correctly classified) and of the *representation coefficient* (size of the obtained hypothesis divided by the size of the target automaton).

5.1 Error rate.

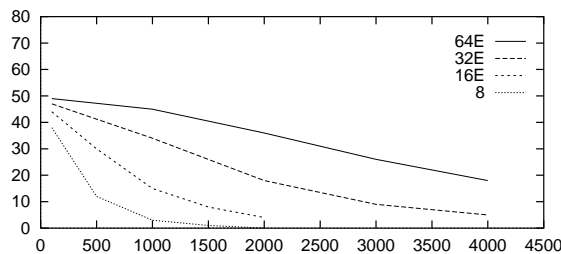


Fig. 4. Error rate obtained running Gold algorithm with target automata of size from 8 to 64 states as the number of sample used for inference varies.

Figures 4 and 5 show respectively the variation of the error rate (percentage of words not correctly classified) obtained by Gold and *RPNI* algorithms for different sizes of target automata when the number of samples used for inference also varies. As one may hope, for a given inference sample size, if the size of the target automaton increases so does the error rate. One may observe that the error rates of *RPNI* algorithm are better than those of Gold algorithm.

In figure 6 we compare the behavior of both algorithms for the case of automata of size 16 (for the rest of the cases, the situation remains equivalent so we do not depict them).

Figures 7 and 8 show the variation of the mean and of the typical deviation of the error rates of the algorithms subject of study for automata of size 16 as the number of samples used for inference vary. We can observe that the typical deviation obtained with *RPNI* algorithm decreases to zero more rapidly than that obtained with Gold algorithm.

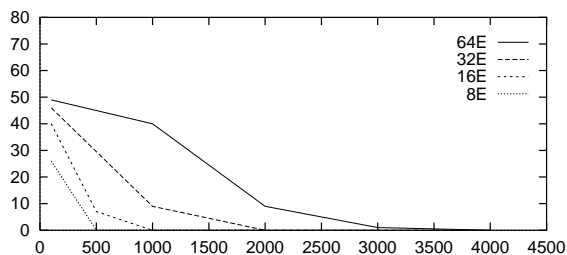


Fig. 5. Error rate obtained running *RPNI* algorithm with target automata of size from 8 to 64 states as the number of sample used for inference varies.

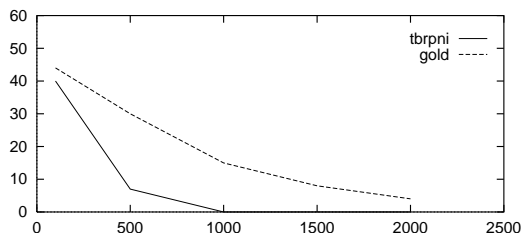


Fig. 6. A comparison of the error rate average obtained by Gold and *RPNI* algorithms when the number of samples used automata of size 16 varies from 100 to 2000.

5.2 Representation coefficient.

Figures 9 and 10 show the representation coefficient (size of the hypothesis divided by the size of the target automaton) of the automata obtained when running Gold and *RPNI* algorithms respectively.

We can see that both algorithms converge to identify the number of states of the automata correctly, although Gold algorithm obtains increasing values as the number of words used in the inference becomes larger (we recall that the size of the hypothesis obtained using Gold algorithm is less than or equal to that of the target automaton) while *RPNI* obtains decreasing values.

6 Conclusions.

We have revised Trakhtenbrot and Barzdin (*TB*), Gold, *RPNI* and Lang algorithms. As it can be seen in [1], the first two ones are in fact the same, while the first description that Lang does of *TB* algorithm agrees with it only in case of a uniformly complete sample. The extension he gives to obtain consistent hypothesis is in fact *RPNI* algorithm.

We have also formulated *TB*/Gold algorithm as a procedure of merging states in the prefix Moore machine of the sample for a better understanding of the comparison with *RPNI*. By other hand we have observed that the implementation

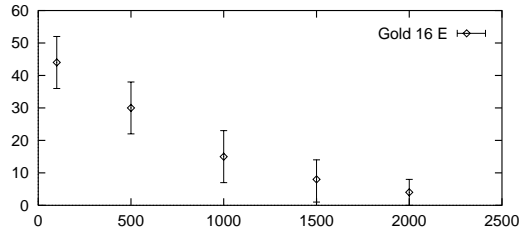


Fig. 7. Error Diagram (mean and typical deviation) for Gold algorithm for automata of 16 states as the number of samples used for inference vary from 100 to 2000.

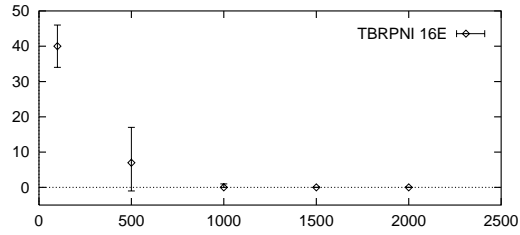


Fig. 8. Error Diagram (mean and typical deviation) for *RPN* algorithm for automata of 16 states as the number of samples used for inference vary from 100 to 2000.

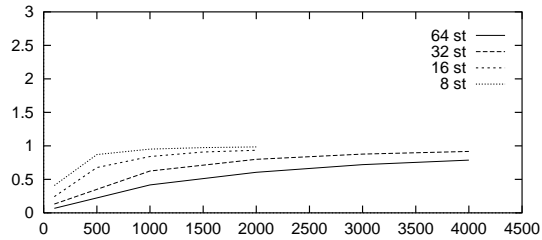


Fig. 9. Average of the representation coefficient of the automata inferred using Gold algorithm.

of this new version works much faster than the original one. That fact can be deduced from theoretical estimations [1].

We have observed, both in theoretical analysis as in experiments done, that generally *RPN* algorithm obtains better results than Gold algorithm.

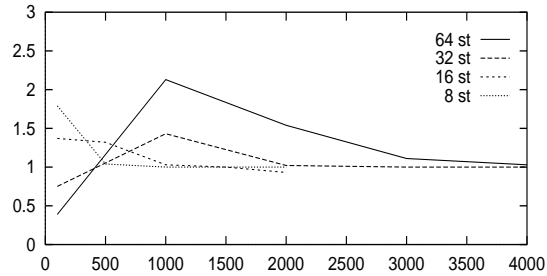


Fig. 10. Average of the representation coefficient of the automata inferred using *RPNI* algorithm.

References

1. García, P. Cano, A. and Ruiz, J. *Estudio comparativo de dos algoritmos de identificación de autómatas*. Internal Report DSIC-II/1/00 Univ. Politécnica Valencia. (2000) (In Spanish).
2. Gold, M. *Complexity of Automaton Identification from Given Data*. Information and Control 37, pp 302-320 (1978).
3. Hopcroft, J. and Ullman, J. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley (1979).
4. Lang, K.J. *Random DFA's can be Approximately Learned from Sparse Uniform Examples*. In Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory, pp 45-52. (1992).
5. Oncina, J. and García, P. *Inferring Regular Languages in Polynomial Updated Time*. In Pattern Recognition and Image Analysis. Pérez de la Blanca, Sanfeliú and Vidal (Eds.) World Scientific. (1992).
6. Trakhtenbrot B. and Barzdin Ya. *Finite Automata: Behavior and Synthesis*. North Holland Publishing Company. (1973).