How considering incompatible state mergings may reduce the DFA induction search tree

François Coste and Jacques Nicolas

IRISA- INRIA
Campus de Beaulieu, F-35042 Cedex, France

Abstract. A simple and effective method for DFA induction from positive and negative samples is the state merging method. The corresponding search space may be tree-structured, considering two subspaces for a given pair of states: the subspace where states are merged and the subspace where states remain different. Choosing different pairs leads to different sizes of space, due to state mergings dependencies. Thus, ordering the successive choices of these pairs is an important issue. Starting from a constraint characterization of incompatible state mergings, we show that this characterization allows to achieve better choices, i.e. to reduce the size of the search tree. Within this framework, we address the issue of learning the set of all minimal compatible DFA's. We propose a pruning criterion and experiment with several ordering criteria. The prefix order and a new entropy based criterion exhibit the best results in our test sets.

keywords: grammatical inference, DFA, constraint system, search tree.

Grammatical inference is an important machine learning problem with applications in language processing, pattern recognition and genetics. It is defined as the process of learning a grammatical representation of a language from examples of words of this language and words that do not belong to the language. In this paper, we address the issue of learning canonical automata of regular languages. More precisely, we are concerned with the following DFA induction problem. Given a set of positive strings I_+ and a set of negative strings I_-, find the DFA's A satisfying:

1. I_+ is *structurally complete* wrt A (i. e. there exists an acceptance of I_+ such that every transition of A is exercised and every final state of A is used as an accepting state);
2. A is *compatible* with I_+ and I_- (i. e. $L(A)$, the language accepted by A, contains all the strings of I_+ and does not contain any string of I_-);
3. $L(A)$ is a most general language
   (i.e. no other solution A' is such that $L(A) \subseteq L(A')$).

Most of the proposed algorithms focus on the production of a single solution. They proceed either with a depth first strategy (Oncina & Garcia 1992, Lang 1992), an evidence driven state merging strategy (Lang 1997, Lang & al 1998, Juillé & Pollack 1998), a beam search strategy (Miclet & de Gentille 1994), or with an optimization method (Dupont 1994).

But these algorithms do not ensure the minimal solution. Indeed, the problem of finding the minimal compatible DFA has been proven NP-complete in the worst case (Angluin 1978,Kearns & Valiant 1989). Using an interactive or incremental process, it seems interesting to continue the search if the proposed DFA is not satisfactory. Algorithms seeking more than one solution are then needed. Furthermore, from a scientific point of view, complete algorithms producing all the solutions are also useful to better understand and characterize the search space and the solution set, before deriving more specialized algorithms.

Few algorithms seeking a set of solutions have been proposed. Miclet has proposed in Miclet & de Gentille 1994 a heuristic algorithm. An algorithm using graph coloring methods can be found in Coste & Nicolas 1997. Preliminary studies have also been attempted in the context free case (Giordano 1993, Vanlehn & Ball 1987).

Looking for all solutions is a difficult problem due to the size of the search space and, in some cases, due to the solution set size itself. We present in section 2 a constraint system allowing to manage the set of all compatible DFA's in a compact and implicit representation.

Using a state merging algorithm, we show in section 3 how the knowledge of constraints reflecting incompatible mergings may help to reduce the size of the search space. It allows to define two criteria: a pruning criterion detecting dead ends in the search space and an entropy based ordering criterion.

To begin with, the next section is devoted to basic definitions, notations and theorems on regular grammatical inference.

1 Search space in regular grammatical inference

1.1 Definitions and notations

We first recall basic definitions of the field of grammatical inference.

Definition 1. (DFA) A finite state automaton is a quintuplet $(Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, Σ an alphabet, δ a transition function from $Q \times \Sigma$ to 2^Q (extended to $Q \times \Sigma^* \to 2^Q$), q_0 is the initial state and $F \subseteq Q$ is the set of accepting or final states.
If $\forall q \in Q, \forall a \in \Sigma$, $\delta(q,a)$ has at most one element, the automaton is said deterministic and we denote DFA such an automaton.
A DFA A accepts a regular language $L(A)$.

Definition 2. ($A(L)$, MCA, PTA) The canonical automaton of a language L, $A(L)$ is the DFA accepting L which has the minimal number of states.
The maximal canonical automaton with respect to a set of words I, $MCA(I)$, is the automaton A with the largest number of states such that $L(A) = I$ and I is structurally complete with respect to A.
The prefix tree acceptor of I, $PTA(I)$, is obtained from $MCA(I)$ by merging states sharing the same prefixes.

The natural generality relation between languages is inclusion, corresponding to inclusion between sets of accepted words. While working on representations of languages, one has to find a compatible relation with this inclusion relation. Such a relation is known for automata (Miclet 1990,Pao & Carr 1978) : the set of finite automata may be partially ordered with a derivation relation, corresponding to the merging of states in the automata.

Definition 3. (Derived automaton A/π) Given an automaton $A = (Q, \Sigma, \delta, q_0, F)$ and a partition $\pi = (B_0, B_1, \dots, B_r)$ of Q, the derived or quotient automaton $A/\pi = (\pi, \Sigma, \Delta, B_0, R)$ is defined as follows:

- $q_0 \in B_0$;
- $R = \{B_i \in \pi, \exists q \in B_i \; st \; q \in F\}$;
- $B_j \in \Delta(B_i, a)$ iff $\exists q \in B_i, \exists q' \in B_j$ such that $q' \in \delta(q, a)$.

The set of all automata derived from A is a lattice $Lat(A)$.

1.2 Search Space : a lattice of automata

We can now characterize more precisely the search space of DFA's for an inference problem (Dupont& al. 1994).

Theorem 4. *Let I_+ be a positive sample. Let L be the target language. If I_+ is structurally complete with respect to $A(L)$, then $A(L)$ is an element of $Lat(PTA(I_+))$.*

Another way to look at this theorem is to state that every language, such that I_+ is structurally complete with respect to its canonical automaton, is learn-able in the lattice, and as a consequence, our search space will be this lattice in the rest of this paper:

Corollary 5. *Let I_+ be a positive sample.*
A regular language L is said to be admissible if I_+ is structurally complete with respect to $A(L)$, the canonical automaton of L. The set of canonical automata of all admissible languages is included in $Lat(PTA(I_+))$.

As the search space of automata is finite and may be partially ordered, we can use a version space approach (Mitchell 1982) where all solutions of a learning problem are characterized using two sets: the set S of maximally specific solutions and the set G of maximally general solutions. An incremental approach using membership queries has been proposed within this framework (Parekh & Honavar 1993). In our case, S may be built straightforwardly, using $PTA(I_+)$. G is the set of all the solutions to the automata induction problem as stated in the introduction. Building G (also called *Border Set*, Dupont& al. 1994) is much more difficult, due to the size of the search space and, in some case, to the size of G itself. When the language of generalization consists of vectors of attribute-value pairs, several authors have proposed to manage an implicit representation of the version space (Hirsh 1992,Nicolas 1993,Sebag 1994).

We propose in the next section a compact representation of compatible DFA's set, based on the set of incompatible state mergings.

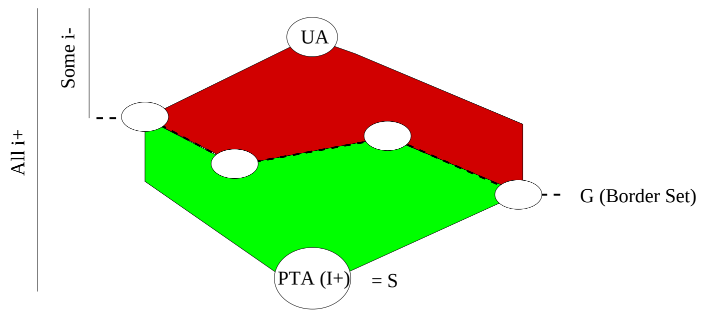

Fig. 1. Search Space: the set of compatible DFA's lies between $PTA(I_+)$ and G

2 Constraint characterization

In this section we propose to characterize the set of compatible DFA's, such that the positive sample is structurally complete wrt the DFA. The structural completeness condition ensures that solutions are in $Lat(PTA(I_+))$. The solutions are then the deterministic automata in the lattice accepting no string of the negative sample.

In order to further explain our approach, we need to introduce two auxiliary concepts, closely related to the search space to be explored. In section 2.1 we introduce the merging procedure for determinization allowing to stay in the space of deterministic automata. In section 2.2, we introduce the *augmented PTA* (APTA) following in this the data structure proposed in Higuera & al. 1996 and Alquézar & Sanfeliu 1995, in order to handle simultaneously the positive sample I_+ and the negative sample I_-. The set of compatible DFA's can then be characterized by a system of constraints presented in section 2.3.

2.1 Merging for determinization

Since the merging procedure may produce a non deterministic automaton, an extended version of the merging procedure (the merging procedure for determinization) is used by algorithms such that RPNI to consider only deterministic automata.

Such a procedure corresponds to the existence of a binary relation between pairs of states. Formally, if we denote $\rightarrow$ such a relation, we have

$$(q_1, q_2) \rightarrow (q'_1, q'_2) \; iff \; \exists a \in \Sigma \; \delta(q_1, a) = q'_1 \wedge \delta(q_2, a) = q'_2 \qquad (1)$$

The merging procedure for determinization can be stated as follow: *each time q_1 and q_2 such that $(q_1, q_2) \rightarrow (q'_1, q'_2)$ are merged, merge also q'_1 and q'_2.*

The result of this procedure is a (eventually smaller) DFA in the lattice.

The transitive closure of $\rightarrow$ is denoted $\rightarrow^*$.

2.2 Augmented PTA

Intuitively, the APTA may be considered as the superposition of $PTA(I_+)$ and $PTA(I_-)$, each state being labeled into three classes : accepting state, rejecting state or intermediate state, accepting no positive and no negative instance.

Definition 6. An augmented PTA (APTA) with respect to a positive sample I_+ and a negative sample I_-, denoted $APTA(I_+, I_-)$, is a 6-tuple $(Q, \Sigma, \delta, q_0, F^+, F^-)$ where

- $PTA(I_+ \cup I_-) = (Q, \Sigma, \delta, q_0, F^+ \cup F^-)$
- F^+ and F^- are subsets of Q respectively identifying accepting states of I_+ and I_-
- F^+ is the set of final states of $APTA(I_+, I_-)$

We give as an illustration, the APTA of sample 1 of language 2 in our benchmark (cf section 4). In this sample, $I_+ = \{ababab\}$, $I_- = \{a, b, abaa, abbbb, ababbbba\}$.

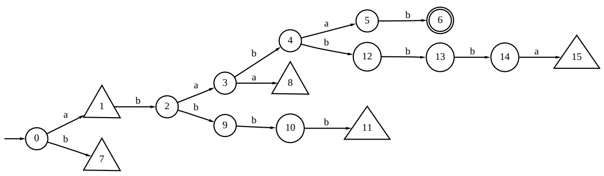

Fig. 2. APTA of sample 1 of language 2

States of F^+ are double circles and states of F^- are triangles.

By construction, for a partition π of states of the $PTA(I_+)$, we get the following property :

$$L(APTA(I_+, I_-)/\pi) = L(PTA(I_+)/\pi) \tag{2}$$

Considering the APTA, instead of the PTA, has the advantage of explicitly propagating the consequences of merging some states in the PTA on the acceptance of negative instances.

2.3 Constraint system

To ensure determinism, the merging procedure will use the merging procedure for determinization presented in section 2.1.

The set of compatible DFA's can then be characterized by a constraint system on incompatible state mergings in the APTA. Intuitively, an incompatible state merging involves, after a chain reaction, the merging of a state accepting a positive string and a state accepting a negative string.

Such considerations can be formalized in the following constraint system:

If we denote q_π the block in the partition of the state q for a partition π, the set of compatible DFA's will be the set of automata derived from $PTA(I_+)$ such that:

$$q_\pi^+ \neq q_\pi^- \quad \forall q^+ \in F^+, \forall q^- \in F^- \tag{3}$$

$$q'_{1\pi} \neq q'_{2\pi} \Rightarrow q_{1\pi} \neq q_{2\pi} \ \forall q'_1, q'_2 \in APTA(I_+, I_-)/\ (q_1, q_2) \rightarrow (q'_1, q'_2) \tag{4}$$

Note that constraints of type 4 may be reversed in form of equality constraints $q_{1\pi} = q_{2\pi} \Rightarrow q'_{1\pi} = q'_{2\pi}$. Moreover, the system of constraints may be simplified, applying modus ponens and discarding redundancy (e.g. $\{1 \neq 2$ and $1 \neq 2 \Rightarrow 3 \neq 4\}$ is reduced to $\{1 \neq 2,\ 3 \neq 4\}$). The subset of constraints of type 3 (after simplifications) is called F_{Nok}. It represents incompatible pairs of states mergings. It may be represented as a graph on the set of states. In the next section, we propose to maintain the validity of this subset and to use it to reduce the search tree in state merging algorithms.

3 Reducing the search tree in state merging algorithms

3.1 An extended state merging algorithm

State merging algorithm explores the PTA lattice by merging states of the PTA until the border set G is reached. The search space is then a set of partitions and there is one operation, namely merging two blocks in a partition, to move from one node to another one, more general in the search space.

Algorithm 1 Extended State Merging Algorithm with incompatibilities

```
ESMA(A)
if (s1, s2) ← ChooseStatesToMerge(A) then
  if (A' ← A[s1 = s2]) /* merges states and propagates consequences */ then
    ESMA(A')
  end if
  /* Extension allowed by F_Nok management */
  if (A' ← A[s1 ≠ s2]) /* adds the constraint and propagates consequences */
  then
    ESMA(A')
  end if
else if A is solution then
  Output A
end if
end

main()
init_constraint_system()
ESMA(PTA(I+))
end
```

As we explicitly and dynamically manage the set of incompatible mergings F_{Nok}, we can consider an alternative representation of this search space such that each pair of states in PTA is an attribute with two possible values : $=$ if states are merged in the derived automaton, $\neq$ if we decide not to merge states in the derived automaton. We give the sketch of the corresponding algorithm (Algorithm 1). At each step, a pair of states is chosen and assigned successively the two possible values ($=$ and $\neq$), the constraint system being updated accordingly. An example of search tree for this algorithm is given (figure 3.2).

Adressing now the issue of producing all minimal (in the sense of the number of states) solution DFA's, we propose two ways of reducing the search tree: a new heuristic ordering of pairs of states, based on entropy measure(section 3.2) and pruning with an incompatibility clique (section 3.3).

3.2 Ordering pairs of states, entropy based selection

Heuristics of every known algorithm working on the search of a single solution may be used for the selection of attributes in this tree and thus these algorithms may be naturally extended to the search of all solutions.

In Trakhenbrot & Barzdin 1973, the order in which the states are merged is not important since the presentation is complete. In algorithms such as RPNI (Oncina & Garcia 1992) or the algorithm of K. Lang (Lang 1992), the states to be merged are chosen in a breadth-first predetermined order. Recent results in the Abbadingo One competition show that better results can be obtained by considering candidate mergings in order of the amount of evidence supporting them, as claimed in Higuera & al. 1996. The most interesting order seems to be merging states according to the implied number of states merged together with the same label (Lang & al 1998).

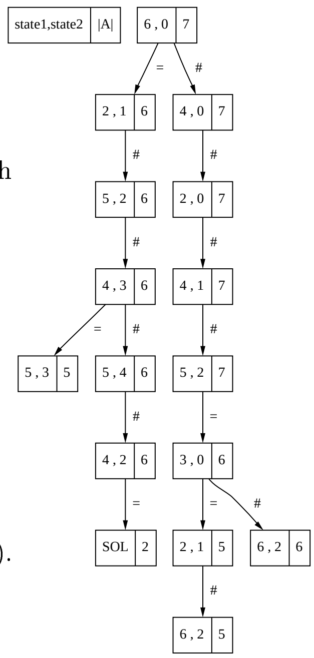

Fig. 3. Search tree of sample 1 of language 2, with a random order selection

The question is then to find the most efficient heuristic with respect to this new "all solutions" goal, and this is not a trivial task since a given algorithm may well find very quickly a first solution, taking advantage of a property of this solution, and then be laborious in finding the other ones, that do not respect this property.

We have developed an entropy based criterion (section 3.2), using the same idea of dynamically computing the evidence of a merging. However, it is no longer based on labels counts but uses instead a more abstract view of merging constraints (positive *and* negative) existing between states.

The search tree as a decision tree Another way of looking at the search space is to consider the search tree as a kind of decision tree, its leaves being labeled either success or failure, according as the corresponding automaton is or not an acceptable solution. Given this framework, reducing the amount of search to be done is equivalent to finding the smallest tree where all leaves contain either a single solution automaton or lead to a failure. Following the methodology TDIDT (Quinlan 1986), we have tried various criteria in order to select the best attribute at each node. However, one has to keep in mind that this is an analogy and not the usual framework, since we do not know how many solutions or failures are below a given node. All we can do is to estimate this number. Defining this estimate is the purpose of the next part of this section.

Estimating the number of solutions In order to estimate the number of solutions, we split the set of states into two parts. The first part, *the pseudo-clique* (PC), corresponds to a set of candidates such that every other state (in the second part denoted $\overline{PC}$) is supposed to be merged with to obtain the target automaton. The name pseudo-clique refers to graph F_{Nok}, where states PC are likely to be in relation and to form a clique (or a densest 'almost' clique).

Once a pseudo-clique is chosen, we estimate the number of solutions by considering possible relations between states in PC and states in $\overline{PC}$.

Let m be the size of PC (i. e. the size of the target automaton), l be the size of $\overline{PC}$ and $|A|$ be the number of states of the current automaton. States of the pseudo-clique PC are denoted p_i and states of the $\overline{PC}$ set are denoted q_i. Let d_i be the number of states of PC not related to state q_i in $\overline{PC}$.

The total number of configurations (possible completions of graph F_{Nok} by adding new edges) tot is: $tot = \prod_{1\leq i\leq l} 2^{d_i}$

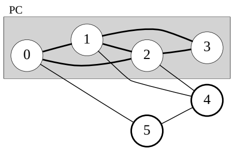

Fig. 4. If the pseudo-clique found for this graph is the set of states in the grey box, then $tot = 32$, $p = 6$, $n = 11$, $u = 15$

Then, we can roughly estimate the number of solutions p as the number of possible mappings of states from $\overline{PC}$ to PC. The number of failure configurations n is the number of configurations where a state of $\overline{PC}$ can not be merged with any state of PC.

$$p = \prod_{1\leq i\leq l} d_i \ ; \qquad n = tot - \prod_{1\leq i\leq l} (2^{d_i} - 1)$$

Therefore, in this node, the number of undefined configurations is u:

$$u = tot - (p+n) = \prod_{1\leq i\leq l} (2^{d_i} - 1) - p$$

Since the value of p may be neglected compared to u and n, the estimated entropy of a given node finally writes:

$$I = -(1-r)log_2(1-r) - rLog_2(r) \quad where \quad r = \prod_{1\leq i\leq l} \frac{(2^{d_i} - 1)}{2^{d_i}}$$

Finally, by averaging entropies of the *left* and *right* edges of a given node in the search tree, we propose to select the attribute corresponding to the pair of states minimizing E:

$$E = tot_{left} I_{left} + tot_{right} I_{right}$$

3.3 Clique in the graph of incompatible mergings

When an upper bound of the number of states of the DFA is available (either user provided or estimated and refined or by a solution found before), the search may be pruned if there exists a clique of states in the graph of F_{Nok} whose size is greater than the maximal expected size of the DFA. In this case the states of the clique cannot be merged together and the size of the resulting DFA is bigger or equal to the size of the clique.

The search of the largest clique is NP-complete. However, if the size of the clique is not too large and an approximation is sufficient, efficients algorithm are known. Our experimentation (section 4) shows good results with a greedy search. Moreover, we have observed that starting from the root of the $PTA(I_+)$ provides a good approximation of the size of the best clique.

4 Experimentation

4.1 Settings

We have used a benchmark described in Dupont 1996 for a first validation of ideas developed in this paper. Target automata are small (< 6 states) but structurally complex. For instance, L_{11} is the language accepting an even number of a and an odd number of b. For each language, 20 training samples have been drawn. The size of these sets of words is less than or equal to 50. The size of the words themselves varies from 1 to 40.

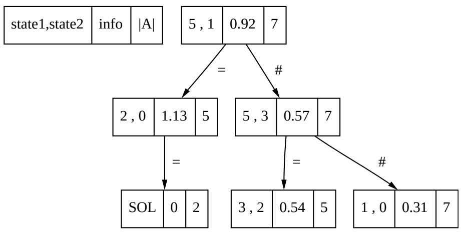

Fig. 5. Search tree of the entropy criterion on sample 1 of language 2.

We have used five orders to build a complete search tree of the solutions: Random order (20 trials have been made for each training set); Prefix order; RPNI order (same as Prefix order except that only pairs of states that can be merged are considered); Entropy-based order and Rlb order (Lang 1997). In each case, pseudo-cliques have been computed and used to detect failure nodes. Indeed, if a clique of size greater than m is detected in F_{Nok}, it means that the corresponding automaton has more than m states, and the search may be pruned. Using RPNI order, we have also tried experiments without clique pruning to show the influence of it. In this case, the maximal number of nodes, fixed to 150000, has often been reached. The average number of nodes have been calculated using this maximal value when the number of nodes was greater.

4.2 Results

Lang.	#s/#t	#sol	APTA	#FN	#n !p	#n rpni	#n rand	#n pref	#n I	#n Rlb
L1	1/1	1	3	0	9	2	3	2	2	2
L2	2/2	1	25	43	>15760	5	10	3	3	3
L3	4/7	1.75	180	998	>133500	17	4326	14	15	53
L4	3/5	8.5	102	192	>115900	84	3298	59	71	269
L5	4/8	2.1	65	218	>150000	51	30837	32	38	73
L6	3/6	2.45	40	108	>106300	28	9843	19	19	73
L7	4/7	4.75	85	545	>150000	217	27291	93	134	406
L8	2/2	1.25	19	43	393	3	4	3	3	3
L9	4/7	1.65	61	358	>150000	18	115	6	7	7
L10	5/6	2.4	81	432	>68200	22	98	16	10	17
L11	4/8	2.85	65	215	>150000	36	37035	27	34	74
L12	3/3	1.5	38	120	>23100	5	9	4	4	4
L13	2/4	1.25	32	71	>90521	6	476	6	6	6
L14	3/4	1.4	38	101	>41200	5	30	6	6	6
L15	4/6	2.1	71	353	>60900	9	36	9	9	9

#s/#t characterizes the size of target automata in terms of number of states and transitions, #sol is the mean number of minimal automata found in BS for the 20 samples. #FN is the mean size of F_{Nok} at the root node. #n !p #n rpni, #n rand, #n pref, #n Rlb, #n I are the mean number of nodes of the tree for respectively a rpni without pruning, a rpni, a random, a prefix, a Rlb, and an entropy selection of pairs of states.

First, the influence of pruning is so important that it would have been much more difficult to compare the orders without it. Results show also how important is the choice of a selection criterion for the reduction of the search space. A random selection of nodes may increase the size of the tree by several orders of magnitude, with respect to best criteria. Results are in favor of the prefix order. The refined Rlb order is not only not necessary but gives worse results in almost every case. The more expensive criterion, entropy, seems comparable or slightly worse than the prefix order.

In order to better differentiate the behavior of the best criteria, we ran a second series of experiments comparing prefix order and entropy order as the amount of samples was increasing. Ten DFA's of size 5 have been drawn from a uniform distribution (Same generation of DFA's and sample strings than for Abbadingo One Competition (Lang & al 1998)). Figure 6 shows the average number of nodes in the search tree for the ten DFA's for sample files ranging from 10 strings to 390 strings. Results are clearly in favor of the entropy order : the number of nodes is an order of magnitude smaller and less data are needed to converge towards the best ordering of mergings.

We have to temperate these observations with an important remarks : our benchmarks contain target automata of small size. Scaling effects may appear with larger DFA.

Fig. 6. Average number of nodes in the search tree

5 Conclusion

We have presented a system of constraints characterizing the set of compatible DFA's. Using the corresponding knowledge on incompatible state mergings (F_{Nok}) allows to reduce the search tree in state merging algorithms. It is a key point since usually, automata inferring algorithms do consider possible mergings but not impossible ones.

The benchmark needs of course to be completed with more difficult problems, increasing the size of the vocabulary and the size of the target automaton. The first important result of this study is that a greedy search of clique in F_{Nok} allows to prune efficiently the search tree and leads to very small search trees in our experiment. Ordering the choice of the pairs of states to be merged is also an important point when considering the "all solutions" search problem. The simple prefix order seems to be a good candidate for this search. We have studied a new entropy based criterion that behaves well, especially in our second experiment, but is expensive to compute. We expect it to be interesting in large search spaces.

References

R. Alquézar and A. Sanfeliu, "Incremental grammatical inference from positive and negative data using unbiased finite state automata," In *Shape, Structure and Pattern Recognition*, Proc. Int. Workshop on Structural and Syntactic Pattern Recognition, SSPR'94, Nahariya (Israel), World Scientific Pub., Singapore, 291–300, 1995.

Angluin, D. On the complexity of minimum inference of regular sets. *Information and Control* 29(3):741 – 765, 1978.

Coste, F. and Nicolas, J. : Regular Inference as a Graph Coloring Problem. Workshop on Grammar Inference, Automata Induction, and Language Acquisition (ICML' 97), Nashville, TN, 1997.

Dupont, P. Regular Grammatical Inference from Positive and Negative Samples by Genetic Search : the GIG method. *ICGI'94, Grammatical inference and Applications* 236–245. Springer Verlag, 1994.

Dupont, P.; Miclet, L.; and E.Vidal. What is the search space of the regular inference ? *ICGI'94, Grammatical inference and Applications* 25–37. Springer Verlag, 1994.

Dupont, P. *Utilisation et apprentissage de modèles de langages pour la reconnaissance de la parole continue.* Ph.D. Dissertation, Ecole Nationale Supérieure des Télécommunications, 1996.

Giordano, J. Espace des versions et inférence de grammaires algébriques. In *Journées Francophones d'Apprentissage*, 1993.

Higuera (de la), C.; Oncina, J.; and Vidal, E. : Identification of DFA : data-dependent versus data-independent algorithms. Grammatical inference Learning Syntax from Sentences, ICGI'96 311–325. Springer Verlag, 1996.

Hirsh, H. Polynomial-time Learning with Version Spaces *National Conference on Artificial Intelligence*, San Jose, CA, 117–122 july 1992.

Juillé, H., and Pollack, J.B. SAGE: a Sampling-based Heuristic for Tree Search. Submitted paper.

Kearns, M., and Valiant, L. Cryptographic limitation on learning boolean formulae and finite automata. In *Proceedings of the Twenty First Annual Symposium on Theory of Computing*, 433 – 444, 1989.

Lang, K. Random DFA's can be Approximately Learned from Sparse Uniform Examples. In *proceedings of the fifth annual ACM Workshop on Computational Learning Theory* 45–52, July 1992.

Lang, K. Merge Order count NECI Tech Report, Sept26, 1997

Lang, K., and Pearlmutter, B., and Price, R. Results of the Abbadingo One DFA Learning Competition and a New Evidence Driven State Merging Algorithm. Submitted paper.

Miclet, L. Grammatical Inference. *Syntactic and Structural Pattern Recognition : Theory and Applications*, Bunke, H. and Sanfeliu, A. Eds, Singapore, World Scientific, 237–290, 1990.

Miclet, L., and de Gentille, C. Inférence grammaticale à partir d'exemples et de contre-exemples : deux algorithmes optimaux : (big et rig) et une version heuristique (brig). *JAVA94, Journées Acquisition, Validation, Apprentissage* F1–F13, Strasbourg France, 1994.

Mitchell, T. Generalization as search. *Artificial Intelligence* (18):203–226, 1982.

Nicolas, J. Une représentation efficace pour les espaces de versions. *8ièmes JFA, Saint-Raphael*, 1993.

Oncina, J., and Garcia, P. Inferring regular languages in polynomial update time. *Pattern Recognition and Image Analysis* 49 – 61, 1993.

Pao, T., and Carr, J. A solution for the syntactic induction inference problem for regular languages in *Computer Language vol3* 53 – 64, 1978.

Parekh, R., and Honavar, V. Efficient Learning of Regular Languages using Teacher Supplied Positive Examples and Learner Generated Queries. In Proceedings of the Fifth UNB Conference on AI , Fredrickton, Canada. 195–203, August 1993.

Quinlan, J.R. Induction of decision trees. *Machine Learning* (1):81–106, 1986.

Sebag, M. Une approche par contraintes de l'espace des versions. *RFIA'94* 17–25,1994.

Trakhenbrot, B. and Barzdin, Y., Finite Automata : Behavior and Synthesis *Amsterdam, North Holland Pub. Comp*, 1973.

Vanlehn, K., and Ball, W. A version space approach to learning context-free grammars. *Machine Learning* (2):39–74, 1987.