Improved SAT models for NFA learning

Frédéric Lardeux[0001−8636−3870] and Eric Monfroy[0001−7970−1368]

LERIA, University of Angers, France firstname.lastname@univ-angers.fr

Abstract. Grammatical inference is concerned with the study of algorithms for learning automata and grammars from words. We focus on learning Nondeterministic Finite Automaton of size k from samples of words. To this end, we formulate the problem as a SAT model. The generated SAT instances being enormous, we propose some model improvements, both in terms of the number of variables, the number of clauses, and clauses size. These improvements significantly reduce the instances, but at the cost of longer generation time. We thus try to balance instance size vs. generation and solving time. We also achieved some experimental comparisons and we analyzed our various model improvements.

Keywords: Constraint problem modeling · SAT · model reformulation.

1 Introduction

Grammatical inference [7] is concerned with the study of algorithms for learning automata and grammars from words. It plays a significant role in numerous applications, such as compiler design, bioinformatics, speech recognition, pattern recognition, machine learning, and others. The problem we address in this paper is learning a finite automaton from samples of words $S = S^+ \cup S^-$, which consist of positive words (S^+) that are in the language and must be accepted by the automaton, and negative words (S^-) that must be rejected by the automaton. A non deterministic automaton (NFA) being generally a smaller description for a language than an equivalent deterministic automaton (DFA), we focus here on NFA inference. An NFA is represented by a 5-tuple $(Q, \Sigma, \Delta, q_1, F)$ where Q is a finite set of states, the vocabulary Σ is a finite set of symbols, the transition function $\Delta : Q \times \Sigma \to \mathcal{P}(Q)$ associates a set of states to a given state and a given symbol, $q_1 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states.

Not to mention DFA (e.g., [6]), the problem for NFA has been tackled from a variety of angles. In [15] a wide panel of techniques for NFA inference is given. Some works focus on the design of ad-hoc algorithms, such as *DeLeTe2* [3] that is based on state merging methods. More recently, a new family of algorithms for regular languages inference was given in [14]. Some approaches are based on metaheuristic, such as in [12] where hill-climbing is applied in the context of regular language, or [4] which is based on genetic algorithm. In contrast to metaheuristics, complete solvers are always able to find a solution if there exists one, to prove the unsatisfiablility of the problem, and to find the optimal solution in case of optimization problems. In this case, generally, the problem is modeled

as a Constraint Satisfaction Problem (CSP [11]). For example, in [15], an Integer Non-Linear Programming (INLP) formulation of the problem is given. Parallel solvers for minimizing the inferred NFA size are presented in [8, 9]. The author of [10] proposes two strategies, based on variable ordering, for solving the CSP formulation of the problem.

In this paper, we are not interesting in designing or improving a solver, but we focus in improving models of the problem in order to obtain faster solving times using a standard SAT solver. Modeling is the process of translating a problem into a CSP consisting in decision variables and constraints linking these variables. The INLP model for NFA inference of [15] cannot be easily modified to reduce the instances: to our knowledge, only Property 1 of our paper could be useful for the INLP model, and we do not see any other possible improvement. We thus start with a rather straightforward conversion of the INLP model into the propositional satisfiablity problem (SAT [5]). This is our base SAT model to evaluate our improvements. The model, together with a training sample, lead to a SAT instance that we solve with a standard SAT solver. The generated SAT instances are very huge: the order of magnitude is $|S|.(|\omega|+1).k^{|\omega|}$ clauses, where k is the number of states of the NFA, ω is the longest word of S, and $|S|$ is the number of words of the training sample. We propose three main improvements to reduce the generated SAT instances. The first one prevents generating subsumed constraints. Based on a multiset representation of words, the second one avoid generating some useless constraints. The last one is a weaker version of the first one, based on prefixes of words. The first improvement returns smaller instances than the second one, which in turn returns smaller instances than the third one. However, the first improvement is very long and costly, whereas the third one is rather fast. We are thus interested in balancing generation and solving times against instance sizes. We achieved some experiments with the Glucose solver [1] to compare the generated SAT instances. The results show that our improvements are worth: larger instances could be solved, and faster. Generating the smallest instances can be too costly, and the best results are obtained with a good balance between instance sizes and generation/solving time.

This paper is organized as follows. In Section 2, we describe the problem and we give the basic SAT model. We also evaluate the size of the generated instances. Section 3 presents 3 model improvements, together with sketches of algorithms to generate them. Section 4 exposes our experimental results and some analysis. We finally conclude in Section 5.

2 Modeling the problem in SAT

The non-linear integer programming (INLP) model of [15, 9] cannot be easily improved or simplified. Indeed, the only improvement proposed in [15] and [9] corresponds to Property 1 (given in the next section). In this section, we thus present a SAT formulation of the NFA inference problem. This SAT model permits many improvements to reduce the size of the generated SAT instances.

The NFA inference problem Consider an alphabet $\Sigma = \{s_1, \ldots, s_n\}$ of n symbols; a training sample $S = S^+ \cup S^-$, where S^+ (respectively S^-) is a set of *positive words* (respectively *negative words*) from Σ^*; and an integer k. The problem consists in building a NFA of size k which validates words of S^+, and rejects words of S^-. The problem can be extended to an optimization problem: it consists in inferring a minimal NFA for S, i.e., an NFA minimizing k. However, we do not consider optimization in this paper.

Notations Let $A = (Q, \Sigma, q, F)$ be a NFA with: $Q = \{q_1, \ldots, q_k\}$ a set of states, Σ a finite alphabet (a set of symbols), q the initial state, and F the set of final states. The symbol λ represents the empty word. We denote by K the set $\{1, \ldots, k\}$. A transition from q_j to q_k with the symbol s_i is denoted by $\tau_{s_i,q_j \to q_k}$. Consider the word $w = w_1 \ldots w_n$ with $w_1, \ldots, w_n$ in Σ. Then, the notion of transition is extended to w by $T_{w,q_{i_1} \to q_{i_{n+1}}}$ which is a sequence of transitions $\tau_{w_1,q_{i_1} \to q_{i_2}}, \ldots, \tau_{w_n,q_{i_n} \to q_{i_{n+1}}}$. The set of candidate transitions for w between the states q_{i_1} and q_{i_l} in a NFA of size k is $\mathcal{T}_{w,q_{i_1} \to q_{i_l}} = \{T_{w,q_{i_1} \to q_{i_l}} \mid \exists i_2, \ldots i_{l-1} \in K,\ T_{w,q_{i_1} \to q_{i_l}} = \tau_{w_1,q_{i_1} \to q_{i_2}}, \ldots, \tau_{w_l,q_{i_l-1} \to q_{i_l}}\}$.

A SAT model Our base model is a conversion into SAT of the nonlinear integer programming problem given in [15] or [9]. Consider the following variables:

- k the size of the NFA we want to build,
- $F = \{f_1, \ldots, f_k\}$ a set of k Boolean variables determining whether states q_1 to q_k are final or not,
- and $\Delta = \{\delta_{s,q_i \to q_j} | s \in \Sigma \text{ and } i, j \in K\}$ a set of $n.k^2$ variables determining whether there is or not a transition $\delta_{s,q_i \to q_j}$, i.e., a transition from state q_i to state q_j with the symbol s, for each q_i, q_j, and s.

A transition $T_{w_1 \ldots w_n,q_{i_1} \to q_{i_{n+1}}} = \tau_{w_1,q_{i_1} \to q_{i_2}}, \ldots \tau_{w_n,q_{i_n} \to q_{i_{n+1}}}$ exists if and only if the conjunction $d = \delta_{w_1,q_{i_1} \to q_{i_2}} \wedge \ldots \wedge \delta_{w_n,q_{i_n} \to q_{i_{n+1}}}$ is true. We call d a c_transition, and we say that d models $T_{w_1 \ldots w_n,q_{i_1} \to q_{i_{n+1}}}$. We denote by D_{w,q_i,q_j} the set of all c_transitions for the word w between states q_i and q_j.

The problem can be modeled with 3 sets of equations:

1. If the empty word λ is in S^+ or in S^-, we can determine whether the first state is final or not:

$$\text{if } \lambda \in S^+, \qquad f_1 \tag{1}$$
$$\text{if } \lambda \in S^-, \qquad \neg f_1 \tag{2}$$

2. For each word $w \in S^+$, there is at least a transition starting in q_1 and ending in a final state q_j:

$$\bigvee_{j \in K} \bigvee_{d \in D_{w,q_1,q_j}} (d \wedge f_j) \tag{3}$$

With the Tseitin transformations [13], we create one auxiliary variable for each combination of a word w, a state $j \in K$, and a transition $d \in D_{w,q_1,q_j}$:

$$aux_{w,j,d} \leftrightarrow d \wedge f_j$$

For each w, we obtain a formula in CNF:

$$\bigwedge_{j \in K} \bigwedge_{d \in D_{w,q_1,q_j}} [(\neg aux_{w,j,d} \vee (d \wedge f_j))] \tag{4}$$

$$\bigwedge_{j \in K} \bigwedge_{d \in D_{w,q_1,q_j}} (aux_{w,j,d} \vee \neg d \vee \neg f_j) \tag{5}$$

$$\bigvee_{j \in K} \bigvee_{d \in D_{w,q_1,q_j}} aux_{w,j,d} \tag{6}$$

d is a conjunction, and thus $\neg aux_{w,j,d} \vee d$ is a conjunction of $|w|$ binary clauses: $(\neg aux_{w,j,d} \vee \delta_{w_1,q_1 \to q_{i_2}}) \wedge \ldots \wedge (\neg aux_{w,j,d} \vee \delta_{w_{|w|},q_{i_{|w|}} \to q_{i_{|w|+1}}})$.
$|D_{w,q_1,q_j}| = k^{|w|-1}$ since for each symbol of w there is k possible moves in the NFA, except for the last symbol which leads to q_j. Thus, we have $(|w|+1).k^{|w|}$ binary clauses for Constraints (4), $k^{|w|}$ $(|w|+2)$-ary clauses for Constraints (5), and one $k^{|w|}$-ary clause for Constraints (6). We have added $k^{|w|}$ auxiliary variables.

3. For each $w \in S^-$ and each state q_j, either there is no complete transition from state q_1 to q_j, or q_j is not final:

$$\neg \left[\bigvee_{j \in K} \bigvee_{d \in D_{w,q_1,q_j}} (d \wedge f_j) \right] \tag{7}$$

Constraints (7) are already in CNF, and we have $k^{|w|}$ $(|w+1|)$-ary clauses.

Thus, the constraint model M_k for building a NFA of size k is:

$$M_k = \bigwedge_{w \in S^+} \Big((4) \wedge (5) \wedge (6) \Big) \wedge \bigwedge_{w \in S^-} (7)$$

and is possibly completed by (1) or (2) if $\lambda \in S^+$ or $\lambda \in S^-$.

Size of the models Considering ω_+, the longest word of S^+, and ω_-, the longest word of S^-, the number of constraints in model M_k is bounded by:

- $|S^+|.(|\omega_+|+1).k^{|\omega_+|}$ binary clauses;
- $|S^+|.k^{|\omega_+|}$ $(|\omega_+|+2)$-ary clauses;
- $|S^+|$ $k^{|\omega_+|}$-ary clauses;
- $|S^-|.k^{|\omega_-|}$ $(|\omega_-|+1)$-ary clauses.

The number of Boolean variables is bounded by:

- k variables in F determining final states;
- $n.k^2$ variables determining existence of transitions;
- $|S^+|.k.^{|\omega_+|}$ auxiliary variables $aux_{w,j,d}$.

It is thus obvious that it is important to improve the model M_k.

3 Improving the SAT model

We now give some properties that can be used for improving the SAT model. By abuse of language, we will say that a model M_1 is smaller than a model M_2 whereas we should say that the SAT instance generated with M_1 and data D is smaller than the instance generated with M_2 and D. A first and simple improvement is based on the following property.

Property 1 (Empty word λ). If $\lambda \in S^-$, then each c_transition ending in q_1 does not have to be considered when generating the constraints related to the word $w \in S$.

Indeed, if w is positive, it cannot be accepted by a transition ending in q_1; similarly, if w is negative, $\neg d \vee \neg f_1$ is always true. When $\lambda \in S^+$, the gain is not very interesting: f_1 can be omitted in Constraints (7), (4), and (5). This does not really reduce the instance, and a standard solver would simplify it immediately.

Whereas a transition is an ordered sequence, the order of conjuncts in a c_transition is not relevant, and equal conjuncts can be deleted. Thus, a c_transition may model several transitions, and may correspond to several words. By abuse of language, we say that a c_transition ends in a state q_j if it corresponds to at least a transition ending in q_j. Thus, a c_transition may end in several states. We consider an order on c_transitions. Let d and d'' be two c_transitions. Then, $d \preceq d''$ if and only if there exists a c_transition d' such that $d \wedge d' = d''$. In other words, each transition variable $\delta_{s,q_i \to q_j}$ appearing in d also appears in d''. This order is used in the two first model improvements which are based on c_transitions. The third model improvement is based on transitions. We now consider some redundant constraints.

Property 2 (Redundant constraints). When a state q_i cannot be reached, each outgoing transition becomes free (it can be assigned true or false), and q_i can be final or not. In order to help the solver, all the corresponding variables can be assigned an arbitrary value. For each state q_j, $j \neq 1$:

$$\Big(\bigwedge_{i \in K, i \neq j} \bigwedge_{s \in \Sigma} \neg \delta_{s,q_i \to q_j}\Big) \to \neg f_j \wedge \Big(\bigwedge_{i \in K} \bigwedge_{s \in \Sigma} \neg \delta_{s,q_j \to q_i}\Big)$$

In CNF, these constraints generate (for all q_j), $(k-1).(k.n+1)$ redundant clauses of size $n.(k-1)+1$.

These constraints are useful when looking for a NFA of size k when k is not the minimal size of the NFA. Compared to SAT instance size, these redundant constraints can be very helpful without being too heavy.

Note that in our implementation, for all the models, we always simplify instances using Property 1 and removing duplicate transition variables in c_transitions (i.e., $\delta_{s,q_i \to q_j} \wedge \ldots \wedge \delta_{s,q_i \to q_j}$ is simplified into $\delta_{s,q_i \to q_j} \wedge \ldots$). Moreover, we also generate the redundant constraints as defined in Property 2.

Improvement based on c_transitions subsumption. This first improvement consists in removing tautologies for negative words, and some constraints and unsatisfiable disjuncts for positive words.

Property 3 (c_transition subsumption). Let v be a negative word from S^-, and $\neg d_v \vee \neg q_j$ be a Constraint (7) generated for the c_transition d_v for v ending in state q_j. We denote this constraint c_{v,d_v,q_j}. Consider a positive word w from S^+, and d_w a c_transition for w ending in q_j such that $d_v \preceq d_w$. Then, each $d_w \wedge f_j$ will be false due to c_{v,d_v,q_j}. Thus, Constraints (4) and (5) corresponding to w, d_w, and q_j will force to satisfy $\neg aux_{w,j,d_w}$; hence, they can be omitted and aux_{w,j,d_w} can be removed from Constraints (7). Similarly, consider ω from S^-, and d_ω a c_transition for w ending in q_j such that $d_v \preceq d_w$. Then, Constraint (7), $\neg d_v \vee \neg q_j$, will always be true (due to the constraint c_{v,d_v,q_j}), and can be omitted.

We can compute the size of the reduced SAT instance when the smaller word is a prefix. Let $v \in S^-$ and $w \in S$ be words such that $w = v.v'$, i.e., $v \subseteq w$ and v is a prefix of w. Then, using Property 3: if $w \in S^-$, the number of clauses generated for w is reduced to $(k-1).k^{|w|-1}$ clauses of size $|w+1|$; if $w \in S^+$, the number of clauses generated for w is reduced to $(|w|+1).(k-1).k^{|w|-1}$ binary clauses for Constraints (4), $(k-1).k^{|w|-1}$ $(|w|+2)$-ary clauses for Constraints (5), and one clause of size $(k-1).k^{|w|-1}$ for Constraint (6). The number of auxiliary variables is reduced to $(k-1).k^{|w|-1}$.

Operationally, we have a two step mechanism. First, for each negative word, each c_transition together with its ending state is generated and stored in a database of couples (c_transition, ending state) that we call c_couple. Then, for generating constraints for a word w, each of its c_couple is compared to the database. If a c_transition for w ending in q_j is smaller than a c_transition from the database also ending in q_j, then the corresponding constraints are not generated, as shown above. We call $M_{k,all}$ this reduced model.

Improvement based on Multisets. Although efficient in terms of generated instance sizes, the previous improvement is very costly in memory and time. It becomes rapidly intractable. This second improvement also uses Property 3. It is a weakening of the above operational mechanism that does not omit every subsumed c_transition. This mechanism is less costly. Hence, generated instances will be a bit larger, but the balance generation time against instance size is very good. The idea is to order words in order to search in a very smaller database of c_couples (c_transition, ending state) when generating constraints for a word w. Moreover, this order will also imply the order for generating constraints.

We associate each word to a multiset which support is the vocabulary Σ. The word w, is thus associated with the multiset $ms(w) = \{s_1^{|w|_{s_1}}, \ldots, s_n^{|w|_{s_n}}\}$ where $|w|_{s_i}$ is the number of occurrences of the symbol s_i in w. Note that several words can have the same multiset representation. Based on multiset inclusion $(\{s_1^{a'_1}, \ldots, s_n^{a'_n}\} \subseteq_{\mathcal{M}} \{s_1^{a_1}, \ldots, s_n^{a_n}\} \Leftrightarrow \forall i, a'_i \leq a_i)$, we can now define the notion of word inclusion, noted $\subseteq_\omega$. Consider w and w', two words of Σ^*, then:

$$w' \subseteq_\omega w \Leftrightarrow ms(w') \subseteq_{\mathcal{M}} ms(w)$$

Consider a sample $S = S^+ \cup S^-$. Let $\top(S)$ be the multiset defined as

$$\top(S) = \{s_1^{1+max_{w\in(S)}\{|w|_{s_1}\}}, \ldots, s_n^{1+max_{w\in(S)}\{|w|_{s_n}\}}\}$$

and $\bot = \{s_1^0, \ldots, s_n^0\}$. Then, $\top(S)$ represents words which are not in the sample S, and $\bot$ represents the empty word λ which may be in S.

Consider the sample $S = S^+ \cup S^-$. Let $MS(S) = \{ms(w)|w \in S^+ \cup S^-\}$ be the set of the representations of words of S. Then, $(MS(S) \cup \{\bot, \top(S)\}, \subseteq_{\mathcal{M}})$ is a lattice. Let m be a multiset of $MS(S)$. Then, $inf(m)$ is the set of multisets $\{m' \in MS(S) \mid m' \subseteq_{\mathcal{M}} m\}$. This lattice of multisets defines the data structure used for constraint generation. For generating constraint of a word w of a multiset m, we now only compare its c_couples with the database of c_couples of words $w' \in S^-$ with $w' \subseteq_\omega w$, i.e., words represented by multisets smaller than m.

The negative words that allow to reduce the most, are the ones represented by the smallest multiset. We thus also propose a mechanism to reduce the database (c_transition, ending state) with the most useful c_couples, i.e., the ones from smallest words. Let $level(m)$ be the "level" of the multiset defined by: $level(m) = 0$ if $m = \bot$, $1 + max_{m'\in inf(m)}(level(m'))$ otherwise. Given a multiset m, and a threshold l, the *base* function returns all the multisets m' of level smaller than l, and such that $m' \subseteq_{\mathcal{M}} m$: $base(p, l) = \{n \in inf(p) \mid level(n) \leq l\} \bigcup \left(\bigcup_{p'\in inf(p)} base(p', l)\right)$ if $p \neq \bot$, $\emptyset$ otherwise.

Based on Property 3, c_couples of the negative words of these multisets will be used to reduce constraint generation of the words of m. We call this model $M_{k,mset,l}$, with l a given threshold. If *base* is called with the threshold 0, the database will be empty and the complete instance will be generated: $M_{k,mset,0} = M_k$. If *base* is called with the maximum level of the lattice, then, the database will be the largest one built with all the smaller words, and we will thus obtain the smallest instances with this notion of lattice. However, the larger the threshold, the longer the generation time, and the smaller the SAT instance. With the maximal threshold, the generated instances will be a bit larger than with the previous improvement ($M_{k,all} \subseteq M_{k,mset,max}$), but the generation is significantly faster. For lack of space, we cannot give here the complete algorithms for generating this improved model.

Improvements based on Prefixes. Although faster to generate, the second model is still costly. We now propose a kind of weakening of Property 3, restricting its use to prefix.

Property 4 (Prefix). Let $w \in S$ be a word from the sample. Consider D^*_{w,q_i,q_j} the set of c_transitions defined by:

$$D^*_{w,q_i,q_j} = \bigvee_{l\in K, l\neq j} \left(\left(\bigvee_{d_u \in D^*_{u,q_i,q_l}} d_u \wedge \left(\bigvee_{d_v \in D^*_{v,q_l,q_i}} d_v \right) \right) \right)$$

if $w = u.v$, and $u \in S^-$; otherwise, $D^*_{w,q_i,q_j} = D_{w,q_i,q_j}$. Then,

$$\forall d \in D_{w,q_i,q_j} \setminus D^*_{w,q_i,q_j}, \neg d \vee \neg f_j$$

Hence, this property allows us to directly generate the reduced constraints, for negative or positive words, without comparing c_couples with a database.

Let $w = u_1 \dots u_n$ be a word from S such that $u_1 \in S^-$, $u_1.u_2 \in S^-$, and $u_1 \dots u_{n-1} \in S^-$ and for each $i < n$, there does not exist a decomposition $u_i = u'_i.u''_i$ such that $u_1 \dots u_{i-1}.u'_i \in S^-$. Then, if $w \in S^+$, using several times Property 4, Constraints (4), (5), and (6) can be replaced by Constraints (8), (9), and (10) where $l_0 = q_1$ and $N = [1, \dots, n]$:

$$\bigwedge_{i \in N, l_i \in K \setminus \{l_j | 1 \leq j < i\}} \bigwedge_{i \in N, d_i \in D_{u_i, q_{l_i - 1}, q_l}} [(\neg aux_{w,l_1,\dots,l_n} \vee (d_1 \wedge \dots \wedge d_n \wedge f_j))] \quad (8)$$

$$\bigwedge_{i \in N, l_i \in K \setminus \{l_j | 1 \leq j < i\}} \bigwedge_{i \in N, d_i \in D_{u_i, q_{l_i - 1}, q_l}} (aux_{w,l_1,\dots,l_n} \vee \neg d_1 \vee \dots \vee \neg d_n \vee \neg f_j) \quad (9)$$

$$\bigvee_{i \in N, l_i \in K \setminus \{l_j | 1 \leq j < i\}} \bigvee_{i \in N, d_i \in D_{u_i, q_{l_i - 1}, q_l}} aux_{w,l_1,\dots,l_n} \quad (10)$$

Similarly, if $w \in S^-$, using several times Property 4, Constraints (7) can be replaced by Constraints (11):

$$\bigwedge_{i \in N, l_i \in K \setminus \{l_j | 1 \leq j < i\}} \bigwedge_{i \in N, d_i \in D_{u_i, q_{l_i - 1}, q_l}} (\neg d_1 \vee \dots \vee \neg d_n \vee \neg f_j) \quad (11)$$

The number of clauses and variables generated for $w \in S^+$ is reduced to:

- $(|w| + 1).\big(\prod_{i=1}^{n}(k - i + 1)\big).k^{|w|-n}$ binary clauses for Constraints (8),
- $\big(\prod_{i=1}^{n}(k - i + 1)\big).k^{|w|-n}$ $(|w| + 2)$-ary clauses for Constraints (9),
- one clause of size $\big(\prod_{i=1}^{n}(k - i + 1)\big)$ for Constraint (10),
- and the number of auxiliary variables is reduced to $\big(\prod_{i=1}^{n}(k - i + 1)\big)$.

For $w \in S^-$, Constraints (11) are already in CNF and they correspond to $\big(\prod_{i=1}^{n}(k - i + 1)\big).k^{|w|-n}$ $(|w + 1|)$-ary clauses. Interestingly, these new counts of clauses (and more especially the factor $k - i + 1$ with $i = n$) also give us a lower bound for k: k must be greater than or equal to n, the number of nested prefixes in a word. This new improved model, that we call $M_{k,pref}$, is not much larger than $M_{k,mset}$, but it is significantly faster to generate.

Improvement order. We have defined various models for inference of NFA of size k that can be ordered by their sizes: $M_{k,all} \subseteq M_{k,mset,l\_max} \subseteq m_{k,pref} \subseteq M_k$. Note that $M_{k,mset,l}$ with $l \neq l\_max$, and $M_{k,pref}$ cannot be compared in the general case; their sizes depend on the instance, the number and size of prefixes, and on the given level l. In the next section, we compare these models not only in terms of instance size, but also in terms of generation and resolution time.

4 Experimental results

We suspect that, with respect to their generation time, the models are in reverse order of the order given above. Thus, we are interested in findng the best balance between three parameters: model size v.s. generation time + SAT solving time.

The experiments were carried out on a computing cluster with Intel-E5-2695 CPUs and 128 GB of memory. Running times were limited to 2 hours for the generation of SAT instances, and 3 hours to solve them. We used the Glucose [1] SAT solver with the default options.The benchmarks are based on the training set of the StaMinA Competition (http://stamina.chefbe.net). We selected 12 instances[1] with a sparsity $s \in \{12.5\%, 25\%, 50\%, 100\%\}$ and an alphabet size $|\Sigma| \in \{2, 5, 10\}$. For each of them, we limited the number of words to $|S^+| = |S^-| = 10$ and 20 for a maximal size of words equal to 7 and to $|S^+| = |S^-| = 20$ for a maximal size of words equal to 10. We generate CNF instances for different NFA sizes ($k \in \{3, 4, 5\}$). Consequently, we obtained 96 instances.

Table 1 presents a synthetic view of our experiments. The 4 first columns detail the instances: size of the NFA (k), size of the longest word ($|\omega|$), number of positive (and negative) words ($|S^+|$), and the model. The next columns provide average values over the 12 instances for the modeling time (T_{Model}), the number of variables ($\#Var$), the number of clauses ($\#Cl$), the solving time (T_{solve}), and the total modeling+solving time (T_{total}). We do not indicate the standard deviations but they are very close to zero. "-" indicates that no result was obtained before the time-out. From Table 1, we can draw some general conclusions about model improvements. As expected, $M_{k,all}$ always returns the smallest instances, and also the instances that Glucose solve the fastest. However, the generation time of these instances is very long. Thus, the total CPU time, i.e., generation + solving, is not the best. We can also see that when we increase the maximum length of words, this model does not permit to generate the instances in less than 2 hours (e.g., Table 1, for $k = 4$, $\omega = 10$, and $|S^+| = 20$). This model is thus tractable, but only for small instances, with short words and small samples.

$M_{k,mset,l_{max}}$ generates instances a bit larger than $M_{k,all}$. Consider the negative word $v = aaab$, and the positive word $w = ba$. $M_{k,all}$ uses some c_transitions of v to ignore some clauses of w that $M_{k,mset,l_{max}}$ will not detect. For example, a loop on aaa from v with the same transition in v is used in $M_{k,all}$ but not in $M_{k,mset,l_{max}}$. However, with the multiset data structure, we obtain a much faster generation of instances. The total time is thus more interesting with $M_{k,mset,l_{max}}$ than with $M_{k,all}$. The generation time of $M_{k,mset,l_{max}}$ is still very high, and its interest is not always significant. For large instances, not presented in the table, $M_{k,mset,l_{max}}$ could not be generated in less than 2 hours.

For $M_{k,pref}$, we can see that the generation time becomes reasonable, and much smaller than with the two previous improvements. Although smaller than with M_k, the instances are larger than with $M_{k,mset,l_{max}}$. In various experiments, this improvement was the best for the total time. Note also that our training samples are not so big, and that the number of prefixes is not so important. With larger $|S^+|$, for a fixed k, we should obtain better performances of $M_{k,pref}$.

We also tried two more improvements of $M_{k,mset,l}$ with $l \in \{1, 3\}$. The generation time of these models is logically faster than the ones of $M_{k,mset,l_{max}}$; as planned, the SAT instances are also larger. However, we were pleasantly surprised by the total time which is much better than for $M_{k,mset,l_{max}}$. The three

[1] We conserved the "official" name used during the Stamina Competition.

Table 1. Comparison on 96 generated instances between the models $m_{k,all}$, $m_{k,mset,l_{max}}$, $m_{k,mset,1}$, $m_{k,mset,3}$, and $m_{k,pref}$. Instances are grouped by size of the NFA (k), size of the longest word ($|\omega|$), and number of positive (and negative) words ($|S^+|$). For each line, obtained values are average on 12 instances.

k	$\|\omega\|$	$\|S^+\|$	Model	T_{model}	#Var.	#Cl.	T_{solve}	T_{total}
3	7	10	m_k	0.19	6742	61366	0.22	0.41
			$m_{k,all}$	0.68	4310	37789	0.14	0.82
			$m_{k,mset,l_{max}}$	0.17	4742	42020	0.14	0.31
			$m_{k,mset,1}$	0.18	5517	49484	0.16	0.34
			$m_{k,mset,3}$	0.17	4822	42850	0.14	0.31
			$m_{k,pref}$	0.18	6466	58645	0.2	0.38
		20	m_k	0.48	14830	134302	1.58	2.06
			$m_{k,all}$	2.62	8274	72569	1.64	4.26
			$m_{k,mset,l_{max}}$	0.42	8929	79030	1.22	1.64
			$m_{k,mset,1}$	0.45	11179	99811	1.39	1.84
			$m_{k,mset,3}$	0.46	9148	81188	1.27	1.73
			$m_{k,pref}$	0.43	13689	123390	1.71	2.14
	10	20	m_k	11	303519	3276974	397.68	408.68
			$m_{k,all}$	746.08	108417	1172093	79.98	826.06
			$m_{k,mset,l_{max}}$	9.87	122423	1313463	143.32	153.19
			$m_{k,mset,1}$	9.04	208610	2255307	233.97	243.01
			$m_{k,mset,3}$	9.06	134720	1443357	156.24	165.3
			$m_{k,pref}$	8.88	281408	3040802	270.04	278.92
4	7	10	m_k	1.46	45014	428775	10.3	11.76
			$m_{k,all}$	19.42	32956	302835	5.59	25.01
			$m_{k,mset,l_{max}}$	1.64	35362	328938	5.58	7.22
			$m_{k,mset,1}$	1.42	39242	369600	7.12	8.54
			$m_{k,mset,3}$	1.56	36048	336637	5.58	7.14
			$m_{k,pref}$	1.3	43655	414141	10.69	11.99
		20	m_k	3.93	100984	950473	83.55	87.48
			$m_{k,all}$	93.48	64428	588293	74.55	168.03
			$m_{k,mset,l_{max}}$	4.33	68041	628400	43.08	47.41
			$m_{k,mset,1}$	3.65	83463	777005	32.32	35.97
			$m_{k,mset,3}$	4.27	70720	653396	41.36	45.63
			$m_{k,pref}$	3.37	94829	887943	55.88	59.25
	10	20	m_k	187.59	4670833	53350566	2084.78	2272.37
			$m_{k,all}$	-	-	-	-	-
			$m_{k,mset,l_{max}}$	919.56	2304788	26010946	651	1570.56
			$m_{k,mset,1}$	173.82	3336332	38121787	658.7	832.52
			$m_{k,mset,3}$	375.34	2345238	26693196	107.13	482.47
			$m_{k,pref}$	162.45	4405201	50260648	1331.92	1494.37
5	7	10	m_k	6.61	201651	1962754	215.06	221.67
			$m_{k,all}$	232.47	161828	1526044	51.82	284.29
			$m_{k,mset,l_{max}}$	14.38	169816	1619550	171.92	186.3
			$m_{k,mset,1}$	7.24	182445	1759734	180.98	188.22
			$m_{k,mset,3}$	10.76	172660	1653301	210.1	220.86
			$m_{k,pref}$	6.26	196894	1908623	176.12	182.38
		20	m_k	19.37	456976	4382919	1268.18	1287.55
			$m_{k,all}$	1158.5	320689	2995308	631.14	1789.64
			$m_{k,mset,l_{max}}$	44.01	333799	3148787	1115.9	1159.91
			$m_{k,mset,1}$	20.24	398074	3784691	1192.49	1212.73
			$m_{k,mset,3}$	32.82	348339	3288509	1309.17	1341.99
			$m_{k,pref}$	16.54	434008	4141453	1203.36	1219.9

models $M_{k,pref}$, $M_{k,mset,1}$, and $M_{k,mset,3}$ are very difficult to compare. Depending on the instance, on the number and size of prefixes, on multiset inclusion, one can be better than the other. But for all the instances we tried, one of this 3 models was always the best of the 6 models, and they were better than M_k. Table 2 presents a focus on 2 specific instances (25_training and 35_training, both with $|\Sigma| = 5$) with a fixed value for k, $|\omega|$, and $|S^+|$. The columns correspond exactly to those of Table 1. For the first instance, we clearly see the order presented in Section 3 for instance sizes of improved models. We can also see the reverse order

Table 2. Focus on 2 specific instances.

k	$\|\omega\|$	$\|S^+\|$	Model	T_{model}	#Var.	#Cl.	T_{solve}	T_{total}
25_training								
5	7	20	m_k	16.72	378030	3748314	934.92	951.64
			$m_{k,all}$	854.47	271338	2626880	841.22	1695.69
			$m_{k,mset,l_{max}}$	48.71	275331	2678349	1538.06	1586.77
			$m_{k,mset,1}$	14.25	280899	2733709	895.92	910.17
			$m_{k,mset,3}$	23.67	277359	2696089	1147.41	1171.08
			$m_{k,pref}$	11.76	338880	3377124	687.79	699.55
35_training								
4	10	20	m_k	163.10	5253332	59504339	-	-
			$m_{k,all}$	-	-	-	-	-
			$m_{k,mset,l_{max}}$	676.22	4234500	47661301	2322.42	2998.64
			$m_{k,mset,1}$	209.86	4969772	56092438	-	-
			$m_{k,pref}$	184.56	5253332	59504339	7145.62	7330.18

in terms of generation time. When $|\Sigma|$ is small, the probability of having prefixes is higher than with larger vocabularies, and for this instance, $M_{k,pref}$ returns the best instance in terms of generation+solving time. For the second instance, $M_{k,all}$ could not be generated in less than 2 hours. M_k and $M_{k,mset,3}$ could be generated rather quickly, but could not be solved. $M_{k,pref}$ was even faster for generating the SAT instance. However, we see that there was not prefix in the training set (the size of instances of M_k and $M_{k,pref}$ are the same). The overhead for taking prefixes into account is rather insignificant (12% of generation time). Since the solving time was close to the timeout, the M_k instance did not succeed to be solved while the $M_{k,pref}$ instance succeeded (the small difference of 55 s., i.e., less than 0,8 %, is certainly due to clause order in the SAT instance). This instance shows that $M_{k,mset,l_{max}}$ can be the best model in terms of total time. This is due to the fact that there is no negative word being prefix of another word from S, and that the lattice is rather "wide", with a long branch. Hence, $M_{k,mset,l}$ is interesting when l is large for this training sample.

5 Conclusion

In the context of grammatical inference, we proposeed various model improvements for learning Nondeterministic Finite Automaton of size k from samples of words. Our base model, M_k, is a conversion from an INLP model [15]. The first improvement, $M_{k,all}$, leads to the smallest SAT instances, which are also solved quickly. However, generating this model is too costly. Thus, when problems grow (in terms of k, $|S|$, or length of words), $M_{k,all}$ cannot be generated anymore. We proposed a set of improvements based on multiset representation of words, $M_{k,mset,l}$. The generated SAT instances are a bit larger with the maximal level than with $M_{k,all}$, but generation is still costly. We thus defined a third improvement based on prefix. On average, the best balance between generation and solving time is obtained with $M_{k,pref}$, $M_{k,mset,1}$, or $M_{k,mset,3}$: the generation is rather light and the reductions are significant. The interest of our work is that, to our knowledge, we are the only ones working on CSP model improvements. It is very complicated to compare our results with previous works. Many works on this topics are only formal and experimental results are also difficult

to compare. For examples, the authors of [8, 9] focus on a parallel solver for optimizing k. In [10], experiments are based on samples issued from the Waltz-DB database [2] of amino acid sequences, i.e., all the words are of size 6, and there cannot be any prefix word: in the tests we performed, only anagrams could be used in multisets. Moreover, for all the 50 instances we tried issued from this database, the M_k model could be generated and solved in a reasonable time, without need of any model improvement.

In the future, we plan to hybridize $M_{k,mset,l}$ for small values of l with $M_{k,pref}$. The second idea is to simplify the work of the SAT solver and of the instance generation with simplified and incomplete training samples. We would then evaluate our SAT models with respect to the accurateness of the generated NFA on test set of words.

References

1. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: Proc. of IJCAI 2009. pp. 399–404 (2009)
2. Beerten, J., van Durme, J.J.J., Gallardo, R., Capriotti, E., Serpell, L.C., Rousseau, F., Schymkowitz, J.: WALTZ-DB: a benchmark database of amyloidogenic hexapeptides. Bioinform. **31**(10), 1698–1700 (2015)
3. Denis, F., Lemay, A., Terlutte, A.: Learning regular languages using rfsas. Theor. Comput. Sci. **313**(2), 267–294 (2004)
4. Dupont, P.: Regular grammatical inference from positive and negative samples by genetic search: the GIG method. In: Proc. of ICGI 94. LNCS, vol. 862, pp. 236–245. Springer (1994)
5. Garey, M.R., Johnson, D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness. W.H. Freeman & Company, San Francisco (1979)
6. Heule, M., Verwer, S.: Software model synthesis using satisfiability solvers. Empirical Software Engineering **18**(4), 825–856 (2013)
7. de la Higuera, C.: Grammatical Inference: Learning Automata and Grammars. Cambridge University Press (2010)
8. Jastrzab, T.: On parallel induction of nondeterministic finite automata. In: Proc. of ICCS 2016. Procedia Computer Science, vol. 80, pp. 257–268. Elsevier (2016)
9. Jastrzab, T.: Two parallelization schemes for the induction of nondeterministic finite automata on pcs. In: Proc. of PPAM 2017. LNCS, vol. 10777, pp. 279–289. Springer (2017)
10. Jastrzab, T.: A comparison of selected variable ordering methods for NFA induction. In: Proc. of ICCS 2019. LNCS, vol. 11540, pp. 741–748. Springer (2019)
11. Rossi, F., van Beek, P., Walsh, T. (eds.): Handbook of Constraint Programming. Elsevier Science, 1st edn. (2006)
12. Tomita, M.: Dynamic construction of finite-state automata from examples using hill-climbing. Proc. of the Fourth Annual Conference of the Cognitive Science Society pp. 105–108 (1982)
13. Tseitin, G.S.: On the Complexity of Derivation in Propositional Calculus, pp. 466–483. Springer Berlin Heidelberg, Berlin, Heidelberg (1983)
14. Vázquez de Parga, M., García, P., Ruiz, J.: A family of algorithms for non deterministic regular languages inference. In: Proc. of CIAA 2006. LNCS, vol. 4094, pp. 265–274. Springer (2006)
15. Wieczorek, W.: Grammatical Inference - Algorithms, Routines and Applications, Studies in Computational Intelligence, vol. 673. Springer (2017)