GA and ILS for optimizing the size of NFA models

Frédéric Lardeux, Eric Monfroy

GA and ILS for optimizing the size of NFA models

Frédéric Lardeux[1] and Eric Monfroy[1]

Univ Angers, LERIA, SFR MATHSTIC, F-49000 Angers, France
firstname.lastname@univ-angers.fr

Abstract. Grammatical inference consists in learning a formal grammar (as a set of rewrite rules or a finite state machine). We are concerned with learning Nondeterministic Finite Automata (NFA) of a given size from samples of positive and negative words. NFA can naturally be modeled in SAT. The standard model [1] being enormous, we also try a model based on prefixes [2] which generates smaller instances. We also propose a new model based on suffixes and a hybrid model based on prefixes and suffixes. We then focus on optimizing the size of generated SAT instances issued from the hybrid models. We present two techniques to optimize this combination, one based on Iterated Local Search (ILS), the second one based on Genetic Algorithm (GA). Optimizing the combination significantly reduces the SAT instances and their solving time, but at the cost of longer generation time. We, therefore, study the balance between generation time and solving time thanks to some experimental comparisons, and we analyze our various model improvements.

Mots-Clefs. Constraint problem modeling, Grammar inference, SAT, model reformulation, NFA inference.

1 Introduction

Grammatical inference [3] (or grammar induction) is concerned with the study of algorithms for learning automata and grammars from some observations. The goal is thus to construct a representation that accounts for the characteristics of the observed objects. This research area plays a significant role in numerous applications, such as compiler design, bioinformatics, speech recognition, pattern recognition, machine learning, and others.
In this article, we focus on learning a finite automaton from samples of words $S = S^+ \cup S^-$, such that S^+ is a set of positive words that must be accepted by the automaton, and S^- is a set of negative words to be rejected by the automaton. Due to their determinism, deterministic finite automata (DFA) are generally faster than non deterministic automata (NFA). However, NFA are significantly smaller than DFA in terms of the number of states. Moreover, the space complexity of the SAT models representing the problem is generally due to the number of states. Thus, 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.
The problem of inferring NFA has been undertaken with various approaches (see, e.g., [1]). Among them, we can cite ad-hoc algorithms such as *DeLeTe2* [4] that is based on state merging methods, or the technique of [5] that returns a collection of NFA. Some approaches use metaheuristics for computing NFA, such as hill-climbing [6] or genetic algorithm [7].
A convenient and declarative way of representing combinatorial problems is to model them as a Constraint Satisfaction Problem (CSP [8]) (see, e.g., [1] for an INLP model for inferring NFA, or [9] for a SAT (the propositional satisfiability problem [10]) model of the same problem). Parallel solvers have also been used for minimizing the inferred NFA size [11, 2].
Orthogonally to the approaches cited above, we do not seek to improve a solver, but to generate a model of the problem that is easier to solve with a standard SAT solver. Our approach is similar to DFA inference with graph coloring [12], or NFA inference with complex data structures [9]. Modeling thus consists in translating a problem into a CSP made of decision variables and constraints over these variables. As a reference for comparisons, we start with the basic SAT model of [9]. The model, together with a sample of positive and negative words, lead to a SAT instance to be solved by a classic SAT solver that we use as a black box. However, SAT instances are gigantic, e.g., our base model space complexity is in the order of $\mathcal{O}(k^{|\omega_+|})$ variables, and in $\mathcal{O}(|\omega_+|.k^{|\omega_+|})$ clauses, where k is the number of states of the NFA, and ω_+ is the size of the longest positive word of the sample. The second model,

PM, is based on intermediate variables for each prefix [2] which enables to compute only once parts of paths that are shared by several words. We propose a third model, SP, based on intermediate variables for suffixes. Although the two models could seem similar, their order of size is totally different. Indeed, PM is in $\mathcal{O}(k^2)$ while SM is in $\mathcal{O}(k^3)$. We then propose hybrid models consisting in splitting words into a prefix and a suffix. Modeling the beginning of the word is made with PM while the suffix is modeled by SM. The challenge is then to determine where to split words to optimize the size of the generated SAT instances. To this end, we propose two approaches, one based on iterated local search (ILS), the second one on genetic algorithm (GA). Both permit to generate smaller SAT instances, much smaller than with the DM model and even the PM model. However, with GA, the generation time is too long and erases the gain in solving with the Glucose SAT solver [13]. But the hybrid instances optimized with the ILS are smaller, and the generation time added to the solving time is faster than with PM. Compared to [9], which is the closest work on NFA inferring, we always obtain significantly smaller instances and solving time.
This paper is organized as follows. In Section 2 we present the direct model, the prefix model, and we propose the suffix model. We then combine suffix and prefix model to propose the new hybrid models (Section 3). Hybrid models are optimized with iterated local search (Sub-section 3.2), and with genetic algorithm in Sub-section 3.3. We then compare experimentally our models in Section 4 before concluding in Section 5.

2 SAT Models

Given 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 NFA inference problem** consists in building a NFA with k states which validates words of S^+, and rejects words of S^-. Note that the satisfaction problem we consider in this paper can be extended to an optimization problem minimizing k [2].
Let us introduce some notations. Let $A = (Q, \Sigma, q_1, F)$ be a NFA with: $Q = \{q_1, \ldots, q_k\}$ a set of k states, Σ a finite alphabet, q_1 the initial state, and F the set of final states. The empty word is noted λ. We denote by K the set of integers $\{1, \ldots, k\}$.
We consider the following variables:

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

The path $i_1, i_2, \ldots, i_{n+1}$ for $w = w_1 \ldots w_n$ exists if and only if $d = \delta_{w_1,\overrightarrow{q_{i_1} q_{i_2}}} \wedge \ldots \wedge \delta_{w_n,\overrightarrow{q_{i_n} q_{i_{n+1}}}}$ is true. We say that the conjunction d is a c_path, and $D_{w,\overrightarrow{q_i q_j}}$ is the set of all c_paths for the word w between states q_i and q_j.

2.1 Direct Model

This simple model has been presented in [9]. It is based on 3 sets of equations:

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

$$\text{if } \lambda \in S^+, \quad f_1 \tag{1}$$

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

2. For each word $w \in S^+$, there is at least a path from q_1 to a final state q_j:

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

With the Tseitin transformations [14], we create one auxiliary variable for each combination of a word w, a state $j \in K$, and a c_path $d \in D_{w,\overrightarrow{q_1 q_j}}$: $aux_{w,j,d} \leftrightarrow d \wedge f_j$. Hence, we obtain a formula in CNF for each w:

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

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

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

number of cl.	arity	Constraints
$\|S^+\|.(\|\omega_+\|+1).k^{\|\omega_+\|}$	2	(4)
$\|S^+\|.k^{\|\omega_+\|}$	$\|\omega_+\|+2$	(4)
$\|S^+\|$	$k^{\|\omega_+\|}$	(4)
$\|S^-\|.k^{\|\omega_-\|}$	$\|\omega_-\|+1$	(7)

Table 1. Clauses for DM_k

number of var	reason
k	final states F
$n.k^2$	transitions δ
$\|S^+\|.k.^{\|\omega_+\|}$	Constraints (3)

Table 2. Variables for DM_k

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

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

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

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

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

Size of the models *(see [9] for details)* Consider ω_+ and ω_-, the longest word of S^+ and S^- respectively. Table 1 presents the number of clauses (Column 1) and their arities (Column 2), which are an upper bound of a given constraint group (last column) for the model SM_k. Table 2 presents the upper bound of the number of Boolean variables that are required and why the are required. We can see on Tables 1 and 2 that the space complexity of the DM_k is huge ($\mathcal{O}(|S^+|.k.^{|\omega_+|})$ variables, and $\mathcal{O}(|S^+|.(|\omega_+|+1).k^{|\omega_+|})$ clauses) and with large clauses (up to arity of $|\omega_+|+2$), and that only small instances for a small number of states will be tractable. It is thus obvious that it is important to improve the model DM_k.

2.2 Prefix Model [2]

Let $Pref(w)$ be the set of all the non-empty prefixes of the word w and, by extension, $Pref(W) = \cup_{w \in W} Pref(w)$ the set of prefixes of the words of the set W. For each $w \in Pref(S)$, we add a Boolean variable $p_{w,\overrightarrow{q_1 q_i}}$ which determines whether there is or not a c_path for w from state q_1 to q_i. Note that these variables can be seen as labels of the Prefix Tree Acceptor (PTA) for S [3]. The problem can be modeled with the following constraints:

1. For all prefix $w = a$ with $w \in Pref(S)$, and $a \in \Sigma$, there is a c_path of size 1 for w:

$$\bigvee_{i \in K} \delta_{a,\overrightarrow{q_1 q_i}} \leftrightarrow p_{a,\overrightarrow{q_1 q_i}} \tag{8}$$

   With the Tseitin transformations, we can derive a CNF formula. It is also possible to directly encode $\delta_{a,\overrightarrow{q_1 q_i}}$ and $p_{a,\overrightarrow{q_1 q_i}}$ as the same variable. Thus, no clause is required.
2. For all words $w \in S^+ - \{\lambda\}$:

$$\bigvee_{i \in K} p_{w,\overrightarrow{q_1 q_i}} \wedge f_i \tag{9}$$

   With the Tseitin transformations [14], we create one auxiliary variable for each combination of $p_{w,\overrightarrow{q_1 q_i}}$ and the status (final or not) of the state q_i: $aux_{w,i} \leftrightarrow p_{w,\overrightarrow{q_1 q_i}} \wedge f_i$. Hence, for each w, we obtain a formula in CNF:

$$\bigwedge_{i \in K} ((\neg aux_{w,i} \vee p_{w,\overrightarrow{q_1 q_i}}) \wedge (\neg aux_{w,i} \vee f_i)) \tag{10}$$

$$\bigwedge_{i \in K} (aux_{w,i} \vee \neg p_{w,\overrightarrow{q_1 q_i}} \vee \neg f_i) \tag{11}$$

$$\bigvee_{i \in K} aux_{w,i} \tag{12}$$

3. For all words $w \in S^- - \{\lambda\}$, we obtain the following CNF constraint:

$$\bigwedge_{i \in K} (\neg p_{w,\overrightarrow{q_1 q_i}} \vee \neg f_i) \tag{13}$$

number of cl.	arity	Constraints
$2.k.\|S^+\|$	2	(10)
$k.\|S^+\|$	3	(11)
k	$k+1$	(12)
$k.\|S^-\|$	2	(13)
$\pi.k^2$	2	(15)
$\pi.k^2$	2	(16)
$\pi.k^2$	3	(17)
$\pi.k$	$k+1$	(18)
$\pi.k^2$	2	(19)

Table 3. Clauses for PM_k

number of var	reason
k	final states F
$n.k^2$	transitions δ
$\|S^+\|.k$	Constraints (9)
$\pi.k^2$	Constraints (14)

Table 4. Variables for PM_k

4. For all prefix $w = va$, $w \in Pref(S)$, $v \in Pref(S)$ and $a \in \Sigma$:

$$\bigwedge_{i \in K} (p_{w,\overrightarrow{q_1 q_i}} \leftrightarrow (\bigvee_{j \in K} p_{v,\overrightarrow{q_1 q_j}} \wedge \delta_{a,\overrightarrow{q_j q_i}})) \tag{14}$$

Applying the Tseitin transformations, we create one auxiliary variable for each combination of existence of a c_path from q_1 to q_i ($p_{v,\overrightarrow{q_1 q_j}}$) and the transition $\delta_{a,\overrightarrow{q_j q_i}}$: $aux_{v,a,j,i} \leftrightarrow p_{v,\overrightarrow{q_1 q_j}} \wedge \delta_{a,\overrightarrow{q_j q_i}}$. Then, (14) becomes:

$$\bigwedge_{i \in K} (p_{w,\overrightarrow{q_1 q_i}} \leftrightarrow (\bigvee_{j \in K} aux_{v,a,j,i}))$$

For each $w \in Pref(S)$, we obtain constraints in CNF:

$$\bigwedge_{(i,j) \in K^2} (\neg aux_{v,a,j,i} \vee p_{w,\overrightarrow{q_1 q_i}}) \tag{15}$$

$$\bigwedge_{(i,j) \in K^2} (\neg aux_{v,a,j,i} \vee \delta_{a,\overrightarrow{q_j q_i}}) \tag{16}$$

$$\bigwedge_{(i,j) \in K^2} (aux_{v,a,j,i} \vee \neg p_{w,\overrightarrow{q_1 q_i}} \vee \neg \delta_{a,\overrightarrow{q_j q_i}}) \tag{17}$$

$$\bigwedge_{i \in K} (\neg p_{w,\overrightarrow{q_1 q_i}} \vee (\bigvee_{j \in K} aux_{v,a,j,i})) \tag{18}$$

$$\bigwedge_{(i,j) \in K^2} (p_{w,\overrightarrow{q_1 q_i}} \vee \neg aux_{v,a,j,i})) \tag{19}$$

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

$$PM_k = \bigwedge_{w \in S^+} \Big((10) \wedge \ldots \wedge (12) \Big) \wedge \bigwedge_{w \in S^-} (13) \wedge \bigwedge_{w \in Pref(S)} (15) \wedge \ldots \wedge (19)$$

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

Size of the models Consider ω_+, the longest word of S^+, ω_-, the longest word of S^-, $\sigma = \Sigma_{w \in S}|w|$, and π, the number of prefix obtained by $Pref(S)$ with a size larger than 1 ($\pi = |\{x | x \in Pref(S), |x| > 1\}|$), then:

$$max(|\omega_+|, |\omega_-|) \leq \pi \leq \sigma \leq |S^+|.|\omega_+| + |S^-|.|\omega_-|$$

The space complexity of the PM_k model is thus in $\mathcal{O}(\sigma.k^2)$ variables, and in $\mathcal{O}(\sigma.k^2)$ binary and ternary clauses, and $\mathcal{O}(\sigma.k)$ $(k+1)$-ary clauses.

2.3 Suffix Model

We now propose a suffix model (SM_k), based on $Suf(S)$, the set of all the non-empty suffixes of all the words in S. The main difference is that the construction starts from every state and terminates in state q_1. For each $w \in Suf(S)$, we add a Boolean variable $p_{w,\overrightarrow{q_i q_j}}$ which determines whether there is or not a c_path for w from state q_i to q_j. To model the problem,

Constraints (10), (11), (12), and (13) remain unchanged and creation of the corresponding auxiliary variables $aux_{w,i}$ as well.

For each suffix $w = a$ with $w \in Suf(S)$, and $a \in \Sigma$, there is a c_path of size 1 for w:

$$\bigvee_{(i,j)\in K^2} \delta_{a,\overrightarrow{q_i q_j}} \leftrightarrow p_{a,\overrightarrow{q_i q_j}} \tag{20}$$

We can directly encode $\delta_{a,\overrightarrow{q_i q_j}}$ and $p_{a,\overrightarrow{q_i q_j}}$ as the same variable. Thus, no clause is required.

For all suffix $w = av$, $w \in Suf(S)$, $v \in Suf(S)$ and $a \in \Sigma$:

$$\bigwedge_{(i,j)\in K^2} (p_{w,\overrightarrow{q_i q_j}} \leftrightarrow (\bigvee_{k\in K} \delta_{a,\overrightarrow{q_i q_k}} \wedge p_{v,\overrightarrow{q_k q_j}})) \tag{21}$$

We create one auxiliary variable for each combination of existence of a c_path from q_k to q_j ($p_{v,\overrightarrow{q_k q_j}}$) and the transition $\delta_{a,\overrightarrow{q_i q_k}}$: $aux_{v,a,i,k,j} \leftrightarrow \delta_{a,\overrightarrow{q_i q_k}} \wedge p_{v,\overrightarrow{q_k q_j}}$
For each $w = av$, we obtain the following constraints (CNF formulas):

$$\bigwedge_{(i,j,k)\in K^3} (\neg aux_{v,a,i,k,j} \vee p_{w,\overrightarrow{q_k q_j}}) \tag{22}$$

$$\bigwedge_{(i,j,k)\in K^3} (\neg aux_{v,a,i,k,j} \vee \delta_{a,\overrightarrow{q_i q_k}}) \tag{23}$$

$$\bigwedge_{(i,j,k)\in K^3} (aux_{v,a,i,k,j} \vee \neg p_{w,\overrightarrow{q_k q_j}} \vee \neg \delta_{a,\overrightarrow{q_i q_k}}) \tag{24}$$

$$\bigwedge_{(i,j)\in K^2} (\neg p_{w,\overrightarrow{q_i q_j}} \vee (\bigvee_{k\in K} aux_{v,a,i,k,j})) \tag{25}$$

$$\bigwedge_{(i,j,k)\in K^3} (p_{w,\overrightarrow{q_i q_j}} \vee \neg aux_{v,a,i,k,j})) \tag{26}$$

Note that some clauses are not worth being generated. Indeed, it is useless to generate paths starting in states different from the initial state q_1, except when the w is in S, and w is also the suffix of another word from S. Removing these constraints does not change the complexity of the model. This can easily be done at generation time, or we can leave it to the solver, which will detect it and remove the useless constraints.
Thus, the constraint prefix model PM_k for building a NFA of size k is:

$$SM_k = \bigwedge_{w\in S^+} \Big((10) \wedge \ldots \wedge (12)\Big) \wedge \bigwedge_{w\in S^-} (13) \wedge \bigwedge_{w\in Pref(S)\setminus S} (22) \wedge \ldots \wedge (26)$$

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

Size of the models Consider ω_+, ω_-, σ, and π as defined in the prefix model. Table 5 presents the number of clauses (first column) and their arities (Column 2) which are an upper bound of a given constraint group (last column) for the model SM_k. Table 6 presents the upper bound of the number of Boolean variables that are required, and the reason of their requirements. To simplify, the space complexity of SM_k is thus in $\mathcal{O}(\sigma.k^3)$ variables, and in $\mathcal{O}(\sigma.k^3)$ binary and ternary clauses, and $\mathcal{O}(\sigma.k^2)$ $(k+1)$-ary clauses.

3 Hybrid Models

We now propose a family of models based on both the notion of prefix and the notion of suffix. The idea is, in fact, to take advantage of the construction of a prefix p and a suffix s of a word w such that $w = p.s$ to pool both prefixes and suffixes. The goal is to reduce the size of generated SAT instances. The process is the following:

1. For each word w_i of S, we split w_i into p_i and s_i such that $w = p_i.s_i$. We thus obtain two sets, $S_p = \{p_i \mid \exists i, w_i \in S \text{ and } w_i = p_i.s_i\}$ and $S_s = \{s_i \mid \exists i, w_i \in S \text{ and } w_i = p_i.s_i\}$.
2. We then consider S_p as a sample, i.e., a set of words. For each w of S_p, we generate Constraints (15) to (19).
3. We consider S_s in turn to generate Constraints (22) to (26) for each $w \in S_s$.

number of cl.	arity	Constraints
$2.k.\|S^+\|$	2	(10)
$k.\|S^+\|$	3	(11)
k	$k+1$	(12)
$k.\|S^-\|$	2	(13)
$\pi.k^3$	2	(22)
$\pi.k^3$	2	(23)
$\pi.k^3$	3	(24)
$\pi.k^2$	$k+1$	(25)
$\pi.k^3$	2	(26)

Table 5. Clauses for SM_k

number of var	reason
k	final states F
$n.k^2$	transitions δ
$\|S^+\|.k$	Constraints (9)
$\pi.k^3$	Constraints (21)

Table 6. Variables for SM_k

4. Then, for each $w_i = p_i.s_i$, clauses corresponding to p_i must be linked to clauses of s_i.
   - if $w_i = p_i.s_i \in S^-$, the constraints are similar to the ones of (13) including the connection of p_i and s_i:

$$\bigwedge_{(j,k)\in K^2} (\neg p_{p_i,\overrightarrow{q_1 q_j}} \vee \neg p_{s_i,\overrightarrow{q_j q_k}} \vee \neg f_i) \tag{27}$$

   - if $w_i = p_i.s_i \in S^+$, the constraints are similar to (9):

$$\bigvee_{(j,k)\in K^2} p_{p_i,\overrightarrow{q_1 q_j}} \wedge p_{s_i,\overrightarrow{q_j q_k}} \wedge f_k \tag{28}$$

   We transform (28) using auxiliary variables $aux_{w_i,j,k} \leftrightarrow p_{w,\overrightarrow{q_1 q_j}} \wedge p_{w,\overrightarrow{q_j q_k}} \wedge f_i$ to obtain the following CNF constraints:

$$\bigwedge_{(j,k)\in K^2} ((\neg aux_{w_i,j,k} \vee p_{w,\overrightarrow{q_1 q_j}}) \wedge (\neg aux_{w_i,j,k} \vee p_{w,\overrightarrow{q_j q_k}}) \wedge (\neg aux_{w_i,j,k} \vee f_k)) \tag{29}$$

$$\bigwedge_{(j,k)\in K^2} (aux_{w_i,j,k} \vee \neg p_{w,\overrightarrow{q_1 q_j}} \vee p_{w,\overrightarrow{q_j q_k}} \vee \neg f_k) \tag{30}$$

$$\bigvee_{(j,k)\in K^2} aux_{w_i,j,k} \tag{31}$$

Thus, the hybrid model HM_k for building a NFA of size k is:

$$HM_k = \bigwedge_{w\in S^+} \Big((29)\wedge\ldots\wedge(31)\Big) \wedge \bigwedge_{w\in S^-} (27) \wedge \bigwedge_{p_i\in Pref(S_p)} (22)\wedge\ldots\wedge(26) \bigwedge_{s_i\in Suf(S_s)} (15)\wedge\ldots\wedge(19)$$

and it is possibly completed by (1) or (2) if $\lambda \in S^+$ or $\lambda \in S^-$.
We do not detail it here, but in the worst case, the complexity of the model is the same as SM_k. It is obvious that the split of each word into a prefix and a suffix will determine the size of the instance. The next sub-sections are dedicated to the computation of this separation $w_i = p_i.s_i$ to minimize the size of the generated hybrid instances with the HM_k model.

3.1 Search Space and Evaluation Function For Metaheuristics

The search space $\mathcal{X}$ of this problem corresponds to all the hybrid models: for each word w of S, we have to determine a n such that $w = p.s$ with $|p| = n$ and $|s| = |w| - n$. The size of the search space is thus: $|\mathcal{X}| = \Pi_{w\in S}|w| + 1$.
Even though we are aware that smaller instances are not necessarily easier to solve, we choose to define the first evaluation function as the number of generated SAT variables. However, this number cannot be computed a priori: first, the instance has to be generated, before counting the variables. This function being too costly, we propose an alternative evaluation function for approximating the number of variables. This fitness function is based on the number of prefixes in $Pref(S_p)$ and suffixes in $Suf(S_s)$. Since the complexity of SM_k is in $\mathcal{O}(k^3)$ whereas the complexity of PM_k is in $\mathcal{O}(k^2)$, suffixes are penalized by a coefficient corresponding to the number of states.

$$fitness(S_p, S_s) = |Pref(S_p)| + k.|Suf(S_s)|$$

Empirically, we observe that the results of this $fitness$ function are proportional to the actual number of generated SAT variables. This approximation of the number of variables will thus be the fitness function in our ILS and GA algorithms.

3.2 Iterated Local Search Hybrid Model HM_ILS_k

We propose an Iterated Local Search (ILS) [15] for optimizing our hybrid model. Classically, a best improvement or a first improvement neighborhood is used in ILS to select the next move. In our case, a first improvement provides very poor results. Moreover, it is clearly impossible to evaluate all the neighbors at each step due to the computing cost. We thus decide to randomly choose a word in S with a roulette wheel selection based on the word weights. Each word w has a weight corresponding for 75% to a characteristic of S, and 25% to the length of the word:

$$\text{weight}_w = 75\%/|S| + 25\% * |w|/(\sum_{w_i \in S} |w_i|)$$

The search starts generating a random couple of prefixes and suffixes sets (S_p,S_s), i.e., for each word w of S an integer is selected for splitting w into a prefix p and a suffix s such that $w = p.s$. Hence, at each iteration, the best couple (p, s) is found for the selected word w. This process is iterated until a maximum number of iterations is reached.
In our ILS, it is not necessary to introduce noise with random walks or restarts because our process of selection of word naturally ensures diversification.

Algorithm 1: Iterated Local Search

Input: set of words S, maximum number of iterations max_iter
maximum of consecutive iterations allowed without improvement $max\_iter\_without\_improv$
Output: set of prefixes S_p^*, set of suffixes S_s^*
1: Couple of prefixes and suffixes sets (S_p,S_s) is randomly generated
2: (S_p^*,S_s^*) = (S_p,S_s)
3: **repeat**
4: Choose a word w in S with a roulette wheel selection
5: (S_p,S_s) is updated by the best couple of the sub-search space corresponding only to a modification of the prefix and the suffix of word w
6: **if** $fitness(S_p, S_s) < fitness(S_p^*, S_s^*)$ **then**
7: (S_p^*,S_s^*) = (S_p,S_s)
8: **end if**
9: **until** maximum number of iterations max_iter is reached or (S_p^*,S_s^*) is not improved since $max\_iter\_without\_improv$ iterations
10: **return** (S_p^*, S_s^*)

3.3 Genetic Algorithm Hybrid Model HM_GA_k

We propose a classical genetic algorithm (GA) based on the search space and fitness function presented in Section 3.1. A population of individuals, represented by a couple of prefixes and suffixes sets, is improved generation after generation. Each generation keeps a portion of individuals as parents and creates children by crossing the selected parents. Crossover operator used in our GA is the well-known uniform crossover. For each word, children inherit the prefix and the suffix of one of their parents randomly chosen. Since the population size is the same during all the search, we have a steady-state GA. A mutation process is applied over all individuals with a probability p_{mut}. For each word w, each prefix and suffix are randomly mutated by generating an integer n between 0 and $|w|$ splitting w into a new prefix of size n and a new suffix $|w| - n$. The search stops when the maximum number of generations is reached or when no improvement is observed in the population during $max\_gen\_without\_improv$ generations.

Algorithm 2: Genetic Algorithm

Input: set of words S, population size $s_{\mathcal{P}}$, mutation probability p_{mut},
maximum number of generations max_gen,
portion of population conserve in the next generation $p_{parents}$,
maximum of consecutive generations allowed without improvement $max\_gen\_without\_improv$
Output: set of prefixes S_p^*, set of suffixes S_s^*
1: Population $\mathcal{P}$ of couples of prefixes and suffixes sets (S_p,S_s) is randomly generated
2: (S_p^*,S_s^*) = Argmin$_{fitness}(\mathcal{P})$
3: **repeat**
4: Select as parents set Par a portion $p_{parents}$ of $\mathcal{P}$
5: Generate $(1 - p_{parents}).s_{\mathcal{P}}$ children by uniform crossover over parents in a set $Children$
6: $\mathcal{P} = Par \cup Children$
7: Mutate for each individual of $\mathcal{P}$ the prefix/suffix for each words of S with a probability p_{mut}
8: Update the population
9: Update (S_p^*,S_s^*) if necessary
10: **until** maximum number of generations max_gen is reached or (S_p^*,S_s^*) is not improved since $max\_gen\_without\_improv$ generations
11: **return** S_p^* and S_s^*

4 Experimental results

To test our new models, we work on the training set of the StaMinA Competition (see http://stamina.chefbe.net). We use 11 of the instances selected in [2][1] with a sparsity $s \in \{12.5\%, 25\%, 50\%, 100\%\}$ and an alphabet size $|\Sigma| \in \{2, 5, 10\}$. We try to generate SAT instances for NFA sizes (k) near to the threshold of the existence or not of an NFA.

4.1 Experimental Protocol

All our algorithms are implemented in Python using specific libraries such as Pysat. The experiments were carried out on a computing cluster with Intel-E5-2695 CPUs, and a limit of 10 GB of memory was fixed. Running times were limited to 10 minutes, including generation of the model and solving time. We used the Glucose [13] SAT solver with the default options. For stochastic methods (ILS and GA), 30 runs are realized to exploit the results statistically. Parameters used for our hybrid models are:

ILS		AG	
max_iter	10 000	$s_{\mathcal{P}}$	100
$max\_iter\_without\_improv$	100	max_gen	3000
		$max\_gen\_without\_improv$	100
		p_{mut}	0.05
		$p_{parents}$	0.03

4.2 Results

Our experiments are reported in Table 7. The first column ($Instance$) corresponds to the official name of the instance, and the second one (k) to the number of states of the expected NFA. Then, we have in sequence the model name ($Model$), the number of SAT variables ($Var.$), the number of clauses ($Cl.$), and the instance generation time (t_M). The right part of the table corresponds to the solving part with the satisfiability of the generated instance (SAT), the decisions number ($Dec.$), and the solving time (t_S) with Glucose. Finally, the last column (t_T) corresponds to the total time (modeling time + solving time). Results for hybrid models based on ILS (HM_ILS_k) and GA (HM_GA_k) correspond to average values over 30 runs. We have decided to only provide the average since the standard deviation values are very small.
The last lines of the table correspond to the cumulative values for each column and each model. When an instance is not solved (time-out), the maximum value needed for solving

[1] We kept the "official" name used in [2].

the other model instances is considered. For the instance generation time (t_M), a credit of 600 seconds is applied when generation did not succeed before the time-out.

We can clearly confirm that the direct model is not usable in practice, and that instances cannot be generated in less than 600s. The prefix model allows the fastest generation when it terminates before the time out (on these benchmarks, it did not succeed once and was thus penalize for cumulative values). It also provides instances that are solved quite fast. As expected, the instances optimized with GA are the smallest ones. However, the generation is too costly: the gain in solving time is not sufficient to compensate the long generation time. In total, in terms of solving+generation time, GA based model is close to prefix model. As planned with its space complexity (in $\mathcal{O}(k^3)$), suffix based instances are huge and long to solve. However, we were surprised for 2 benchmarks (ww-10-40 and ww-10-50) for which the generated instances are relatively big (5 times the size of the GA optimized instances), but their solving is the fastest. We still cannot explain what made these instances easy to solve, and we are still investigating their structure. The better balance is given with the ILS model: instances are relatively small, the generation time is fast, and the solving time as well. This is thus the best option of this work.
It is very difficult to compare our results with the results of [2]. First of all, in [2], they try to minimize k, the number of states. Moreover, they use parallel algorithms. Finally, they do not detail the results for each instance and each k, except for st-2-30 and st-5-50. For the first one, with $k = 9$ we are much faster. But for the second one, with $k = 5$ we are slower.

5 Conclusion

In this paper, we have proposed to use some metaheuristics algorithms, namely ILS and GA, to improve the size of SAT models for the NFA inferring problem. Our hybrid model, optimized with GA gives, on average, the smallest SAT instances. Solving these instances is also faster than with the direct or prefix models. However, generation of the optimized instances with GA is really too long and is not balanced out with the gain in solving time; it is at the level of the prefix model w.r.t. total CPU time. The ILS model generates optimized instances a bit larger than with GA and a bit smaller than with prefixes. Moreover, the solving time is the best of our experiments, and the generation time added to the solving time makes of the HM_ILS_k our better model.
In the future, we plan to speed up GA to make it more competitive. We also plan to consider more complex fitness functions, not only based on the number of SAT variables but also on the length of clauses. We also plan a model portfolio approach for larger samples.

References

1. Wieczorek, W.: Grammatical Inference - Algorithms, Routines and Applications. Volume 673 of Studies in Computational Intelligence. Springer (2017)
2. Jastrzab, T., Czech, Z.J., Wieczorek, W.: Parallel algorithms for minimal nondeterministic finite automata inference. Fundam. Informaticae **178** (2021) 203–227
3. de la Higuera, C.: Grammatical Inference: Learning Automata and Grammars. Cambridge University Press (2010)
4. Denis, F., Lemay, A., Terlutte, A.: Learning regular languages using rfsas. Theor. Comput. Sci. **313** (2004) 267–294
5. Vázquez de Parga, M., García, P., Ruiz, J.: A family of algorithms for non deterministic regular languages inference. In: Proc. of CIAA 2006. Volume 4094 of LNCS., Springer (2006) 265–274
6. Tomita, M.: Dynamic construction of finite-state automata from examples using hill-climbing. Proc. of the Annual Conference of the Cognitive Science Society (1982) 105–108
7. Dupont, P.: Regular grammatical inference from positive and negative samples by genetic search: the GIG method. In: Proc. of ICGI 94. Volume 862 of LNCS., Springer (1994) 236–245
8. Rossi, F., van Beek, P., Walsh, T., eds.: Handbook of Constraint Programming. 1st edn. Elsevier Science (2006)
9. Lardeux, F., Monfroy, E.: Improved SAT models for NFA learning. In: Proc. of OLA 2021. Communications in Computer and Information Science, Springer (2021) In press.
10. Garey, M.R., Johnson, D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness. W.H. Freeman & Company, San Francisco (1979)

Table 7. Comparison on 11 instances between the models DM_k, PM_k, SM_k, HM_ILS_k, and HM_GA_k.

Instance	k	Model	Var.	Cl.	t_M	SAT	Dec.	t_S	t_T
st-2-10	4	DM_k	190 564	1 817 771	13.46	True	973 213	88.62	102.09
		PM_k	1 276	4 250	1.27	True	3 471	0.10	**1.37**
		SM_k	5 196	17 578	1.30	True	4 332	0.21	1.51
		HM_ILS_k	1 188	4 179	4.14	True	2 503	**0.05**	4.18
		HM_GA_k	**1 107**	3 884	14.45	True	2 368	**0.05**	14.50
st-2-20	6	DM_k	-	-	-	-	-	-	-
		PM_k	4 860	17 150	1,34	False	1 625 706	241,24	242,59
		SM_k	-	-	-	-	-	-	-
		HM_ILS_k	5 688	21 073	5,39	False	662 354	98,35	**103,74**
		HM_GA_k	**4 735**	17 611	34,65	False	708 356	**94,95**	129,61
st-2-30	9	DM_k	-	-	-	-	-	-	-
		PM_k	-	-	-	-	-	-	-
		SM_k	-	-	-	-	-	-	-
		HM_ILS_k	20 637	78 852	7.55	True	1 998 574	**228.53**	**236.07**
		HM_GA_k	**16 335**	62 832	66.94	True	4 079 686	527.44	594.38
st-5-20	4	DM_k	-	-	-	-	-	-	-
		PM_k	4 024	13 464	1.49	True	2 641	**0.08**	**1.57**
		SM_k	14 964	50 660	1.68	True	23 540	3.23	4.91
		HM_ILS_k	3 608	12 514	7.83	True	14 584	0.72	8.56
		HM_GA_k	**3 522**	12 180	47.84	True	18 344	0.94	48.78
st-5-30	4	DM_k	-	-	-	-	-	-	-
		PM_k	5 364	18 054	1.43	True	177 711	21.57	**23.00**
		SM_k	21 084	71 502	1.87	True	362 318	128.02	129.89
		HM_ILS_k	4 837	16 955	9.90	True	156 631	**19.90**	29.81
		HM_GA_k	**4 705**	16 478	119.42	True	171 062	21.67	141.09
st-5-40	4	DM_k	-	-	-	-	-	-	-
		PM_k	6 284	21 216	1.52	False	7 110	0.55	**2.07**
		SM_k	23 604	80 104	1.55	False	15 708	1.74	3.29
		HM_ILS_k	5 745	20 290	10.29	False	6 206	**0.34**	10.62
		HM_GA_k	**5 548**	19 517	150.50	False	6 204	0.35	150.85
st-5-50	5	DM_k	-	-	-	-	-	-	-
		PM_k	11 150	38 745	1.59	False	1 943 735	562.80	564.39
		SM_k	-	-	-	-	-	-	-
		HM_ILS_k	11 085	40 258	10.80	False	911 280	**238.10**	**248.90**
		HM_GA_k	**10 040**	36 350	279.87	False	1 093 093	287.46	567.33
st-5-60	5	DM_k	-	-	-	-	-	-	-
		PM_k	14 200	49 455	1.52	False	1 245 538	383.37	384.89
		SM_k	-	-	-	-	-	-	-
		HM_ILS_k	13 920	50 568	13.47	False	800 920	**231.82**	**245.29**
		HM_GA_k	**13 180**	47 755	313.30	False	950 601	270.97	584.26
ww-10-40	4	DM_k	15 012	112 039	2.07	True	69 219	1.52	3.59
		PM_k	3 624	11 900	1.38	True	977	0.03	1.41
		SM_k	13 844	46 648	1.25	True	4 173	**0.02**	**1.28**
		HM_ILS_k	2 896	10 342	5.94	True	3 897	0.06	6.00
		HM_GA_k	**2 761**	9 839	75.57	True	2 842	0.04	75.60
ww-10-50	4	DM_k	80 548	694 641	5.61	True	483 153	103.52	109.14
		PM_k	5 364	17 850	1.28	True	167 390	20.29	21.57
		SM_k	20 844	70 482	1.49	True	74 482	11.58	**13.07**
		HM_ILS_k	4 633	16 514	7.71	True	73 534	5.46	13.17
		HM_GA_k	**4 517**	15 940	123.21	True	52 894	**3.38**	126.59

Cumulative values	DM_k	397 451	3 014 842	4 221,15	-	10 821 816	2 041,50	6 262,65
	PM_k	76 783	270 936	612,82	-	9 253 965	1 757,47	2 370,29
	SM_k	151 211	525 099	2 409,15	-	9 379 218	1 879,65	4 268,80
	HM_ILS_k	74 237	271 543	**83,03**	-	4 630 483	**823,33**	**906,36**
	HM_GA_k	**66 450**	242 386	1 225,75	-	7 085 451	1 207,23	2 432,98

11. Jastrzab, T.: Two parallelization schemes for the induction of nondeterministic finite automata on pcs. In: Proc. of PPAM 2017. Volume 10777 of LNCS., Springer (2017) 279–289
12. Heule, M., Verwer, S.: Software model synthesis using satisfiability solvers. Empirical Software Engineering **18** (2013) 825–856
13. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: Proc. of IJCAI 2009. (2009) 399–404
14. Tseitin, G.S. In: On the Complexity of Derivation in Propositional Calculus. Springer Berlin Heidelberg, Berlin, Heidelberg (1983) 466–483
15. Stützle, T., Ruiz, R. In: Iterated Local Search. Springer International Publishing, Cham (2018) 579–605