Optimized models and symmetry breaking for the NFA inference problem

Frédéric Lardeux, Eric Monfroy

Optimized models and symmetry breaking for the NFA inference problem

Frédéric Lardeux
Univ Angers, LERIA,
F-49000 Angers, France
Email: Frederic.Lardeux@univ-angers.fr

Eric Monfroy
Univ Angers, LERIA,
F-49000 Angers, France
Email: Eric.Monfroy@univ-angers.fr

***Abstract*—Grammatical inference is concerned with the study of algorithms for learning automata and grammars from words. We propose some models for learning Nondeterministic Finite Automaton (NFA) of size k from samples of words of the language and words not belonging to the language we want to describe. To this end, we formulate the problem as a SAT model trying to reduce the size of generated SAT instances.**

We propose new models to generate even smaller SAT instances. We also suggest some techniques for breaking some symmetries, hence reducing the search space, and consequently, speeding-up solving. We also achieved some experimental comparisons and we analyze our various model improvements and over-constraint propositions. Compared to [1], our models are easier and faster to solve. Compare to the parallel solver of [2], we find some new bounds for some instances for which the minimal size of NFA is not known yet.

1. Introduction

A classic way to identify and learn regular languages is to infer a finite automaton. While deterministic finite automata (DFAs) are generally faster for accepting or rejecting a word, they are also larger, sometimes exponentially larger (in terms of the number of states), than non-deterministic finite automata (NFAs). The spatial complexity of models for inferring automata being polynomial in the number of states, NFAs become a very good alternative. The problem can be formulated as follows: given a sample $S = S^+ \cup S^-$ made of some positive words (S^+) that are in the regular language L, some negative words (S^-) that are not element of the language L, and k a number of states, find an automaton of size k to characterize L by accepting words from S^+ and rejecting words from S^-.

DFA inference is generally based on the generation of a prefix tree acceptor (PTA), and then, on some algorithms for merging states while keeping the determinism, see for example [3] for the RPNI (Regular Positive and Negative Inference) algorithm, or [4] for merge based on graph coloring. From the idea that a non deterministic automaton (NFA) is usually a smaller description of a regular language than its equivalent DFAs, various techniques have been proposed. Among them, algorithms, such as $DeLeTe2$ [5] based on Residual Finite State Automata, or algorithm for Unambiguous Automata [6]. Other approaches use metaheuristics for computing NFA, such as hill-climbing [7] or genetic algorithm [8].

Constraint Satisfaction Problem (CSP [9]) makes it possible to declaratively represent combinatorial problems: the user defines variables together with their domains (i.e., the candidate values) and relations between these variables (i.e., the constraints). This task is called modeling, the result (variables and constraints) is a model, and a model together with some data forms an instance. In our case, the model is a NFA inference model, the data are S and k. For example, an INLP (Integer Non Linear Programming) model for inferring NFA is given in [10], in [2] a SAT (the propositional satisfiability problem [11]) model is given, and in [12], [1] several improvements of the SAT model are given.

Whereas some approaches focus on improving the solver, such as specific strategies [13], or parallel techniques [2] for optimizing k, we have an othogonal approach that consists in improving the model and using a standard solver. In [12], we studied complex data structures to generate smaller instances, and in [1], we proposed various models, and we use metaheuristics to optimize instance sizes. In this paper, we propose four new models for the inference of NFA based on the optimization of the splitting of words into a prefix and a suffix. The idea is to try to share prefixes (respectively suffixes) among as many words as possible and/or to obtain the longest prefixes (respectively suffixes). Our new $S_k^\star$ model surpass our previous models [1], both in the size of the generated instances and in the solving speed. The new model based on Iterated Local Search is very efficient in terms of instance size. However, the gain in solving time is not always sufficient to compensate for the time lost in instance generation.

We also propose some extra constraints to over-constrain the model, either removing symmetric solutions (in this case, the "deleted" solutions can be recovered by permutation of remaining solutions) or loosing solutions (these solutions cannot be recovered, but as we will see, these over-constrained models can help finding solutions of complex instances). The first extra-constraints consists in enforcing, for each symbol, the first derivation from the initial state q_1. The second one consists in reducing undeterminism of the first transitions from q_1.

We present some results of experimentation. We use the

benchmarks of [2]. Our new models with symmetry breaking techniques give even better results than the models presented in [1]. Compared to [2] which use parallelism, we find interesting results, and for some instances, we were able to find better bounds for some instances for which the optimal size k is not known.

This paper is organized as follows. The next section (Section 2), describes the models for inferring NFAs. We first describe the necessary variables, and then, some previous models (Section 2.1) that will be used to compare our new models presented in Section 2.2. Complementarily to the models, we propose some extra constraints to over constraint the model. The first one (Section 3.1), only remove some symmetric solutions that can be recovered by permutations of nodes. The second one is also a symmetry breaking technique for the final states (Section 3.2). The last one may loose some solutions that cannot be recovered latter (Section 3.3). We then present our experimentation in Section 4. The results are given in Section 5 together with some discussions and comparisons with previous works. We finally conclude in Section 6.

2. Models

Let $\Sigma = \{s_1, \ldots, s_n\}$ be an alphabet of n symbols. Consider a training sample $S = S^+ \cup S^-$, where S^+ (respectively S^-) is a set of *positive words* (respectively *negative words*) from Σ^*, and k, an integer. **The NFA inference problem** is to learn a NFA with k states, that accepts each word of S^+, and rejects each word of S^-. We say that S^+ is a set of "positive" words, and S^- is a set of "negative" words. This satisfaction problem can be extended to an optimization problem for minimizing k [2].

A NFA is a 4-tuple $A = (Q, \Sigma, q_1, F)$ 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. λ denotes the empty word, and K is the set of integers $\{1, \ldots, k\}$.

We will consider the following variables in our models:

- k, an integer, the size of the NFA we want to learn,
- a set of k Boolean variables $F = \{f_1, \ldots, f_k\}$ determining whether state q_i is final or not,
- and $\Delta = \{\delta_{s,\overrightarrow{q_i q_j}} | s \in \Sigma \text{ and } (i,j) \in K^2\}$ 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 i, 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. Previous models

For each model, we have to consider λ:

$$(\lambda \in S^+ \longrightarrow f_1) \wedge (\lambda \in S^- \longrightarrow \neg f_1) \quad (1)$$

The **Direct model** (see e.g., [2], [1]) to infer a NFA of size k is composed of 3 sets of constraints. $D_k = (1) \wedge (2) \wedge (3)$ with:

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

$$\bigvee_{j \in K} \bigvee_{d \in D_{w,\overrightarrow{q_1 q_j}}} (d \wedge f_j) \quad (2)$$

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

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

After Tseitin transformations [14], to convert D_k in CNF, the spacial complexity is in $\mathcal{O}(|S^+| \cdot (|\omega_+|+1) \cdot k^{|\omega_+|})$ for the number of clauses, and in $\mathcal{O}(|S^+| \cdot k^{\cdot |\omega_+|})$ for the number of variables with ω_+ (respectively ω_-) the longest word of S^+ (respectively of S^-). See [1] for more details about the Tseitin transformations and complexity of the model.

The **Prefix Model** is defined as follows. Let $Pref(w)$ be the set of all the non-empty prefixes of the word w and, $Pref(W) = \cup_{w \in W} Pref(w)$. For each $w \in Pref(S)$, consider a Boolean variable $p_{w,\overrightarrow{q_1 q_i}}$ to determine the existence of a c_path for w from state q_1 to q_i. The prefix model is defined by $P_k = (1) \wedge (4) \wedge (5) \wedge (6) \wedge (7)$ with Constraints (4), (5), (6), and (7) defined as follows:

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

$$\bigvee_{i \in K} \delta_{a,\overrightarrow{q_1 q_i}} \leftrightarrow p_{a,\overrightarrow{q_1 q_i}} \quad (4)$$

- For each prefix $w = va$, $w \in Pref(S)$, $v \in Pref(S)$, and $a \in \Sigma$:

$$\bigwedge_{i \in K} (p_{w,\overrightarrow{q_1 q_i}} \leftrightarrow (\bigvee_{j \in K} p_{v,\overrightarrow{q_1 q_j}} \wedge \delta_{a,\overrightarrow{q_j q_i}})) \quad (5)$$

- For each word $w \in S^+ \setminus \{\lambda\}$:

$$\bigvee_{i \in K} p_{w,\overrightarrow{q_1 q_i}} \wedge f_i \quad (6)$$

- For each word $w \in S^- \setminus \{\lambda\}$:

$$\bigwedge_{i \in K} (\neg p_{w,\overrightarrow{q_1 q_i}} \vee \neg f_i) \quad (7)$$

After transformations, P_k is converted in CNF, and its spacial complexity is in $\mathcal{O}(\sigma \cdot k^2)$ variables, and $\mathcal{O}(\sigma \cdot k^2)$ clauses with $\sigma = \Sigma_{w \in S}|w|$. See [1] for details.

The **Suffix Model**, S_k, is based on $Suf(S)$, the set of all the non-empty suffixes of all the words in S. The construction starts from every state and terminates in state q_1. For each $w \in Suf(S)$, we add a Boolean variable $p_{w,\overrightarrow{q_i q_j}}$

to determine the existence of a c_path for w from q_i to q_j. $S_k = (1) \wedge (8) \wedge (9) \wedge (6) \wedge (7)$ with:

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

$$\bigvee_{(i,j)\in K^2} \delta_{a,\overrightarrow{q_i q_j}} \leftrightarrow p_{a,\overrightarrow{q_i q_j}} \quad (8)$$

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

$$\bigwedge_{(i,j)\in K^2} (p_{w,\overrightarrow{q_i q_j}} \leftrightarrow (\bigvee_{k\in K} \delta_{a,\overrightarrow{q_i q_k}} \wedge p_{v,\overrightarrow{q_k q_j}})) \quad (9)$$

Although P_k models, and S_k models could seem similar, their complexities are very different. S_k models are in $\mathcal{O}(\sigma \cdot k^3)$ variables, and in $\mathcal{O}(\sigma \cdot k^3)$ clauses [1].

The **Hybrid Models** consists in splitting each word $w \in S$ into a prefix p and a suffix s such that $w = p.s$. We then consider two samples, $S_p = S_p^+ \cup S_p^-$ with $S_p^+ = \{p \mid \exists w \in S^+, w = p.s\}$ and $S_p^- = \{p \mid \exists w \in S^-, w = p.s\}$, and $S_s = S_s^+ \cup S_s^-$ with $S_s^+ = \{s \mid \exists w \in S^+, w = p.s\}$ and $S_s^- = \{s \mid \exists w \in S^-, w = p.s\}$.

For each prefix of $Pref(S_p)$ we generate Constraints (4) and (5), and for each suffix of $Suf(S_s)$, Constraints (8) and (9). For each $w = p.s$, clauses generated for p must be linked to clauses generated for s as follows:

- if $w = p.s \in S^-$:

$$\bigwedge_{(j,k)\in K^2} (\neg p_{p,\overrightarrow{q_1 q_j}} \vee \neg p_{s,\overrightarrow{q_j q_k}} \vee \neg f_k) \quad (10)$$

- if $w = p.s \in S^+$:

$$\bigvee_{(j,k)\in K^2} p_{p,\overrightarrow{q_1 q_j}} \wedge p_{s,\overrightarrow{q_j q_k}} \wedge f_k \quad (11)$$

We thus have: $H_k = (1) \wedge 4) \wedge (5) \wedge (8) \wedge (9) \wedge (10) \wedge (11)$ Obviously, the way we split each word into a prefix and a suffix will determine the size of the instance.

In [12], we proposed two strategies based on metaheuristics for optimizing models, and consequently for optimizing the split of each word of S. For both of them, the search space corresponds to all the hybrid models, i.e., all the possible split for all words of S. The fitness we used is: $f(S_p, S_s) = |Pref(S_p)| + k \cdot |Suf(S_s)|$.

The first strategy is based on an Iterated Local Search (ILS) [15] with the fitness f for optimizing our hybrid model $ILS_k(rand, f)$. The search starts with a random split for each word. At each iteration, the best split $w = p.s$ is found for the selected word w: w is selected randomly with a roulette wheel selection based on the weights of words defined by $\text{weight}_w = 75\%/|S| + 25\% * |w|/(\sum_{w_i \in S} |w_i|)$. The number of iterations is given. We do not need to introduce random walks or restarts since our word selection process ensures diversification.

The second hybrid strategy was based on genetic algorithms (GA). A population is made of individuals, each individual being a split for each word of S. Each generation keeps a portion of individuals as parents and creates children by crossing (well-known uniform crossover) the selected parents. For each word, children inherit the prefix and the suffix of one of their parents randomly chosen. The population size is constant during all the search. Each individual $w = p.s$ is applied a mutation process (i.e., a new split of w into p' and s') with a probability p_{mut}. The search stops when the maximum number of generations is reached or when no improvement is observed in the population during a given time. The corresponding model is noted $GA_k(rand, f)$.

2.2. New hybrid models

We now propose 4 new hybrid models. In [1], we have noticed that $ILS_k(rand, f)$ and $GA_k(rand, f)$, produce smaller instances, but the time for generating an instance with $GA_k(rand, f)$ is too long compared to the gain in solving time. For $ILS_k(rand, f)$, generation is much faster, but remain costly. The first two models thus focus on optimizing the splitting of words in a deterministic manner.

The spatial complexity of the S model is quite bad. Thus, the idea of the **Best suffix model** is to optimize the construction of the suffixes. The Best suffix model consists in ordering the set $Suf(S)$ of suffixes of words of S. For each suffix s of $Suf(S)$, we consider $\Omega(s)$, the set of words of S accepting s as a suffix:

$$\Omega(s) = \{w \in S \mid s \in Suf(w)\}$$

We can now define the order $\succcurlyeq$ as follows. Consider s_1 and s_2, two suffixes of $Suf(S)$, then:

$$s_1 \succcurlyeq s_2 \quad \Leftrightarrow \quad |s_1| * |\Omega(s_1)| \geq |s_2| * |\Omega(s_2)|$$

with $|.|$ being both the length of a word and the cardinal of a set. The set of best suffixes, $\mathcal{S}$, is defined as follows: $\mathcal{S} \subseteq Suf(S)$ such that

- $\bigcup_{s\in\mathcal{S}} \Omega(s) = S$, each word of S has a suffix in $\mathcal{S}$;
- $\forall s' \in Suf(S) \setminus \mathcal{S},\ \forall s \in \mathcal{S},\ \ s \succcurlyeq s'$, i.e., suffixes of $\mathcal{S}$ are the most important ones regarding the $\succcurlyeq$ order;
- $\forall s' \in \mathcal{S}, (\bigcup_{s\in\mathcal{S}\setminus\{s'\}} \Omega(s)) \subset S$, i.e., if a suffix is removed from $\mathcal{S}$, at least a word of S has no suffix in $\mathcal{S}$.

For each word $w \in S$, we define the best suffix for w as $s_w^\star \in \mathcal{S} \cap Suf(w)$, and $\forall s \in Suf(w),\ s_w^\star \succcurlyeq s$, i.e., the suffix $s_w^\star$ we consider for w is the largest suffix possible w.r.t. $\succcurlyeq$ in $\mathcal{S}$. Hence, $\mathcal{S}$ is the set S_s of our $S^\star$ hybrid model and $S_p = \{p \in Pref(S) \mid \exists w \in S, w = s_w^\star.p\}$. The hybrid model $S_k^\star$ is then built with S_s and S_p has describe above for H_k.

The second model is the **Best prefix model**. This model is built in a similar way as the Best suffix model, starting with a selection of the best prefixes. It consists in first determining which are the best prefixes with respect to the

order $\succsim$ defined as above for prefixes: consider p_1 and p_2, two prefixes of $Pref(S)$, then:

$$p_1 \succsim p_2 \quad \Leftrightarrow \quad |p_1| * |\alpha(p_1)| \geq |p_2| * |\alpha(p_2)|$$

with $\alpha(p) = \{w \in S \mid p \in Pref(w)\}$. Then, similarly as with suffix, the set of best prefixes is built, and consequently, we define S_p and S_s that will be used in H_k to generate the constraints of the best prefix hybrid model $P_k^\star$.

The third model we propose, **the $ILS_k(S_k^\star, f)$ model** combines techniques from the Best suffix model and from the ILS model. The main difference with $ILS_k(rand, f)$ is the initial configuration. Indeed, we first compute word splitting of the $S_k^\star$ model to initialize the ILS algorithm. Note that we do not need to generate the $S_k^\star$ model itself, only the splitting is necessary. Hence, we know that the time for generating SAT instances with this model will be long, but we hope to obtain even smaller instances, and thus, we hope to be able to tackle even larger problems.

The last model, **the $ILS_k(P_k^\star, f)$ model**, is similar to the previous one, but the initial configuration is based on the word splitting of $P_k^\star$.

3. Over-constraining

In this section we propose some extra constraints, either to break symmetries, or to over-constrain the models. Note that these extra constraints are compatible with each of the previous models.

3.1. Symmetry Breaking for first derivations

A symmetric solution can be restored from another solution by some permutations. In constraint programming, symmetry breaking is a technique which consists in adding extra constraints for removing symmetric solution, and thus reducing the search space. This way, lost solutions can be recovered, and the problem remains satisfiable if it was before.

We propose some symmetry breaking that consist in enforcing the derivations from the state q_1. Remember that q_1 has a kind of special behavior when $\lambda \in S$. Consider a word $w = s_1 v$ from S^+ with $s_1 \in \Sigma$. Then, the following constraints can be added to remove symmetric NFA:

$$\delta_{s_1,\overrightarrow{q_1 q_1}} \vee \delta_{s_1,\overrightarrow{q_1 q_2}} \tag{12}$$

This means that we force to have at least a transition for symbol s_1 to be either from state q_1 to q_1 or from q_1 to q_2. For the second symbol, s_2, we can enforce to have a transition labeled with s_2 from q_1 to q_1, from q_1 to q_2, or from q_1 to q_3. The last possibility is required since we already have extra Constraints (12) over q_1 and q_2. Consider $\Sigma' = \{a_1, \ldots, a_l\}$, such that $\Sigma' \subseteq \Sigma$, and for each symbol $a_j \in \Sigma'$ there exists a word $w = a_j v$ in S^+, and for each $s \in \Sigma \setminus \Sigma'$, there does not exists a word $w = sv$ in S^+. For each $i \in [1..k-1] \cap [1..l]$, we consider the following extra constraints that break symmetries w.r.t. possibilities of derivations from state q_1:

$$\bigvee_{j \in [1..i+1]} \delta_{a_i,\overrightarrow{q_1 q_j}} \tag{13}$$

Figure 1 shows all possible NFA for instance st-2-10 ($|S^+| = |S^-| = 10$ and $\Sigma = \{0, 1\}$). Adding symmetry breaking constraint (13), only one solution, Sub-Figure (c), remains.

3.2. Over-constraining final states

We can also remove symmetric solutions by over-constraining final states with the following constraint:

$$f_1 \vee f_2 \tag{14}$$

However, Constraint (14) is not compatible with Constraints (13), and in our experiments, we will only consider Constraints (13) which are stronger.

Note that in [2], some extra constraints are added on final states: since they are computing in parallel, for a NFA of size k, they consider $2.k - 1$ different instances: each instance enforces to have a given number of final states, between 1 and k, including or not State q_1. It is clear that most of these instances are UNSAT even for a SAT instance. But thanks to parallelism, the gain in time is sensible.

3.3. Removing some undeterminism

We now over-constrain instances by removing some undeterminism. To this end, for each symbol, we consider that there is only one derivation from q_1. It is clear that we can remove complete groups of solutions, and thus, we can transform SAT instances into UNSAT instances. We thus have a semi-decision procedure, that can be helpful to quickly find SAT solutions, and lower upper bounds when the optimal k is not known (open instances). Consider Σ' defined as above. For each $i \in [1..k-1] \cap [1..l]$, we consider the following extra constraints:

$$\delta_{a_i,\overrightarrow{q_1 q_1}} \vee \delta_{a_i,\overrightarrow{q_1 q_{i+1}}} \tag{15}$$

$$\neg\delta_{a_i,\overrightarrow{q_1 q_1}} \vee \neg\delta_{a_i,\overrightarrow{q_1 q_{i+1}}} \tag{16}$$

$$\bigwedge_{j \in K \setminus \{1, i+1\}} \neg\delta_{a_i,\overrightarrow{q_1 q_j}} \tag{17}$$

4. Experimentations

Our algorithms are implemented in Python using specific libraries such as Pysat [16]. 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 [17] SAT solver with its default options.

For the stochastic processes ($ILS_k(rand, f)$, $ILS_k(P_k^\star, f)$, and $ILS_k(S_k^\star, f)$), 30 runs were realized. The maximum number of iterations is fixed to 10000 but

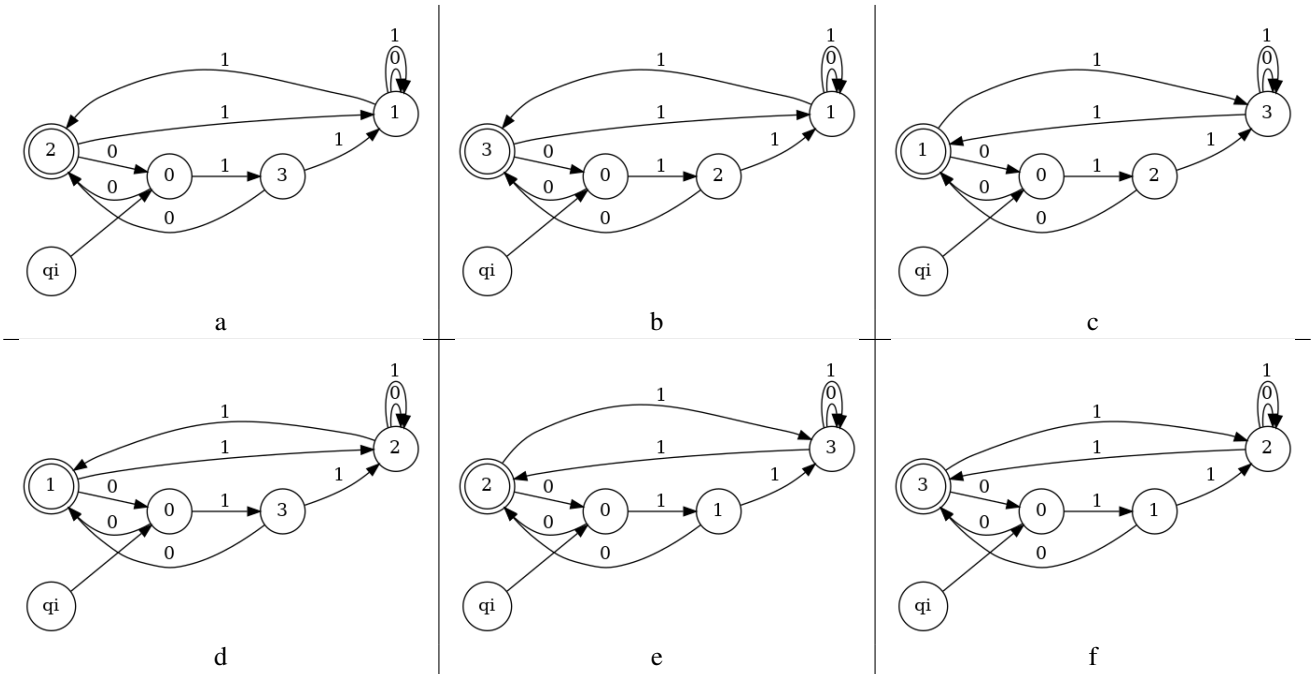

Figure 1. Instance st-2-10 with $\Sigma = \{0, 1\}$ and $k = 4$. Six NFA respect S and can be inferred, Sub-figures (a) to (f). With symmetry breaking (Constraints (13)), only NFA of Sub-figure (c) is valid.

if no improvement is obtained during 100 iterations, the search stops.

Experiments are realized on 30 benchmarks used in [2][1]. Some of them are based on the training set of the StaMinA Competition (http://stamina.chefbe.net) and the others were randomly generated. From the StaMinA competition, 10 benchmarks have an alphabet of size 2, and 10 others have an alphabet of size 5. The 10 random benchmarks have an alphabet of size 10. The numbers of positive and negative words ($|S^+|$, and $|S^-|$) vary depending on the benchmarks. All the characteristics of each benchmark are summarized in Table 1. When no optimal NFA size is known, the name is written in bold.

	StaMinA		ww
$\|S^+\| = \|S^-\|$	$\|\Sigma\| = 2$	$\|\Sigma\| = 5$	$\|\Sigma\| = 10$
10	st-2-10	st-5-10	ww-10-10
20	st-2-20	st-5-20	ww-10-20
30	st-2-30	st-5-30	ww-10-30
40	**st-2-40**	st-5-40	ww-10-40
50	**st-2-50**	st-5-50	ww-10-50
60	**st-2-60**	**st-5-60**	ww-10-60
70	**st-2-70**	**st-5-70**	ww-10-70
80	**st-2-80**	**st-5-80**	**ww-10-80**
90	**st-2-90**	**st-5-90**	**ww-10-90**
100	**st-2-100**	**st-5-100**	**ww-10-100**

TABLE 1. 30 BENCHMARKS USED WITH THEIR CHARACTERISTICS

1. The benchmarks we used are accessible on the GitHub repository https://github.com/tjastrzab/min-nfa

For each of the benchmarks, the NFA minimal size mv (or the lower bound for the NFA minimal size when the instance is still open) is given in [2]. We try to generate and solve CNF instances for $k \in \{mv - 1, mv, mv + 1\}$. Consequently, we obtained 90 instances.

5. Results and Discussions

In this section, we analyze and compare the results we obtained.

5.1. Comparative Results

Each of the 90 instances is tested with each of the SAT models presented above, and one of the following options: without symmetry breaking, with symmetry breaking (Extra Constraints (13)), or over-constraining (Extra Constraints (15)–(17)). In total, this leads to 2160 combinations.

Some instances, 26, cannot be generated by any of the models in the given time. We discard these 26 instances, and thus propose results for the 64 remaining instances.

We summarize all these results in Table 2 which presents the average or the sum for various indicators. The first column (Model) corresponds to the selected model. The second column (Sym.) indicates the symmetry breaking option. Three values are possible: "no" for no symmetry breaking, "yes" for symmetry breaking with Constraint (13), and "OC" for the Over-Constraints (15–17). Then, we have in sequence the total number of generated SAT instances in the allocated time (#Gen.), the average number of SAT variables ($Var.$) and the average number of clauses ($Cl.$)

Model	Sym.	#Gen.	Var.	Cl.	t_M	#Sol.	t_S	t_T	t_P
D_k	no	15	15654	134 009	2.51	14	0.08	2.25	469.24
	yes	15	15654	134 011	2.52	14	0.22	2.41	469.28
	OC	15	15654	134 018	2.55	14/0/0	0.04	2.26	469.25
$ILS_k(rand, f)$	no	61	11813	44 003	7.81	**46**	58.76	65.82	**216.06**
	yes	61	11827	44 060	7.74	**48**	58.16	65.41	199.06
	OC	61	11814	44 031	7.74	**58/2/1**	46.21	53.81	105.01
P_k	no	61	10641	37 616	1.75	42	16.92	18.57	218.43
	yes	61	10641	37 619	1.73	45	27.09	28.74	198.33
	OC	61	10641	37 637	1.72	56/2/1	47.22	48.92	117.80
$P_k^\star$	no	61	47842	173 800	2.35	37	22.90	24.01	267.00
	yes	61	47842	173 803	2.35	39	42.19	41.91	259.91
	OC	61	47842	173 821	2.33	48/1/2	41.04	42.97	182.22
$ILS_k(P_k^\star, f)$	no	61	11892	44 307	7.77	45	55.99	62.99	222.41
	yes	61	11879	44 262	7.74	44	26.42	33.41	210.47
	OC	61	11832	44 106	7.71	57/2/1	41.07	48.61	108.92
S_k	no	61	67597	243 066	2.89	39	42.60	44.61	261.56
	yes	61	67597	243 069	2.88	40	29.23	31.22	244.52
	OC	61	67597	243 087	2.86	42/2/0	15.38	17.41	217.67
$S_k^\star$	no	61	9984	37 164	1.62	43	33.38	33.48	219.37
	yes	61	9984	37 167	1.62	46	26.77	28.33	**189.11**
	OC	61	9984	37 186	1.62	57/3/0	23.15	24.75	**87.67**
$ILS_k(S_k^\star, f)$	no	61	9237	34 668	7.68	45	49.82	56.75	218.03
	yes	61	9232	34 653	7.66	46	47.61	53.66	207.32
	OC	61	9231	34 668	7.66	57/2/0	20.41	27.89	90.47

TABLE 2. COMPRESSED RESULTS FOR EXPERIMENTS ON ALL INSTANCES WITH 3 DIFFERENT VALUES AND AND THE PRESENTED MODELS WITH AND WITHOUT SYMMETRY BREAKING OPTIONS.

in the instance, and the average instance generation time (t_M). The middle part of the table corresponds to the solving part with the total number of solved instances (#Sol.), and the average solving time (t_S) with Glucose. Finally, the right part is made up of t_T and t_P which are the average total times, i.e., modeling time + solving time, without and with penalties. For each unsolved instance a penalty of 600 seconds is applied in order to correspond to the effective time which has been consumed.

Note also that for column #Sol., lines corresponding to the over-constraint symmetry breaking (OC) indicate 3 values. The first one indicate the number of solved instances, the second one is the number of errors (SAT instances that became UNSAT by adding the over-constraints), and finally, the number of SAT instances that were not found by the other combinations Model/Sym.

Over the 64 instances of our benchmark tests, each model allowed to generate 61 SAT instances in the allowed time, except the D_k model which allowed to generate only 15 SAT instances. One of these 15 instances was also too complex to be solved before the time-out. Among the 4 new hybrid models proposed, all improve the number of solved instances compared to the previously published models, except the $P_k^\star$ model. This is certainly due to the size of the generated instances. Indeed, $P_k^\star$ optimizes the prefix management, but we have shown that the suffix treatment has a higher complexity. On the contrary, models optimizing the treatment of suffixes ($S_k^\star$ and $ILS_k(S_k^\star, f)$) produce instances with fewer variables.

Model $ILS_k(rand, f)$ solves slightly more instances than other models. However, the model that generates SAT instances that are solved the fastest, is the $S_k^\star$ model. Moreover, the stochastic aspects of the ILS model requires to perform several executions that are then averaged to obtain the results. The standard deviation is quite high and the solving times usually varies from simple to double. For instance, benchmark st-5-30 with $k = 5$ generated by $ILS_k(S_k^\star, f)$ is solved in average in 15.23 seconds but the standard deviation is 12.32. The shortest solving time is 2.40 seconds and the highest solving time is 33.49 seconds. Whereas solving time for instances generated with the $S_k^\star$ model is constant, solving time for the $ILS_k(*, f)$ models is not homogeneous at all.

We note that adding symmetry breaking constraints only slightly increases the number of clauses of the generated instances (the number of variables remains the same, except for models using stochastic search based on ILS). This low number of extra clauses, however, significantly improves solving time of the instances.

Over-constrain instances obtained by removing undeterminism provide better results (#Sol. and t_P) but the only exploitable results are those returning SAT. However, given the number of solved instances, it seems relevant and appropriate to consider this approach. For example, in the next section, we use it for decreasing upper bounds of open instances.

5.2. Bounds

As shown in the previous section, our new models allowed us to obtain solutions for open instances in a reasonable time. In order to improve the best known values for the 30 benchmarks proposed in [2], we allowed a running time of 30 minutes to generate and solve instances using the $S_k^\star$ model with symmetry breaking and over-constraining. Table 3 summarizes the results we obtained. The first column, $|S^+| = |S^-|$, corresponds to the number of positive and negative words of the benchmarks. Then, for each family of benchmarks (i.e., StaMinA with alphabet of size 2 (st-2), and alphabet of size 5 (st-5), and random benchmarks with alphabet of size 10 (ww-10)), we have two columns BK and NB. Columns BK correspond to the best known bounds as reported in [2]. Columns NB correspond to the new bounds: lower bounds were obtained by the $S_k^\star$ model and symmetry breaking (13), and upper bounds were obtained by the model $S_k^\star$ with either symmetry breaking (13) or over-constraints (15-17). Results are represented as intervals $[x-y]$ for which UNSAT is found for $k = x - 1$ and SAT for $k = y$. When two consecutive values of k provide respectively UNSAT and SAT, the optimal solution is found: hence, there is only one value written in the cell, the minimal k for which there exists a NFA for the benchmark.

	st-2		st-5		ww-10	
$\|S^+\| = \|S^-\|$	BK	NB	BK	NB	BK	NB
10	4	4	3	**2**	2	2
20	7	7	4	4	3	3
30	9	9	4	4	3	**4**
40	[9-11]	[9-**10**]	5	**6**	4	4
50	[9-12]	[**10-11**]	6	6	4	4
60	[9-15]	[9-**13**]	[6-8]	[6-8]	5	5
70	[9-17]	[9-**13**]	[7-8]	[7-8]	5	5
80	[9-18]	[9-**16**]	[7-9]	[7-9]	[5-6]	[5-6]
90	[9-22]	[9-**16**]	[7-11]	[7-11]	[5-6]	[5-6]
100	[9-26]	[9-**20**]	[8-10]	[8-10]	[5-7]	[5-6]

TABLE 3. COMPARISONS BETWEEN BEST KNOWN BOUNDS (BK) AND NEW BOUNDS OBTAINED (NB) WITH THE MODEL $S_k^\star$.

New results are in bold in Table 3. We can see that we succeeded in reducing the range of some open instances. Concerning 3 other instances, we feel that some typos appeared in the table given in [2]:

- st-5-10 has a solution with $k = 2$. A NFA solution is given in Figure 2,
- st-5-40 has no solution for $k = 5$, and the smallest k is 6,
- and ww-10-30 has no solution for $k < 4$, the smallest k being 4.

For all StaMinA benchmarks with $|\Sigma| = 2$ (st-2), we have reduced the upper bounds of the intervals. For st-2-50 we have also increased the lower bound.

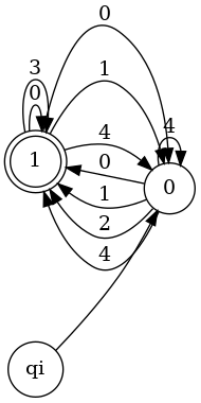

Figure 2. NFA with $k = 2$ for instance st-5-10.

6. Conclusion

In this paper, we have presented four new SAT models for inferring NFA of given size k. We have completed these models with some extra constraints to over-constrain the problem, either just for breaking symmetries, or loosing solutions. The new models allow us to generate smaller instances than previously [1]. Symmetry breaking reduces the search space, and help finding solutions. Over-constraining permits to find solutions to more instances than before. To summarize, the best model without symmetry breaking is the $ILS_k(rand, f)$ model, with symmetry breaking, $S_k^\star$, and for tackling more complex instances, $S_k^\star$ with over-constraints. Compared to the parallel approach of [2], we managed to improve some bounds of instances for which the minimal size is not yet known.

Our implementation is able to generate the SAT instances, but also to generate the equivalent INLP (variables in the domain $\{0, 1\}$, and non-linear equations and inequations) instances for each model, using PyCSP3 [18]. The results we obtained were very bad compared to the SAT instances, and that is why we do not report about them in this article. However, we plan to design hybrid models especially for optimizing INLP constraints (for example, trying to minimize the number of non-linear equations and favoring linear ones) with ILS, or a complete order, and compare them with SAT hybrid models. NFA models present numerous symmetries. We proposed some symmetry breaking techniques in this paper, but we feel that more symmetries can be broken to reduce even more the search space.

References

[1] F. Lardeux and E. Monfroy, "GA and ILS for optimizing the size of NFA models," in *The 8th International Conference on Metaheuristics and Nature Inspired Computing (META)*, Marrakech, Morocco, Oct. 2021. [Online]. Available: https://hal.univ-angers.fr/hal-03284541

[2] T. Jastrzab, Z. J. Czech, and W. Wieczorek, "Parallel algorithms for minimal nondeterministic finite automata inference," *Fundam. Informaticae*, vol. 178, no. 3, pp. 203–227, 2021. [Online]. Available: https://doi.org/10.3233/FI-2021-2004

[3] J. Oncina and P. García, *Inferring Regular Languages In Polynomial Updated Time*, pp. 49–61.

[4] M. Heule and S. Verwer, "Software model synthesis using satisfiability solvers," *Empirical Software Engineering*, vol. 18, no. 4, pp. 825–856, 2013.

[5] F. Denis, A. Lemay, and A. Terlutte, "Learning regular languages using rfsas," *Theor. Comput. Sci.*, vol. 313, no. 2, pp. 267–294, 2004. [Online]. Available: https://doi.org/10.1016/j.tcs.2003.11.008

[6] F. Coste and D. Fredouille, "Unambiguous automata inference by means of state-merging methods," in *Machine Learning: ECML 2003, 14th European Conference on Machine Learning, Cavtat-Dubrovnik, Croatia, September 22-26, 2003, Proceedings*, ser. Lecture Notes in Computer Science, N. Lavrac, D. Gamberger, L. Todorovski, and H. Blockeel, Eds., vol. 2837. Springer, 2003, pp. 60–71.

[7] M. Tomita, "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.

[8] P. Dupont, "Regular grammatical inference from positive and negative samples by genetic search: the GIG method," in *Proc. of ICGI 94*, ser. LNCS, vol. 862. Springer, 1994, pp. 236–245.

[9] F. Rossi, P. van Beek, and T. Walsh, Eds., *Handbook of Constraint Programming*, 1st ed. Elsevier Science, 2006.

[10] W. Wieczorek, *Grammatical Inference - Algorithms, Routines and Applications*, ser. Studies in Computational Intelligence. Springer, 2017, vol. 673.

[11] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*. San Francisco: W.H. Freeman & Company, 1979.

[12] F. Lardeux and E. Monfroy, "Improved SAT models for NFA learning," in *International Conference in Optimization and Learning (OLA)*, Catania, Italy, Jun. 2021. [Online]. Available: https://hal.univ-angers.fr/hal-03284571

[13] T. Jastrzab, "A comparison of selected variable ordering methods for NFA induction," in *Proc. of ICCS 2019*, ser. LNCS, vol. 11540. Springer, 2019, pp. 741–748.

[14] G. S. Tseitin, *On the Complexity of Derivation in Propositional Calculus*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1983, pp. 466–483.

[15] T. Stützle and R. Ruiz, *Iterated Local Search*. Cham: Springer International Publishing, 2018, pp. 579–605. [Online]. Available: https://doi.org/10.1007/978-3-319-07124-4_8

[16] A. Ignatiev, A. Morgado, and J. Marques-Silva, "PySAT: A Python toolkit for prototyping with SAT oracles," in *SAT*, 2018, pp. 428–437. [Online]. Available: https://doi.org/10.1007/978-3-319-94144-8_26

[17] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern SAT solvers," in *Proc. of IJCAI 2009*, 2009, pp. 399–404.

[18] C. Lecoutre and N. Szczepanski, "PYCSP3: modeling combinatorial constrained problems in python," *CoRR*, vol. abs/2009.00326, 2020. [Online]. Available: https://arxiv.org/abs/2009.00326