Inference of over-constrained NFA of size k + 1 to efficiently and systematically derive NFA of size k for grammar learning

Tomasz Jastrząb, Frédéric Lardeux, Eric Monfroy

Inference of over-constrained NFA of size $k+1$ to efficiently and systematically derive NFA of size k for grammar learning

Tomasz Jastrząb[1][0000−0002−7854−9058], Frédéric Lardeux[2][0000−0001−8636−3870], and Eric Monfroy[2][0000−0001−7970−1368]

[1] Silesian University of Technology
Poland
[2] University of Angers, LERIA
France
Tomasz.Jastrzab@polsl.pl
{Frederic.Lardeux,Eric.Monfroy@}@univ-angers.fr

Abstract. Grammatical inference involves learning a formal grammar as a finite state machine or set of rewrite rules. This paper focuses on inferring Nondeterministic Finite Automata (NFA) from a given sample of words: the NFA must accept some words, and reject others. Our approach is unique in that it addresses the question of whether or not a finite automaton of size k exists for a given sample by using an over-constrained model of size $k+1$. Additionally, our method allows for the identification of the automaton of size k when it exists. While the concept may seem straightforward, the effectiveness of this approach is demonstrated through the results of our experiments.

Keywords: grammatical inference · nondeterministic automata · SAT models.

1 Introduction

Grammatical inference [4] is the process of learning formal grammars, such as finite automata or production rules, from a given learning sample of words. This method is useful in various fields, such as compiler design, bioinformatics, pattern recognition, and machine learning. The problem we tackle is to learn a finite automaton, specifically a Nondeterministic Finite Automaton (NFA), that can accept a set of positive examples and reject a set of negative examples. The complexity of this problem is determined by the number of states in the automaton. Nondeterministic automata are often smaller in size than deterministic automata for the same language, thus the focus is on learning NFAs. The goal is to minimize the number k of states in the automaton—this is typically done by determining lower (such as 1) and upper bounds (such as the size of a prefix

tree acceptor[3]) on the number of states, and using optimization algorithms to find the smallest possible number of states.

The problem of learning formal grammars from a given sample of words has been explored from multiple angles. Several algorithms have been proposed, including ad-hoc methods such as DeLeTe2 [2] that focuses on merging states from the prefix tree acceptor (PTA), and newer approaches like the family of algorithms for regular languages inference presented in [18]. Some studies have employed metaheuristics, such as hill-climbing in [16], while others have used complete solvers that can always find a solution if one exists, prove unsatisfiability, and find the global optimum in optimization problems. The problem is often modeled as a Constraint Satisfaction Problem (CSP) and various techniques have been employed, such as Integer Non-Linear Programming (INLP) in [19] and parallel solvers in [6, 7]. Additionally, the author of [8] proposed two strategies for solving the CSP formulation of the problem, and [9] presents a parallel approach for solving the optimization variant of the problem.

In this paper, we aim to enhance SAT models for grammatical inference, as opposed to creating a new solver. A SAT (the propositional SATisfiability problem [3]) model consists in defining the problem with Boolean variables and a Boolean formula in Conjunctive Normal Form (CNF). More precisely, we focus here on the definition of over-constrained models of size $k+1$ with properties that allow deducing information about the classical model of size k. The benefit of these over-constrained models is in terms of spacial complexity. Whereas the complexity of the generation of an NFA of size k of a learning sample S is in $\mathcal{O}(\sigma \cdot k^3)$ variables, and $\mathcal{O}(\sigma \cdot k^3)$ clauses (with $\sigma = \Sigma_{w \in S}|w|$ for most of the models[4]), the generation of our over-constrained NFA of size $k+1$ is in $\mathcal{O}(\sigma \cdot k^2)$ variables, and $\mathcal{O}(\sigma \cdot k^2)$ clauses.

We made some experiments to test our method. The originality is that we use some of these properties to derive k-state NFAs by building more constrained models of size $k+1$, where the previous model of size k was unable to provide a solution.

The structure of this paper is as follows. Section 2 gives an overview of the NFA inference problem. Section 3 describes extensions of the classical models. In Section 4, we present some properties concerning the extensions. The results of our experiments are discussed in Section 5 and we conclude in Section 6.

2 The NFA inference problem

In this section, we formally introduce the NFA inference problem by utilizing the propositional logic paradigm and providing a generic model. Additionally, we review and highlight some previously established models within the current literature.

[3] A prefix tree acceptor (PTA) is a tree-like Deterministic Finite Automaton built from the sample by using each prefix in the sample as a state

[4] Only the complexity of the prefix model is in $\mathcal{O}(\sigma \cdot k^2)$ variables, and $\mathcal{O}(\sigma \cdot k^2)$ clauses.

2.1 Notations

Let $\Sigma = \{a_1, \ldots, a_n\}$ be an alphabet of n symbols. A learning sample $S = S^+ \cup S^-$ is given by a set S^+ of so called "positive" words from Σ^* that the inferred NFA must accept, and a set S^- of "negative" words that the NFA must reject.

Let K be the set of the k first non-zero integers, $K = \{1, \ldots, k\}$. We consider the following variables:

- k, an integer, the size of the NFA to be generated,
- a set of k Boolean variables $F = \{f_1, \ldots, f_k\}$ determining whether state i is final or not,
- and, $\Delta = \{\delta_{a,\overrightarrow{i,j}} | a \in \Sigma \text{ and } (i,j) \in K^2\}$, a set of nk^2 Boolean variables representing the existence of transitions from state i to state j with the symbol $a \in \Sigma$, for each i, j, and a.

We define $p_{w,\overline{i_1,i_{m+1}}}$ as the path $i_1, i_2, \ldots, i_{m+1}$ for a word $w = a_1 \ldots a_m$.

$$p_{w,\overline{i_1,i_{m+1}}} = \delta_{a_1,\overrightarrow{i_1,i_2}} \wedge \ldots \wedge \delta_{a_m,\overrightarrow{i_m,i_{m+1}}}$$

Although the path is directed from i_1 to i_{m+1} (it is a sequence of derivations), we will build it either starting from i_1, starting from i_{m+1}, or starting from both sides. Thus, to avoid confusion of path and building, we prefer keeping $\overline{i_1, i_{m+1}}$ without any direction.

2.2 A "meta-model"

A meta-model to define an NFA of size k (noted k_NFA) can be done with the 3 following constraints:

- Special cases for the empty word λ; if it is a word of S^+, the initial state must be final, if it is a negative word, the initial state must not be final:

$$(\lambda \in S^+ \implies f_1) \quad \wedge \quad (\lambda \in S^- \implies \neg f_1) \tag{1}$$

- A positive word must terminate on a final state of the k_NFA, i.e., there must be a path from the initial state 1 to a final state i (f_i must be true):

$$\bigvee_{i \in K} p_{w,\overline{1,i}} \wedge f_i \tag{2}$$

- A negative word w must not terminate on a final state of the k_NFA, i.e., either there is no path for w, or each path terminates in a non-final state:

$$\bigwedge_{i \in K} (\neg p_{w,\overline{1,i}} \vee \neg f_i) \tag{3}$$

Of course, the notion of path can be defined and built in many ways. In [10], prefix, suffix, and hybrid approaches are proposed.

2.3 Some previous models

As seen in the previous section, several models can be considered for learning an NFA. We skip the direct model (see [9, 11]) which has a bad complexity and does not behave well in practice: its space complexity is in $\mathcal{O}(|S^+| \cdot (|\omega_+| + 1) \cdot k^{|\omega_+|})$ clauses, and $\mathcal{O}(|S^+| \cdot k^{|\omega_+|})$ variables with ω_+ the longest word of S^+. We also discard models with 0/1 variables, either from INLP [19] or CSP [14]: we made some tests with various models with [13] and obtained some disastrous results: the NFA inference problem is intrinsically a Boolean problem, and thus, well suited for SAT solvers.

In [12], 7 different models were defined: the prefix model (P), the suffix model (S), and several hybrid models combining prefix and suffix models with different splitting strategies of words, such as the best prefix model ($P^\star$) to optimize size and use of prefixes, the best suffix model ($S^\star$) to optimize size and use of suffixes, and 3 hybrid models ($ILS(Init)$) based on a local search optimization [15] of word splittings (starting with an initial configuration $Init$, being either a random splitting of words, the splitting found by the $P^\star$ model, or by the $S^\star$ model).

The main difference is the definition and construction of paths. For example, in the prefix model (P), some extra Boolean variables represent paths for each prefix of the learning sample. Thus, paths are defined with the following two constraints: (each word is represented as $w = va$ with $a \in \Sigma$):

- Case for a word of length 1:

$$\bigvee_{i \in K} \delta_{a,\overrightarrow{1,i}} \leftrightarrow p_{a,\overline{1,i}} \tag{4}$$

- Recursive definition of paths for each prefix w of each word of the sample:

$$\bigwedge_{i \in K} (p_{w,\overline{1,i}} \leftrightarrow (\bigvee_{j \in K} p_{v,\overline{1,j}} \wedge \delta_{a,\overrightarrow{j,i}})) \tag{5}$$

The suffix model (S) is obtained similarly, but with extra variables representing suffixes (i.e., with words represented as $w = av$ with $a \in \Sigma$).

Hybrid models are built with both prefixes and suffixes constructions, each word being considered as the concatenation of a prefix and a suffix. Some constraints are also added to "join" prefix paths and suffix paths.

After transformation in Conjunctive Normal Form (CNF) using Tseitin transformations [17], the spacial complexity of the prefix model 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 [11] for details). Although similar, the spacial complexity of the suffix model is in $\mathcal{O}(\sigma \cdot k^3)$ variables and $\mathcal{O}(\sigma \cdot k^3)$ clauses. The reason is that we build prefixes from the initial state 1, whereas suffixes are built from one of the k states (see [11]). Hybrid models are thus also in $\mathcal{O}(\sigma \cdot k^3)$ variables and clauses.

3 *k*_NFA extensions

For a given sample, if there is a k_NFA, i.e., an NFA of size k, to recognize words of S^+ and reject words of S^-, there is also an NFA of size $k+1$. Although

obvious and rather useless, this property can be refined to generate k_NFA. To this end, some more constraints are added to the $(k+1)$_NFA to build what we call $(k+1)$_NFA extensions.

3.1 Buiding a $(k+1)$_NFA from a k_NFA

Let $A = (Q^A, \Sigma, \Delta^A, q_1, F^A)$ be an NFA of size k. Then, there always exists an NFA of size $k+1$, $A' = (Q^{A'}, \Sigma, \Delta^{A'}, q_1, F^{A'})$, such that $Q^{A'} = Q^A \cup \{q_{k+1}\}$, $F^{A'} = \{q_{k+1}\}$ and $\Delta^{A'}$:

$$\forall_{i,j \in (Q^A)^2} \ \delta^A_{a,\overrightarrow{i,j}} \leftrightarrow \delta^{A'}_{a,\overrightarrow{i,j}}$$
$$\forall_{i \in Q^A, j \in F^A} \ \delta^A_{a,\overrightarrow{i,j}} \leftrightarrow \delta^{A'}_{a,\overrightarrow{i,k+1}}$$

In other words, A' has only one final state which is the new state $k+1$, each transition of A also exists in A', and transitions from a state i to a final state of A are duplicated as new transitions from i to state $k+1$. The obvious but important property for the rest of this paper is that the language recognized by A' is the same as the one recognized by A.

In the following, the main idea is to over-constrain a $(k+1)$_NFA model (i.e., a model to generate an NFA of size $k+1$) to obtain a model closer or equal to A' as described above. Moreover, the idea is also that the over-constrained $(k+1)$_NFA model can be solved and reduced to an NFA of size k more efficiently than solving directly the k_NFA model. We propose two $(k+1)$_NFA model extensions for which a new state and new constraints are added. The first model extension, the $(k+1)$_NFA$^+$, over-constrains the $(k+1)$_NFA to be able to reduce a generated $(k+1)$_NFA into a k_NFA with a reduction algorithm (see [10]). However, as shown in [10], we are not always able to reduce a $(k+1)$_NFA$^+$ into a k_NFA, and the gain is really poor. In fact, a $(k+1)$_NFA$^+$ instance mainly provides information when it is unsatisfiable: in this case, we also know that there is no k_NFA. The second model extension, the $(k+1)$_NFA* model, does not require any algorithm to reduce a generated $(k+1)$_NFA to a $(k+1)$_NFA: the satisfiability (respectively unsatisfiability) of a $(k+1)$_NFA* implies the satisfiability (respectively unsatisfiability) of a k_NFA: moreover, in case of satisfiability, the k_NFA can always be directly derived from the $(k+1)$_NFA* by removing a state and some transitions. This operation has no cost.

3.2 $(k+1)$_NFA$^+$ extension

Let $K = \{1, \ldots, k\}$ be the k first non-zero integers, and $K_+ = \{1, \ldots, k+1\}$ be the $k+1$ first non-zero integers.

A $(k+1)$_NFA$^+$ is a $(k+1)$_NFA with some extra properties that ensure that it may be reduced to a k_NFA by a reduction algorithm [10]. The extra properties of a $(k+1)$_NFA$^+$ are:

- a $(k+1)$_NFA$^+$ has one and only one final state, i.e., state $k+1$;

- state $k+1$ has no outgoing transition;
- each transition from a state i to state $k+1$ reading the symbol a has an equivalent transition from the state i to a state j ($j \neq k+1$) with a. State i is called a possibly final state.

A $(k+1)$_NFA$^+$ is defined by the same variables as a k_NFA with an addition of those related to state $k+1$ (a variable for the final state, transitions, and paths). The extra constraints are the following:

- The $(k+1)$_NFA$^+$ has only one final state, state $k+1$:

$$\bigwedge_{i \in K} (\neg f_i) \wedge f_{k+1} \tag{6}$$

  Note that this constraint mainly impacts the suffix model by unit propagation.
- There is no outgoing transition from the $(k+1)$_NFA$^+$ final state:

$$\bigwedge_{a \in \Sigma} \bigwedge_{i \in K_+} \neg \delta_{a,\overrightarrow{k+1,i}} \tag{7}$$

- Each incoming transition of the $(k+1)$_NFA$^+$ final state $k+1$ must also finish in another state:

$$\bigwedge_{a \in \Sigma} \left(\bigwedge_{i \in K} \left(\delta_{a,\overrightarrow{i,k+1}} \rightarrow \bigvee_{j \in K} \delta_{a,\overrightarrow{i,j}} \right) \right) \tag{8}$$

3.3 k_NFA* extension

A $(k+1)$_NFA* is a $(k+1)$_NFA$^+$ with some extra properties on words and a new set of Boolean variables representing possibly final states for the corresponding k_NFA ($F^* = \{f_1^*, ..., f_k^*\}$). $(k+1)$_NFA* may be reduced to a k_NFA by removing state $k+1$ and its incoming transitions, and fixing the final states among the possible final states, i.e., determining the f_i^* of $\{f_1^*, \ldots, f_k^*\}$ which are final states of the k_NFA. To determine these final states, we have to ensure:

- A negative word cannot terminate in a possible final state:

$$\bigwedge_{i \in K} \left(f_i^* \rightarrow \bigwedge_{w \in S^-} \neg p_{w,\overrightarrow{1,i}} \right) \tag{9}$$

- Each possibly final state validates at least one positive word of S^+:

$$\bigwedge_{i \in K} \left(f_i^* \rightarrow (\bigvee_{va \in S^+} \bigvee_{j \in K} (p_{v,\overrightarrow{1,j}} \wedge \delta_{a,\overrightarrow{j,i}} \wedge \delta_{a,\overrightarrow{j,k+1}}) \right) \tag{10}$$

- Each positive word terminates in at least one possible final state:

$$\bigwedge_{w \in S^+} \bigvee_{i \in K} (p_{w,\overrightarrow{1,i}} \wedge f_i^*) \tag{11}$$

3.4 Complexity

Extensions $(k+1)\_\text{NFA}^{+}$ and $(k+1)\_\text{NFA}^{\star}$ are of course bigger than $k_$NFA since the corresponding models contain an extra state, some extra transitions, and some extra constraints. $(k+1)\_\text{NFA}^{\star}$ model needs also k new variables for the possible final states. New constraints to transform $(k+1)_$NFA to the extensions also increase the number of clauses and variables. Table 1 provides the cost in terms of variables and clauses. Some details about the arity of the generated clauses are given (unary, binary, and greater). A blank in a cell corresponds to 0.

Table 1. Complexity in terms of variables and clauses for each new constraint allowing the definition of the $(k+1)\_\text{NFA}^{+}$ (Constraints (6–8)) and the $(k+1)\_\text{NFA}^{\star}$ models (Constraints (6–11)).

	Variables	Clauses			
		total	unary	binary	$>$
Constraint 6		$k+1$	$k+1$		
Constraint 7		$n(k+1)$	$n(k+1)$		
Constraint 8		nk			nk
Constraint 9		$k\|S^-\|$		$k\|S^-\|$	
Constraint 10	$k\|S^+\|$	$k(k+4)\|S^+\|$		$k^2\|S^+\| + 4k\|S^+\|$	$k\|S^+\|$
Constraint 11	$k\|S^+\|$	$(3k+1)\|S^+\|$		$2k\|S^+\|$	$(k+1)\|S^+\|$

Both extensions increase the number of variables and clauses of the initial $(k+1)_$NFA model. We can observe a lot of unary and binary clauses allowing solvers to perform very well. The global complexity for each extension remains unchanged or decreases, despite the increase in clauses and variables. As we will observe in Section 5.2, the implementation of these new constraints significantly lowers the complexity of many of the models ($\mathcal{O}(\sigma \cdot k^3)$ clauses to $\mathcal{O}(\sigma \cdot (k+1)^2)$ clauses) just by simplification.

4 Properties of the extensions

Extensions $(k+1)\_\text{NFA}^{+}$ and $(k+1)\_\text{NFA}^{\star}$ are over-constrained models for $(k+1)_$NFA. Indeed, some properties allow them to infer a $k_$NFA solution. A k_NFA^{+} is a "weak" extension because only the existence of a $k_$NFA can be proved, whereas $k\_\text{NFA}^{\star}$ is a "strong" extension that proves the existence or not of a $k_$NFA.

4.1 $(k+1)\_\text{NFA}^{+}$

Major property Adding Constraints (6), (7), and (8) to $k_$NFA model leads to the following property:

$$\exists\ k\_\text{NFA} \Rightarrow \exists\ (k+1)\_\text{NFA}^{+} \tag{12}$$

The contraposition allows us to prove the unsatisfiability of k_NFA with $(k+1)$_NFA$^+$:

$$\nexists\ (k+1)\text{\_NFA}^+ \Rightarrow \nexists\ k\text{\_NFA} \tag{13}$$

Proof. The main idea is: there always exists a transformation from any k_NFA to a $(k+1)$_NFA$^+$ which recognizes the same language.

1. A new state $k+1$ is added to the k_NFA.
2. Each of incoming transitions of k_NFA final states are duplicated by transitions outgoing to state $k+1$. Thus, a path $p_{w,\overline{1,k+1}}$ exists from the initial state to state $k+1$ if and only if a path exists from the initial state to k_NFA final states.
3. This induces:
   - Each positive word can now finish in state $k+1$,
   - No negative word can finish in state $k+1$.
4. All k_NFA final states are then redundant with state $k+1$ thus state $k+1$ can be considered as the unique final state.
5. The automaton is then a $(k+1)$_NFA$^+$ since it has only one final state, and it validates positive words and rejects negative words.

Minor property While the lack of solutions for $(k+1)$_NFA$^+$ allows to conclude the lack of solutions for k_NFA, a solution for $(k+1)$_NFA$^+$ is not sufficient to conclude existence of a k_NFA:

$$\exists\ (k+1)\text{\_NFA}^+ \not\Rightarrow \exists\ k\text{\_NFA} \tag{14}$$

The simple following example (Figure 1) shows a k_NFA$^+$ solution with $k=3$, whereas there is no k_NFA with $k=2$.

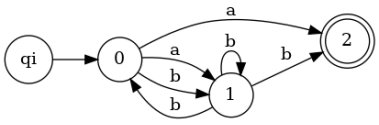

Fig. 1. Example of a $(k+1)$_NFA$^+$ solution for $k=3$ and $S = (\{a, ab, abb, bbb\}, \{aab, b, ba, bab, aaa\})$. It is not possible to find a k_NFA with $k=2$.

It is possible to try to obtain a k_NFA from a $(k+1)$_NFA$^+$ solution with the algorithm presented in [10]. However this algorithm rarely succeeds, and its worst-case complexity is in $O(k \cdot |S|)$.

4.2 $(k+1)$_NFA*

Major property Adding Constraints (9), (10), and (11) to $(k+1)$_NFA$^+$ model allows to obtain the equisatisfiability between k_NFA and $(k+1)$_NFA*:

$$\exists\ k\text{\_NFA} \equiv \exists\ (k+1)\text{\_NFA}^\star \tag{15}$$

Proof. The main idea is: there are some transformations from k_NFA to $(k+1)$_NFA* and from $(k+1)$_NFA* to k_NFA preserving acceptance of positive words and rejections of negative words.

$\Rightarrow$ From k_NFA to $(k+1)$_NFA*
1. It is similar to the proof of Section 4.1.

$\Leftarrow$ From $(k+1)$_NFA* to k_NFA
1. All transitions $\delta_{a,\overrightarrow{i,k+1}}$ are removed, and states j such that $\delta_{a,\overrightarrow{i,k+1}} \wedge \delta_{a,\overrightarrow{i,j}}$ are now considered as final states.
2. State $k+1$ is removed.
3. Each positive word can terminate in a possible final state because there are some transitions to a possible final state redundant to transitions to state $k+1$.
4. No negative word can terminate in a possible final state otherwise, by construction, it would also finish in state $k+1$.
5. The automaton is then a k_NFA validating words of S^+ and rejecting words of S^-.

Minor property The lack of solutions for $(k+1)$_NFA* allows concluding that there are no solutions for the k_NFA model. However, nothing can be deduced for the $(k+1)$_NFA model.

$$\nexists\ (k+1)\text{\_NFA}^\star \not\Rightarrow \nexists\ (k+1)\text{\_NFA} \tag{16}$$

It is easy to prove Property 16 with its contraposition and its rewritting using Property 15:

$$\begin{aligned} &\exists\ (k+1)\text{\_NFA} \not\Rightarrow \exists\ (k+1)\text{\_NFA}^\star \\ &\exists\ (k+1)\text{\_NFA} \not\Rightarrow \exists\ k\text{\_NFA} \end{aligned} \tag{17}$$

Property 17 is correct, otherwise, the existence of a $(k+1)$_NFA solution would imply the existence of a k_NFA solution. As a counterexample, Figure 1 proposes a $(k+1)$_NFA with $k=3$ but no k_NFA with $k=2$ exists.

5 Experimentation

We present an experimental study of two equisatisfiable models, k_NFA and $(k+1)$_NFA*. Our experiments are conducted on state-of-the-art instances that vary in the sizes of the alphabet and the number of positive and negative words in the learning sample. We present a simplified version of the $(k+1)$_NFA* model: indeed, the models briefly presented in Section 2.3 can be rewritten and simplified when combined with Constraints (6–11), leading to SAT instances with fewer variables and constraints. We also compare the performance of the two models in terms of the number of solutions and running time.

5.1 Context for reproductibility

The algorithms were implemented in Python using libraries such as PySAT [5]. The experiments were carried out on a computing cluster with Intel-E5-2695 CPUs, and a fixed limit of 10 GB of memory. Running times were limited to 15 minutes, including model generation and solving time. We used the Glucose [1] SAT solver with default options.

These experiments were carried out on state-of-the-art instances, described in [12]. These instances can be divided into three categories corresponding to the sizes of the alphabet (2, 5, and 10). The number of positive and negative words is the same in each instance and varies from 10 to 100 in increments of 10 for each category. There are thus 30 instances in total. We test all the possible values of k from 1 to k_{PTA} where k_{PTA} is the upper bound obtained by the size of the computed prefix tree acceptor.

We focus our experiments on k_NFA and $(k+1)$_NFA* excluding $(k+1)$_NFA$^+$ because we want to focus on the comparison of two equisatisfiable models.

Due to the deterministic behavior of Glucose, we run only one execution of each couple (instance, k).

5.2 Simplified models

The $(k+1)$_NFA* model is defined by over-constraining a k_NFA model. Mixing and combining all these constraints allows us to obtain a simplified model with fewer variables (f disappear) and fewer constraints (Constraints (1) and (7) are deleted and Constraints (2) and (3) are simplified). The final $(k+1)$_NFA* models that we use can be defined as follows:

- Variables:
  - a set of k Boolean variables determining whether state i is a possibly final state or not: $F^* = \{f_1^*, \ldots, f_k^*\}$,
  - a set of $nk(k+1)$ Boolean variables representing the transitions from state i to state j with the symbol $a \in \Sigma$: $\Delta = \{\delta_{a,\overrightarrow{i,j}} | a \in \Sigma \text{ and } i \in K \text{ and } j \in K_+\}$, and
  - a set of Boolean variables representing the paths (sequence of transitions from a state i to a state j reading a word w):

$$\Pi = \{p_{w,\overline{i,j}} | (i,j) \in K_+^2, w \in \Sigma^*\}$$

- Constraints:
  - $\bigwedge_{w \in S^+} p_{w,\overline{1,k+1}}$ *// Simplification of Constraint 2*
  - $\bigwedge_{w \in S^-} \neg p_{w,\overline{1,k+1}}$ *// Simplification of Constraint 3*
  - $\bigwedge_{a \in \Sigma} \left(\bigwedge_{i \in K} \left(\delta_{a,\overrightarrow{i,k+1}} \rightarrow \bigvee_{j \in K} \delta_{a,\overrightarrow{i,j}} \right) \right)$ *// Constraint 8*

Table 2. Results for k_NFA and $(k+1)$_NFA* on instances with all values coming from PTA. Each instance corresponds to the average for all samples of words with all k between 1 and the value given by PTA.

		Instances		Results			
	Model	Vars	Clauses	Sat	Unsat	?	Time
k_NFA	P	56,256	212,229	124	131	158	368
	S	386,508	1,459,936	80	116	217	217
	$ILS(r)$	114,218	446,854	104	133	176	418
	$ILS(P^\star)$	114,781	449,122	110	132	171	423
	$ILS(S^\star)$	51,401	205,593	105	134	174	415
	$P^\star$	412,120	1,574,192	74	126	213	504
	$S^\star$	51,683	206,564	103	134	176	409
$(k+1)$_NFA*	P	70,362	267,368	128	133	152	356
	S	381,308	1,417,649	86	126	201	169
	$ILS(r)$	122,553	477,081	125	137	151	376
	$ILS(P^\star)$	123,707	479,036	124	138	151	423
	$ILS(S^\star)$	55,192	219,637	125	137	151	362
	$P^\star$	391,773	1,475,492	120	122	171	406
	$S^\star$	55,567	220,898	122	137	154	356

- $\bigwedge_{i\in K}\left(f_i^* \rightarrow \bigwedge_{w\in S^-} \neg p_{w,\overline{1,i}}\right)$ // *Constraint 9*
- $\bigwedge_{i\in K}\left(f_i^* \rightarrow (\bigvee_{v.a\in S^+} \bigvee_{j\in K}(p_{v,\overline{1,j}} \wedge \delta_{a,\overrightarrow{j,i}} \wedge \delta_{a,\overrightarrow{j,k+1}})\right)$ // *Constraint 10*
- $\bigwedge_{w\in S^+} \bigvee_{i\in K}(p_{w,\overline{1,i}} \wedge f_i^*)$ // *Constraint 11*

Additionally, the constraints applied to define the paths either starting from the end, the beginning, or both extremities of words (as outlined in Section 2.3) can also be simplified. The most significant simplification is for the suffix model, where the space complexity changes from $\mathcal{O}(\sigma \cdot k^3)$ clauses to $\mathcal{O}(\sigma \cdot (k+1)^2)$ clauses. All other models, which partially use the suffix model, are also affected except for the prefix model.

5.3 Results and Discussions

Table 2 shows results for k_NFA and $(k+1)$_NFA* with 7 models: the prefix model P, the suffix model S, and 5 hybrid models, the local search model with random initial configuration $ILS(r)$, with $P^\star$ (respectively $S^\star$) initial configuration, the best prefix model $P^\star$, and finally the best suffix model $P^\star$. The combination of learning samples and the number of states (k values) produces 413 instances, and thus, 413 executions for each model. Table 2 is divided into two parts: instances description and results. Instances are described by the model and the average number of variables and clauses. As result, we consider the number of satisfiable instances (Sat), unsatisfiable instances (Unsat), and Unknown solutions (noted ?). Moreover, we give the average running time (in seconds).

Table 3. Difference (in %) from k_NFA to $(k+1)$_NFA* based on results in Table 2.

Model	Vars	Clauses	Sat	Unsat	?	Times
P	+25.07	+25.98	+3.23	+1.53	-2.35	-3.44
S	-1.35	-2.9	+7.5	+8.62	-8.16	-22.04
$ILS(r)$	+7.3	+6.76	+20.19	+3.01	-10.55	-10.16
$ILS(P^\star)$	+7.78	+6.66	+12.73	+4.55	-8.26	-0.13
$ILS(S^\star)$	+7.37	+6.83	+19.05	+2.24	-9.62	-12.9
$P^\star$	-4.94	-6.27	+62.16	-3.17	-21.00	-19.5
$S^\star$	+7.51	+6.94	+18.45	+2.24	-9.28	-13.01

Table 3 shows the relative difference from k_NFA to $(k+1)$_NFA* for each model. Column names are the same as in Table 2, but they indicate the percentage difference compared to the results obtained with k_NFA instances.

Tables 2 and 3 show that the number of variables and clauses in $(k+1)$_NFA* instances increase with respect to k_NFA, but they remain very close; except for the prefix model P with a +25% increase, and for the suffix model S and the best prefix model $P^\star$ with a decrease (respectively -1 to -6%), the instances are around +8% larger. Model P increases by 25% because it is the only one that does not use the suffixes (which give a $\mathcal{O}(\sigma k^3)$ complexity). The others, although adding extra Constraints (6–11), keep similar sizes since the $(k+1)$_NFA* models allow decreasing complexity of suffix constructions from $\mathcal{O}(\sigma k^3)$ to $\mathcal{O}(\sigma k^2)$.

We can observe that the number of Sat solutions increases for all models when we use $(k+1)$_NFA*. It is similar for Unsat solutions (except for model $P^\star$). The number of instances not solved in the given time is therefore logically reduced. The very interesting result is that this improvement in the number of resolved instances is coupled with a diminution of the resolution time.

In terms of global efficiency, best results are obtained by $ILS(r)$, $ILS(P^\star)$, and $ILS(S^\star)$ with $(k+1)$_NFA* (only 151 instances remain unsolved). Adding the running time in the comparison, model $ILS(S^\star)$ is the best one. Compared to the best model for k_NFA (P), it finds one more Sat and 6 more Unsat solutions with a faster average running time (6 seconds less).

6 Conclusion

Grammatical inference is the process of learning formal grammars, in our case as a Nondeterministic Finite Automaton. In this paper, we proposed an over-constrained model of size $k+1$ ($(k+1)$_NFA*) with properties that allow deriving a solution for the classical model of size k (k_NFA). If such an automaton of size $k+1$ exists, it can be freely reduced to an automaton of size k, and if it does not exist, we have the proof there is no automaton of size k. The advantage of using the $(k+1)$_NFA* model is to obtain shorter resolution times while increasing the rate of solved instances.

Working with a model of size $k+1$ increases the number of clauses and variables but the additional constraints allow us to simplify the model and limit

the combinatorial explosion. Moreover, these constraints allow us to lower the global complexity of most of our models that use a suffix construction (such as the model S).

In the future, we plan to work on adding new constraints to further reduce the complexity of the model $(k+1)\_\text{NFA}^\star$. Moreover, a parallel execution of some well-chosen models could lead even more often to solved instances.

References

1. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: Proc. of IJCAI 2009. pp. 399–404 (2009)
2. Denis, F., Lemay, A., Terlutte, A.: Learning regular languages using rfsas. Theor. Comput. Sci. **313**(2), 267–294 (2004)
3. Garey, M.R., Johnson, D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness. W.H. Freeman & Company, San Francisco (1979)
4. de la Higuera, C.: Grammatical Inference: Learning Automata and Grammars. Cambridge University Press (2010)
5. Ignatiev, A., Morgado, A., Marques-Silva, J.: PySAT: A Python toolkit for prototyping with SAT oracles. In: SAT. pp. 428–437 (2018), https://doi.org/10.1007/978-3-319-94144-8_26
6. Jastrzab, T.: On parallel induction of nondeterministic finite automata. In: Proc. of ICCS 2016. Procedia Computer Science, vol. 80, pp. 257–268. Elsevier (2016)
7. 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)
8. 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)
9. Jastrzab, T., Czech, Z.J., Wieczorek, W.: Parallel algorithms for minimal nondeterministic finite automata inference. Fundam. Informaticae **178**(3), 203–227 (2021), https://doi.org/10.3233/FI-2021-2004
10. Jastrząb, T., Lardeux, F., Monfroy, E.: Taking advantage of a very simple property to efficiently infer NFAs. In: 34th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2022, virtual, November 1-2, 2022. IEEE (2022)
11. Lardeux, F., Monfroy, E.: 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), https://hal.univ-angers.fr/hal-03284541
12. Lardeux, F., Monfroy, E.: Optimized models and symmetry breaking for the NFA inference problem. In: 33rd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2021, Washington, DC, USA, November 1-3, 2021. pp. 396–403. IEEE (2021), https://doi.org/10.1109/ICTAI52525.2021.00065
13. Lecoutre, C., Szczepanski, N.: PYCSP3: modeling combinatorial constrained problems in python. CoRR **abs/2009.00326** (2020), https://arxiv.org/abs/2009.00326
14. Rossi, F., van Beek, P., Walsh, T. (eds.): Handbook of Constraint Programming, Foundations of Artificial Intelligence, vol. 2. Elsevier (2006)
15. Stützle, T., Ruiz, R.: Iterated Local Search, pp. 579–605. Springer International Publishing, Cham (2018), https://doi.org/10.1007/978-3-319-07124-4_8
16. 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)

17. Tseitin, G.S.: On the Complexity of Derivation in Propositional Calculus, pp. 466–483. Springer Berlin Heidelberg, Berlin, Heidelberg (1983)
18. 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)
19. Wieczorek, W.: Grammatical Inference – Algorithms, Routines and Applications, Studies in Computational Intelligence, vol. 673. Springer (2017)