Robust models to infer flexible nondeterministic finite automata

Tomasz Jastrzab, Frédéric Lardeux, Eric Monfroy

Highlights

Robust models to infer flexible nondeterministic finite automata

Tomasz Jastrzab, Frédéric Lardeux, Eric Monfroy

- Proposed over-constrained inference models for NFA of size $k+1$ allow us to efficiently and systematically derive NFA of size k.
- Experimental results indicate that proposed models help to reduce the number of unsolved instances for selected benchmark sets.
- Inferred NFA can be post-processed to reduce or increase the number of accepting states and to increase the number of transitions, producing automata that are globally more or less accepting.

Robust models to infer flexible nondeterministic finite automata

Tomasz Jastrzab[a], Frédéric Lardeux[b], Eric Monfroy[b]

[a] *Silesian University of Technology, Gliwice, Poland*
[b] *LERIA, University of Angers, Angers, France*

Abstract

Grammatical inference involves learning a formal grammar as a finite state machine or a 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 the others. To address this task we construct several over-constrained inference models capable of finding NFA of size $k+1$, which are directly convertible to NFA of size k. Additionally, we propose an NFA evaluation method based on random walks along with two methods used to enhance the acceptance rates of the inferred NFA. The effectiveness of our approach is demonstrated through the results of comprehensive experiments conducted on several benchmark sets. This paper is an extended version of the ICCS 2023 paper entitled "Inference of over-constrained NFA of size $k+1$ to efficiently and systematically derive NFA of size k for grammar learning" [1].

Keywords: grammatical inference, nondeterministic automata, SAT models

1. Introduction

As defined in [2], grammatical inference consists in learning formal grammars in terms of finite automata or production rules, from a given learning sample of words. This research topic has various applications going from compiler design, bioinformatics, and pattern recognition up to machine learning. In this article, we are interested in learning finite automata: more precisely, given a set of positive examples and negative examples, we want to learn nondeterministic finite automaton (NFA) that will accept the positive words and reject the negative words.

Email addresses: Tomasz.Jastrzab@polsl.pl (Tomasz Jastrzab), Frederic.Lardeux@univ-angers.fr (Frédéric Lardeux), Eric.Monfroy@univ-angers.fr (Eric Monfroy)

Although deterministic automata are much simpler and more efficient to use, they are generally much larger (in terms of the number of states) than nondeterministic automata for the same language. Moreover, in most of the models (a model is a description of the problem in a declarative language), the complexity is related to the number of states. Hence, as most of the works in the state of the art, we focus on learning NFAs.

Our goal is to generate the smallest NFA, i.e., 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[1]) on the number of states, and using some optimization algorithms taking advantage of the properties of the problem, to find the smallest possible number of states.

Given a sample of positive and negative words, the problem of learning (generating) a finite automaton has been studied in various communities, using a variety of tools. For example, DeLeTe2 [3] consists in building the prefix tree acceptor (PTA), and then, reducing the size of the NFA by merging states having similar behavior. Some newer approaches can be cited, such as the family of algorithms for regular languages inference presented in [4]. Some studies are based on metaheuristics [5], such as hill-climbing in [6].

Some other techniques, such as constraint programming [7] are based on complete solvers, i.e., some generic algorithms for solving a determined type of variables and constraints (e.g., a SAT solver for solving a Boolean formula over some Boolean variables) that can always find a solution if one exists, prove unsatisfiability, and find the global optimum in optimization problems. Constraint programming (CP) [8] is a problem-solving paradigm for solving combinatorial problems using techniques originating from artificial intelligence, computer science, and operations research. In CP, the user does not describe how to solve the problem, but they state the problem with decision variables linked by some relations called constraints. Various types of variables (Boolean, bounded integers, ...) and various types of constraints (linear, non-linear arithmetic, symbolic, such as unification, ...) can be considered. In CP, a model represents a problem, and an instance is given by a model and some data. In our case, the problem consists of

[1] 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. Note that it is possible to build the PTA for only positive words or both types of words.

generating an NFA with a generic sample, (i.e., a not-yet-defined sample in which the positive and negative words and their numbers are parameters), and an instance is the instantiation of the model with a given sample.

In CP, the problem is generally modeled (or stated) as a Constraint Satisfaction Problem (CSP) or a Constrained Optimization Problem (COP). Again, various techniques, in terms of types of variables and constraints, have been employed: for example, Integer Non-Linear Programming (INLP) in [9] or parallel SAT solvers in [10, 11]. Additionally, the author of [12] proposed two strategies for solving the CSP formulation of the problem, and [13] presents a parallel approach for solving the optimization variant of the problem.

We believe that a good model is at least as important as a good solver. Thus, contrary to many works, we are not interested to adapt a solver to our problem, nor to design an ad-hoc solver. Instead, we aim at enhancing SAT models for grammatical inference and NFA generation. SAT is the well-known SATisfiability problem which was defined in the seventies [14]. Modeling in SAT consists in defining and stating the problem with Boolean variables and a Boolean formula in Conjunctive Normal Form (CNF) [2]. In a previous paper [15], we have proposed several SAT models, either starting from prefixes or suffixes of words, or trying to optimize both prefixes and suffixes at the same time. These models applied to given samples enable us to generate smaller SAT instances than a standard model. Moreover, these instances are solved more easily by standard SAT solvers.

Here, we want to exploit a property of the problem based on the size (i.e., the number of states) of the generated NFA. More precisely, we focus on creating over-constrained models[3] of NFA of size $k+1$ (i.e., NFA with $k+1$ states) with some special properties that allow reducing them to classical NFA of size k. The benefit of these over-constrained models mainly resides in their spatial complexity. Whereas the complexity of the generation of an NFA of size k for a learning sample S is in $\mathcal{O}(\sigma k^3)$ variables, and $\mathcal{O}(\sigma k^3)$

[2]In fact, most of the solvers only accept SAT instances in CNF, and converting a formula into CNF can bring an exponential explosion to the size of the formula.

[3]Over-constraining a model consists in adding some extra constraints to reduce the number of solution of the model, and thus the search space, expecting the solver to be more efficient on this over-constrained model. In our case, we reduce the number of possible NFA of size $k+1$, but we are sure to have a solution if there is an NFA of size k.

clauses (with $\sigma = \sum_{w \in S} |w|$ for most of the models[4]), the generation of our over-constrained NFA of size $k+1$ is in $\mathcal{O}(\sigma k^2)$ variables and clauses.

Moreover, we propose a technique for ensuring that a k+1 NFA can always be reduced to a k NFA. It relies on the notion of possibly final states—these are the states from the $k+1$ NFA (except for the $(k+1)$'th state) such that: 1) they have outgoing nondeterministic transitions for the same symbol, one going to the $k+1$ state, and another one going to some other state, and 2) they are not the last state of a path reading some negative word of the sample. For the k NFA, these states may thus be final. Consequently, the simplest reduction from $k+1$ to k NFA consists in transforming each possibly final state of the $(k+1)$ NFA into a final state of the k NFA.

We also try to increase the number of transitions in the NFA. Usually, transitions that are not defined (i.e., transitions whose existence do not change the behavior of the NFA w.r.t. the given sample) are not considered in the NFA. Densifying consists in adding some of these transitions. Given the densified NFA, we try to reduce or increase the number of final states, without violating the constraints imposed by the sample. The idea is to obtain NFAs that accept positive words of the sample, reject the negative ones, but are also globally more accepting or more rejecting. For example, if we want to classify peptides, it is better to accept fewer peptides that are sure to be safe than accepting more, with some harmful ones among them.

Our contribution is thus three-fold. Firstly, we propose some over-constrained inference models for NFA of size $k+1$, which allow us to straightforwardly reduce the found NFA to one of size k. Secondly, we propose some methods to reduce or extend the number of accepting states and to densify the automaton to influence its global acceptance rates—this way we obtain a flexible method of generating NFAs of various characteristics. Finally, we propose an evaluation method based on random walks which allows us to compare the obtained NFA from different perspectives.

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

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

2. Inference of automata: related works and SAT models

Let $\Sigma = \{a_1, \ldots, a_n\}$ be an alphabet of n symbols, and $S = S^+ \cup S^-$ be a learning sample made of two sets. The problem consists in inferring a grammar for a language $\mathcal{L}$ (called the target language) such that words from S^+ (called positive words) belong to $\mathcal{L}$, and words from S^- (called negative words) do not belong to $\mathcal{L}$: $S^+ \subseteq \mathcal{L}$ and $S^- \cap \mathcal{L} = \emptyset$.

Automata are finite state machines used to recognize strings. A string is read from left to right, and at each step, the next state is chosen based only on the previous state and the symbol being read. This makes the automata powerful enough to accept a limited class of languages that are called regular languages (e.g., [16]), the class of languages we are interested in.

The classification of a word with an automaton can be made deterministic by allowing only one action to be possible at each step. These machines, called deterministic finite automata (DFA), are generally easier to manipulate. However, nondeterminism may be a partial solution to the difficulties one has to face when learning: incomplete data, problem complexity, etc.

We focus here on inferring a regular language by learning nondeterministic finite automata. This means that words from S^+ must be accepted by the generated automaton, and words from S^- must be rejected. A nondeterministic finite automaton (NFA) is a quintuple $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ where Σ is an alphabet, Q is a finite set of states, $I \subseteq Q$ is the set of initial states (in the following, we consider only one initial state, i.e., state q_1), F is the set of final (accepting) states, and $\delta : Q \times \Sigma \Longrightarrow 2^Q$ is a transition function. A word $w = a_1 ... a_l$ is recognized by $\mathcal{A}$ if there is a sequence of states $q_{i_1}, \ldots, q_{i_l}$ with the i_j in $[1..l]$ and $q_{i_l} \in F$ such that there exists a sequence of transitions $\delta(q_{i_1}, a_1), \ldots, \delta(q_{i_l}, a_l)$ with $q_{i_{j+1}} \in \delta(q_{i_j}, a_j)$.

Before defining formally the NFA inference problem, we motivate the use of NFA and SAT models.

2.1. Why using SAT models for inferring NFA?

Many approaches have been developed for the generation of automata in the context of grammatical inference. We motivate and justify here the use of SAT models.

First, note that our goal is to obtain a flexible method that can be easily adapted to NFA, DFA, and specific automata (e.g., automata with three sorts of states as presented in [2]). We also want to be able to generate various languages $\mathcal{L}$ corresponding to S, able to be more "accepting" or "rejecting"

as shown in our experiments. Moreover, we do not want to develop an ad-hoc tool, nor a too specific algorithm.

The pioneer work of Dana Angluin (e.g., [17]) is based on an oracle (called the Teacher) which can answer membership queries about a word, and can also test equivalence of 2 grammars (in case of negative answer, it also provides a counterexample. Although a sample could replace the Teacher, it is not known "a priori" how many loops Angluin's algorithm must perform to succeed in inferring the grammar, and thus, what should be the sample size. In our case, we want to be able to generate a grammar, even with a very small sample[5].

In [3], Denis et al. proposed an algorithm designed to discern regular languages. This algorithm operates by generating a specialized form of nondeterministic finite automaton known as a residual finite state automaton (RFSA). Then, they propose a new learning algorithm, DeLeTe2, based on the search for the inclusion relations between the residual languages of the target language. Thus, the algorithm yields an RFSA as its output. The authors establish that unless $|\Sigma| = 1$, the characterization of the class of regular languages over Σ using RFSAs cannot be achieved in polynomial time.

RPNI is one of the methods which are based on merging states from the Prefix Tree Acceptor (PTA). A PTA is a tree-like DFA built by using each prefix of the sample (only from S^+ in the case of RPNI) as a state. Then, this kind of method tries to merge states and to propagate the merges under the condition of not merging states that represent positive samples with those which represent negative ones. In the case of RPNI, the method also keeps the automaton deterministic.

Some newer approaches based on state merging can be cited, such as the family of algorithms for regular languages inference presented in [4]: the method is based on building, for each word of S^+, an irreducible subautomaton (i.e., merging two states would lead to acceptance of some word from S^-). Each subautomaton is obtained by merging states in the automaton that just recognizes the word. The output of any algorithm from the family consists of a collection of selected NFAs for each positive word. Each member of the family infers the class of regular languages in the limit. The authors claim that this method gives better results than DeLeTe2, and RPNI [18].

[5]However, it is obvious that the quality of the generated NFA in terms of classification is also depending on the size and quality of the sample.

However, the final NFAs may differ from each other: indeed, depending on the algorithm used, the number of subautomata inferred for each word or the order of state merging can vary.

Another state merging algorithm, called order independent learning (OIL), was proposed in [19]. The OIL algorithm, employing the notion of a universal automaton, converges towards generating an NFA for recognizing the target language, irrespective of the sequence in which states are merged. The authors demonstrate that while the merging of states in varying orders may result in distinct NFAs, the language accepted by these NFAs consistently corresponds to the target language. However, OIL requires a *universal sample*. Although there exists such a sample for each regular language, we cannot be dependent of such a property since for real applications, samples are obtained by observation generally.

To our knowledge, the result of state merging algorithms always depend on the order of merges (or works with some specific samples in the case of OIL). Moreover, it is not possible to know beforehand the size of the obtained NFA/DFA, nor to know whether this size is minimal. It is also not possible to influence the acceptance rates of the automaton, i.e., respecting a sample S, will the DFA/NFA be globally more accepting or more rejecting towards a set of words to be classified?

The problem of methods based on metaheuristics [5] is the impossibility to ensure the existence or not of a solution. Thus, as with hill-climbing in [6], one may fail to find a correct solution.

Compared to the previously cited methods, one can decide to use some already existing tools, such as constraint solvers [7] that are complete, i.e., they can always find a solution if one exists, prove unsatisfiability otherwise, and find the global optimum in optimization problems. To use these solvers, the problem is generally modeled as a Constraint Satisfaction Problem (CSP) or a Constrained Optimization Problem (COP). Various types of variables and constraints can be used. In [9], Integer Non-Linear Programming (INLP) has been used. Although rather "elegant", such models are not well suited for the NFA problem. We performed some tests with PyCSP [20] and the ACE [21] solver, and only trivial and extremely small samples could be tackled.

Using a SAT model and a classical SAT solver is thus an alternative to infer an NFA with k states, k being given. However, one has to be careful since as shown in [22], a model such as the *direct model* can have a spatial complexity in the order of $\mathcal{O}(|S^+|.k^{|\omega_+|})$ with ω_+ the longest word of S^+, and k the number of states of the generated NFA. Better models have a

complexity in the order of $\mathcal{O}(k^3)$ or even $\mathcal{O}(k^2)$. The models can easily be completed to infer DFAs by just adding some few constraints: for each state and for each symbol a of Σ, they enforce at most one outgoing transition with a. The problem thus becomes more constrained than with NFA, and one could think it would be easier to solve it. However, the generated automata are much larger (exponentially larger than NFA in the worst case), and this significantly limits the inference of DFA.

Note also that the satisfiability problem which consists in computing an NFA of size k, can be easily extended to an optimization problem: for each k smaller than the minimum NFA size k^* the problem is not satisfiable, and for each $k > k^*$ the problem is satisfiable; it is thus easy to find the minimal NFA by a dichotomic use of the satisfiability problem. It is also rather easy to parallelize this process, see [13]. As shown in the current paper, it is also possible to derive some more or less accepting NFA using the notion of "possibly accepting states". Slightly modifying the models, it is also possible to infer some special NFA, such as NFA with three types of states: accepting, rejecting, or whatever states (see [23] for inference of 3 sorts automata, and [2] for motivations for using them).

A SAT model thus gives us the required flexibility for tackling other types of automata and to make them more or less "accepting". This is at the cost of a less scalable method. However, the performance have been significantly improved using hybrid models, some properties, and parallelism, and we can now tackle most of the applications we are interested in.

2.2. Meta-model

We now formally define the NFA learning problem. To this end, we rely on the propositional logic paradigm in order to propose a generic SAT meta-model. We then instantiate this meta-model to present some previously established models [24] deriving words from prefixes, suffixes, or trying to optimize hybrid derivations.

2.2.1. Notations

We first need a few notations. For a given k, we also consider $K = \{1, \ldots, k\}$, i.e., the set of the first k non-zero integers. The following integer and Boolean variables are the ones that appear in our meta-model:

- The integer k represents the size of the NFA to be generated; it is given.

- For each state i, we have one Boolean variable f_i determining whether state i is final or not. We thus have the set F of k Boolean variables $F = \{f_1, \ldots, f_k\}$.

- For each pair of states (i, j) (i possibly equal to j), and each symbol a of Σ we have a Boolean variable $\delta_{a,\overrightarrow{i,j}}$ determining whether there is a transition from state i to state j with the symbol a. We thus obtain a set $\Delta = \{\delta_{a,\overrightarrow{i,j}} | a \in \Sigma \text{ and } (i,j) \in K^2\}$ of nk^2 Boolean variables.

The "path" $p_{w,\overline{i_1,i_{m+1}}}$ is a Boolean variable representing the existence of a derivation from state i_1 to state i_{m+1} for the word $w = a_1 \ldots a_m$, i.e., the successive transitions from state i_l to i_{l+1} with the symbol a_l. In terms of Boolean variable, $p_{w,\overline{i_1,i_{m+1}}}$, is defined as follows:

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

A path from i_1 to i_{m+1} is a sequence of transitions (a derivation), it is thus directed. However, we will incrementally build such path either starting from i_1, starting from i_{m+1}, or starting from both sides. Thus, to avoid possible confusion between paths and their building, we will refer to it as $\overline{i_1, i_{m+1}}$ without any direction.

2.2.2. The meta-model

The aim of the meta-model is to be generic and parameterized. Hence, we can then instantiate it in order to obtain various models. Our meta-model to define an NFA of size k (that we denote by k_NFA) can be described with the following constraints:

- This constraint is only related to the empty word λ, and the status of the initial state: if λ is a word of S^+, then the initial state must be final; oppositely, if it is a negative word, the initial state cannot be final:

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

- A positive word w from S^+ must terminate in 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) for the word w:

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

- Similarly, a negative word w must not terminate in a final state of the k_NFA, i.e., either there is no path for w from state 1 to a state from K, or each path for w from 1 to a state i implies that i in a not a final state:

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

Of course, the notion of a path can be defined and built in many ways. In [24], prefix, suffix, and some hybrid constructions are proposed.

2.3. From the meta-model to models

As said before, we can consider various models for inferring NFA. These models can differ in terms of the types of constraints and variables, or can use the same type of variables, but with a different approach to the problem.

Instead of Boolean, some models use 0/1 variables to represent transitions, and final states. Then, depending on the kind of constraints, the model can be of the INLP family [9] or the CSP family [8]. We made some preliminary tests with various models with the constraint modeler PyCSP3 [20] and we obtained some disastrous results: the models are elegant and concise, but the constraints do not permit sufficient filtering[6] or sufficiently efficient filtering of the search space. Indeed, for example, some of these constraints are based on n-ary sums of max: both of these constraints have costly and weak filtering (see, e.g., [8]), and their combination is even worse. Intrinsically, the NFA learning problem is a Boolean problem, and thus, it is well suited for SAT solvers, i.e., solvers that decide the satisfiability of a Boolean formula, generally in CNF.

One of the most standard models for NFA, the direct model (see [13, 15]), consists in considering each possible path for each word, without sharing any sub-paths between words. This model which is an instantiation of our meta-model, has a very bad space complexity of $\mathcal{O}(|S^+|(|\omega_+|+1)k^{|\omega_+|})$ clauses, and $\mathcal{O}(|S^+|k^{|\omega_+|})$ variables with ω_+ the longest word of S^+. Moreover, this model does not behave well in practice.

In [25], 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

[6] Using a constraint, filtering consists in removing values from variable domains that will never be part of a solution [8].

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 [26] 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).

Prefix model. Some models such as the prefix model (P), need some auxiliary Boolean variables for representing paths for each prefix of each word (positive and negative) of the learning sample. This means that words are built from state 1, incrementally, adding each time a new symbol, i.e., symbols are concatenated one by one, forming all the prefixes of the word, up to the word itself. Given a word ω:

- For the first symbol of the word ω:

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

- The path for each prefix w of ω with $w = va$, $v \in \Sigma^+, a \in \Sigma$, is built recursively:

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

After the transformation into CNF using Tseitin transformations [27], the spatial complexity of the prefix model is in $\mathcal{O}(\sigma k^2)$ variables, and $\mathcal{O}(\sigma k^2)$ clauses with $\sigma = \Sigma_{w \in S}|w|$ (see [15] for details).

Suffix model. Models such as the suffix model (S) also need some extra variables. This time, we consider recursive concatenation of symbols on the left (i.e., av with $a \in \Sigma, v \in \Sigma^+$). Thus, given a word ω, its construction starts from the end of the word ω:

- For the last symbol of the word ω:

$$\bigvee_{i,j \in K} \delta_{a,\overrightarrow{i,j}} \leftrightarrow p_{a,\overline{i,j}} \tag{6}$$

  Note that here, the transition is between states i and j since we do not know where the word terminates.

- The path for each suffix w of ω with $w = av$ is built recursively:

$$\bigwedge_{i,j \in K} \left(p_{w,\overline{i,j}} \leftrightarrow \left(\bigvee_{l \in K} \delta_{a,\vec{i,l}} \wedge p_{v,\overline{l,j}} \right) \right) \tag{7}$$

  As Constraint (6) before, we do not know where the words terminate. We thus consider paths between i and j for the suffixes.

After the transformation in CNF using Tseitin transformations, the spatial complexity of the suffix model is in $\mathcal{O}(\sigma k^3)$ variables, and $\mathcal{O}(\sigma k^3)$ clauses with σ being the total length of suffixes for all the words of the sample [15]. Due to not knowing where a word terminates, in contrast to the prefix model, where we always start from state 1, we are now in a $O(\sigma k^3)$ space complexity.

Hybrid models. Whereas the prefix (respectively suffix) model consists in splitting words as $w = va$ (respectively as $w = av$) with $a \in \Sigma$, hybrid models consist in splitting words into a prefix and a suffix that are words from Σ^+, i.e., $w = uv$ with $u, v \in \Sigma^+$. Then, the prefix u is generated with Constraints (4), (5) and the suffix v with Constraints (6), (7). We thus need to "glue" together these two sub-paths with the constraints:

$$p_{w,\overline{1,i}} \equiv \bigvee_{j \in K} p_{u,\overline{1,j}} \wedge p_{v,\overline{j,i}} \tag{8}$$

Since the suffix part is in $\mathcal{O}(\sigma k^3)$, the hybrid models are also in $\mathcal{O}(\sigma k^3)$.

The question is now, how to split each word of S in order to obtain a model that can be solved more efficiently? In what follows, *Suf*(S) represents the set of all suffixes of each word of S, and similarly, *Pref*(S) is the set of all prefixes of words of S. To answer this question, we present three types of hybrid models. Each word $w \in S$ is thus seen as the concatenation of a prefix u and a suffix v (i.e., $w = uv$) that have to be determined. We thus now consider two samples, $S_u = S_u^+ \cup S_u^-$ with $S_u^+ = \{u \mid \exists w \in S^+, w = uv\}$ and $S_u^- = \{u \mid \exists w \in S^-, w = uv\}$, and $S_v = S_v^+ \cup S_v^-$ with $S_v^+ = \{v \mid \exists w \in S^+, w = uv\}$ and $S_v^- = \{v \mid \exists w \in S^-, w = uv\}$.

- **Best suffix model ($S_k^\star$)**: as building suffixes is in $\mathcal{O}(\sigma k^3)$, the idea is to try to optimize the construction of suffixes. To this end, the best suffix model $S_k^\star$ is based on an ordering of the set *Suf*(S). Let

$\Omega(v) = \{w \in S \mid v \in \mathit{Suf}(w)\}$ be the set of words accepting v as a suffix. Then,

$$v_1 \succcurlyeq_s v_2 \text{ iff } |v_1| \cdot |\Omega(v_1)| \geq |v_2| \cdot |\Omega(v_2)|$$

with $|.|$ being both the length of a word and the cardinality of a set.

The set of best suffixes $\mathcal{B}_s$ is composed of the best suffixes (w.r.t. to $\succcurlyeq_s$) that cover S [25] and is minimal. Thus, $\mathcal{B}_s \subseteq \mathit{Suf}(S)$ such that:

1. $\bigcup_{v \in \mathcal{B}_s} \Omega(v) = S$, i.e., each word of S has a suffix in $\mathcal{B}_s$;
2. $\forall v' \in \mathcal{B}_s, (\bigcup_{v \in \mathcal{B}_s \setminus \{v'\}} \Omega(v)) \subset S$, i.e., if a suffix is removed from $\mathcal{B}_s$, at least one word of S has no suffix in $\mathcal{B}_s$.
3. $\forall \mathcal{B}'_s \subseteq \mathit{Suf}(S)$, if $\mathcal{B}'_s \neq \mathcal{B}_s$ and $\mathcal{B}'_s$ satisfies Conditions 1 and 2 above, then $\sum_{v \in \mathcal{B}_s} |v||\Omega(v)| \geq \sum_{v \in \mathcal{B}'_s} |v||\Omega(v)|$, i.e., $\mathcal{B}_s$ is the largest set of suffixes regarding the $\succcurlyeq_s$ order, and respecting Conditions 1 and 2 above.

For each word $w \in S$, $v^\star_w$, the best suffix for w in $\mathcal{B}_s$ is:

$$v^\star_w \in \mathcal{B}_s \cap \mathit{Suf}(w) \text{ and } \forall v \in \mathit{Suf}(w),\ v^\star_w \succcurlyeq_s v$$

i.e., the suffix $v^\star_w$ for w is the largest suffix possible w.r.t. $\succcurlyeq_s$ in $\mathcal{B}_s$. Set S_v will thus be $\mathcal{B}_s$. Then, for each w, the prefix is built as $w = uv^\star_w$, and the $u's$ constitute S_u.

- **Best prefix model ($P^\star_k$)**: this model is built similarly as the best suffix model above. This time, we have an order between prefixes. Consider u_1 and u_2, two prefixes of $\mathit{Pref}(S)$, the set of all prefixes of all words in S, then:

$$u_1 \succcurlyeq_p u_2 \quad \Leftrightarrow \quad |u_1| * |A(u_1)| \geq |u_2| * |A(u_2)|$$

with $A(u) = \{w \in S \mid u \in \mathit{Pref}(w)\}$. Then, the set $\mathcal{B}_p$ of best prefixes is equivalent to the set of best suffixes above, but w.r.t. the ordering $\succcurlyeq_p$. Finally, the suffixes are built similarly as above.

- **Local search optimization of the splitting ($ILS_k(Init)$)**: we consider here that the search space corresponds to all the hybrid models, i.e., all the possible splits for all words of S. The fitness function we

use is: $f(S_p, S_s) = |Pref(S_p)| + k \cdot |Suf(S_s)|$[7], and the optimization of the prefixes and suffixes is based on Iterated Local Search (ILS) [26] with the fitness function f for optimizing our hybrid model $ILS_k(Init)$. The search starts with an initial split $Init$ for each word of S. At each iteration, the best split $w = uv$ is found for the selected word w: w is selected randomly with a roulette wheel selection based on the weights of words defined by $\text{weight}_w = 75\%/|S| + 25\% \cdot |w|/(\sum_{w_i \in S} |w_i|)$. Note that the coefficients (75% and 25%) ensure that every word receives a minimum weight, augmented by a bonus proportional to its length. The number of iterations is given. Since our word selection process ensures diversification, there is no need to introduce random walks or restarts.

The $Init$ split can be a random split, but it can also be a split based on the best suffix (respectively prefix) model. Note that we do not need the $S_k^\star$ (respectively the $P_k^\star$) model itself, but only the splitting: once we have determined where to split each word (e.g., for the best suffix model, by determining $\mathcal{B}_s$ and the best prefix of each word) we have the initial split (the initial configuration of the local search), and generating the constraints would be useless since this splitting will be improved by the local search. In this case, since building $\mathcal{B}_s$ or $\mathcal{B}_p$ is the costly part of the suffix and prefix models, we know that the generation will probably be longer than usual models, but with the hope of obtaining smaller instances.

3. Taking advantage of the problem characteristics: model extensions

We can now present a property that we will use in what follows. Consider a sample S. Then, if there is a k_NFA to accept words of S^+ and reject words of S^-, there is also an NFA of size $k+1$, also accepting words of S^+ and rejecting words of S^-. This property is quite obvious, for example, it is sufficient to add a new state $k+1$, and no transition outgoing from or incoming to it. It is also rather useless: why generating an NFA of size $k+1$

[7]Since the size of suffix based models ($\mathcal{O}(k^3)$) is larger than the size of the prefix model ($\mathcal{O}(k^2)$) by a factor k, we intuitively multiplied the penalty for suffixes by k. Experimentation showed that this fitness function is suitable.

knowing that the instance is larger than the k NFA instance? However, we can refine and complete this property in order to generate k_NFA from a $(k+1)$_NFA in cases when k_NFA cannot be inferred directly. To do so, some extra constraints are added to the $(k+1)$_NFA to over-constrain it. We obtain what we call $(k+1)$_NFA extensions.

3.1. From k_NFA to (k + 1)_NFA

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

- $Q^{A'} = Q^A \cup \{q_{k+1}\}$,
- $F^{A'} = \{q_{k+1}\}$,
- and $\Delta^{A'}$ defined by:

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

In fact, there is only one final state for A', i.e., the newly created state $k+1$. All the transitions of A are kept in A'. For each final state j, for each symbol a, for each incoming transition from state i to j with a, a new transition from i to $k+1$ with a is created. In other words, each transition to a final state is duplicated to state $k+1$. Following this construction, state $k+1$ does not have any outgoing transition.

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.

Sketch of the proof:

1. For each word $w \in S^+ \setminus \{\lambda\}$:
   Let $w = va$ with $v \in \mathit{Pref}(S)$ and $a \in \Sigma$. Then v is recognized by A and can finish in several states $T \subseteq Q$ (not necessary final states). As $w \in S^+$, at least one transition $\delta^A_{a,i,\overrightarrow{j}}$ with $i \in T$ and $j \in F$ exists. By the rules of transitions creation, $\delta^{A'}_{a,i,\overrightarrow{k+1}}$ exists and so word w is recognized by A'.

2. For each word $w \in S^- \setminus \{\lambda\}$:
   Let $w = va$ with $v \in \mathit{Pref}(S)$ and $a \in \Sigma$. There may be a path for v in A that terminates in states $T \subseteq Q$. As $w \in S^-$, if $p^A_{v,\overline{1,j}}$ exists then $\delta^A_{a,\overrightarrow{j,i}}$ with $j \in T$, $i \in F$ does not exist. Thus, $\delta^{A'}_{a,\overrightarrow{1,k+1}}$ is not created and w is then rejected by A'.

Note that if $\lambda \in S^+$, we can do a very similar construction by considering state $k+1$ as the second initial state. Since there are no outgoing transitions from this state, it will accept λ without affecting the rest of the NFA. If $\lambda \in S^-$, no changes need to be introduced to the model, since $k+1 \neq 1$.

In what follows, we over-constrain an NFA of size $k+1$, i.e., we over-constrain a $(k+1)$_NFA model, to obtain an NFA as close as possible to A'. The goal is to obtain over-constrained $(k+1)$_NFA model and instances that can be solved as easily (or even easier as we will see) as an NFA of size k. Moreover, the idea is also that the NFA generated with the over-constrained $(k+1)$_NFA model can be reduced to an NFA of size k. To this end, we present two $(k+1)$_NFA model *extensions* for which a new state and new constraints are added.

To represent the fact that the new extensions contain one more state than the initial k_NFA, we name them $(k+1)$_NFA$^+$ and $(k+1)$_NFA*. Of course, new Boolean variables are added for the new final state and the new transitions.

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

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

A $(k+1)$_NFA$^+$ is a $(k+1)$_NFA with some extra properties to help to reduce it to a k_NFA by a reduction algorithm (as in [24]). The properties that we enforce are:

- a $(k+1)$_NFA$^+$ has one and only one final state, i.e., state $k+1$;
- there is no outgoing transition for state $k+1$;
- each transition from a state i to state $k+1$ with symbol a has an equivalent transition from state i to a state j ($j \neq k+1$) reading the same symbol a. In what follows, such states i are called possibly final states.

We need the same variables as a k_NFA to model a $(k+1)$_NFA$^+$, together with some extra variables, each one related to state $k+1$: a variable to fix that state $k+1$ is final, some variables for the new transitions incoming to state $k+1$, and some new variables to build paths going to state $k+1$. These variables are linked with the following new constraints:

- No state is final, except state $k+1$ which is the only final state:

$$\bigwedge_{i \in K} (\neg f_i) \wedge f_{k+1} \tag{9}$$

  Note that this constraint will mainly impact instances generated with the suffix model. Indeed, some steps of unit propagation will be achieved directly.

- There is no outgoing transition from the $(k+1)$_NFA$^+$ final state:

$$\bigwedge_{a \in \Sigma} \bigwedge_{i \in K_+} \neg \delta_{a,\overrightarrow{k+1,i}} \tag{10}$$

- Each incoming transition from state i to the $(k+1)$_NFA$^+$ final state $k+1$ must also be a transition from i finishing in another state:

$$\bigwedge_{a \in \Sigma} \left(\bigwedge_{i \in K} \left(\delta_{a,i,\overrightarrow{k+1}} \rightarrow \bigvee_{j \in K} \delta_{a,\overrightarrow{i,j}} \right) \right) \tag{11}$$

As said before, this 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 the reduction algorithm presented in Fig. 1. The worst-case complexity of this algorithm is in $O(k|S|)$ and for a $(k+1)$_NFA$^+$ obtained by P^R_{k+1} model[8] the algorithm will always succeed.

However, we cannot always reduce a $(k+1)$_NFA$^+$ into a k_NFA [24]. In fact, we experimentally observed that the reduction algorithm (Fig. 1) rarely succeeds. Indeed, a $(k+1)$_NFA$^+$ instance mainly provides information when it is unsatisfiable, because then we also know that there is no k_NFA.

[8]The P^R_{k+1} model is the reducible prefix model. See [24] for details.

Algorithm ReduceToKNFA *(S, $p_{w,\overline{1,i}}$)*

1. *Based on the variables $p_{w,\overline{1,i}}$ for $i \in K$ and $w \in S^-$, determine the set of candidate final states as $Q^A \setminus \{q_j\}$, where $j \in K$ and $p_{w,\overline{1,j}} =$ **true** (i.e., states in which any negative example can be reached cannot be final).*

2. *If the set of candidate final states is empty, the algorithm stops without being able to compute a k_NFA.*

3. *Otherwise, given a non-empty set of candidate final states, for each word $w \in S^+$ test whether the word can be reached in any candidate state by investigating variables $p_{w,\overline{1,i}}$, for these states.*

4. *If the test in the previous step returns a negative result (i.e., there are words $w \in S^+$ that cannot be reached in any of the candidate final states), the algorithm stops, without being able to compute a k_NFA.*

5. *Otherwise, the k_NFA can be obtained by removing the transitions leading to state $(k+1)$ and setting all candidate states to be final.*

Figure 1: Sketch of our reduction algorithm.

3.3. The $(k+1)$_NFA* extension

Contrary to the first extension, the second model extension, the $(k+1)$_NFA* model, does not need any reduction algorithm. Indeed, the satisfiability (respectively unsatisfiability) of a $(k+1)$_NFA* implies the satisfiability (respectively unsatisfiability) of a k_NFA. Moreover, in case of satisfiability, the $(k+1)$_NFA* can always be reduced to a k_NFA by just removing the $k+1$ state and some transitions. Thus, this reduction has also no cost.

We can define a $(k+1)$_NFA* as a $(k+1)$_NFA$^+$ with some more properties on words and a new set of Boolean variables for possibly final states in the reduced k_NFA ($F^* = \{f_1^*, ..., f_k^*\}$). A $(k+1)$_NFA* is reduced to a k_NFA by removing state $k+1$ and all its incoming transitions. Moreover, the final states are chosen among the possible final states, by determining the f_i^* of $\{f_1^*, \ldots, f_k^*\}$ that 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:

$$\bigwedge_{i \in K} \left(f_i^* \rightarrow \bigwedge_{w \in S^-} \neg p_{w,\overline{1,i}} \right) \tag{12}$$

- A possibly final state must validate at least one positive word of S^+:

$$\bigwedge_{i \in K} \left(f_i^* \rightarrow \left(\bigvee_{va \in S^+} \bigvee_{j \in K} \left(p_{v,\overline{1,j}} \wedge \delta_{a,\overrightarrow{j,i}} \wedge \delta_{a,\overrightarrow{j,k+1}} \right) \right) \right) \tag{13}$$

- Each positive word must terminate in at least one possibly final state:

$$\bigwedge_{w \in S^+} \bigvee_{i \in K} (p_{w,\overline{1,i}} \wedge f_i^*) \tag{14}$$

As said before, we do not need any reduction algorithm in this case.

3.4. Complexity

Obviously, extensions $(k+1)$_NFA$^+$ and $(k+1)$_NFA* have worse complexity than the k_NFA model since they require one more state, some more transitions to this state, and some more constraints as defined before. Moreover, k new Boolean variables are required to define the possibly final states.

The extra constraints for constructing the extensions from $(k+1)$_NFA also increase the number of clauses and variables. Table 1 provides the cost in terms of variables and clauses, and some details about the arity of the generated clauses (unary, binary, and greater). Note that a blank cell corresponds to 0.

Table 1: Complexity in terms of variables and clauses for each new constraint allowing the definition of the $(k+1)$_NFA$^+$ (Constraints (9)–(11)) and the $(k+1)$_NFA* models (Constraints (9)–(14)).

	Variables	Clauses			
		total	unary	binary	n-ary
Constraint 9		$k+1$	$k+1$		
Constraint 10		$n(k+1)$	$n(k+1)$		
Constraint 11		nk			nk
Constraint 12		$k\|S^-\|$		$k\|S^-\|$	
Constraint 13	$k^2\|S^+\|$	$4k^2\|S^+\|+k$		$3k^2\|S^+\|$	$k^2\|S^+\|+k$
Constraint 14	$k\|S^+\|$	$(3k+1)\|S^+\|$		$2k\|S^+\|$	$(k+1)\|S^+\|$

Note that there are numerous unary and binary clauses that will help SAT solvers to be efficient. The global order of complexity remains unchanged or decreases: since there is now only one final state, we fall down to $\mathcal{O}(\sigma k^2)$ for building suffixes, and thus for the hybrids. As we will observe in Section 3.5, the implementation of these new constraints significantly lowers the complexity of many of the models ($\mathcal{O}(\sigma k^3)$ clauses to $\mathcal{O}(\sigma(k+1)^2)$ clauses) just by simplification.

3.5. Simplified models

Despite the additional complexity resulting from the extra variables and clauses, the $(k+1)$_NFA* extension allows us to simplify all other models by reducing the number of variables (f disappear) and constraints (Constraints (1) and (10) are removed and Constraints (2) and (3) are simplified). The final $(k+1)$_NFA* models that we use can be defined in terms of the following variables:

- a set of k Boolean variables, $F^* = \{f_1^*, \ldots, f_k^*\}$, determining whether state i is a possibly final state or not,

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

and constraints:

- simplified Constraint (2) due to the existence of only one final state $k+1$: $\bigwedge_{w \in S^+} p_{w,\overline{1,k+1}}$
- simplified Constraint (3) due to the existence of only one final state $k+1$: $\bigwedge_{w \in S^-} \neg p_{w,\overline{1,k+1}}$
- Constraint (11): $\bigwedge_{a \in \Sigma} \left(\bigwedge_{i \in K} \left(\delta_{a,\overrightarrow{i,k+1}} \rightarrow \bigvee_{j \in K} \delta_{a,\overrightarrow{i,j}} \right) \right)$
- Constraint (12): $\bigwedge_{i \in K} \left(f_i^* \rightarrow \bigwedge_{w \in S^-} \neg p_{w,\overline{1,i}} \right)$
- Constraint (13): $\bigwedge_{i \in K} \left(f_i^* \rightarrow \left(\bigvee_{va \in S^+} \bigvee_{j \in K} \left(p_{v,\overline{1,j}} \wedge \delta_{a,\overrightarrow{j,i}} \wedge \delta_{a,\overrightarrow{j,k+1}} \right) \right) \right)$
- Constraint (14): $\bigwedge_{w \in S^+} \bigvee_{i \in K} \left(p_{w,\overline{1,i}} \wedge f_i^* \right)$

The extension also affects positively the constraints used for path definitions, regardless of the way in which they are constructed (as outlined in Section 2.3). The most significant simplification is for the suffix model, where the space complexity changes from $\mathcal{O}(\sigma k^3)$ clauses to $\mathcal{O}(\sigma(k+1)^2)$ clauses. All other models, which at least partially use the suffixes, also undergo this complexity reduction.

4. Properties of the model extensions

The extensions are over-constrained models for generating $(k+1)$_NFA. This means, that some extra constraints are not required for learning a $(k+1)$_NFA, but in our case, these constraints are useful to reduce the generated $(k+1)$_NFA to a k_NFA. We can qualify the k_NFA$^+$ model as a "weak" extension because with it, only the existence of a k_NFA can be proved. On the other hand, $(k+1)$_NFA is a "strong" extension that proves the existence or not of a k_NFA.

4.1. The $(k+1)$_NFA$^+$ extension

Let us first present the properties of the weak extension.

Main property. Completing the k_NFA model with Constraints (9), (10), and (11) we obtain the following main property:

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

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

$$\nexists\ (k+1)\text{\_NFA}^+ \Rightarrow \nexists\ k\text{\_NFA} \tag{16}$$

Proof. The proof is based on the main idea: 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. For each final state of the k_NFA, each incoming transition is duplicated into a transition going to state $k+1$, with the same transition's start state and the same symbol. 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 the k_NFA final states.
3. Thus:
   - each positive word now terminates in state $k+1$,
   - and, a negative word cannot terminate in the state $k+1$.
4. Each final state of the k_NFA is now redundant with state $k+1$. Hence, 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 accepts positive words and rejects negative words.

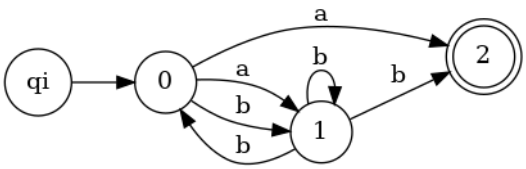

Figure 2: Example of a $(k+1)$_NFA$^+$ solution for $k = 3$ and $S = (\{a, ab, abb, bbb\}, \{aab, b, ba, bab, aaa\})$. It is not possible to find a k_NFA with $k = 2$.

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

$$\exists\ (k+1)\text{\_NFA}^+ \not\Rightarrow \exists\ k\text{\_NFA} \tag{17}$$

The simple example shown in Fig. 2 illustrates this property: a k_NFA$^+$ solution with $k = 3$ exists, whereas there is no k_NFA with two states ($k = 2$).

The reduction algorithm presented in Fig. 1 (and initially in [24]) computes a k_NFA from a $(k+1)$_NFA$^+$ solution. Unfortunately, as already mentioned, this algorithm rarely succeeds in computing a k_NFA.

4.2. The $(k+1)$_NFA* model extension

We can now observe the strong model extension and its more interesting properties. Note that this extension is built on the previous one.

Main property. Completing the $(k+1)$_NFA$^+$ model with Constraints (12), (13), and (14), we obtain the equisatisfiability between k_NFA and $(k+1)$_NFA*:

$$\exists\ k\text{\_NFA} \equiv \exists\ (k+1)\text{\_NFA}^\star \tag{18}$$

Proof. The idea of the proof is based on the fact that: there are some transformations from k_NFA to $(k+1)$_NFA*, and from $(k+1)$_NFA* to k_NFA preserving validation of positive words and rejections of negative words.

$\Rightarrow$ from k_NFA to $(k+1)$_NFA*

1. the proof is similar to the one in Section 4.1.

$\Leftarrow$ from $(k+1)$_NFA* to k_NFA

1. Each transition $\delta_{a,\overrightarrow{i,k+1}}$ is deleted, 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. Since there are some transitions to a possible final state redundant to transitions to state $k+1$, each positive word now terminates in at least a possible final state.
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. If there is no solution for a $(k+1)$_NFA*, this means that there are also no solutions for the corresponding k_NFA model. However, nothing can be deduced for the $(k+1)$_NFA model.

$$\nexists\ (k+1)\text{\_NFA}^\star \not\Rightarrow \nexists\ (k+1)\text{\_NFA} \tag{19}$$

It is easy to prove Property (19) with its contradiction and its rewriting using Property (18):

$$\begin{aligned} \exists\ (k+1)\text{\_NFA} &\not\Rightarrow \exists\ (k+1)\text{\_NFA}^\star \\ \exists\ (k+1)\text{\_NFA} &\not\Rightarrow \exists\ k\text{\_NFA} \end{aligned} \tag{20}$$

Property (20) is correct, otherwise, the existence of a $(k+1)$_NFA solution would imply the existence of a k_NFA solution. As a counterexample, Fig. 2 proposes a $(k+1)$_NFA with $k=3$ but no k_NFA with $k=2$ exists.

5. Experiments

We now present the results of a comprehensive experimental study of two equisatisfiable models, k_NFA and $(k+1)$_NFA*. The models were compared taking into account three different perspectives:

1. Models' performance in terms of the number of solutions and running times for k ranging from 1 to the $k_{\max}$ with $k_{\max} = \min(30, \text{size of the PTA built for set } S^+)$,

2. Models' ability to find NFA of small (possibly minimum) size—we compare the sizes of the smallest NFA found for each sample by each model,

3. Selected characteristics of found NFA established based on the random walks algorithm.

In the experiments, we use five different models for each of the k_NFA and $(k+1)$_NFA*, namely the best prefix model $P^\star$, the best suffix model $S^\star$ and three ILS-based models, $ILS(r)$, $ILS(P^\star)$, and $ILS(S^\star)$. For the $(k+1)$_NFA* we use the simplified models as described in Section 3.5.

5.1. Datasets

The experiments were conducted on three different datasets. They possess varying characteristics in terms of alphabet size, sample size, word length distribution within the sample, as well as the sizes of the PTAs.

The first dataset was composed of the samples described first in [13]—in what follows this dataset is termed STAMINA. The basic characteristics of all 30 samples contained in this dataset are gathered in Table 2. As can be observed the samples are based on alphabets of size 2, 5, and 10, with sample sizes ranging from 20 to 200 words split equally between sets S^+ and S^-. Thus, the samples are balanced in terms of S^+ and S^- sizes. Note also that the ww-10-30..99 samples possess the strict superset property, i.e., the larger samples are the strict supersets of the smaller ones. The st-* samples are not related to each other.

The second dataset was based on WaltzDB—a database of peptide sequences known to be amyloidogenic (harmful) or non-amyloidogenic [28, 29]. The samples were based on the subsets of peptides as defined in the database (see http://waltzdb.switchlab.org/sequences). This dataset is called PEPTIDES in the following sections. Samples' characteristics are collected in Table 3. Note that for each sample mentioned in the table we also generated sub-samples containing 10%, 30%, and 50% of the first peptides sequences. As can be seen, the alphabet sizes are much bigger than for the STAMINA dataset, the samples are also quite imbalanced.

Finally, the third dataset, REGEXES, was constructed based on regular expressions (regexp). For each sample we defined a regexp describing the words in S^+ and we generated randomly a given number of words having the lengths between 1 and 15 symbols. Set S^- was constructed by randomly shuffling the positive words and ensuring they do not match the regular expression. The regular expressions used were as follows:

Table 2: Characteristics of the STAMINA samples. Note that for the st-2-x samples, with x $= \{30, 40, \ldots, 100\}$ and st-5-y samples, with y $= \{20, 30, \ldots, 100\}$ we observed the same word length characteristics, so the rows were compacted. $|\Sigma|$ – size of alphabet, $|S^+|$, $|S^-|$ – sizes of the sets of positive and negative examples, $|w|$ – length of word.

Name	$\|\Sigma\|$	$\|S^+\|$	$\|S^-\|$	$\min_{w\in S^+} \|w\|$	$\max_{w\in S^+} \|w\|$	$\min_{w\in S^-} \|w\|$	$\max_{w\in S^-} \|w\|$
st-2-10	2	10	10	3	8	3	8
st-2-20	2	20	20	3	10	3	10
st-2-30..100	2	30..100	30..100	2	10	3	10
st-5-10	5	10	10	3	8	3	8
st-5-20..100	5	20..100	20..100	2	10	2	10
ww-10-10	10	10	10	2	6	2	7
ww-10-20	10	20	20	2	10	2	10
ww-10-30	10	30	30	2	4	2	4
ww-10-40	10	40	40	2	5	2	6
ww-10-50	10	50	50	2	6	2	7
ww-10-60	10	60	60	2	7	2	7
ww-10-70	10	70	70	2	8	2	8
ww-10-80	10	80	80	2	9	2	9
ww-10-90	10	90	90	2	10	2	10
ww-10-99	10	99	99	2	10	2	10

Table 3: Characteristics of the PEPTIDES samples. Sample names correspond to the following subsets: AH – Amylhex, AMS – Apoai mutant set, Fun – Functionals, FUS – FUS, Lin – Lindquist, Lit – Literature, NC – Newcores, Seb – Seb set, SOD1 – SOD1, TMS – Tau mutant set, TDP43 – TDP43. $|\Sigma|$ – size of alphabet, $|S^+|$, $|S^-|$ – sizes of the sets of positive and negative examples, $|w|$ – length of word.

Name	$\|\Sigma\|$	$\|S^+\|$	$\|S^-\|$	$\min_{w\in S^+} \|w\|$	$\max_{w\in S^+} \|w\|$	$\min_{w\in S^-} \|w\|$	$\max_{w\in S^-} \|w\|$
AH	19	79	121	6	6	6	6
AMS	20	79	36	6	6	6	6
Fun	19	11	126	6	6	6	6
FUS	18	1	48	6	6	6	6
Lin	20	3	102	6	6	6	6
Lit	20	204	66	5	6	6	6
NC	18	32	15	6	6	6	6
Seb	20	4	46	6	6	6	6
SOD1	17	8	22	6	6	6	6
TMS	20	92	22	6	6	6	6
TDP43	15	2	92	6	6	6	6

- regex1: $(0|11)(001|000|10)^*0$,
- regex2: $[0-4][0-4][0-4](012|123|234)^*(0|1)$,

- regex3: $[0-9][0-4][5-9](024|135|(98|87))^*(0|6)$,
- regex4: $[0-4]^*[5-9]^*(024|135|(98|87))^*(0|6)$,
- regex5: $[0-2]^*[3-4](0|2)^*(0|1|4)$.

Table 4 lists the basic parameters of this dataset. Similarly to the PEPTIDES dataset, we also generated sub-samples containing 10%, 30%, and 50% of the words in each sample. Due to the way the negative examples were produced, the set is balanced in terms of S^+ and S^- sizes.

Table 4: Characteristics of the REGEXES samples. $|\Sigma|$ – size of alphabet, $|S^+|$, $|S^-|$ – sizes of the sets of positive and negative examples, $|w|$ – length of word.

Name	$\|\Sigma\|$	$\|S^+\|$	$\|S^-\|$	$\min_{w\in S^+}\|w\|$	$\max_{w\in S^+}\|w\|$	$\min_{w\in S^-}\|w\|$	$\max_{w\in S^-}\|w\|$
regex1	2	100	100	2	15	3	15
regex2	5	100	100	4	13	4	13
regex3	10	100	100	4	15	4	15
regex4	10	100	100	4	15	2	15
regex5	5	100	100	3	15	3	15

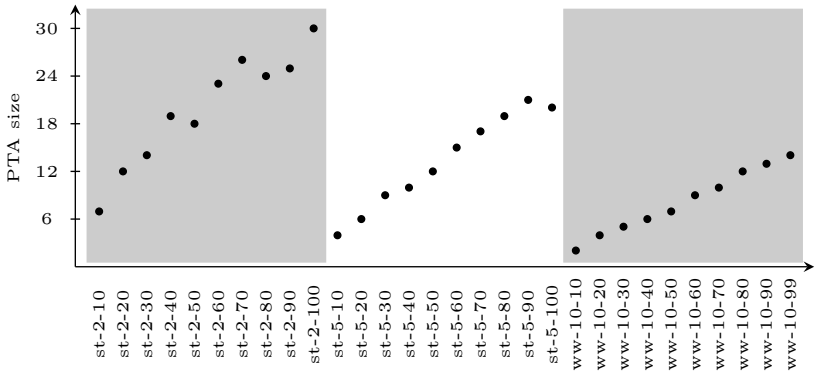

Figure 3: Sizes of prefix tree acceptors built for all STAMINA samples.

To observe further differences between the datasets and the challenges they could pose for the SAT models we analyzed the sizes of PTAs built for each sample. These sizes are depicted in Figs. 3, 4, and 5, respectively for the STAMINA, PEPTIDES, and REGEXES datasets. As can be seen, the samples belonging to different datasets differ significantly from each other in terms of PTA sizes. For STAMINA samples we observe approximately linear

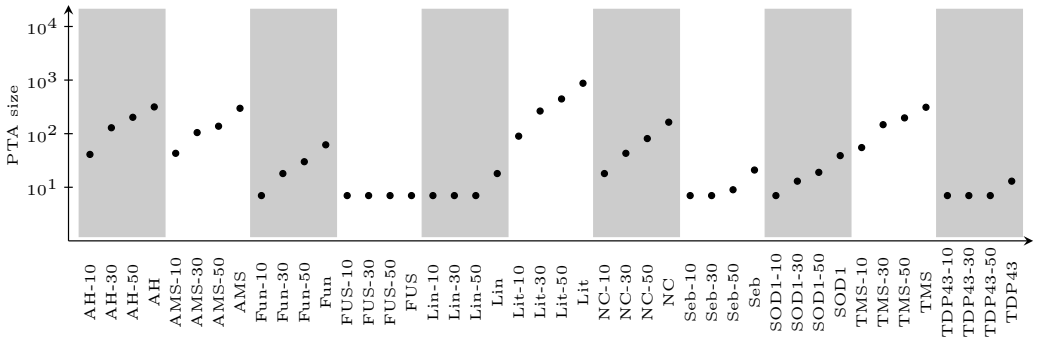

Figure 4: Sizes of prefix tree acceptors built for all PEPTIDES samples. Sample names ending in '-10', '-30' and '-50' represent the 10%, 30%, and 50% sub-samples. Note the logarithmic scale on the Y axis.

increase in the size of the PTAs[9]. The linear trend may be attributed to the strict superset property in case of ww-10-30..99 samples, and to the almost invariant characteristics of the samples st-2-* and st-5-*. For the PEPTIDES samples we either observe no change in PTA size or an exponential increase (note the logarithmic scale on the Y axis). The lack of PTA size change results from the sample characteristics—the number of positive examples on which the PTA is based is very small, and consequently almost the same number of positive examples is included in each sub-sample.

5.2. Random walks algorithm

Apart from being nondeterministic, the generated automata can also contain cycles or loops[10], which make the languages accepted by them infinite. Therefore, it is not feasible to generate a set of all the words accepted by such automata, in order to compare their acceptance capabilities. Thus, to evaluate the obtained NFA we designed a simple random walks algorithm with the aim of capturing the behavior of the NFA based on a finite subset of words. The algorithm starts from the initial state and chooses one of the outgoing transitions at random with uniform probability. We continue this process until one of the following stop conditions is met:

[9]Note that for STAMINA samples we used the PTA size after the application of randomized state merging algorithm as reported in [13].

[10]Note that *all* the NFA generated in our experiments contained at least one cycle.

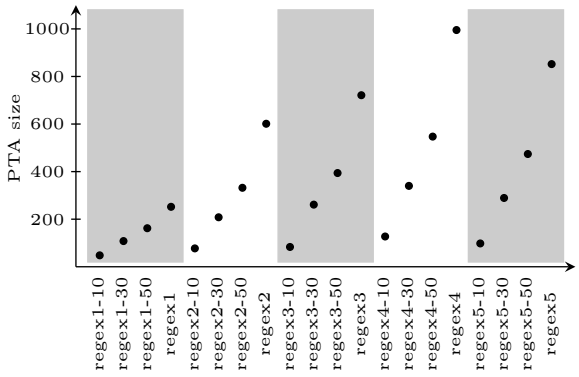

Figure 5: Sizes of prefix tree acceptors built for all REGEXES samples. Sample names ending in '-10', '-30' and '-50' represent the 10%, 30%, and 50% sub-samples.

SC1: A final (accepting) state was reached. If the state has no outgoing transitions, the walk stops here. Otherwise, the walk either continues or stops with a certain probability β (in the experiments $\beta = 0.3$ was used).

SC2: A state (either final or non-final) without outgoing transitions was reached—no further walks are possible. Note that this condition implies that dead-end states may exist in the inferred automata. It results from the SAT solver focusing on finding a solution to a CNF formula generated for a given number of states k, which may not be the smallest k possible for the given sample. If a solution containing such states is found, they can be removed thus producing a smaller NFA.

SC3: A maximum allowed length of a word was reached. The maximum length is sample-dependent and corresponds to the length of the longest word in the sample (see Tables 2, 3, and 4).

Note that condition SC1 may work differently depending on the used model, i.e., for the $(k+1)$_NFA* the only final state that exists has no outgoing transitions, so the probability β is irrelevant. For the k_NFA model however, we can either continue building the path for a word, or we can stop at the given state and accept the word built so far. Note that the same applies to the k_NFA obtained after reduction from $(k+1)$_NFA*. Condition SC2 naturally always ends the walk in all the models. Finally, condition SC3 was added to consider only words of similar characteristics to the ones in the

sample, and to avoid infinite walks.

Given the above conditions, the lengths of accepted words may vary between 1 and the maximum allowed word length, whereas the lengths of rejected words depend on the existence of condition SC2. If it is effective, i.e., the automaton contains dead-end states, then rejected words' lengths may also vary between 1 and the maximum length. Otherwise, the rejected words will always have the maximum allowed length.

For each NFA, the random walking was performed T times (in the experiments $T = 1000$ was used). For a set of T walks, the following metrics were then collected:

- Number of unique words U_w – since some of the walks could have ended up reading the same words (either using the same or different paths) we counted the number of words excluding duplicates. Given two automata with metrics values U_{w1} and U_{w2}, such that $U_{w1} \ll U_{w2}$, we can conclude that the latter automaton is more diversified and thus has greater generalization and acceptance capabilities than the former one.

- Number of unique word paths U_p – similarly to U_w metric, this one also takes into consideration the fact that some walks may have used the same paths. Note that $U_p \geq U_w$ is always true, since it is possible that the same word was read on more than one path. If the number of unique word paths is low, it may indicate that the automaton is more deterministic and thus, the same path is selected multiple times during the random walks.

- Number of words with multiple unique paths M_{wp} – this metric gives us a hint on how many words can be produced using more than a single path. In other words, it shows the nondeterminism of the automaton on a higher level, i.e., complete paths rather than single transitions.

- Number of SC1 stops N_{SC1} – this metric reflects the number of words accepted by the NFA. Note that if the walk ended due to SC2 or SC3 in a final state, we count it as an SC1 stop.

- Number of SC2 stops N_{SC2} – this metric reflects the number of times we reached a state without outgoing transitions that is not a final state.

A large value of this metric may indicate a significant number of dead-end states in the NFA. Consequently, the automaton's size could be reduced by removing these states.

- Number of SC3 stops N_{SC3} – this metric shows how often the algorithm wandered around the NFA without reaching the final state (or reaching it and escaping from it due to the existence of outgoing transitions in case of k_NFA model or the reduced $(k+1)$_NFA*).

To sum up, the random walks algorithm aims at providing some measurable statistics reflecting the behavior of the inferred NFA escaping the problem of generating all possible words of an infinite language accepted by the automaton.

5.3. NFA acceptance rate enhancements

Given an NFA for sample S, we know that it accepts positive words of the sample and rejects the negative ones. In this experiment, we want to investigate whether the NFA can be also globally more accepting or more rejecting. To address this, we propose a method for NFA densification, aimed at increasing the number of transitions, thereby mitigating potential biases in the generation process. We then proceed to investigate the effects of minimizing and maximizing the number of final states on the NFA's acceptance rates.

5.3.1. Densification algorithm

The generation of NFAs using SAT solvers can introduce a bias in the structure of the NFA due to the solver's preference for assigning value `false` rather than `true` to Boolean variables. This bias can affect the creation of certain transitions. To address this issue, we propose a strategy to densify the NFA by adding as many transitions as possible.

The objective is to respect the properties of the NFA (acceptance of positive words and rejection of negative words) while maximizing the number of transitions. This can only enhance the acceptability of the NFA. The principle is rather straightforward: if the incorporation of a transition does not lead to an NFA acceptance of a word from set S^-, then that transition is incorporated into the NFA. The order of inclusion does influence the final NFA, but in our case, we add the transitions starting from the initial state and following the numerical order of states. The alphabet is also processed in alphanumeric order.

5.3.2. Changing the set of final states

We identify NFAs with the minimal number of states k. The number of accepting states among these k states can also significantly impact the NFA's acceptance rates. Thus, we explore two approaches: minimization and maximization of the set of accepting states.

For minimization, we test all combinations of states that are smaller in size than the initial NFA's set of accepting states. We only consider a new set of accepting states if it respects NFA's properties (accepting positive words and rejecting negative words) and has the smallest possible size. Consequently, we are able to construct an NFA with a size of k and the smallest number of accepting states.

For maximization, we follow a similar process, but we test combinations of states that are strictly larger in size than the initial NFA's set of accepting states.

5.4. Experimental setup

The models were implemented in Python using the PySAT library [30]. 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 [31] SAT solver with default options. Due to its deterministic behavior, we run only one execution of each couple (instance, k), with k ranging from 1 to $k_{\max}$.

The random walks algorithm was implemented in Java and executed on a computer with Intel i7-7560U CPU and 8 GB of memory. The number of walks was set to $T = 1000$ and no time limit was assumed. It took around 720 s to execute the algorithm for all NFA. To ensure repeatability of the results, the pseudo-random number generator was initialized with a fixed seed value of 0. The densification algorithm as well as the accepting states sets modifications described in Sect. 5.3 were implemented in Python.

5.5. Results and discussion

5.5.1. Models' performance analysis

Tables 5, 6, and 7 show the results for k_NFA and $(k+1)$_NFA* with the 5 models described before for the three analyzed datasets. Each table is divided into two parts: instances description and results. Instances are described by the model and the average number of variables and clauses. As the results, we consider the number of satisfiable instances (Sat), unsatisfiable

instances (Unsat), and Unknown solutions (noted ?). Moreover, we give the average running time (in seconds). The running time corresponds to the sum of the modeling time and the SAT solver resolution time. If no result was obtained within the time limit, the running time of 900 seconds was assumed. Note that the bold value indicates the model with the minimum number of Unknown solutions.

Table 5: Results for k_NFA and $(k+1)$_NFA* on STAMINA instances. Each instance corresponds to the average for all samples with all k between 1 and $k_{\max}$. The bold value indicates the best model in terms of the number of unsolved instances.

Instances				Results			
Model		Vars	Clauses	Sat	Unsat	?	Time
k_NFA	$ILS(r)$	134,610	526,978	117	101	195	465
	$ILS(P^\star)$	135,352	529,964	119	102	192	467
	$ILS(S^\star)$	59,246	237,169	158	104	181	425
	$P^\star$	450,030	1,716,896	85	92	236	467
	$S^\star$	59,696	238,750	147	104	182	425
$(k+1)$_NFA*	$ILS(r)$	143,335	558,587	143	104	166	403
	$ILS(P^\star)$	144,631	563,706	142	104	167	406
	$ILS(S^\star)$	63,429	252,733	175	107	**161**	389
	$P^\star$	459,938	1,735,419	114	93	206	338
	$S^\star$	63,989	254,655	172	107	164	383

The tables show that applying $(k+1)$_NFA* models generally increases the size of the model both in terms of the number of variables and clauses. Nevertheless, the sizes remain close to each other and the relative differences are acceptable.

In terms of results, we can note that for STAMINA instances the number of unsolved instances drops from 181 for the $ILS(S^\star)$ k_NFA model to 161 achieved by the $ILS(S^\star)$ $(k+1)$_NFA* model. It is also worth noting that each of the $(k+1)$_NFA* models allows to solve from 18 to 30 more instances compared to its k_NFA equivalent. Furthermore, despite the increased sizes of the models, the running time decreases for all $(k+1)$_NFA* models.

The PEPTIDES dataset (cf. Table 6) is less favorable for the $(k+1)$_NFA* models. The results are often identical for k_NFA and $(k+1)$_NFA*, with some models showing slightly better performance. This may be attributed to the relative simplicity of the peptide samples having big alphabets, small word lengths and small NFA sizes. For these samples, the k_NFA models

are sufficient to find most of the NFA (note that only 7 instances remain unsolved).

Finally, for the REGEXES (cf. Table 7) we observe again a positive impact of the $(k+1)$_NFA* models (except for the $P^\star$ and $S^\star$ models) which allows us to reduce the number of unsolved instances by 8. We note a decrease in the running times (again except for the $S^\star$ model), although they generally remain very close to each other. Similarly to the STAMINA dataset, the ILS-based models tend to achieve the best results, with $ILS(S^\star)$ model being again the winner.

Table 6: Results for k_NFA and $(k+1)$_NFA* on PEPTIDES instances. Each instance corresponds to the average for all samples with all k between 1 and $k_{\max}$. The bold values indicate the best models in terms of the number of unsolved instances.

Instances				Results			
Model		Vars	Clauses	Sat	Unsat	?	Time
k_NFA	$ILS(r)$	114,249	431,757	862	11	14	30
	$ILS(P^\star)$	116,224	439,296	863	11	13	29
	$ILS(S^\star)$	103,895	391,925	869	11	**7**	22
	$P^\star$	327,260	1,245,545	720	11	156	168
	$S^\star$	319,378	1,218,815	751	11	125	137
$(k+1)$_NFA*	$ILS(r)$	118,469	445,595	864	11	12	27
	$ILS(P^\star)$	122,350	460,553	863	11	13	28
	$ILS(S^\star)$	110,992	417,222	869	11	**7**	22
	$P^\star$	346,580	1,311,033	727	11	149	161
	$S^\star$	329,763	1,252,342	749	11	127	139

To conclude, the $(k+1)$_NFA* models generally allow to reduce the number of unsolved instances of the problem, keeping or even lowering the time budget necessary to arrive at a solution. Hence, they should be considered as a good alternative to the k_NFA models.

5.5.2. Small size NFA analysis

To address the second aspect of comparison, namely the ability of the models to find small-sized NFA, we collected the minimum sizes of NFA found by each of the k_NFA and $(k+1)$_NFA* models. Let us recall that the existence of $(k+1)$_NFA* for a certain $k+1$ implies also the existence of a corresponding k_NFA. Thus, we can expect the following cases:

Table 7: Results for k_NFA and $(k+1)$_NFA* on REGEXES instances. Each instance corresponds to the average for all samples with all k between 1 and $k_{\max}$. The bold value indicates the best model in terms of the number of unsolved instances.

Instances				Results			
Model		Vars	Clauses	Sat	Unsat	?	Time
k_NFA	$ILS(r)$	360,014	1,397,255	465	26	109	218
	$ILS(P^\star)$	346,288	1,343,062	472	26	102	216
	$ILS(S^\star)$	281,128	1,095,440	521	26	53	138
	$P^\star$	412,111	1,565,067	313	26	261	412
	$S^\star$	475,580	1,834,252	387	26	187	313
$(k+1)$_NFA*	$ILS(r)$	367,142	1,421,137	474	26	100	204
	$ILS(P^\star)$	355,958	1,376,936	480	26	94	197
	$ILS(S^\star)$	284,430	1,105,832	529	26	**45**	129
	$P^\star$	424,728	1,599,539	308	26	266	405
	$S^\star$	464,506	1,785,121	378	26	196	325

$C_=$: The smallest NFA found by the k_NFA model has the size of k states, and the smallest NFA found by the $(k+1)$_NFA* model has the size $k+1$. We can say then that both models performed equally well.

C_+: The smallest NFA found by the k_NFA model has the size of k states, and the smallest NFA found by the $(k+1)$_NFA* model has the size of l such that $l \leq k$. This implies that an NFA of size $l-1$ exists as well and so there is an advantage of using $(k+1)$_NFA* (recall that we can directly obtain a k_NFA from a $(k+1)$_NFA* by removing state $k+1$ and its incoming transitions and marking the possibly final states as final).

C_-: The smallest NFA found by the k_NFA model has the size of k states, and the smallest NFA found by the $(k+1)$_NFA* model has the size greater than $k+1$. This implies that the k_NFA model enabled finding a smaller automaton than $(k+1)$_NFA* and so the use of the latter does not bring any value.

Apart from the above cases we may also consider situations in which either k_NFA or $(k+1)$_NFA* model failed to find any NFA for the given sample within the established time limit. These cases give advantage to the model which managed to find an NFA, if any was found at all. However, for simplicity we include these cases in the above ones.

Table 8: Comparison of the smallest NFA sizes found by k_NFA and $(k+1)$_NFA* models according to the evaluation cases $C_=$ (same size NFA found by both models), C_+ ($(k+1)$_NFA* model found smaller NFA), and C_- (k_NFA model found smaller NFA).

	STAMINA			PEPTIDES			REGEXES		
Model	$C_=$	C_+	C_-	$C_=$	C_+	C_-	$C_=$	C_+	C_-
$ILS(r)$	16	11	0	44	0	0	19	0	1
$ILS(P^\star)$	17	10	0	44	0	0	20	0	0
$ILS(S^\star)$	15	12	0	44	0	0	20	0	0
$P^\star$	16	11	0	44	0	0	20	0	0
$S^\star$	20	7	0	44	0	0	20	0	0
Total	**84**	**51**	**0**	**220**	**0**	**0**	**99**	**0**	**1**

Table 8 shows the performance of $(k+1)$_NFA* model vs. k_NFA model in terms of the aforementioned evaluation cases —the table columns are also labelled after them[11]. Note that for the STAMINA dataset, for three largest samples all the models failed to find any NFA, hence these samples are not included in the analysis.

As can be seen, the STAMINA dataset shows the most diverse characteristics when it comes to the performance of the models in terms of small-sized NFA. The PEPTIDES and REGEXES on the other hand do not benefit from the $(k+1)$_NFA* model, as there were no cases this model allowed us to obtain smaller NFA. However, PEPTIDES dataset is also the only one, in which the k_NFA model did not turn out to be better for any sample. The results in Table 8 clearly indicate that the proposed $(k+1)$_NFA* model is useful and acts on par or better than the k_NFA model in a vast majority of cases. What is more, a detailed analysis has shown that for all analyzed samples, there always exists a model which is able to achieve the same or better NFA size using the $(k+1)$_NFA* model than using the k_NFA model.

Comparing the performance of respective models we can see that $ILS(r)$ is the only model which failed to always achieve the same or better results for the $(k+1)$_NFA*. In turn, when we consider C_+ evaluation case, the $ILS(S^\star)$ model is the winner, being slightly better than the $P^\star$ and $ILS(r)$ models.

To sum up, if the goal of the inference is to find the minimal NFA for the

[11]Due to the amount of data, detailed smallest sizes of NFA found by the respective models can be found at `https://gitlab.com/tjastrzab/jcs2023`.

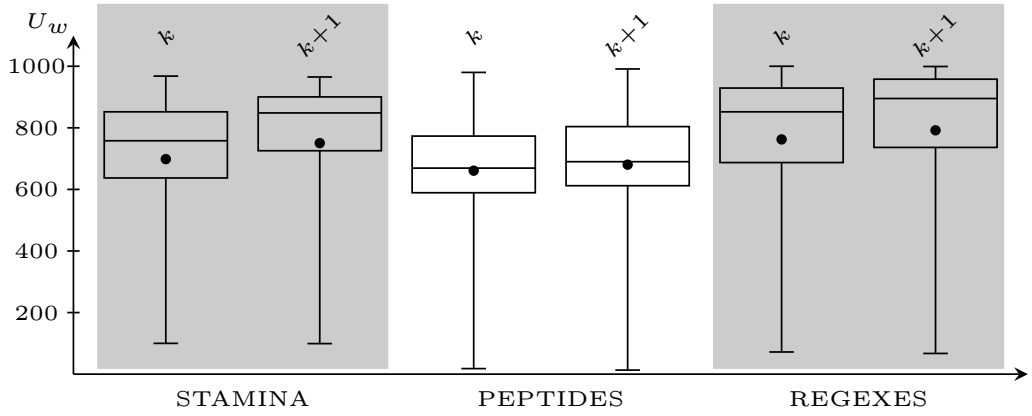

Figure 6: Number of unique words obtained within 1000 random walks for the STAMINA, PEPTIDES, and REGEXES sets and NFA obtained using $ILS(S^\star)$ models. Black dots represent average values.

given sample, it is profitable to apply the $(k+1)$_NFA* models, especially the $ILS(S^\star)$ model.

5.5.3. Random walks outcomes analysis

Let us now compare the metrics obtained by the NFA using the random walk algorithm described in Section 5.2. To make the analysis clearer, we decided to run the random walks algorithm only for the $ILS(S^\star)$ models, since they achieved the best results in the previous two experiments. Moreover, for each $(k+1)$_NFA* we determined the reduced k_NFA, so that all NFA can have multiple final states.

Figure 6 depicts the box plots along with the average values of the U_w metric. Based on these results, we established that the differences in the average number of unique words for the STAMINA and PEPTIDES datasets are not statistically significant at $\alpha = 0.05$. On the other hand, for the REGEXES dataset, we obtain significant differences at $\alpha = 0.05$ (based on the unpaired t-test, with $p = .0490$). Comparing the averages and the median values in Fig. 6, it is clear that the $(k+1)$_NFA* models produce typically more words than k_NFA models. Statistical analysis of the average values achieved by the $ILS(S^\star_k)$ and $ILS(S^\star_{k+1})$ models shows that they differ significantly (with $p = .0176$ at $\alpha = 0.05$). These observations indicate that the $(k+1)$_NFA* models are a bit denser in terms of the number of transitions, allowing more words to be read.

Looking at the extreme values, we can get as low as 13 unique words for the PEPTIDES, 67 for the REGEXES, and 99 for the STAMINA dataset. The maximum values are typically above 950 words, with some NFA achieving the value of 1000. This diversity of values indicates that depending on the size of the NFA, it may actually be more or less deterministic.

Figure 7 shows the values of the U_p metric for all the NFA. Similarly to the U_w metric we can observe a trend of lower average values obtained by the k_NFA-based models. The behavior observed for the extreme metric values varies slightly as compared to the previous metric, with higher—for STAMINA and REGEXES datasets—or equal minimum values obtained by U_p. This indicates that even when the number of read words was small, at least a portion of them appeared on different paths within the given NFA.

The conducted statistical analysis revealed that the average values of U_p metric differ significantly at $\alpha = 0.05$ for the STAMINA ($p = .0168$) and PEPTIDES ($p = .0161$) datasets, while this time we do not observe any statistical differences for the REGEXES dataset. Comparing the $ILS(S^\star_k)$ and $ILS(S^\star_{k+1})$ models across all datasets we also achieve statistically significant results ($p = .0004$).

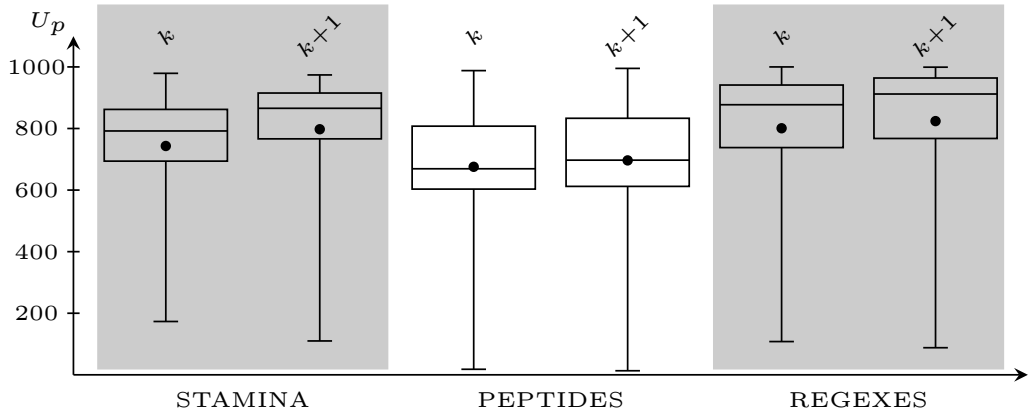

Figure 7: Number of unique word paths obtained within 1000 random walks for the STAMINA, PEPTIDES, and REGEXES sets and NFA obtained using $ILS(S^\star)$ models. Black dots represent average values.

The analysis of M_{wp} metric has shown that out of the total number of 2 946 NFA found, in 2 099 cases (71%) multiple paths were found for at least one unique word. Furthermore, in 1 262 cases (43%) at least one word possessed more than 2 unique paths. This shows that despite the partially

Table 9: Selected statistical parameters for the M_{wp} metric obtained for 1000 random walks. In the table, we consider the NFAs inferred by $ILS(S_k^\star)$ and $ILS(S_{k+1}^\star)$ models together. The Q1 and Q3 denote the first and the third quartile.

Dataset	Min	Q1	Median	Q3	Max
STAMINA	0	9	17	34	200
PEPTIDES	0	0	2	12	139
REGEXES	0	3	10	27	165

deterministic nature of the NFA, they still reflect their nondeterministic capabilities.

Table 9 shows the extreme values as well as the first (Q1), second (median), and third (Q3) quartile for M_{wp}. It is clear that regardless of the dataset, the number of words reachable by multiple paths is mostly less than 35. Moreover, around 10%–12% of M_{wp} metric values would be considered as outliers[12] for each dataset, based on the analysis of inter-quartile range (IQR). This means that obtaining large numbers of words reachable via multiple paths is unlikely for the inferred NFA.

Based on the above, we conclude that the $ILS(S_{k+1}^\star)$ model provides better diversity in terms of both the words and the paths identified using the random walks algorithm, as compared to the $ILS(S_k^\star)$ model.

The metrics related to the stop conditions SC1–SC3 will be discussed in the following section to enable the comparison of the "base" automata with the automata enhanced with densification and final states' number modifications.

5.5.4. Acceptance rates analysis

Similarly to the random walks analysis discussed before, the acceptance rate analysis for the models with minimized and maximized number of final states with and without densification was performed for the NFA inferred using $ILS(S^\star)$ models. The execution of the densification algorithm or changes to the number of final states was stopped after 2 hours if no result could be found. The algorithm for increasing the number of final states turned out to be particularly costly. Consequently, we failed to obtain the results for

[12] A value is considered an outlier when it falls below $Q1 - 1.5 \cdot IQR$ or above $Q3 + 1.5 \cdot IQR$, where $Q1$ and $Q3$ are the first and third quartile, and $IQR = Q3 - Q1$ is the inter-quartile range.

the $ILS(S_k^\star)$ model in case of STAMINA and REGEXES datasets. Since the main purpose of this research was to evaluate the impact on the number of final states and the number of transitions on the acceptance rates, we focus on the analysis of N_{SC1}–N_{SC3} metrics of the random walks algorithm.

Figures 8 to 10 depict the contribution of SC1–SC3 stop conditions, expressed as the percentage of unique word paths. In other words, the bars represent the values of $N_{SCi}/U_p \cdot 100\%$, for $i = 1, 2, 3$. Based on them we can notice a few things.

Firstly, we can clearly see that the increase in the number of accepting states allows us to achieve better acceptance rates, i.e., the contribution of SC1 stop condition always increases when we compare the bars labelled as Ext. or Den. & Ext. to the others bars. Specifically, the Ext. variant achieves the highest acceptance rates among all analyzed variants (within the set of NFAs obtained using $ILS(S_k^\star)$ or $ILS(S_{k+1}^\star)$ models, respectively), regardless of the dataset.

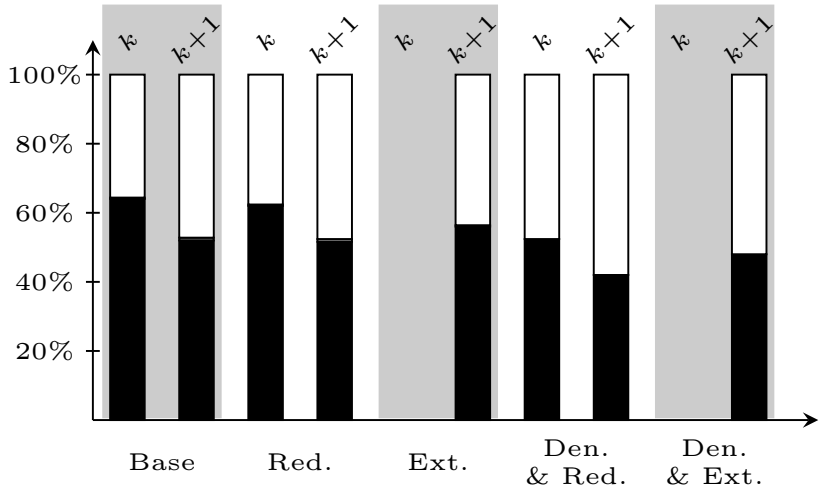

Figure 8: Distribution of SC1 (black), SC2 (gray), and SC3 (white) stop conditions for the STAMINA samples, and NFA inferred without any modification (Base), with final states number reduction (Red.), final states number increase (Ext.), final states number reduction with densification (Den. & Red.), and final states number increase with densification (Den. & Ext.).

Secondly, when we compare the results obtained by $ILS(S_k^\star)$ and $ILS(S_{k+1}^\star)$ models, we note that in most cases, the acceptance rates of the $(k+1)$_NFA* are lower—the exceptions involve Ext. and Den. & Ext. variants in which the

k_NFA is either unable to produce any outcome, or both models achieve the maximum acceptance rate of 100%. For the Base NFAs as well as reduction-based variants, the differences range between 2% to 12%.

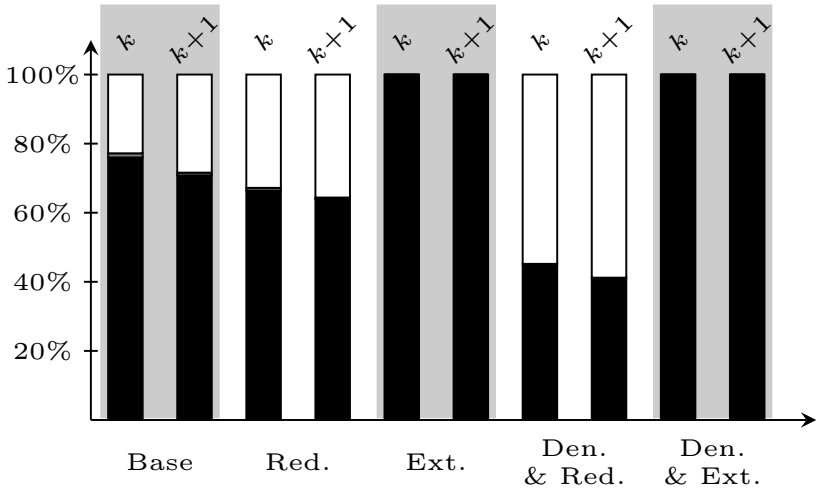

Figure 9: Distribution of SC1 (black), SC2 (gray), and SC3 (white) stop conditions for the PEPTIDES samples and NFA inferred without any modification (Base), with final states number reduction (Red.), final states number increase (Ext.), final states number reduction with densification (Den. & Red.), and final states number increase with densification (Den. & Ext.).

Thirdly, when we compare the densified and non-densified NFA (i.e., Red. vs. Den. & Red., or Ext. vs. Den. & Red.) we can see that the densification either lowers the acceptance rates or keeps them at the same level as before (only for PEPTIDES and Ext. vs. Den. & Red. variants). This trend can be observed regardless of the dataset and the model.

Finally, referring to the overall behavior of the NFA, they are typically more accepting than rejecting—the most extreme exception from this rule can be observed in Fig. 10 for the Den. & Red. variant where the acceptance rates drop to 28% and 25%, respectively for $ILS(S^{\star}_{k})$ and $ILS(S^{\star}_{k+1})$. Increasing the number of states has an expected, positive effect on the acceptance rates, reaching up to 59% increase for the PEPTIDES dataset for a comparison between Den. & Red. and Den. & Ext. variants of $(k+1)$_NFA*.

In terms of the other stop conditions, we can see that the contribution of SC2 condition is generally low, ranging from 0% to 7%, with the highest values achieved only for the REGEXES dataset. For the STAMINA and

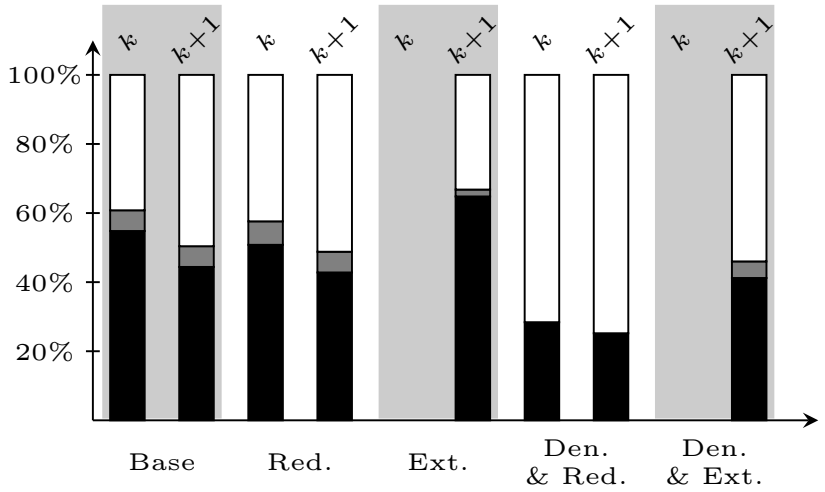

Figure 10: Distribution of SC1 (black), SC2 (gray), and SC3 (white) stop conditions for the REGEXES samples and NFA inferred without any modification (Base), with final states number reduction (Red.), final states number increase (Ext.), final states number reduction with densification (Den. & Red.), and final states number increase with densification (Den. & Ext.).

PEPTIDES datasets it does not reach levels higher than 1%. A deeper analysis of SC2 condition occurrences revealed that there were only 2 situations in which a dead-end state was present in the smallest automaton found by the $ILS(S^\star)$ models. It occurred once for the Base NFA, and once for the Red. variant. In both cases it applied to the st-2-70 sample. However, the NFA found was the *only* automaton found for this sample using $ILS(S^\star)$ models, and so also the smallest one. As a consequence, removing the dead-end state we could have achieved a smaller automaton. Note however, that a yet smaller automaton was found by the $ILS_{k+1}(r)$ model, so the overall goal of finding the minimal NFA was not affected by the dead-end state.

A key take-away from this experiment is that increasing the number of final states is enough to increase the acceptance rates of the automaton. When done in opposite way or is combined with other modifications (densification), the acceptance rates will certainly drop.

5.5.5. Length-dependent acceptance rates analysis

To finalize the analysis of acceptance rates we verified how they depend on the lengths of random walks performed. To this end we selected the Base and

Ext. variants and performed 1000 random walks for maximum word lengths equal to 1, 2, ..., 15. The Base variant was chosen because it represents the standard output of our inference models. The Ext. variant, in turn, was selected as the one providing the best acceptance rates so far. As before, we focused only on the NFA produced by $ILS(S^\star)$ models.

Figure 11 shows a linear trend of acceptance rates increase with increasing length of random walks—this trend is only not preserved in case of PEPTIDES dataset and the Ext. variant, which achieves a constant 100% acceptance rate regardless of walk lengths. This results from the fact that extended NFA for PEPTIDES have all their states final, because the negative examples typically contain symbols outside of the alphabet of S^+.

Moreover, we can observe that Ext. variant always achieves higher acceptance rates than its Base counterpart (compare white circles to white squares or the two-colored circles to squares of both types). This is consistent with the previous results and proves that increasing the number of final states makes the automata more accepting.

Finally, note that the values achieved by the analyzed models correspond to the values presented in Figs. 8 to 10, for $\max|w| = 10$ (STAMINA), $\max|w| = 6$ (PEPTIDES), and $\max|w| = 13, 15$ (REGEXES). This is because in the previous experiments we limited the maximum walk length to the maximum word lengths in the samples.

The main conclusion from the above analysis is that there is a linear correspondence between the acceptance rates and the length of the performed walk. Therefore, increasing the walk length further should allow achieving better acceptance rates, diverging however, from the characteristics of the initial sample.

6. Conclusion

Grammatical inference is the process of learning formal grammars, in our case as a nondeterministic finite automaton. In this paper, we proposed over-constrained models to infer an NFA of size $k+1$ ($(k+1)$_NFA*) with properties that allow deriving a solution for the classical model of size k (k_NFA). With the new models we take advantage of an important property, i.e., if a $(k+1)$_NFA* automaton exists, it can be directly 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.

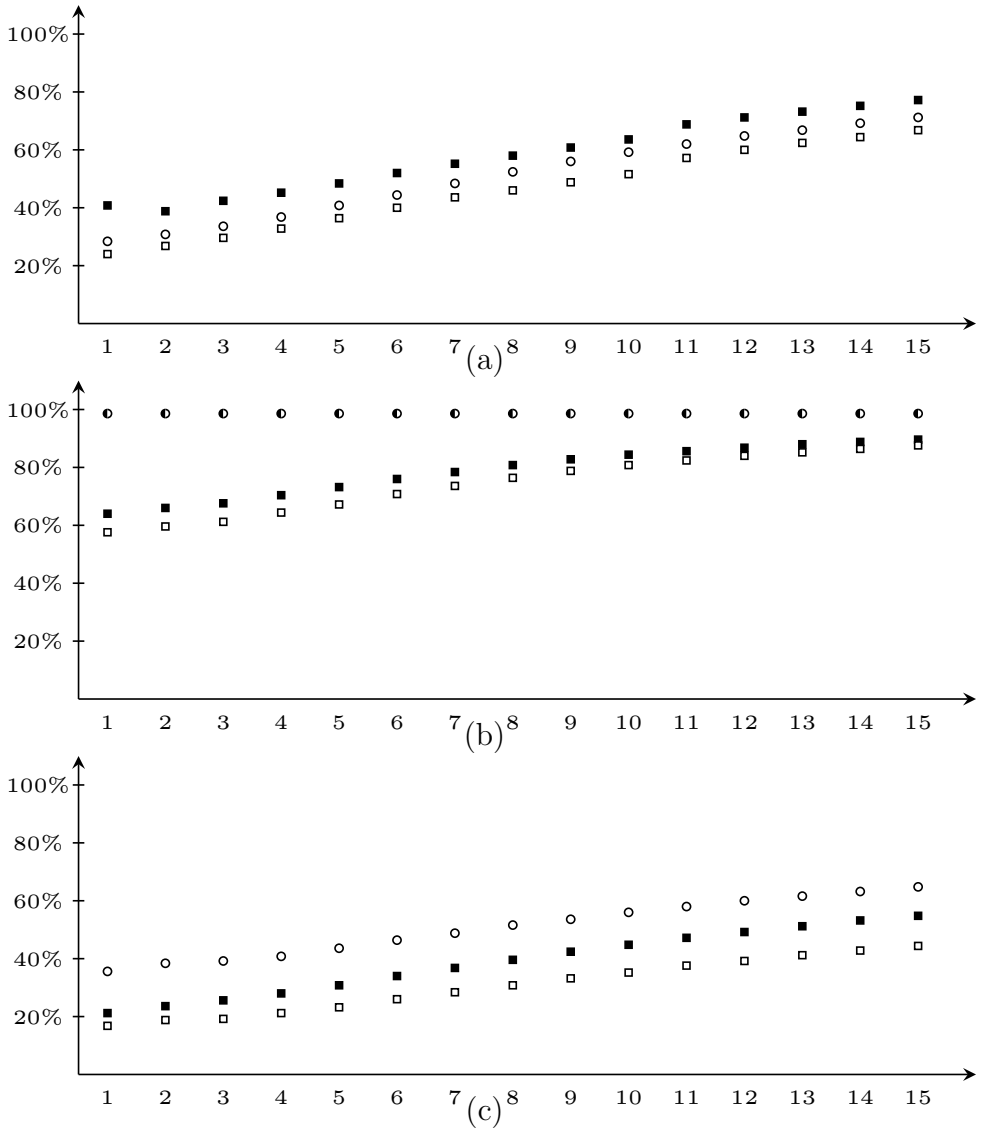

Figure 11: Acceptance rates of NFAs generated using $ILS(S^\star)$ models for the Base (squares) and Ext. (circles) variants and lengths of random walks varying between 1 and 15, for the STAMINA (a), PEPTIDES (b), and REGEXES (c) datasets. Black items denote k_NFA, while white items denote $(k+1)$_NFA*. The two-colored circles indicate that both models achieved exactly the same results.

Despite increased number of variables and clauses in the model of size $k + 1$, these additional constraints allow us to simplify the model and limit the combinatorial explosion. Furthermore, they allow us to lower the global complexity of most of our models that use a suffix construction.

Given an NFA, we can preserve its characteristics and reduce (or increase) the number of its accepting states. Moreover, we can also try to densify the automaton by adding transitions excluded from the NFA by the SAT solver bias. These modifications allow us to gain better control over the NFA's acceptability, i.e., it may become more accepting or more rejecting. Consequently, using the proposed models and post-processing techniques we obtain flexible NFAs, which can be adjusted to one's needs.

Finally, using random walks we can gain some insight into the structure and behavior of the inferred NFA that accept infinite languages. In this paper, we analyzed their capability of generating unique words and paths, as well as the possible reasons for random walk stops.

In the future, we plan to employ parallel computing capabilities to some of the models to further reduce the number of unsolved instances. Moreover, we plan to use the inferred NFA to not only generate new words, but also to classify them.

References

[1] T. Jastrzab, F. Lardeux, É. Monfroy, Inference of over-constrained NFA of size k+1 to efficiently and systematically derive NFA of size k for grammar learning, in: J. Mikyska, C. de Mulatier, M. Paszynski, V. V. Krzhizhanovskaya, J. J. Dongarra, P. M. A. Sloot (Eds.), Proc. of the 23rd ICCS Conference, Part I, volume 14073 of *LNCS*, Springer, 2023, pp. 134–147. doi:10.1007/978-3-031-35995-8_10.

[2] C. de la Higuera, Grammatical Inference: Learning Automata and Grammars, Cambridge University Press, 2010.

[3] F. Denis, A. Lemay, A. Terlutte, Learning regular languages using rfsas, Theor. Comput. Sci. 313 (2004) 267–294.

[4] M. Vázquez de Parga, P. García, J. Ruiz, A family of algorithms for non deterministic regular languages inference, in: Proc. of CIAA 2006, volume 4094 of *LNCS*, Springer, 2006, pp. 265–274.

[5] M. Gendreau, J.-Y. Potvin, Handbook of Metaheuristics, Springer, 2009.

[6] M. Tomita, Dynamic construction of finite-state automata from examples using hill-climbing., Proc. of the Fourth Annual Conference of the Cognitive Science Society (1982) 105–108.

[7] K. Apt, Principles of Constraint Programming, Cambridge University Press, 2003.

[8] F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, volume 2 of *Foundations of Artificial Intelligence*, Elsevier, 2006.

[9] W. Wieczorek, Grammatical Inference – Algorithms, Routines and Applications, volume 673 of *Studies in Computational Intelligence*, Springer, 2017.

[10] T. Jastrzab, On parallel induction of nondeterministic finite automata, in: Proc. of ICCS 2016, volume 80 of *Procedia Computer Science*, Elsevier, 2016, pp. 257–268.

[11] T. Jastrzab, Two parallelization schemes for the induction of nondeterministic finite automata on PCs, in: Proc. of PPAM 2017, volume 10777 of *LNCS*, Springer, 2017, pp. 279–289.

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

[13] T. Jastrzab, Z. J. Czech, W. Wieczorek, Parallel algorithms for minimal nondeterministic finite automata inference, Fundam. Informaticae 178 (2021) 203–227. doi:10.3233/FI-2021-2004.

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

[15] F. Lardeux, 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, 2021. URL: `https://hal.univ-angers.fr/hal-03284541`.

[16] S. Yu, Regular languages, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Volume 1: Word, Language, Grammar, Springer, 1997, pp. 41–110. doi:10.1007/978-3-642-59136-5_2.

[17] D. Angluin, Learning regular sets from queries and counterexamples, Information and Computation 75 (1987) 87–106. doi:10.1016/0890-5401(87)90052-6.

[18] A. Sanfeliu, N. P. de la Blanca, E. Vidal, Pattern Recognition and Image Analysis, WORLD SCIENTIFIC, 1992. doi:10.1142/1582.

[19] P. García, M. Vázquez de Parga, G. I. Álvarez, J. Ruiz, Universal automata and nfa learning, Theoretical Computer Science 407 (2008) 192–202. doi:https://doi.org/10.1016/j.tcs.2008.05.017.

[20] C. Lecoutre, N. Szczepanski, PYCSP3: modeling combinatorial constrained problems in python, CoRR abs/2009.00326 (2020). URL: https://arxiv.org/abs/2009.00326.

[21] C. Lecoutre, Ace, a generic constraint solver, 2023. arXiv:2302.05405.

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

[23] T. Jastrzab, F. Lardeux, É. Monfroy, Cclassifying words with 3-sort automata, in: 16th International Conference on Agents and Artificial Intelligence, ICAART 2024, SCITEPRESS, 2024. In press.

[24] T. Jastrzab, F. Lardeux, E. Monfroy, 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.

[25] F. Lardeux, E. Monfroy, 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, IEEE, 2021, pp. 396–403. doi:10.1109/ICTAI52525.2021.00065.

[26] T. Stützle, R. Ruiz, Iterated Local Search, Springer International Publishing, Cham, 2018, pp. 579–605. doi:10.1007/978-3-319-07124-4_8.

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

[28] J. Beerten, J. J. J. van Durme, R. Gallardo, E. Capriotti, L. C. Serpell, F. Rousseau, J. Schymkowitz, WALTZ-DB: a benchmark database of amyloidogenic hexapeptides, Bioinform. 31 (2015) 1698–1700.

[29] N. Louros, K. Konstantoulea, M. De Vleeschouwer, M. Ramakers, J. Schymkowitz, F. Rousseau, WALTZ-DB 2.0: an updated database containing structural information of experimentally determined amyloid-forming peptides, Nucleic Acids Research 48 (2019) D389–D393. doi:10.1093/nar/gkz758.

[30] A. Ignatiev, A. Morgado, J. Marques-Silva, PySAT: A Python toolkit for prototyping with SAT oracles, in: SAT, 2018, pp. 428–437. doi:10.1007/978-3-319-94144-8_26.

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