



**HAL**  
open science

# Robust models to infer flexible nondeterministic finite automata

Tomasz Jastrzab, Frédéric Lardeux, Eric Monfroy

► **To cite this version:**

Tomasz Jastrzab, Frédéric Lardeux, Eric Monfroy. Robust models to infer flexible nondeterministic finite automata. *Journal of computational science*, 2024, 79, pp.102309. 10.1016/j.jocs.2024.102309 . hal-04568947

**HAL Id: hal-04568947**

**<https://univ-angers.hal.science/hal-04568947v1>**

Submitted on 6 May 2024

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

## 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<sup>a</sup>, Frédéric Lardeux<sup>b</sup>, Eric Monfroy<sup>b</sup>

<sup>a</sup>*Silesian University of Technology, Gliwice, Poland*

<sup>b</sup>*LERIA, University of Angers, Angers, France*

---

## 1 Abstract

2 Grammatical inference involves learning a formal grammar as a finite state  
3 machine or a set of rewrite rules. This paper focuses on inferring nonde-  
4 terministic finite automata (NFA) from a given sample of words: the NFA  
5 must accept some words, and reject the others. To address this task we  
6 construct several over-constrained inference models capable of finding NFA  
7 of size  $k + 1$ , which are directly convertible to NFA of size  $k$ . Additionally,  
8 we propose an NFA evaluation method based on random walks along with  
9 two methods used to enhance the acceptance rates of the inferred NFA.  
10 The effectiveness of our approach is demonstrated through the results of  
11 comprehensive experiments conducted on several benchmark sets. This paper  
12 is an extended version of the ICCS 2023 paper entitled “Inference of over-  
13 constrained NFA of size  $k + 1$  to efficiently and systematically derive NFA of  
14 size  $k$  for grammar learning” [1].

*Keywords:* grammatical inference, nondeterministic automata, SAT models

---

## 15 1. Introduction

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

---

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

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

30 Our goal is to generate the smallest NFA, i.e., the goal is to minimize the  
31 number  $k$  of states in the automaton. This is typically done by determining  
32 lower (such as 1) and upper bounds (such as the size of a prefix tree acceptor<sup>1</sup>)  
33 on the number of states, and using some optimization algorithms taking  
34 advantage of the properties of the problem, to find the smallest possible  
35 number of states.

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

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

---

<sup>1</sup>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.

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

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

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

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

---

<sup>2</sup>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.

<sup>3</sup>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  $k$ .

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

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

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

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

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

---

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

## 121 2. Inference of automata: related works and SAT models

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

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

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

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

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

### 149 2.1. Why using SAT models for inferring NFA?

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

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

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

159 The pioneer work of Dana Angluin (e.g., [17]) is based on an oracle (called  
160 the Teacher) which can answer membership queries about a word, and can  
161 also test equivalence of 2 grammars (in case of negative answer, it also pro-  
162 vides a counterexample. Although a sample could replace the Teacher, it is  
163 not known “a priori” how many loops Angluin’s algorithm must perform to  
164 succeed in inferring the grammar, and thus, what should be the sample size.  
165 In our case, we want to be able to generate a grammar, even with a very  
166 small sample<sup>5</sup>.

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

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

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

---

<sup>5</sup>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.



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

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

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

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

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

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

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

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

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

## 252 *2.2. Meta-model*

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

### 258 *2.2.1. Notations*

259 We first need a few notations. For a given  $k$ , we also consider  $K =$   
260  $\{1, \dots, k\}$ , i.e., the set of the first  $k$  non-zero integers. The following integer  
261 and Boolean variables are the ones that appear in our meta-model:

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

- 263 • For each state  $i$ , we have one Boolean variable  $f_i$  determining whether  
 264 state  $i$  is final or not. We thus have the set  $F$  of  $k$  Boolean variables  
 265  $F = \{f_1, \dots, f_k\}$ .
- 266 • For each pair of states  $(i, j)$  ( $i$  possibly equal to  $j$ ), and each symbol  $a$   
 267 of  $\Sigma$  we have a Boolean variable  $\delta_{a, \overrightarrow{i, j}}$  determining whether there is a  
 268 transition from state  $i$  to state  $j$  with the symbol  $a$ . We thus obtain a  
 269 set  $\Delta = \{\delta_{a, \overrightarrow{i, j}} \mid a \in \Sigma \text{ and } (i, j) \in K^2\}$  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  $w = a_1 \dots a_m$ , 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:

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

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

### 275 2.2.2. The meta-model

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

- 280 • This constraint is only related to the empty word  $\lambda$ , and the status  
 281 of the initial state: if  $\lambda$  is a word of  $S^+$ , then the initial state must  
 282 be final; oppositely, if it is a negative word, the initial state cannot be  
 283 final:

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

- 284 • A positive word  $w$  from  $S^+$  must terminate in a final state of the  $k\_NFA$ ,  
 285 i.e., there must be a path from the initial state 1 to a final state  $i$  ( $f_i$   
 286 must be true) for the word  $w$ :

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

287 • Similarly, a negative word  $w$  must not terminate in a final state of the  
 288  $k$ -NFA, i.e., either there is no path for  $w$  from state 1 to a state from  
 289  $K$ , or each path for  $w$  from 1 to a state  $i$  implies that  $i$  is not a final  
 290 state:

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

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

### 293 2.3. From the meta-model to models

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

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

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

315 In [25], 7 different models were defined: the prefix model ( $P$ ), the suffix  
 316 model ( $S$ ), and several hybrid models combining prefix and suffix models with  
 317 different splitting strategies of words, such as the best prefix model ( $P^*$ ) to

---

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

318 optimize size and use of prefixes, the best suffix model ( $S^*$ ) to optimize size  
 319 and use of suffixes, and 3 hybrid models ( $ILS(Init)$ ) based on a local search  
 320 optimization [26] of word splittings (starting with an initial configuration  
 321  $Init$ , being either a random splitting of words, the splitting found by the  $P^*$   
 322 model, or by the  $S^*$  model).

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

- 329 • For the first symbol of the word  $\omega$ :

$$\bigvee_{i \in K} \delta_{a, \overline{1}, i} \leftrightarrow p_{a, \overline{1}, i} \quad (4)$$

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

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

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

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

- 339 • For the last symbol of the word  $\omega$ :

$$\bigvee_{i, j \in K} \delta_{a, i, \overline{j}} \leftrightarrow p_{a, i, \overline{j}} \quad (6)$$

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

- 342 • The path for each suffix  $w$  of  $\omega$  with  $w = av$  is built recursively:

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

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

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

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

$$p_{w,\bar{i},\bar{i}} \equiv \bigvee_{j \in K} p_{u,\bar{i},\bar{j}} \wedge p_{v,\bar{j},\bar{i}} \quad (8)$$

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

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

- **Best suffix model ( $S_k^*$ ):** 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  $S_k^*$  is based on an ordering of the set  $Suf(S)$ . Let

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

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

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

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

- 369 1.  $\bigcup_{v \in \mathcal{B}_s} \Omega(v) = S$ , i.e., each word of  $S$  has a suffix in  $\mathcal{B}_s$ ;
- 370 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  $\mathcal{B}_s$ ,  
371 at least one word of  $S$  has no suffix in  $\mathcal{B}_s$ .
- 372 3.  $\forall \mathcal{B}'_s \subseteq \text{Suf}(S)$ , if  $\mathcal{B}'_s \neq \mathcal{B}_s$  and  $\mathcal{B}'_s$  satisfies Conditions 1 and  
373 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  
374 largest set of suffixes regarding the  $\succ_s$  order, and respecting Con-  
375 ditions 1 and 2 above.

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

$$v_w^* \in \mathcal{B}_s \cap \text{Suf}(w) \text{ and } \forall v \in \text{Suf}(w), v_w^* \succ_s v$$

376 i.e., the suffix  $v_w^*$  for  $w$  is the largest suffix possible w.r.t.  $\succ_s$  in  $\mathcal{B}_s$ . Set  
377  $S_v$  will thus be  $\mathcal{B}_s$ . Then, for each  $w$ , the prefix is built as  $w = uv_w^*$ ,  
378 and the  $u$ 's constitute  $S_u$ .

- **Best prefix model ( $P_k^*$ ):** 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  $\text{Pref}(S)$ , the set of all prefixes of all words in  $S$ , then:

$$u_1 \succ_p u_2 \Leftrightarrow |u_1| * |A(u_1)| \geq |u_2| * |A(u_2)|$$

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

- 382 • **Local search optimization of the splitting ( $ILS_k(\text{Init})$ ):** we con-  
383 sider here that the search space corresponds to all the hybrid models,  
384 i.e., all the possible splits for all words of  $S$ . The fitness function we

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

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

### 408 3. Taking advantage of the problem characteristics: model exten- 409 sions

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

---

<sup>7</sup>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.



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

### 421 3.1. From $k$ \_NFA to $(k + 1)$ \_NFA

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

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

$$\begin{aligned} \forall_{i,j \in (Q^A)^2} \delta_{a,i,j}^A &\leftrightarrow \delta_{a,i,j}^{A'} \\ \forall_{i \in Q^A, j \in F^A} \delta_{a,i,j}^A &\leftrightarrow \delta_{a,i,k+1}^{A'} \end{aligned}$$

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

432 The obvious but important property for the rest of this paper is that the  
 433 language recognized by  $A'$  is the same as the one recognized by  $A$ .

434 *Sketch of the proof:*

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

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

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

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

458 To represent the fact that the new extensions contain one more state  
 459 than the initial  $k$ \_NFA, we name them  $(k + 1)$ \_NFA<sup>+</sup> and  $(k + 1)$ \_NFA<sup>\*</sup>. Of  
 460 course, new Boolean variables are added for the new final state and the new  
 461 transitions.

### 462 3.2. The $(k + 1)$ \_NFA<sup>+</sup> extension

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

465 A  $(k + 1)$ \_NFA<sup>+</sup> is a  $(k + 1)$ \_NFA with some extra properties to help to  
 466 reduce it to a  $k$ \_NFA by a reduction algorithm (as in [24]). The properties  
 467 that we enforce are:

- 468 • a  $(k + 1)$ \_NFA<sup>+</sup> has one and only one final state, i.e., state  $k + 1$ ;
- 469 • there is no outgoing transition for state  $k + 1$ ;
- 470 • each transition from a state  $i$  to state  $k + 1$  with symbol  $a$  has an  
 471 equivalent transition from state  $i$  to a state  $j$  ( $j \neq k + 1$ ) reading the  
 472 same symbol  $a$ . In what follows, such states  $i$  are called possibly final  
 473 states.

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

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

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

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

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

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

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

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

486 As said before, this first model extension, the  $(k+1)\_NFA^+$ , over-constrains  
 487 the  $(k+1)\_NFA$  to be able to reduce a generated  $(k+1)\_NFA$  into a  $k\_NFA$   
 488 with the reduction algorithm presented in Fig. 1. The worst-case complexity  
 489 of this algorithm is in  $O(k|S|)$  and for a  $(k+1)\_NFA^+$  obtained by  $P_{k+1}^R$   
 490 model<sup>8</sup> the algorithm will always succeed.

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

---

<sup>8</sup>The  $P_{k+1}^R$  model is the reducible prefix model. See [24] for details.

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

1. Based on the variables  $p_{w, \bar{1}, i}$  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  $p_{w, \bar{1}, j} = \mathbf{true}$  (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  $p_{w, \bar{1}, i}$ , 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.

495 *3.3. The  $(k + 1)$ \_NFA\* extension*

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

502 We can define a  $(k + 1)$ \_NFA\* as a  $(k + 1)$ \_NFA<sup>+</sup> with some more properties  
 503 on words and a new set of Boolean variables for possibly final states in the  
 504 reduced  $k$ \_NFA ( $F^* = \{f_1^*, \dots, f_k^*\}$ ). A  $(k + 1)$ \_NFA\* is reduced to a  $k$ \_NFA  
 505 by removing state  $k + 1$  and all its incoming transitions. Moreover, the  
 506 final states are chosen among the possible final states, by determining the  $f_i^*$   
 507 of  $\{f_1^*, \dots, f_k^*\}$  that are final states of the  $k$ \_NFA. To determine these final  
 508 states, we have to ensure:

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

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

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

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

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

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

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

513 *3.4. Complexity*

514 Obviously, extensions  $(k + 1)$ \_NFA<sup>+</sup> and  $(k + 1)$ \_NFA\* have worse com-  
 515 plexity than the  $k$ \_NFA model since they require one more state, some more  
 516 transitions to this state, and some more constraints as defined before. More-  
 517 over,  $k$  new Boolean variables are required to define the possibly final states.

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

Table 1: Complexity in terms of variables and clauses for each new constraint allowing the definition of the  $(k + 1)$ \_NFA<sup>+</sup> (Constraints (9)–(11)) and the  $(k + 1)$ \_NFA<sup>\*</sup> 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^+ $

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

### 530 3.5. Simplified models

531 Despite the additional complexity resulting from the extra variables and  
532 clauses, the  $(k + 1)$ \_NFA<sup>\*</sup> extension allows us to simplify all other models  
533 by reducing the number of variables ( $f$  disappear) and constraints (Con-  
534 straints (1) and (10) are removed and Constraints (2) and (3) are simplified).  
535 The final  $(k + 1)$ \_NFA<sup>\*</sup> models that we use can be defined in terms of the  
536 following variables:

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

- 539 • a set of  $nk(k+1)$  Boolean variables representing the transitions from  
540 state  $i$  to state  $j$  with the symbol  $a \in \Sigma$ :  $\Delta = \{\delta_{a,\vec{i},\vec{j}} | a \in \Sigma \text{ and } i \in$   
541  $K \text{ and } j \in K_+\}$ , and
- 542 • a set of Boolean variables representing the paths (sequence of transi-  
543 tions from a state  $i$  to a state  $j$  reading a word  $w$ ):  $\Pi = \{p_{w,\vec{i},\vec{j}} | (i, j) \in$   
544  $K_+^2, w \in \Sigma^*\}$

545 and constraints:

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

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

#### 560 4. Properties of the model extensions

561 The extensions are over-constrained models for generating  $(k+1)$ \_NFA.  
562 This means, that some extra constraints are not required for learning a  $(k+$   
563  $1)$ \_NFA, but in our case, these constraints are useful to reduce the generated  
564  $(k+1)$ \_NFA to a  $k$ \_NFA. We can qualify the  $k$ \_NFA<sup>+</sup> model as a “weak”  
565 extension because with it, only the existence of a  $k$ \_NFA can be proved. On  
566 the other hand,  $(k+1)$ \_NFA is a “strong” extension that proves the existence  
567 or not of a  $k$ \_NFA.

568 4.1. The  $(k + 1)$ \_NFA<sup>+</sup> extension

569 Let us first present the properties of the weak extension.

570 *Main property.* Completing the  $k$ \_NFA model with Constraints (9), (10), and  
571 (11) we obtain the following main property:

$$\exists k\_NFA \Rightarrow \exists (k + 1)\_NFA^+ \quad (15)$$

572 The contradiction allows us to prove the unsatisfiability of  $k$ \_NFA with  
573  $(k + 1)$ \_NFA<sup>+</sup>:

$$\not\exists (k + 1)\_NFA^+ \Rightarrow \not\exists k\_NFA \quad (16)$$

574 *Proof.* The proof is based on the main idea: there always exists a trans-  
575 formation from any  $k$ \_NFA to a  $(k + 1)$ \_NFA<sup>+</sup> which recognizes the same  
576 language.

- 577 1. A new state  $k + 1$  is added to the  $k$ \_NFA.
- 578 2. For each final state of the  $k$ \_NFA, each incoming transition is duplicated  
579 into a transition going to state  $k + 1$ , with the same transition's start  
580 state and the same symbol. Thus, a path  $p_{w,1,k+1}$  exists from the initial  
581 state to state  $k + 1$  if and only if a path exists from the initial state to  
582 the  $k$ \_NFA final states.
- 583 3. Thus:
  - 584 • each positive word now terminates in state  $k + 1$ ,
  - 585 • and, a negative word cannot terminate in the state  $k + 1$ .
- 586 4. Each final state of the  $k$ \_NFA is now redundant with state  $k + 1$ . Hence,  
587 state  $k + 1$  can be considered as the unique final state.
- 588 5. The automaton is then a  $(k + 1)$ \_NFA<sup>+</sup> since it has only one final state,  
589 and it accepts positive words and rejects negative words.



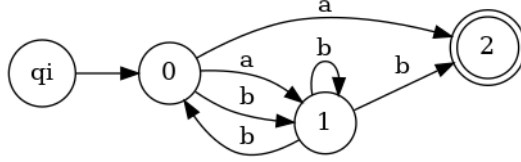


Figure 2: Example of a  $(k + 1)$ \_NFA<sup>+</sup> solution for  $k = 3$  and  $S = (\{a, ab, abb, bbb\}, \{aab, b, ba, bab, aaa\})$ . It is not possible to find a  $k$ \_NFA with  $k = 2$ .

590 *Minor property.* Whereas the lack of solutions for  $(k + 1)$ \_NFA<sup>+</sup> allows us to  
 591 conclude the lack of solutions for  $k$ \_NFA, a solution for  $(k + 1)$ \_NFA<sup>+</sup> is not  
 592 sufficient to affirm the existence of a  $k$ \_NFA:

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

593 The simple example shown in Fig. 2 illustrates this property: a  $k$ \_NFA<sup>+</sup>  
 594 solution with  $k = 3$  exists, whereas there is no  $k$ \_NFA with two states ( $k = 2$ ).

595 The reduction algorithm presented in Fig. 1 (and initially in [24]) com-  
 596 putes a  $k$ \_NFA from a  $(k + 1)$ \_NFA<sup>+</sup> solution. Unfortunately, as already  
 597 mentioned, this algorithm rarely succeeds in computing a  $k$ \_NFA.

#### 598 4.2. The $(k + 1)$ \_NFA<sup>\*</sup> model extension

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

601 *Main property.* Completing the  $(k + 1)$ \_NFA<sup>+</sup> model with Constraints (12),  
 602 (13), and (14), we obtain the equisatisfiability between  $k$ \_NFA and  $(k +$   
 603  $1)$ \_NFA<sup>\*</sup>:

$$\exists k\text{-NFA} \equiv \exists (k + 1)\text{-NFA}^* \tag{18}$$

604 *Proof.* The idea of the proof is based on the fact that: there are some trans-  
 605 formations from  $k$ \_NFA to  $(k + 1)$ \_NFA<sup>\*</sup>, and from  $(k + 1)$ \_NFA<sup>\*</sup> to  $k$ \_NFA  
 606 preserving validation of positive words and rejections of negative words.

607  $\Rightarrow$  from  $k$ \_NFA to  $(k + 1)$ \_NFA<sup>\*</sup>

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

609  $\Leftarrow$  from  $(k + 1)$ \_NFA<sup>\*</sup> to  $k$ \_NFA

- 610 1. Each transition  $\delta_{a,i,k+1}$  is deleted, and states  $j$  such that  $\delta_{a,i,k+1} \wedge$   
611  $\delta_{a,i,j}$  are now considered as final states.
- 612 2. State  $k + 1$  is removed.
- 613 3. Since there are some transitions to a possible final state redundant  
614 to transitions to state  $k + 1$ , each positive word now terminates in  
615 at least a possible final state.
- 616 4. No negative word can terminate in a possible final state. Other-  
617 wise, by construction, it would also finish in state  $k + 1$ .
- 618 5. The automaton is then a  $k$ \_NFA validating words of  $S^+$  and re-  
619 jecting words of  $S^-$ .

620 *Minor property.* If there is no solution for a  $(k + 1)$ \_NFA $^*$ , this means that  
621 there are also no solutions for the corresponding  $k$ \_NFA model. However,  
622 nothing can be deduced for the  $(k + 1)$ \_NFA model.

$$\nexists (k + 1)\text{-NFA}^* \not\Rightarrow \nexists (k + 1)\text{-NFA} \quad (19)$$

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

$$\begin{aligned} \exists (k + 1)\text{-NFA} &\not\Rightarrow \exists (k + 1)\text{-NFA}^* \\ \exists (k + 1)\text{-NFA} &\not\Rightarrow \exists k\text{-NFA} \end{aligned} \quad (20)$$

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

## 628 5. Experiments

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

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

- 635 2. Models’ ability to find NFA of small (possibly minimum) size—we com-  
636 pare the sizes of the smallest NFA found for each sample by each model,
- 637 3. Selected characteristics of found NFA established based on the random  
638 walks algorithm.

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

### 643 5.1. Datasets

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

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

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

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

Table 2: Characteristics of the STAMINA samples. Note that for the st-2-x samples, with  $x = \{30, 40, \dots, 100\}$  and st-5-y samples, with  $y = \{20, 30, \dots, 100\}$  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

671 • regex1: (0|11)(001|000|10)\*0,

672 • regex2: [0 – 4][0 – 4][0 – 4](012|123|234)\*(0|1),

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

676 Table 4 lists the basic parameters of this dataset. Similarly to the PEPTIDES  
 677 dataset, we also generated sub-samples containing 10%, 30%, and 50% of the  
 678 words in each sample. Due to the way the negative examples were produced,  
 679 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

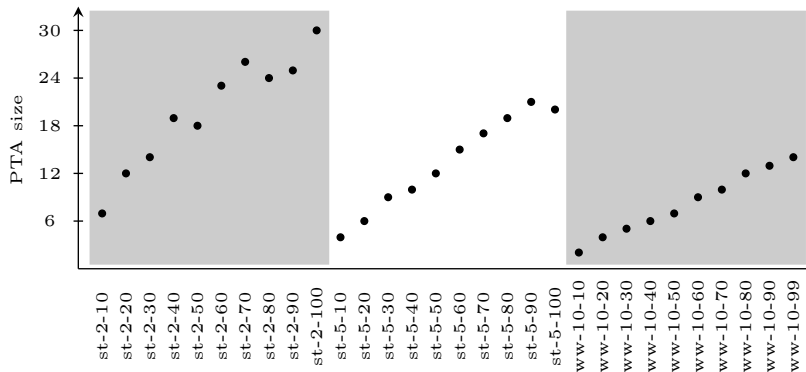


Figure 3: Sizes of prefix tree acceptors built for all STAMINA samples.

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

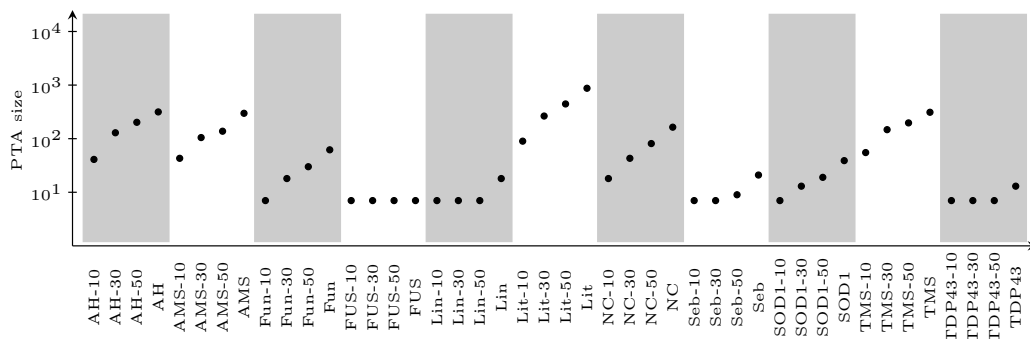


Figure 4: Sizes of prefix tree acceptors built for all PEPTIDES samples. Sample names ending in ‘-10’, ‘-30’ and ‘-50’ represent the 10%, 30%, and 50% sub-samples. Note the logarithmic scale on the Y axis.

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

## 694 5.2. Random walks algorithm

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

<sup>9</sup>Note that for STAMINA samples we used the PTA size after the application of ran-  
 domized state merging algorithm as reported in [13].

<sup>10</sup>Note that *all* the NFA generated in our experiments contained at least one cycle.

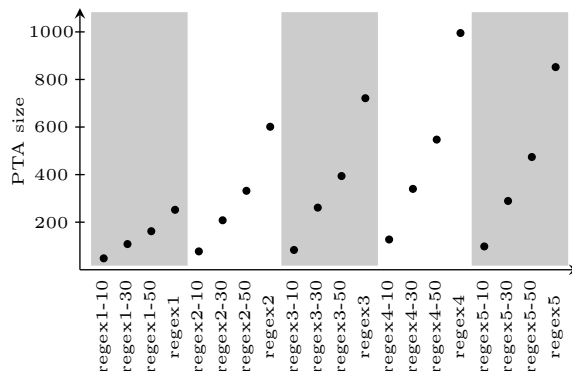


Figure 5: Sizes of prefix tree acceptors built for all REGEXES samples. Sample names ending in ‘-10’, ‘-30’ and ‘-50’ represent the 10%, 30%, and 50% sub-samples.

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

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

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

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

726 sample, and to avoid infinite walks.

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

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

- 736 • Number of unique words  $U_w$  – since some of the walks could have  
737 ended up reading the same words (either using the same or different  
738 paths) we counted the number of words excluding duplicates. Given  
739 two automata with metrics values  $U_{w1}$  and  $U_{w2}$ , such that  $U_{w1} \ll U_{w2}$ ,  
740 we can conclude that the latter automaton is more diversified and thus  
741 has greater generalization and acceptance capabilities than the former  
742 one.
- 743 • Number of unique word paths  $U_p$  – similarly to  $U_w$  metric, this one  
744 also takes into consideration the fact that some walks may have used  
745 the same paths. Note that  $U_p \geq U_w$  is always true, since it is possible  
746 that the same word was read on more than one path. If the number of  
747 unique word paths is low, it may indicate that the automaton is more  
748 deterministic and thus, the same path is selected multiple times during  
749 the random walks.
- 750 • Number of words with multiple unique paths  $M_{wp}$  – this metric gives  
751 us a hint on how many words can be produced using more than a single  
752 path. In other words, it shows the nondeterminism of the automaton  
753 on a higher level, i.e., complete paths rather than single transitions.
- 754 • Number of SC1 stops  $N_{SC1}$  – this metric reflects the number of words  
755 accepted by the NFA. Note that if the walk ended due to SC2 or SC3  
756 in a final state, we count it as an SC1 stop.
- 757 • Number of SC2 stops  $N_{SC2}$  – this metric reflects the number of times  
758 we reached a state without outgoing transitions that is not a final state.



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

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

766 To sum up, the random walks algorithm aims at providing some mea-  
767 surable statistics reflecting the behavior of the inferred NFA escaping the  
768 problem of generating all possible words of an infinite language accepted by  
769 the automaton.

### 770 5.3. NFA acceptance rate enhancements

771 Given an NFA for sample  $S$ , we know that it accepts positive words of  
772 the sample and rejects the negative ones. In this experiment, we want to  
773 investigate whether the NFA can be also globally more accepting or more re-  
774 jecting. To address this, we propose a method for NFA densification, aimed  
775 at increasing the number of transitions, thereby mitigating potential biases  
776 in the generation process. We then proceed to investigate the effects of min-  
777 imizing and maximizing the number of final states on the NFA’s acceptance  
778 rates.

#### 779 5.3.1. Densification algorithm

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

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

794 *5.3.2. Changing the set of final states*

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

799 For minimization, we test all combinations of states that are smaller in  
800 size than the initial NFA’s set of accepting states. We only consider a new set  
801 of accepting states if it respects NFA’s properties (accepting positive words  
802 and rejecting negative words) and has the smallest possible size. Conse-  
803 quently, we are able to construct an NFA with a size of  $k$  and the smallest  
804 number of accepting states.

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

808 *5.4. Experimental setup*

809 The models were implemented in Python using the PySAT library [30].  
810 The experiments were carried out on a computing cluster with Intel-E5-  
811 2695 CPUs, and a fixed limit of 10 GB of memory. Running times were  
812 limited to 15 minutes, including model generation and solving time. We used  
813 the Glucose [31] SAT solver with default options. Due to its deterministic  
814 behavior, we run only one execution of each couple (instance,  $k$ ), with  $k$   
815 ranging from 1 to  $k_{\max}$ .

816 The random walks algorithm was implemented in Java and executed on  
817 a computer with Intel i7-7560U CPU and 8 GB of memory. The number of  
818 walks was set to  $T = 1000$  and no time limit was assumed. It took around  
819 720 s to execute the algorithm for all NFA. To ensure repeatability of the  
820 results, the pseudo-random number generator was initialized with a fixed  
821 seed value of 0. The densification algorithm as well as the accepting states  
822 sets modifications described in Sect. 5.3 were implemented in Python.

823 *5.5. Results and discussion*

824 *5.5.1. Models’ performance analysis*

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

830 instances (Unsat), and Unknown solutions (noted ?). Moreover, we give the  
831 average running time (in seconds). The running time corresponds to the sum  
832 of the modeling time and the SAT solver resolution time. If no result was  
833 obtained within the time limit, the running time of 900 seconds was assumed.  
834 Note that the bold value indicates the model with the minimum number of  
835 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
$k\_NFA$	<i>ILS</i> ( $r$ )	134,610	526,978	117	101	195	465
	<i>ILS</i> ( $P^*$ )	135,352	529,964	119	102	192	467
	<i>ILS</i> ( $S^*$ )	59,246	237,169	158	104	181	425
	$P^*$	450,030	1,716,896	85	92	236	467
	$S^*$	59,696	238,750	147	104	182	425
$(k + 1)\_NFA^*$	<i>ILS</i> ( $r$ )	143,335	558,587	143	104	166	403
	<i>ILS</i> ( $P^*$ )	144,631	563,706	142	104	167	406
	<i>ILS</i> ( $S^*$ )	63,429	252,733	175	107	<b>161</b>	389
	$P^*$	459,938	1,735,419	114	93	206	338
	$S^*$	63,989	254,655	172	107	164	383

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

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

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

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

853 Finally, for the REGEXES (cf. Table 7) we observe again a positive impact  
 854 of the  $(k + 1)$ \_NFA\* models (except for the  $P^*$  and  $S^*$  models) which allows  
 855 us to reduce the number of unsolved instances by 8. We note a decrease in  
 856 the running times (again except for the  $S^*$  model), although they generally  
 857 remain very close to each other. Similarly to the STAMINA dataset, the  
 858 ILS-based models tend to achieve the best results, with  $ILS(S^*)$  model being  
 859 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
	$ILS(P^*)$	116,224	439,296	863	11	13	29
	$ILS(S^*)$	103,895	391,925	869	11	<b>7</b>	22
	$P^*$	327,260	1,245,545	720	11	156	168
	$S^*$	319,378	1,218,815	751	11	125	137
$(k + 1)$ _NFA*	$ILS(r)$	118,469	445,595	864	11	12	27
	$ILS(P^*)$	122,350	460,553	863	11	13	28
	$ILS(S^*)$	110,992	417,222	869	11	<b>7</b>	22
	$P^*$	346,580	1,311,033	727	11	149	161
	$S^*$	329,763	1,252,342	749	11	127	139

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

### 864 5.5.2. Small size NFA analysis

865 To address the second aspect of comparison, namely the ability of the  
 866 models to find small-sized NFA, we collected the minimum sizes of NFA  
 867 found by each of the  $k$ \_NFA and  $(k + 1)$ \_NFA\* models. Let us recall that the  
 868 existence of  $(k + 1)$ \_NFA\* for a certain  $k + 1$  implies also the existence of a  
 869 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
	$ILS(P^*)$	346,288	1,343,062	472	26	102	216
	$ILS(S^*)$	281,128	1,095,440	521	26	53	138
	$P^*$	412,111	1,565,067	313	26	261	412
	$S^*$	475,580	1,834,252	387	26	187	313
$(k + 1)\_NFA^*$	$ILS(r)$	367,142	1,421,137	474	26	100	204
	$ILS(P^*)$	355,958	1,376,936	480	26	94	197
	$ILS(S^*)$	284,430	1,105,832	529	26	<b>45</b>	129
	$P^*$	424,728	1,599,539	308	26	266	405
	$S^*$	464,506	1,785,121	378	26	196	325

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

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

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

885 Apart from the above cases we may also consider situations in which either  
886  $k\_NFA$  or  $(k + 1)\_NFA^*$  model failed to find any NFA for the given sample  
887 within the established time limit. These cases give advantage to the model  
888 which managed to find an NFA, if any was found at all. However, for sim-  
889 plicity 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).

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

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

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

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

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

---

<sup>11</sup>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>.

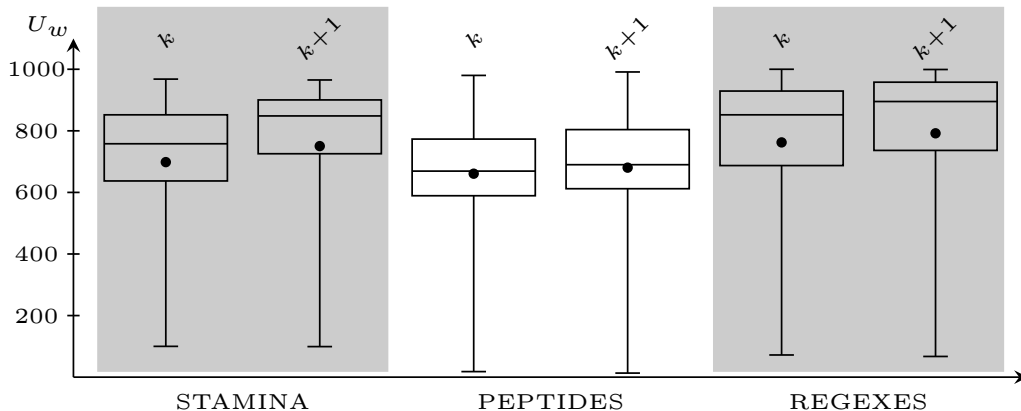


Figure 6: Number of unique words obtained within 1000 random walks for the STAMINA, PEPTIDES, and REGEXES sets and NFA obtained using  $ILS(S^*)$  models. Black dots represent average values.

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

### 914 5.5.3. Random walks outcomes analysis

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

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

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

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

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

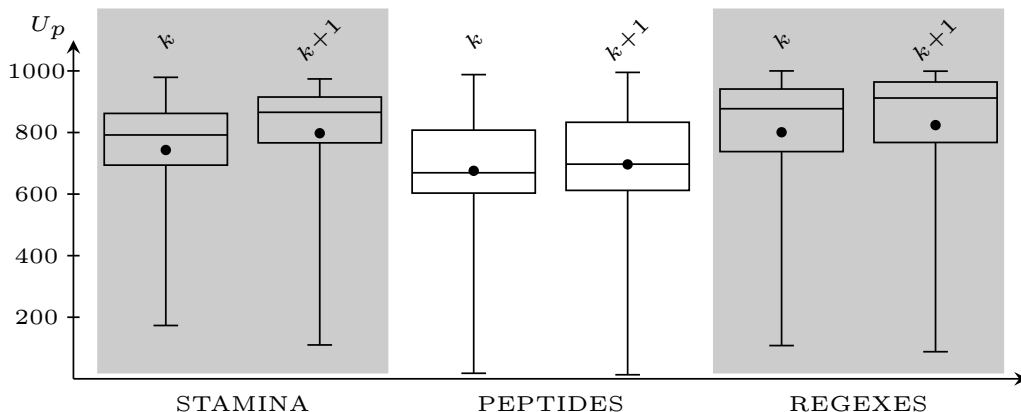


Figure 7: Number of unique word paths obtained within 1000 random walks for the STAMINA, PEPTIDES, and REGEXES sets and NFA obtained using  $ILS(S^*)$  models. Black dots represent average values.

951 The analysis of  $M_{wp}$  metric has shown that out of the total number of  
 952 2 946 NFA found, in 2 099 cases (71%) multiple paths were found for at  
 953 least one unique word. Furthermore, in 1 262 cases (43%) at least one word  
 954 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  $ILS(S_k^*)$  and  $ILS(S_{k+1}^*)$  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

955 deterministic nature of the NFA, they still reflect their nondeterministic ca-  
 956 pabilities.

957 Table 9 shows the extreme values as well as the first (Q1), second (me-  
 958 dian), and third (Q3) quartile for  $M_{wp}$ . It is clear that regardless of the  
 959 dataset, the number of words reachable by multiple paths is mostly less  
 960 than 35. Moreover, around 10%–12% of  $M_{wp}$  metric values would be con-  
 961 sidered as outliers<sup>12</sup> for each dataset, based on the analysis of inter-quartile  
 962 range (IQR). This means that obtaining large numbers of words reachable  
 963 via multiple paths is unlikely for the inferred NFA.

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

967 The metrics related to the stop conditions SC1–SC3 will be discussed in  
 968 the following section to enable the comparison of the “base” automata with  
 969 the automata enhanced with densification and final states’ number modifi-  
 970 cations.

#### 971 5.5.4. Acceptance rates analysis

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

---

<sup>12</sup>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.

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

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

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

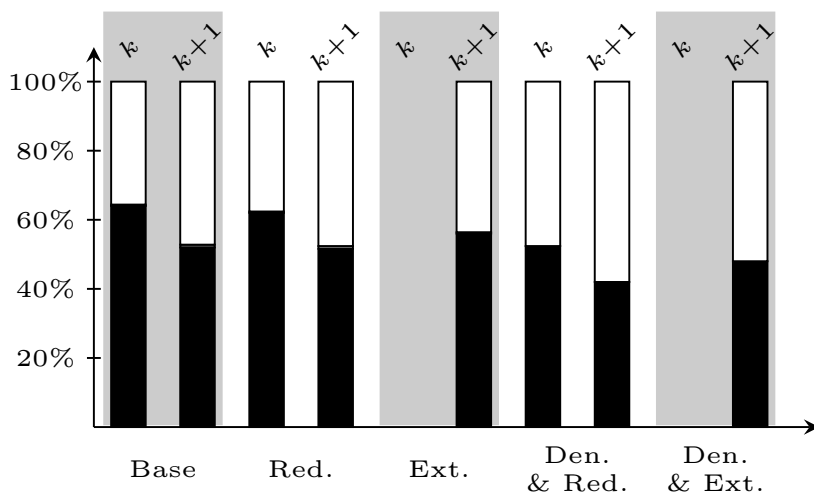


Figure 8: Distribution of SC1 (black), SC2 (gray), and SC3 (white) stop conditions for the STAMINA samples, and NFA inferred without any modification (Base), with final states number reduction (Red.), final states number increase (Ext.), final states number reduction with densification (Den. & Red.),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

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

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

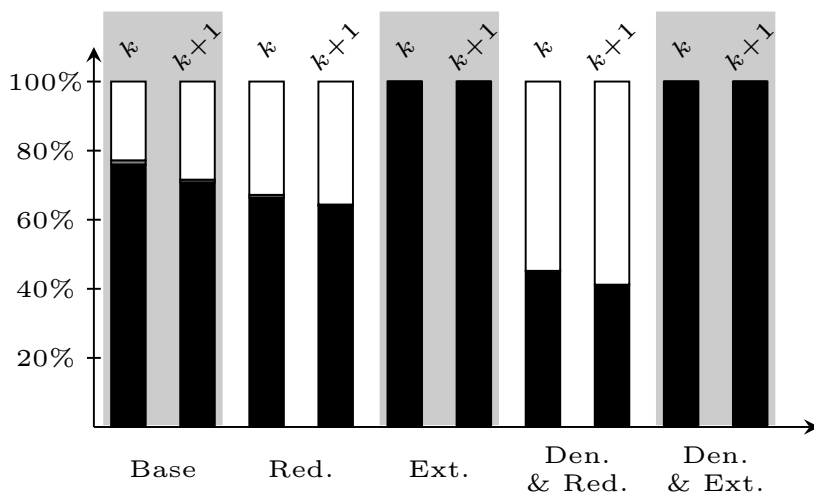


Figure 9: Distribution of SC1 (black), SC2 (gray), and SC3 (white) stop conditions for the PEPTIDES samples and NFA inferred without any modification (Base), with final states number reduction (Red.), final states number increase (Ext.), final states number reduction with densification (Den. & Red.),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

1000 Thirdly, when we compare the densified and non-densified NFA (i.e., Red.  
 1001 vs. Den. & Red., or Ext. vs. Den. & Red.) we can see that the densification  
 1002 either lowers the acceptance rates or keeps them at the same level as before  
 1003 (only for PEPTIDES and Ext. vs. Den. & Red. variants). This trend can  
 1004 be observed regardless of the dataset and the model.

1005 Finally, referring to the overall behavior of the NFA, they are typically  
 1006 more accepting than rejecting—the most extreme exception from this rule can  
 1007 be observed in Fig. 10 for the Den. & Red. variant where the acceptance rates  
 1008 drop to 28% and 25%, respectively for  $ILS(S_k^*)$  and  $ILS(S_{k+1}^*)$ . Increasing  
 1009 the number of states has an expected, positive effect on the acceptance rates,  
 1010 reaching up to 59% increase for the PEPTIDES dataset for a comparison  
 1011 between Den. & Red. and Den. & Ext. variants of  $(k + 1)$ \_NFA\*.

1012 In terms of the other stop conditions, we can see that the contribution  
 1013 of SC2 condition is generally low, ranging from 0% to 7%, with the highest  
 1014 values achieved only for the REGEXES dataset. For the STAMINA and

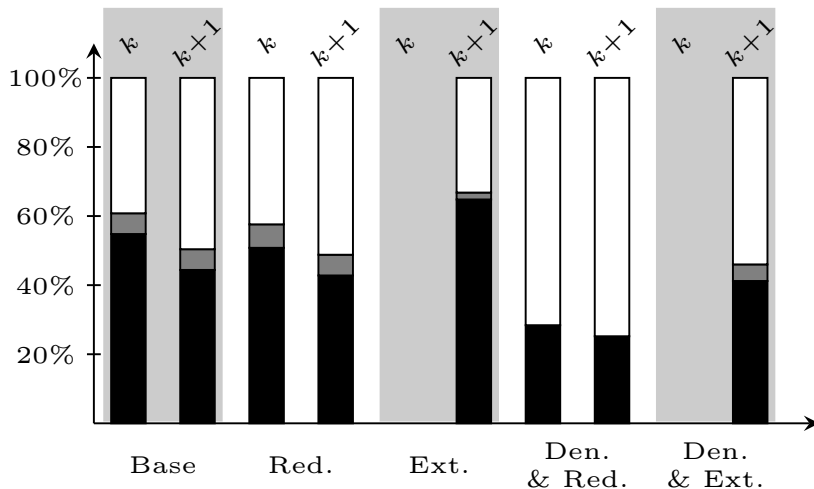


Figure 10: Distribution of SC1 (black), SC2 (gray), and SC3 (white) stop conditions for the REGEXES samples and NFA inferred without any modification (Base), with final states number reduction (Red.), final states number increase (Ext.), final states number reduction with densification (Den. & Red.),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

1015 PEPTIDES datasets it does not reach levels higher than 1%. A deeper anal-  
 1016 ysis of SC2 condition occurrences revealed that there were only 2 situations  
 1017 in which a dead-end state was present in the smallest automaton found by  
 1018 the  $ILS(S^*)$  models. It occurred once for the Base NFA, and once for the  
 1019 Red. variant. In both cases it applied to the st-2-70 sample. However, the  
 1020 NFA found was the *only* automaton found for this sample using  $ILS(S^*)$   
 1021 models, and so also the smallest one. As a consequence, removing the dead-  
 1022 end state we could have achieved a smaller automaton. Note however, that  
 1023 a yet smaller automaton was found by the  $ILS_{k+1}(r)$  model, so the overall  
 1024 goal of finding the minimal NFA was not affected by the dead-end state.

1025 A key take-away from this experiment is that increasing the number of fi-  
 1026 nal states is enough to increase the acceptance rates of the automaton. When  
 1027 done in opposite way or is combined with other modifications (densification),  
 1028 the acceptance rates will certainly drop.

### 1029 5.5.5. Length-dependent acceptance rates analysis

1030 To finalize the analysis of acceptance rates we verified how they depend on  
 1031 the lengths of random walks performed. To this end we selected the Base and

1032 Ext. variants and performed 1000 random walks for maximum word lengths  
1033 equal to 1, 2, ..., 15. The Base variant was chosen because it represents  
1034 the standard output of our inference models. The Ext. variant, in turn, was  
1035 selected as the one providing the best acceptance rates so far. As before, we  
1036 focused only on the NFA produced by  $ILS(S^*)$  models.

1037 Figure 11 shows a linear trend of acceptance rates increase with increas-  
1038 ing length of random walks—this trend is only not preserved in case of PEP-  
1039 TIDES dataset and the Ext. variant, which achieves a constant 100% accep-  
1040 tance rate regardless of walk lengths. This results from the fact that extended  
1041 NFA for PEPTIDES have all their states final, because the negative examples  
1042 typically contain symbols outside of the alphabet of  $S^+$ .

1043 Moreover, we can observe that Ext. variant always achieves higher accep-  
1044 tance rates than its Base counterpart (compare white circles to white squares  
1045 or the two-colored circles to squares of both types). This is consistent with  
1046 the previous results and proves that increasing the number of final states  
1047 makes the automata more accepting.

1048 Finally, note that the values achieved by the analyzed models correspond  
1049 to the values presented in Figs. 8 to 10, for  $\max |w| = 10$  (STAMINA),  
1050  $\max |w| = 6$  (PEPTIDES), and  $\max |w| = 13, 15$  (REGEXES). This is be-  
1051 cause in the previous experiments we limited the maximum walk length to  
1052 the maximum word lengths in the samples.

1053 The main conclusion from the above analysis is that there is a linear  
1054 correspondence between the acceptance rates and the length of the performed  
1055 walk. Therefore, increasing the walk length further should allow achieving  
1056 better acceptance rates, diverging however, from the characteristics of the  
1057 initial sample.

## 1058 6. Conclusion

1059 Grammatical inference is the process of learning formal grammars, in our  
1060 case as a nondeterministic finite automaton. In this paper, we proposed  
1061 over-constrained models to infer an NFA of size  $k + 1$  ( $(k + 1)$ \_NFA $^*$ ) with  
1062 properties that allow deriving a solution for the classical model of size  $k$   
1063 ( $k$ \_NFA). With the new models we take advantage of an important property,  
1064 i.e., if a  $(k + 1)$ \_NFA $^*$  automaton exists, it can be directly reduced to an  
1065 automaton of size  $k$ , and if it does not exist, we have the proof there is no  
1066 automaton of size  $k$ . The advantage of using the  $(k + 1)$ \_NFA $^*$  model is to  
1067 obtain shorter resolution times while increasing the rate of solved instances.

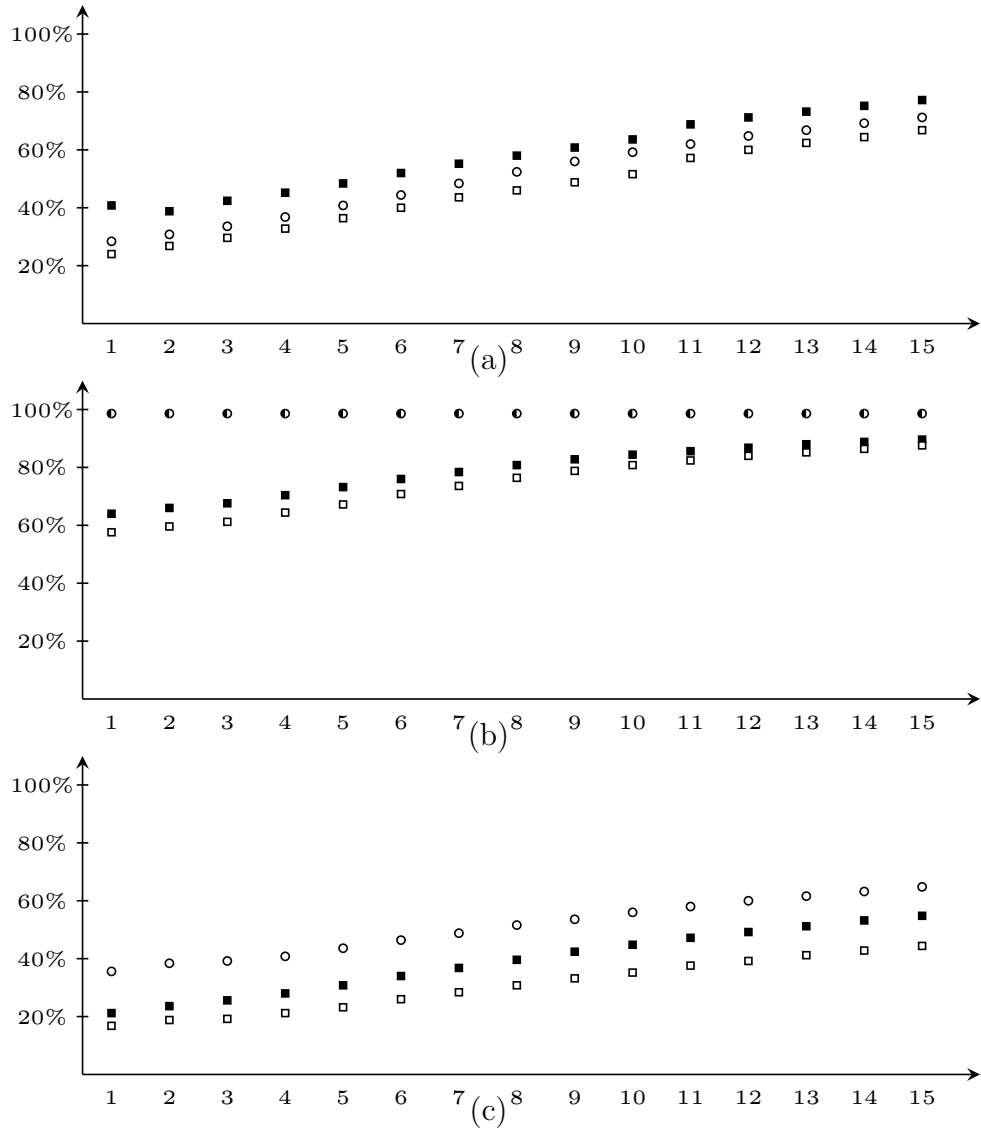


Figure 11: Acceptance rates of NFAs generated using  $ILS(S^*)$  models for the Base (squares) and Ext. (circles) variants and lengths of random walks varying between 1 and 15, for the STAMINA (a), PEPTIDES (b), and REGEXES (c) datasets. Black items denote  $k$ \_NFA, while white items denote  $(k +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

1068 Despite increased number of variables and clauses in the model of size  
1069  $k + 1$ , these additional constraints allow us to simplify the model and limit  
1070 the combinatorial explosion. Furthermore, they allow us to lower the global  
1071 complexity of most of our models that use a suffix construction.

1072 Given an NFA, we can preserve its characteristics and reduce (or increase)  
1073 the number of its accepting states. Moreover, we can also try to densify  
1074 the automaton by adding transitions excluded from the NFA by the SAT  
1075 solver bias. These modifications allow us to gain better control over the  
1076 NFA's acceptability, i.e., it may become more accepting or more rejecting.  
1077 Consequently, using the proposed models and post-processing techniques we  
1078 obtain flexible NFAs, which can be adjusted to one's needs.

1079 Finally, using random walks we can gain some insight into the structure  
1080 and behavior of the inferred NFA that accept infinite languages. In this  
1081 paper, we analyzed their capability of generating unique words and paths, as  
1082 well as the possible reasons for random walk stops.

1083 In the future, we plan to employ parallel computing capabilities to some  
1084 of the models to further reduce the number of unsolved instances. Moreover,  
1085 we plan to use the inferred NFA to not only generate new words, but also to  
1086 classify them.

## 1087 References

- 1088 [1] T. Jastrzab, F. Lardeux, É. Monfroy, Inference of over-constrained NFA  
1089 of size  $k+1$  to efficiently and systematically derive NFA of size  $k$  for  
1090 grammar learning, in: J. Mikyska, C. de Mulatier, M. Paszynski, V. V.  
1091 Krzhizhanovskaya, J. J. Dongarra, P. M. A. Sloot (Eds.), Proc. of the  
1092 23rd ICCS Conference, Part I, volume 14073 of *LNCS*, Springer, 2023,  
1093 pp. 134–147. doi:10.1007/978-3-031-35995-8\_10.
- 1094 [2] C. de la Higuera, Grammatical Inference: Learning Automata and  
1095 Grammars, Cambridge University Press, 2010.
- 1096 [3] F. Denis, A. Lemay, A. Terlutte, Learning regular languages using rfsas,  
1097 Theor. Comput. Sci. 313 (2004) 267–294.
- 1098 [4] M. Vázquez de Parga, P. García, J. Ruiz, A family of algorithms for  
1099 non deterministic regular languages inference, in: Proc. of CIAA 2006,  
1100 volume 4094 of *LNCS*, Springer, 2006, pp. 265–274.

- 1101 [5] M. Gendreau, J.-Y. Potvin, Handbook of Metaheuristics, Springer, 2009.
- 1102 [6] M. Tomita, Dynamic construction of finite-state automata from exam-  
1103 ples using hill-climbing., Proc. of the Fourth Annual Conference of the  
1104 Cognitive Science Society (1982) 105–108.
- 1105 [7] K. Apt, Principles of Constraint Programming, Cambridge University  
1106 Press, 2003.
- 1107 [8] F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Pro-  
1108 gramming, volume 2 of *Foundations of Artificial Intelligence*, Elsevier,  
1109 2006.
- 1110 [9] W. Wiecek, Grammatical Inference – Algorithms, Routines and  
1111 Applications, volume 673 of *Studies in Computational Intelligence*,  
1112 Springer, 2017.
- 1113 [10] T. Jastrzab, On parallel induction of nondeterministic finite automata,  
1114 in: Proc. of ICCS 2016, volume 80 of *Procedia Computer Science*, Else-  
1115 vier, 2016, pp. 257–268.
- 1116 [11] T. Jastrzab, Two parallelization schemes for the induction of nonde-  
1117 terministic finite automata on PCs, in: Proc. of PPAM 2017, volume  
1118 10777 of *LNCS*, Springer, 2017, pp. 279–289.
- 1119 [12] T. Jastrzab, A comparison of selected variable ordering methods for NFA  
1120 induction, in: Proc. of ICCS 2019, volume 11540 of *LNCS*, Springer,  
1121 2019, pp. 741–748.
- 1122 [13] T. Jastrzab, Z. J. Czech, W. Wiecek, Parallel algorithms for minimal  
1123 nondeterministic finite automata inference, *Fundam. Informaticae* 178  
1124 (2021) 203–227. doi:10.3233/FI-2021-2004.
- 1125 [14] M. R. Garey, D. S. Johnson, Computers and Intractability, A Guide  
1126 to the Theory of NP-Completeness, W.H. Freeman & Company, San  
1127 Francisco, 1979.
- 1128 [15] F. Lardeux, E. Monfroy, GA and ILS for optimizing the size of NFA  
1129 models, in: The 8th International Conference on Metaheuristics and  
1130 Nature Inspired Computing (META), Marrakech, Morocco, 2021. URL:  
1131 <https://hal.univ-angers.fr/hal-03284541>.



- 1132 [16] S. Yu, Regular languages, in: G. Rozenberg, A. Salomaa (Eds.), Hand-  
1133 book of Formal Languages, Volume 1: Word, Language, Grammar,  
1134 Springer, 1997, pp. 41–110. doi:10.1007/978-3-642-59136-5\_2.
- 1135 [17] D. Angluin, Learning regular sets from queries and counterexam-  
1136 ples, *Information and Computation* 75 (1987) 87–106. doi:10.1016/0890-  
1137 5401(87)90052-6.
- 1138 [18] A. Sanfeliu, N. P. de la Blanca, E. Vidal, *Pattern Recognition and Image*  
1139 *Analysis*, WORLD SCIENTIFIC, 1992. doi:10.1142/1582.
- 1140 [19] P. García, M. Vázquez de Parga, G. I. Álvarez, J. Ruiz, Universal  
1141 automata and nfa learning, *Theoretical Computer Science* 407 (2008)  
1142 192–202. doi:https://doi.org/10.1016/j.tcs.2008.05.017.
- 1143 [20] C. Lecoutre, N. Szczepanski, PYCSP3: modeling combinatorial con-  
1144 strained problems in python, *CoRR* abs/2009.00326 (2020). URL:  
1145 <https://arxiv.org/abs/2009.00326>.
- 1146 [21] C. Lecoutre, Ace, a generic constraint solver, 2023. [arXiv:2302.05405](https://arxiv.org/abs/2302.05405).
- 1147 [22] F. Lardeux, E. Monfroy, Improved SAT models for NFA learning, in:  
1148 *International Conference in Optimization and Learning (OLA)*, Catania,  
1149 Italy, 2021. URL: <https://hal.univ-angers.fr/hal-03284571>.
- 1150 [23] T. Jastrzab, F. Lardeux, É. Monfroy, Cclassifying words with 3-sort  
1151 automata, in: *16th International Conference on Agents and Artificial*  
1152 *Intelligence, ICAART 2024*, SCITEPRESS, 2024. In press.
- 1153 [24] T. Jastrzab, F. Lardeux, E. Monfroy, Taking advantage of a very simple  
1154 property to efficiently infer NFAs, in: *34th IEEE International Confer-*  
1155 *ence on Tools with Artificial Intelligence, ICTAI 2022*, virtual, November  
1156 1-2, 2022, IEEE, 2022.
- 1157 [25] F. Lardeux, E. Monfroy, Optimized models and symmetry break-  
1158 ing for the NFA inference problem, in: *33rd IEEE International*  
1159 *Conference on Tools with Artificial Intelligence, ICTAI 2021*, Wash-  
1160 ington, DC, USA, November 1-3, 2021, IEEE, 2021, pp. 396–403.  
1161 doi:10.1109/ICTAI52525.2021.00065.

- 1162 [26] T. Stützle, R. Ruiz, *Iterated Local Search*, Springer International Pub-  
1163 lishing, Cham, 2018, pp. 579–605. doi:10.1007/978-3-319-07124-4\_8.
- 1164 [27] G. S. Tseitin, *On the Complexity of Derivation in Propositional Calculus*, Springer Berlin Heidelberg, Berlin, Heidelberg, 1983, pp. 466–483.  
1165
- 1166 [28] J. Beerten, J. J. J. van Durme, R. Gallardo, E. Capriotti, L. C. Serpell,  
1167 F. Rousseau, J. Schymkowitz, WALTZ-DB: a benchmark database of  
1168 amyloidogenic hexapeptides, *Bioinform.* 31 (2015) 1698–1700.
- 1169 [29] N. Louros, K. Konstantoulea, M. De Vleeschouwer, M. Ramak-  
1170 ers, J. Schymkowitz, F. Rousseau, WALTZ-DB 2.0: an updated  
1171 database containing structural information of experimentally deter-  
1172 mined amyloid-forming peptides, *Nucleic Acids Research* 48 (2019)  
1173 D389–D393. doi:10.1093/nar/gkz758.
- 1174 [30] A. Ignatiev, A. Morgado, J. Marques-Silva, PySAT: A Python toolkit  
1175 for prototyping with SAT oracles, in: *SAT*, 2018, pp. 428–437.  
1176 doi:10.1007/978-3-319-94144-8\_26.
- 1177 [31] G. Audemard, L. Simon, Predicting learnt clauses quality in modern  
1178 SAT solvers, in: *Proc. of IJCAI 2009*, 2009, pp. 399–404.