Learning DFAs from Positive Examples Only via Word Counting

Benjamin Bordais[1,2,3], **Daniel Neider**[1,2]

[1]TU Dortmund University, Dortmund, Germany
[2]Center for Trustworthy Data Science and Security, University Alliance Ruhr, Dortmund, Germany
[3]Université libre de Bruxelles, Belgium
benjamin.bordais@ulb.be, daniel.neider@tu-dortmund.de

Abstract

Learning finite automata from positive examples has recently gained attention as a powerful approach for understanding, explaining, analyzing, and verifying black-box systems. The motivation for focusing solely on positive examples arises from the practical limitation that we can only observe what a system is capable of (positive examples) but not what it cannot do (negative examples). Unlike the classical problem of passive DFA learning with both positive and negative examples, which has been known to be NP-complete since the 1970s, the topic of learning DFAs exclusively from positive examples remains poorly understood. This paper introduces a novel perspective on this problem by leveraging the concept of counting the number of accepted words up to a carefully determined length. Our contributions are twofold. First, we prove that computing the minimal number of words up to this length accepted by DFAs of a given size that accept all positive examples is NP-complete, establishing that learning from positive examples alone is computationally demanding. Second, we propose a new learning algorithm with a better asymptotic runtime than the best-known bound for existing algorithms. While our experimental evaluation reveals that this algorithm under-performs state-of-the-art methods, it demonstrates significant potential as a preprocessing step to enhance existing approaches.

Code — https://github.com/BBordais/Learning-DFAs-from-Positive-Examples-Only-via-Word-Counting

Extended version — https://arxiv.org/abs/2511.08431

Introduction

In recent years, the rapid advancement and proliferation of Artificial Intelligence systems have transformed numerous fields, from healthcare (Zhang, Weng, and Lund 2022) to transportation (Bharadiya 2023) or finance (Chen et al. 2023). This surge in AI capabilities has greatly increased the need of interpretable models able to provide insights into the decision-making processes of complex black-box systems.

Interpretable models are synthesized from temporal data. The goal is to obtain a model that is coherent with the observations gathered, which in turn can be used to decipher the system's past behavior. Different kinds of (interpretable) models are used in the literature, often of one of two types: finite-state machines (Weiss, Goldberg, and Yahav 2024) and (temporal) logic formulas (Camacho and McIlraith 2019). In this paper, we focus on Deterministic Finite Automata (DFAs) (Rabin and Scott 1959), that enjoy both an accessible formalism, making them suitable e.g. classifier of sequential data (Shvo et al. 2021), and many desirable properties, such as tractable model checking.

In the literature, synthesizing a DFA from observations is usually formalized as a passive learning task where we are given a positive set of words that a prospective DFA should accept, a negative set of words that it should reject, and an integer bound on its number of states (Gold 1978; Grinchtein, Leucker, and Piterman 2006; Heule and Verwer 2010). However, a significant obstacle on the usefulness of the passive learning task from positive and negative examples (PLPN) is the difficulty of coming up with negative examples: they correspond to observations of a system's undesirable behaviors, which can be intrinsically hard (e.g., with black-box systems) or unrealistic (e.g., with self-driving cars) to gather.

To circumvent that issue, the focus has recently shifted to the passive learning task of synthesizing DFAs from positive examples only (PLP). Without any additional restriction, the DFA accepting exactly the (positive) words of the input set $\mathcal{P}$—on the one end of the spectrum—and the trivial accepting DFA (the one-state DFA accepting every word)—on the other end of the spectrum—both meet the requirement of this PLP task while providing no insight whatsoever w.r.t. the input set $\mathcal{P}$. To forbid these two extreme solutions, constraints on the prospective DFA are added. On the one hand, as in the classical PLPN setting, we consider a bound $n \in \mathbb{N}$ on the number of states of the prospective DFA. The benefit is twofold: first, it prevents synthesizing the DFA accepting exactly the words in $\mathcal{P}$ (assuming n is small enough); second, for explainability purposes, the smaller the DFA, the better. On the other hand, we look for a DFA that is language-minimal, i.e., whose language is minimal (for inclusion) among those DFAs with at most n states that accept all words in $\mathcal{P}$. As argued by Avellaneda and Petrenko (2018), this constraint follows naturally from a parsimony standpoint: we look for a simplest solution. With these restrictions, we obtain a PLP task $\mathcal{T}$.

A counterexample (CE) guided algorithm solving the task $\mathcal{T}$ was first developed by Avellaneda and Petrenko (2018), then a few years later it was improved into a symbolic algo-

rithm by Roy et al. (2023), which now constitutes the state-of-the-art. The CE algorithm relies on iteratively synthesizing DFAs by solving instances of the PLPN task, where $\mathcal{P}$ always constitutes the positive set, while the negative set is initially empty and iteratively grown by adding words witnessing the non language-minimality of candidate DFAs (i.e., counterexamples). On the other hand, the symbolic algorithm starts from the trivial accepting DFA, and iteratively looks for DFAs accepting all words in $\mathcal{P}$ with a smaller language. Both the CE and symbolic algorithms iteratively call SAT solvers at each step to find new candidate DFAs.

By construction, the symbolic algorithm avoids redundancies that could occur in the CE algorithm. However, even with the symbolic algorithm, a significant drawback remains: the best bound on the number of steps of the algorithm, and thus on the number of calls to SAT solvers, is exponential in n. This is due to the fact that there are exponentially many words of length at most (polynomial in) n. Furthermore, while the complexity of the PLPN task (which can be stated as a decision problem) is well understood—it is known to be NP-complete since the 70s (Gold 1978)—the PLP task is poorly understood complexity-wise. In particular, (Avellaneda and Petrenko 2018; Roy et al. 2023) do not provide a complexity lower bound on any closely-related decision problem. As such, it is not clear that solving the PLP task $\mathcal{T}$ even requires the use of SAT solvers.

Our Contributions. In this paper, we tackle the PLP task $\mathcal{T}$ from a new perspective which consists in assigning to each DFA a word-counting metric that greatly enhances our ability to compare two DFAs (as opposed to checking language inclusion). Specifically, this word-counting metric is the number of words accepted by each DFA up to length $2n-2$. To demonstrate the usefulness of this new perspective, we show that a word-counting minimal DFA is necessarily language-minimal. Thus, we focus on a new PLP task $\mathcal{T}'$—that requires a prospective DFA to be word-counting minimal instead of only language-minimal—and on the associated decision problem about the minimal word-counting metric of DFAs with at most n states accepting all words in $\mathcal{P}$. Our main contribution is that this problem is NP-complete, which constitutes the first complexity result directly related to a PLP task. This NP-completeness result has two main consequences: first, because it is in NP, we immediately obtain that the task $\mathcal{T}'$, and thus the task $\mathcal{T}$ as well, can be solved with polynomially many calls to a SAT solver, via a binary search; second, because the problem is NP-hard, it is unlikely that we can do better than polynomially many calls to a SAT solver to solve task $\mathcal{T}'$ (though task $\mathcal{T}$ could be intrinsically easier to solve than task $\mathcal{T}'$).

We have implemented an algorithm solving task $\mathcal{T}'$ (and thus also task $\mathcal{T}$) via an Integer Linear Programming (ILP) encoding. Our experiments show that this algorithm underperforms the symbolic state-of-the-art algorithm solving task $\mathcal{T}$. However, building on the word-counting ideas presented above, we have developed a method that could be used to improve the performance of the symbolic algorithm for solving $\mathcal{T}$. Specifically, we have designed a heuristic algorithm that looks for a DFA with at most n states, accepting all words in $\mathcal{P}$, with a word-counting metric as small as possible. This algorithm is intended to be used as a preprocessing step outputting a DFA that can then be used as a starting point by the symbolic algorithm. Our experiments show promising results of this method.

Missing technical details can be found in the extended version (Bordais and Neider 2025).

Related Work. As already mentioned, there is a large body of work focusing on the classical DFA passive learning. Another widely studied DFA learning setting is active learning, where a learner successively queries a teacher. Angluin's seminal work (Angluin 1987) essentially started the research on this topic, and is still very influential nowadays (Shahbaz and Groz 2009; Bollig et al. 2009).

DFA learning from positive examples only was introduced by Gold (1967), where it is shown that DFAs cannot be identified at the limit. This partly explains why there was no further study of this topic until recently (Avellaneda and Petrenko 2018; Roy et al. 2023). More popular subjects are the passive learning from positive examples only of better-behaved models, such as pattern languages (Angluin 1980), hidden Markov chains (Stolcke and Omohundro 1992), or stochastic machines (Carrasco and Oncina 1999).

Finally, an ILP encoding a learning task (of a logical (LTLf) formula) was first introduced in (Ielo et al. 2023).

Definitions and Notations

We let $\mathbb{N}$ denote the set of non-negative integers. For all $i \leq j \in \mathbb{N}$, we let $[\![i, j]\!]$ denote the set $\{i, i+1, \ldots, j\} \subseteq \mathbb{N}$.

Alphabet and Automata. An alphabet Σ refers to a non-empty finite set of letters. We let Σ^* (resp. Σ^+) denote the set of (resp. non-empty) finite words with letters in Σ. We let $\epsilon \in \Sigma^*$ denote the empty word. For all words $u \in \Sigma^*$, we let $|u| \in \mathbb{N}$ denote the length of the word u, i.e., its number of letters. For all $m \in \mathbb{N}$, we let Σ^m (resp. $\Sigma^{\leq m}$) denote the set of words of length m (resp. at most m). For all $L \subseteq \Sigma^*$, we let $\mathsf{Pref}(L)$ denote the set of (non-strict) prefixes of words in L. Let us now recall the definition of DFAs.

Definition 1. *Consider an alphabet Σ. A* deterministic finite automaton (DFA) *on Σ is a tuple $\mathcal{A} = (Q, \Sigma, q_{\mathsf{init}}, \delta, F)$ where Q is a finite non-empty set of states, $q_{\mathsf{init}} \in Q$ is the initial state, $\delta \colon Q \times \Sigma \to Q$ is the transition function, and $F \subseteq Q$ is the set of final states. The transition function δ is naturally extended to finite words into a function $\delta^* \colon Q \times \Sigma^* \to Q$ defined as follows: for all $q \in Q$, $\delta^*(q, \epsilon) := q$, and for all $u \in \Sigma^*$ and $\alpha \in \Sigma$: $\delta^*(q, u{\cdot}\alpha) := \delta(\delta^*(q, u), \alpha)$.*

A word $u \in \Sigma^*$ is accepted *by the DFA $\mathcal{A}$ if $\delta^*(q_{\mathsf{init}}, u) \in F$. We let $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^*$ denote the language of the DFA $\mathcal{A}$, i.e., the set of words that it accepts: $\mathcal{L}(\mathcal{A}) := \{u \in \Sigma^* \mid \delta^*(q_{\mathsf{init}}, u) \in F\}$. We let $|\mathcal{A}|$ denote the size of the DFA $\mathcal{A}$, i.e., its number of states $|\mathcal{A}| := |Q|$. Furthermore, let $\mathsf{DFA}(\Sigma)$ denote the set of DFAs on the alphabet Σ.*

In the following, unless otherwise stated, a DFA $\mathcal{A}$ on an alphabet Σ will refer to the tuple $\mathcal{A} = (Q, \Sigma, q_{\mathsf{init}}, \delta, F)$.

Given an integer $n \in \mathbb{N}$ and a set of words $\mathcal{P}$, we let $\mathsf{Rec}(\mathcal{P}, n) := \{\mathcal{A} \in \mathsf{DFA}(\Sigma) \mid |\mathcal{A}| \leq n,\ \mathcal{P} \subseteq \mathcal{L}(\mathcal{A})\}$

denote the set of DFAs with at most n states that accept all words in $\mathcal{P}$.

Strict Partial Orders. A relation $\prec \subseteq \mathsf{DFA}(\Sigma) \times \mathsf{DFA}(\Sigma)$ is a strict partial order on $\mathsf{DFA}(\Sigma)$ if it is irreflexive, anti-symmetric, and transitive. For $S \subseteq \mathsf{DFA}(\Sigma)$ and $\mathcal{A} \in S$, $\mathcal{A}$ is said to be S-minimal w.r.t. $\prec$ if, for all $\mathcal{A}' \in S$, we do *not* have $\mathcal{A}' \prec \mathcal{A}$.

Given two DFAs $\mathcal{A}, \mathcal{A}' \in \mathsf{DFA}$, we write $\mathcal{A} \prec_{\mathcal{L}} \mathcal{A}'$ when $\mathcal{L}(\mathcal{A}) \subsetneq \mathcal{L}(\mathcal{A}')$. Clearly, $\prec_{\mathcal{L}}$ is a strict partial order on $\mathsf{DFA}(\Sigma)$.

Learning from Positive Examples Only. In this paper, we focus on the task of synthesizing a DFA with at most $n \geq 1$ states accepting all words in a given set $\mathcal{P}$, i.e., we look for DFAs in $\mathsf{Rec}(\mathcal{P}, n)$. Furthermore, among those DFAs, we look for a language-minimal one (following the parsimony principle), i.e., one that is $\mathsf{Rec}(\mathcal{P}, n)$-minimal w.r.t. the order $\prec_{\mathcal{L}}$. Formally, the task that we consider is defined below.

Task 1. *Given as input an alphabet Σ, a set $\mathcal{P} \subseteq \Sigma^*$, and $n \geq 1$, find a DFA $\mathcal{A}$ such that: a) $\mathcal{A} \in \mathsf{Rec}(\mathcal{P}, n)$; and b) the DFA $\mathcal{A}$ is $\mathsf{Rec}(\mathcal{P}, n)$-minimal w.r.t. $\prec_{\mathcal{L}}$.*

The size of this task is equal to $n + |\mathsf{Pref}(\mathcal{P})|$.

Learning DFAs via Word Counting

Before we detail our approach for solving Task 1, let us describe the state-of-the-art (symbolic) algorithm.

The State-of-the-Art Algorithm. It was developed by Roy et al. (2023) and proceeds as follows. Initially, one starts with the trivial accepting DFA $\mathcal{A}_0 \in \mathsf{Rec}(\mathcal{P}, n)$. Then, at step $i \geq 1$, one computes a DFA $\mathcal{A}_i \in \mathsf{Rec}(\mathcal{P}, n)$ satisfying $\mathcal{A}_i \prec_{\mathcal{L}} \mathcal{A}_{i-1}$ using a SAT solver. The process stops when no such DFA is found. Overall, this algorithm iteratively computes a decreasing chain of DFAs $\mathcal{A}_0 \succ_{\mathcal{L}} \ldots \succ_{\mathcal{L}} \mathcal{A}_k \succ_{\mathcal{L}} \ldots$ with $\mathcal{A}_i \in \mathsf{Rec}(\mathcal{P}, n)$, for all $i \in \mathbb{N}$. Since the languages of DFAs may be infinite, it may not be clear that this chain is necessarily finite. However, it is indeed the case, as argued by Roy et al. (2023): two DFAs with at most n states have the same language if they accept exactly the same words of length at most n^2 (a tighter bound exists, as discussed below). Thus, since there are finitely many words of length at most n^2, the above decreasing chain of DFAs is necessarily finite. However, the best bound on the number of iterations that we can extract from this argument is exponential in n (since there are exponentially many words of length at most n^2), with each iteration involving a call to a SAT solver.

From a Language-Inclusion to a Numerical Order. To design a method that does significantly less calls to SAT solvers, we shift perspective from focusing on language inclusion, via the strict partial order $\prec_{\mathcal{L}}$, to focusing on a numerical order induced by values that we assign to DFAs. As we will see in the remainder of this paper, this has two main benefits: first, numerical orders are compatible with binary searches; second, the values assigned to DFAs can be seen as a score that we aim at optimizing (in this case, minimize). However, it is crucial that this numerical order relates to the language-inclusion order $\prec_{\mathcal{L}}$ previously defined, as this is the one involved in Task 1. In fact, taking as values assigned to DFAs the number of words, up to a certain length, that they accept does the job. This is formally defined below.

Definition 2. *Consider some $h \in \mathbb{N}$. For all DFAs $\mathcal{A}, \mathcal{A}'$, we write $\mathcal{A} \prec_h \mathcal{A}'$ when $|\mathcal{L}(\mathcal{A}) \cap \Sigma^{\leq h}| < |\mathcal{L}(\mathcal{A}') \cap \Sigma^{\leq h}|$.*

In other words, we have $\mathcal{A} \prec_h \mathcal{A}'$ when $\mathcal{A}$ accepts fewer words of length at most h than $\mathcal{A}'$. Clearly, for all $h \in \mathbb{N}$, $\prec_h$ is a strict partial order on DFAs. Furthermore, this order satisfies the following property, thus showing that minimizing the number of accepted words also minimizes the language.

Proposition 1. *For an alphabet Σ, a set $\mathcal{P} \subseteq \Sigma^*$, $n \geq 1$, and $h := 2n - 2$, a DFA $\mathcal{A} \in \mathsf{Rec}(\mathcal{P}, n)$ that is $\mathsf{Rec}(\mathcal{P}, n)$-minimal w.r.t. $\prec_h$ is $\mathsf{Rec}(\mathcal{P}, n)$-minimal w.r.t. $\prec_{\mathcal{L}}$.*

This proposition relies on the folk theorem recalled below.

Theorem 1 (Folk result)**.** *Consider an alphabet Σ, and $\mathcal{A}, \mathcal{A}' \in \mathsf{DFA}(\Sigma)$. If $\mathcal{L}(\mathcal{A}) \neq \mathcal{L}(\mathcal{A}')$, then $\Sigma^{\leq |\mathcal{A}|+|\mathcal{A}'|-2} \cap (\mathcal{L}(\mathcal{A}) \setminus \mathcal{L}(\mathcal{A}') \cup \mathcal{L}(\mathcal{A}') \setminus \mathcal{L}(\mathcal{A})) \neq \emptyset$.*

We provide a proof of this theorem in (Bordais and Neider 2025). It essentially relies on the iterative computation of the Myhill-Nerode equivalence classes of DFAs. We also show that the bound $h = |\mathcal{A}| + |\mathcal{A}'| - 2$ is tight.

The proof of Proposition 1 is now direct.

Proof. Consider a DFA $\mathcal{A} \in \mathsf{Rec}(\mathcal{P}, n)$ that is $\mathsf{Rec}(\mathcal{P}, n)$-minimal w.r.t. $\prec_h$, for $h = 2n - 2$. Consider any DFA $\mathcal{A}' \in \mathsf{Rec}(\mathcal{P}, n)$ and assume towards a contradiction that we have $\mathcal{A}' \prec_{\mathcal{L}} \mathcal{A}$, i.e., $\mathcal{L}(\mathcal{A}') \subsetneq \mathcal{L}(\mathcal{A})$. By Theorem 1, there is some $u \in \Sigma^{\leq h} \cap (\mathcal{L}(\mathcal{A}) \setminus \mathcal{L}(\mathcal{A}'))$. Since we do not have $\mathcal{A}' \prec_h \mathcal{A}$, it follows that $|\mathcal{L}(\mathcal{A}) \cap \Sigma^{\leq h}| \leq |\mathcal{L}(\mathcal{A}') \cap \Sigma^{\leq h}|$. Hence, there must be some $u' \in \Sigma^{\leq h} \cap (\mathcal{L}(\mathcal{A}') \setminus \mathcal{L}(\mathcal{A}))$. Thus, we do not have $\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A})$. Hence, the contradiction. □

Following Proposition 1, we consider a new task.

Task 2. *Given as input an alphabet Σ, a set $\mathcal{P} \subseteq \Sigma^*$, and $n \geq 1$, find a DFA $\mathcal{A}$ such that: a) $\mathcal{A} \in \mathsf{Rec}(\mathcal{P}, n)$; and b) the DFA $\mathcal{A}$ is $\mathsf{Rec}(\mathcal{P}, n)$-minimal w.r.t. $\prec_{2n-2}$.*

Proposition 1 gives that a DFA satisfying the requirements of Task 2 also satisfies the requirements of Task 1. Therefore, let us focus on solving this Task 2.

Computing the Minimal Number of Accepted Words of Length at Most $2n - 2$.

To study how we can solve Task 2, we investigate the associated decision problem, formally defined below.

Decision problem 1. *Given as input an alphabet Σ, a set $\mathcal{P} \subseteq \Sigma^*$, $n \geq 1$, and $k \in \mathbb{N}$, decide if there exists a DFA $\mathcal{A} \in \mathsf{Rec}(\mathcal{P}, n)$ such that: $|\mathcal{L}(\mathcal{A}) \cap \Sigma^{\leq 2n-2}| \leq k$.*

The size of the input of this task is $n + |\mathsf{Pref}(\mathcal{P})| + \log k$.

As a first step in the study of this problem, let us consider how we can compute the number of words, up to a certain length, accepted by a DFA. Actually, this can be done in a direct (iterative) manner, as described in Lemma 1 below.

Lemma 1. *Consider an alphabet Σ and a DFA $\mathcal{A} \in \mathsf{DFA}(\Sigma)$. For all $q \in Q$ and $m \in \mathbb{N}$, we let $\mathsf{N}(q, m) := \{w \in \Sigma^m \mid \delta^*(q_0, w) = q\}$. Then, $\mathsf{N}(q, 0) = 1$ if $q = q_0$, and*

$\mathsf{N}(q,0) = 0$ *otherwise. Furthermore, for all* $(m,q) \in \mathbb{N}\times Q$*:* $\mathsf{N}(q,m+1) = \sum_{q'\in Q} \mathsf{N}(q',m)\cdot|\{\alpha\in\Sigma \mid \delta(q',\alpha)=q\}|$.

In addition, $|\mathcal{L}(\mathcal{A})\cap\Sigma^{\leq m}| = \sum_{i=0}^{m}\sum_{q\in F}\mathsf{N}(q,i)$.

Note that this lemma would not hold with automata that are not deterministic.

Let us now state the main result of this paper.

Theorem 2. *The decision problem 1 is* NP*-complete.*

Proof. We argue the NP-hardness in a dedicated section below. As for Problem 1 being in NP, this problem can be solved as follows: we guess a DFA $\mathcal{A} \in \mathsf{Rec}(\mathcal{P},n)$—we can check that it is indeed the case in time polynomial in $|\mathsf{Pref}(\mathcal{P})|$ by simulating the runs of $\mathcal{A}$ on words in $\mathcal{P}$—, we compute $|\mathcal{L}(\mathcal{A})\cap\Sigma^{\leq 2n-2}| \in \mathbb{N}$ and compare it with k. Although $|\mathcal{L}(\mathcal{A})\cap\Sigma^{\leq 2n-2}|$ may be exponential, we can compute it in polynomial time with the help of Lemma 1. □

Before we present the NP-hardness proof, let us discuss the two major implications of Theorem 2. First, the fact that the decision problem 1 is in NP suggests an algorithm solving Task 2, and thus Task 1 as well, with seemingly better asymptotic guarantees than the state-of-the-art algorithm.

Proposition 2. *Given an alphabet* Σ*, a set* $\mathcal{P}\subseteq\Sigma^*$*, and an integer* $n\in\mathbb{N}$*, we can solve Task 2 with* $\mathcal{O}(poly(|\Sigma|,n))$*-calls to a SAT solver on inputs of size* $\mathcal{O}(poly(|\Sigma|,|\mathsf{Pref}(\mathcal{P})|,n))$.

Proof. We have $|\Sigma^{\leq 2n-2}| = \frac{m^{2n-1}-1}{m-1}$, for $m := |\Sigma| \geq 2$. Hence, a binary search querying, at each step, a SAT solver on an encoding of Problem 1 with $1\leq k\leq|\Sigma^{\leq 2n-2}|$ would take at most $\log(|\Sigma^{\leq 2n-2}|) \leq (2n-1)\cdot\log m$ steps to find a DFA $\mathcal{A}\in\mathsf{Rec}(\mathcal{P},n)$ satisfying $|\mathcal{L}(\mathcal{A})\cap\Sigma^{\leq 2n-2}| = \min_{\mathcal{A}'\in\mathsf{Rec}(\mathcal{P},n)}|\mathcal{L}(\mathcal{A}')\cap\Sigma^{\leq 2n-2}|$. By definition, such a DFA $\mathcal{A}$ would be $\mathsf{Rec}(\mathcal{P},n)$-minimal w.r.t. $\prec_{2n-2}$. □

Second, unless P = NP, the NP-hardness of Decision Problem 1 makes it unlikely that it is possible to solve Task 2 with an asymptotic bound significantly better than that of Proposition 2. This only applies to Task 2, and there might be an efficient way to solve Task 1 without solving Task 2.

NP-Hardness Proof. The NP-hardness proof of Theorem 2 is quite technical, see (Bordais and Neider 2025) for a detailed proof. Here, we only give a bird's eye view of the steps that we take to prove the result.

The NP-hardness proof relies on a reduction from the All-Pos-Neg SAT (APN-SAT) problem that was already used by Lingg, de Oliveira Oliveira, and Wolf (2024) to establish the NP-hardness of the classical passive learning problem with positive and negative words (with a binary alphabet). The APN-SAT problem is a variant of the classical SAT problem where we are given a set of clauses to satisfy, with each clause being constituted of only positive literals (i.e., without negations) or only negative literals (i.e., with negations). This problem is defined as follows.

Decision problem 2. *Consider as input a set* $X := \{x_i \mid i\in[\![1,r]\!]\}$ *of variables, a set of clauses* $\mathcal{C} = \{C_j \mid j\in[\![1,s]\!]\}$ *that is described as the disjoint union of sets of positive and negative clauses:* $\mathcal{C} = \mathcal{C}_+ \uplus \mathcal{C}_-$*, where for all* $j\in[\![1,s]\!]$*, we have* $C_j\subseteq X$*. We have to decide if there is a valuation* $\nu\colon X\to\{\top,\bot\}$ *of the variables satisfying* $(X,\mathcal{C})$*, i.e., such that, for all* $j\in[\![1,s]\!]$*:*

$$\exists x\in C_j:\ \nu(x) = \begin{cases}\top & \text{if } C_j\in\mathcal{C}_+\\ \bot & \text{if } C_j\in\mathcal{C}_-\end{cases}$$

For the remainder of this section, we consider an instance $(X,\mathcal{C})$ of Problem 2 with $X = \{x_i \mid i\in[\![1,r]\!]\}$ the set of variables, and $\mathcal{C} = \{C_j \mid j\in[\![1,s]\!]\} = \mathcal{C}_+\uplus\mathcal{C}_-$ the set of clauses. The alphabet that we consider is $\Sigma := \{a,b\}$. Our goal is to define a set of positive words $\mathcal{P}$ and two integers n and k such that $(X,\mathcal{C})$ is a positive instance of Problem 2 if and only if there is a $(\mathcal{P},n,k)$-suitable DFA $\mathcal{A}$, i.e., such that 1) $\mathcal{A}\in\mathsf{Rec}(\mathcal{P},n)$; and 2) $|\Sigma^{\leq 2n-2}\cap(\mathcal{L}(\mathcal{A})\setminus\mathcal{P})|\leq k$[1].

Let us first discuss what a DFA satisfying conditions 1) and 2) could look like on a simple input of Problem 2. That way, we will be able to illustrate (some of) the kinds of words that we include in $\mathcal{P}$, which the DFA has to accept.

Example 1. *Consider the set of variables* $X = \{x_1,x_2,x_3\}$*, a single positive clause* $C_1 = \{x_1,x_3\}\in\mathcal{C}_+$ *and single a negative clause* $C_2 = \{x_2,x_3\}\in\mathcal{C}_-$*. In Figure 1, we scheme the shape of a DFA* $\mathcal{A}_{\mathsf{ex}}$ *that would be* $(\mathcal{P},n,k)$*-suitable, with* $(\mathcal{P},n,k)$*—which we partly describe here—being the result of the reduction from the instance* $(X,\mathcal{C})$*. The double-circled states are accepting.*

In $\mathcal{P}$*, we define a "*$\top$*-word"* $a\cdot a\cdot a\cdot a$ *and a "*$\bot$*-word"* $a\cdot b\cdot a\cdot a$*. One can see that when the DFA* $\mathcal{A}_{\mathsf{ex}}$ *runs on those words, it visits the right-most (green and red) states.*

We also define "variables words" in $\mathcal{P}$ *that the DFA* $\mathcal{A}_{\mathsf{ex}}$ *processes by visiting the central (blue) states. In particular, these words impose to* $\mathcal{A}_{\mathsf{ex}}$ *that we reach either* $q^1_\top$ *or* $q^1_\bot$ *upon reading the letter* a *from the states* q'_{x_i}*, for* $i\in\{1,2,3\}$*. This corresponds to the bold transitions from the central states to the right-most states. These transitions uniquely define a valuation* $\nu\colon X\to\{\top,\bot\}$*, specifically:* $\nu(x_1)=\top$*,* $\nu(x_2)=\bot$*,* $\nu(x_3)=\top$.

In addition, we define "clause words" in $\mathcal{P}$ *that* $\mathcal{A}_{\mathsf{ex}}$ *processes by visiting the left-most (yellow) states. Some of these words ensure that upon reading the letter* b *from* q_{C_j}*, for* $j\in\{1,2\}$*, we reach a state* q'_{x_i}*, for* $i\in\{1,2,3\}$*, such that* $x_i\in C_j$*. The other "clause words" ensure that a positive (resp. negative) clause-state eventually leads to a* $\top$*-state (resp.* $\bot$*-state). Namely, the word* $b\cdot b\cdot b\cdot a\in\mathcal{P}$ *encodes that* C_1 *is a positive clause, and the word* $b\cdot b\cdot a\cdot b\cdot a\cdot a\in\mathcal{P}$ *encodes that* C_2 *is a negative clause.*

Overall, the valuation $\nu\colon X\to\{\top,\bot\}$ *uniquely defined by* $\mathcal{A}_{\mathsf{ex}}$ *satisfies* $(X,\mathcal{C})$*, e.g., for the negative clause* C_2*, there is a bold edge from* q_{C_2} *to* q'_{x_2}*, thus we must have* $x_2\in C_2$ *and* $\nu(x_2)=\bot$*, which is indeed the case.*

Let us now describe a little more formally how the words that we define in $\mathcal{P}$ ensure that the existence of a $(\mathcal{P},n,k)$-suitable DFA $\mathcal{A}$ implies that $(X,\mathcal{C})$ is a positive instance of Problem 2. Below, we consider a DFA $\mathcal{A} = (Q,\Sigma,q_{\mathsf{init}},\delta,F)$ satisfying conditions 1) and 2).

[1]This condition constitutes a slight change compared to Problem 1. This is however harmless, see the extended version.

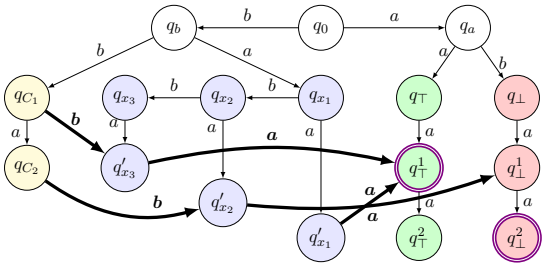

Figure 1: The shape of a DFA $\mathcal{A}_{\mathsf{ex}}$.

1. We define two words $w_\top, w_\bot \in \mathcal{P}$ and we ensure that $\delta^*(q_0, w_\top) \neq \delta^*(q_0, w_\bot)$. This fact will allow us to distinguish when a variable is mapped to true ($\top$) and to false ($\bot$). In the DFA $\mathcal{A}_{\mathsf{ex}}$ of Example 1, we have $\delta^*(q_0, w_\top) = q^1_\top \neq q^1_\bot = \delta^*(q_0, w_\bot)$.
2. For each variable $x \in X$, we define a variable word $w_x \in \mathcal{P}$ and we ensure that, for all $x \neq x' \in X$, $\delta^*(q_0, w_x) \neq \delta^*(q_0, w_{x'})$. In the DFA $\mathcal{A}_{\mathsf{ex}}$, we have $\delta^*(q_0, w_{x_1}) = q'_{x_1}$, $\delta^*(q_0, w_{x_2}) = q'_{x_2}$, and $\delta^*(q_0, w_{x_3}) = q'_{x_3}$.
3. We add words in $\mathcal{P}$ to ensure that there is some $w_{\mathsf{Var}} \in \Sigma^*$, such that, for all $x \in X$, we have $\delta^*(q_0, w_x \cdot w_{\mathsf{Var}}) \in \{\delta^*(q_0, w_\top), \delta^*(q_0, w_\bot)\}$. In Example 1, $w_{\mathsf{Var}} = a$. When this property is ensured, a valuation $\nu\colon X \to \{\top, \bot\}$ is uniquely defined by, for all $x \in X$: $\delta^*(q_0, w_x \cdot w_{\mathsf{Var}}) = \delta^*(q_0, w_{\nu(x)})$. (Note that the valuation $\nu\colon X \to \{\top, \bot\}$ is unique because $\delta^*(q_0, w_\top) \neq \delta^*(q_0, w_\bot)$.)
4. For each clause $C \in \mathcal{C}$, we define a clause word $w_C \in \mathcal{P}$ and we ensure that, for all $C, C' \in \mathcal{C}$, $\delta^*(q_0, w_C) \neq \delta^*(q_0, w_{C'})$. In the DFA $\mathcal{A}_{\mathsf{ex}}$, we have $\delta^*(q_0, w_{C_1}) = q_{C_1}$, and $\delta^*(q_0, w_{C_2}) = q_{C_2}$.
5. We add words in $\mathcal{P}$ to ensure that there is some $w_{\mathsf{Cl}} \in \Sigma^*$, such that, for all $C \in \mathcal{C}$, there is some variable $x_C \in C$ such that we have $\delta^*(q_0, w_C \cdot w_{\mathsf{Cl}}) = \delta^*(q_0, w_{x_C})$. In Example 1, we have $w_{\mathsf{Cl}} = b$.
6. Finally, we add words in $\mathcal{P}$ to ensure that, for all clauses $C \in \mathcal{C}$, we have: $\delta^*(q_0, w_C \cdot w_{\mathsf{Cl}} \cdot w_{\mathsf{Var}}) = \delta^*(q_0, w_\top)$ if $C \in \mathcal{C}_+$, and $\delta^*(q_0, w_C \cdot w_{\mathsf{Cl}} \cdot w_{\mathsf{Var}}) = \delta^*(q_0, w_\bot)$ if $C \in \mathcal{C}_-$. In Example 1, this corresponds to the words $b \cdot b \cdot b \cdot a, b \cdot b \cdot a \cdot b \cdot a \cdot a \in \mathcal{P}$.

With Items 5. and 6., we have that the valuation defined in Item 3. satisfies $(X, \mathcal{C})$. Indeed, consider e.g. a positive clause $C \in \mathcal{C}$. By Fact 5, there is variable $x_C \in C$ such that $\delta^*(q_0, w_C \cdot w_{\mathsf{Cl}}) = \delta^*(q_0, w_{x_C})$. By Fact 6, we have $\delta^*(q_0, w_{x_C} \cdot w_{\mathsf{Var}}) = \delta^*(q_0, w_C \cdot w_{\mathsf{Cl}} \cdot w_{\mathsf{Var}}) = \delta^*(q_0, w_\top)$. Thus, $\nu(x_C) = \top$ and the valuation ν satisfies the clause C. Note that, in the actual reduction described in the extended version (Bordais and Neider 2025), there are many words in $\mathcal{P}$ associated with $\top$, $\bot$, variables and clauses.

Now that we have described the general idea of the construction, we would like to sketch how we prove that a DFA satisfying conditions 1) and 2) necessarily ensures the various properties laid out in Items 1., 2., 3., 4., 5., and 6. above. We use two different kinds of arguments that leverage the two opposing constraints on DFAs satisfying conditions 1) on 2): on the one hand, such DFAs must not make too many errors (i.e., accepting too many words not in $\mathcal{P}$), this is leveraged by TME-arguments ("Too Many Errors"); on the other hand, such DFAs must not have too many states, this is leveraged by TMS-arguments ("Too Many States").

- TME-arguments are used to show that two words $w, w' \in \Sigma^*$ are mapped to two different states by the function $\delta^*(q_{\mathsf{init}}, \cdot)$. Typically, the structure of a TME-argument is as follows. Assume that there are $k+1$ words $(u_\sigma)_{\sigma \in [\![1, k+1]\!]}$ such that we have $w \cdot u_\sigma \in \mathcal{P}$ and $w' \cdot u_\sigma \notin \mathcal{P}$ for all $\sigma \in [\![1, k+1]\!]$. In that case, it cannot be that $\delta^*(q_{\mathsf{init}}, w) = \delta^*(q_{\mathsf{init}}, w')$, since otherwise we would have $w' \cdot u_\sigma \in \mathcal{L}(\mathcal{A}) \setminus \mathcal{P}$, for all $\sigma \in [\![1, k+1]\!]$, which is a contradiction with condition 2) (assuming that $2n-2$ is large enough). This type of argument allows to establish the properties described in Facts 1., 2. and 4.
- TMS-arguments are used to show that in two sets of words $W, W' \subseteq \Sigma^*$ there must be some $w \in W, w' \in W'$ such that $\delta^*(q_{\mathsf{init}}, w) = \delta^*(q_{\mathsf{init}}, w')$. To do so, it suffices to show that we have $|W| + |W'| > n$. TMS-arguments allow us to establish the properties described in Facts 3. and 5.

As for the properties of Fact 6, they are direct consequences of the properties of other facts (specifically, Fact 3 and Fact 5), and of the words that we define in $\mathcal{P}$.

Experiments

Our goal is now to use our new word-counting perspective to improve the state-of-the-art algorithm solving Task 1. All of our code and data is available on GitHub (cf. the top page). Before we discuss our algorithms, let us describe the experimental setup that we have used.

Experimental Setup. We ran all of our experiments on Ubuntu 24.04.2 LTS, using 30 GiB of RAM, 12 CPU cores, and a clock speed of 5.2 GHz.

In order to compare our algorithms with the state-of-the-art symbolic algorithm developed by Roy et al. (2023), we have used the inputs used in that paper to compare the performance of the symbolic algorithm and the counterexample guided algorithm developed by Avellaneda and Petrenko (2018). These inputs are constituted of 28 samples of around 1000 words of lengths from 1 to 10. These words are obtained by sampling the languages of randomly generated small DFAs (of at most 10 states), with an alphabet of size 3. As in (Roy et al. 2023), we consider a timeout (TO) of 1000s to solve instances of Task 1 and Task 2. All experiments are done with a number of states n between 1 and 8. For all samples, we have run the algorithms 10 times, and we use the mean runtime (with the standard deviation).

ILP Algorithm Solving Task 2

To solve Task 2, we encode it as an Integer Linear Programming (ILP) minimization problem. In that setting, the goal is to minimize a linear combination of integer variables, subject to a system of linear inequalities (with integer coefficients). When encoding Task 2 in the ILP setting, the quantity to minimize corresponds to the number of words up to length $2n-2$ accepted by a prospective DFA.

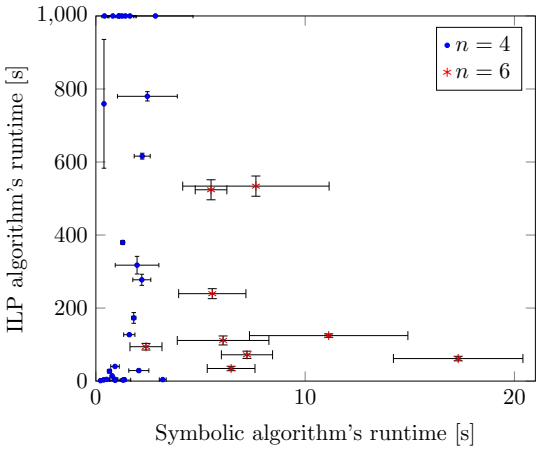

Figure 2: The comparison on the runtime of the symbolic and ILP algorithms.

Let us describe on a high level how our ILP encoding works—the details can be found in the extended version (Bordais and Neider 2025). Our encoding relies on two types of variables: binary variables, encoding the DFA and checking that it is in $\mathsf{Rec}(\mathcal{P}, n)$; and integer variables which are used to count the number of accepted words. The inequalities involving the binary variables are actually very close to the logical constraints used in the SAT-encoding developed by (Roy et al. 2023) (2023). As for the integer variables, our encoding relies on Lemma 1: the inductive relation described in this lemma allows us to store in an integer variable the number of words of length at most $2n-2$ accepted by the prospective DFA, which we can set as a minimization goal, subject to the other inequalities. Note that, to avoid having variable multiplication, we had to use the classical Big-M method (Griva, Nash, and Sofer 2008).

We implemented our algorithm in Gurobi (Gurobi Optimization, LLC 2024) which we have used with a Python3 API. In Figure 2, we have plotted in blue dots the computation time of our ILP algorithm (y-axis) and the computation time of the symbolic algorithm (x-axis) on each 28 samples with $n = 4$. Clearly, the ILP algorithm largely underperforms the symbolic algorithm. The situation is not significantly better for the case $n = 6$ plotted as red stars[2]. Overall, our ILP algorithm does not stand the comparison with the symbolic algorithm. We discuss below three possible reasons for this significant runtime difference.

First, from a theoretical standpoint, the symbolic algorithm could asymptotically take exponentially many steps to terminate. However, in practice, the symbolic algorithm does not take many steps to terminate (around 20, in the worst case), and thus greatly outperforms our ILP algorithm. There could be some kinds of samples for which the symbolic algorithm takes much more steps to conclude, in which case the runtime difference could be much smaller.

Furthermore, ILP solvers and SAT solvers internally use really different mechanisms: branch and bound for ILP solvers and DPLL for SAT solvers. It could be that in the case of these samples, DPLL works much better than branch and bound. This could be tested by implementing a binary search algorithm solving Task 2 by querying SAT solvers.

Finally, the ILP algorithm solves Task 2, while the symbolic algorithm solves Task 1. We have shown in Proposition 1 that Task 2 is at least as hard to solve as Task 1, it could very well be that it is simply much harder to solve.

[2]We only plotted the samples for which the ILP algorithm did not reach a full timeout (i.e., all 10 attempts timeout) before $n = 6$.

A Pre-Processing Algorithm

The symbolic algorithm starts from the trivial accepting DFA and then computes a $\prec_{\mathcal{L}}$-decreasing chain of DFAs until it finds a language-minimal one. To speed-up this process, we have designed a heuristic algorithm that computes a DFA that can then be used as a starting point for the symbolic algorithm. We first describe this heuristic algorithm, and then we discuss its experimental evaluation.

Description of the Heuristic Algorithm. The goal of this heuristic algorithm is to output a DFA $\mathcal{A} \in \mathsf{Rec}(\mathcal{P}, n)$ that accepts as few words of length at most $2n - 2$ as possible. The general idea of the algorithm is to randomly search among DFAs, and pick one that minimizes the number of accepted words. However, it is not clear how to randomly search among DFAs in $\mathsf{Rec}(\mathcal{P}, n)$. Therefore, our heuristic algorithm focuses on transition systems instead. A transition system can be seen as a DFA without starting or final states, i.e., it is described by a tuple $T = (Q, \Sigma, \delta)$ with $\delta \colon Q \times \Sigma \to Q$. Consider then a transition system $T = (Q, \Sigma, \delta)$ with $|Q| \leq n$ and a starting state $q \in Q$. It is clear that choosing $F_{T,q} := \{\delta^*(q, w) \mid w \in \mathcal{P}\}$ minimizes the number of words accepted by the DFA $\mathcal{A}_{T,q} := (Q, \Sigma, \delta, q, F_{T,q})$ while ensuring that $\mathcal{A}_{T,q} \in \mathsf{Rec}(\mathcal{P}, n)$. Following, given a transition system T, we define the *score* of the transition system T as follows: $\min_{q \in Q} |\mathcal{L}(\mathcal{A}_{T,q}) \cap \Sigma^{\leq 2n-2}|$. We naturally extend the notion of score to DFAs. Our heuristic algorithm then proceeds as follows:

1. Randomly search ($\mathsf{InitRand}$ attempts) n-states transition systems and pick one with the best (i.e., lower) score.
2. From the current transition system, look through all possible changes of a single transition. Make the change that improves the score the most, if one exists. Otherwise, stop. (Theoretically, this could take exponentially many steps to terminate, in practice, around 30 steps suffice.)
3. Do the two above steps NbRun times, pick the transition system T with the best score, extract a DFA from T with the best score (over all possible starting states).

This heuristic algorithm has two hyper-parameters: $\mathsf{InitRand}$ and NbRun. From several experiments conducted on sample 02 (with $\mathsf{InitRand} \in \{10, 100, 1000\}$ and $\mathsf{NbRun} \in \{10, 20, 50, 100\}$), we have chosen $\mathsf{InitRand} = 100$ and $\mathsf{NbRun} = 50$ as a compromise between running time and efficiency, though it is plausible that better values could be chosen (even for sample 02).

Experimental Evaluation. We have implemented this heuristic algorithm in C++ and compared the performance

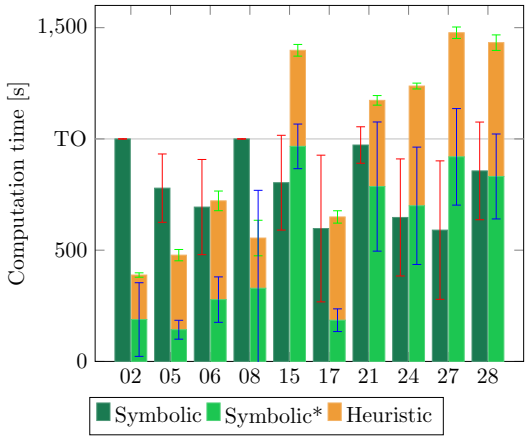

Figure 3: The runtime comparison of the symbolic and the heuristic+symbolic* algorithms.

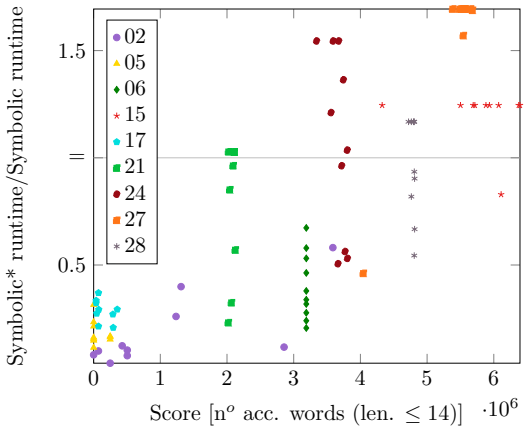

Figure 4: The quotient of the runtime of the symbolic(*) algorithms as a function of the starting score.

of the symbolic algorithm starting from the trivial accepting DFA and of the symbolic algorithm starting from a DFA computed by the heuristic algorithm, referred to as the symbolic* algorithm. Since the runtime of the heuristic algorithm is not negligible, the experiments are performed on the samples (and number of states n) for which the symbolic algorithm takes at least 500s. This covers 10 samples, all with $n = 8$ except for sample 08, where $n = 7$. The results are summarized in Figure 3.

There are two main positive takeaways from these results. First, there are three samples (02, 05, 08) where our algorithm (heuristic+symbolic*) significantly outperforms the symbolic algorithm. This is a promising result with a portfolio approach in mind where both the heuristic+symbolic* and symbolic algorithms would run in parallel. Second, there are five samples (02, 05, 06, 08, 17) where the computation time of the symbolic* algorithm is much lower than that of the symbolic algorithm. This result is promising for harder instances where the runtime of the heuristic algorithm becomes more-or-less negligible compared to the runtime of the symbolic algorithm.

On the negative side, on the five other samples, the symbolic* algorithm does not outperform the symbolic algorithm, and even under-performs it for sample 27. This shows that a lower starting score (i.e., score of a starting DFA) does not always mean a better performance of the symbolic algorithm. To better identify how the starting score impacts the runtime of the symbolic algorithm, we have represented relevant data in Figure 4.

In this Figure, each data point corresponds to a run of the symbolic* algorithm[3]: in the x-axis, the starting score is depicted, in the y-axis is depicted the quotient of the runtime of the symbolic* algorithm and the mean of the runtime of the symbolic algorithm on that sample.

From this figure, we can extract the following facts: when starting from a very low score, the runtime ratio is small (as exemplified for samples 02, 05, 17 and also possibly 27); and when starting from higher scores, the impact of the score on the runtime is hard to see. These facts may suggest that the significant differences on the various samples may mostly come from the fact that for some samples the heuristic algorithm has found a starting DFA with a (very) low score, while it is not the case for the other samples. Following, improving the capacity of our heuristic algorithm at finding DFAs with (very) low score (when possible) may have the potential to greatly lower the runtime of the heuristic+symbolic* algorithm. Nonetheless, how the starting score influences the runtime of the SAT solver remains rather cryptic despite our experiments, and further investigating this relationship seems warranted to improve the efficiency of the minimizing-score heuristics approach.

[3]To compare the scores, we have only depicted the samples where the computation of the heuristic+symbolic* algorithm is done with $n = 8$, thus we excluded sample 08.

Conclusion

In this paper, we have studied the task of learning DFAs from positive examples only with a novel word-counting perspective. The core of our studies is theoretical: we have shown the first complexity result on a decision problem directly related to this task, and we have exhibited a new algorithm solving this task with better theoretical guarantees than the state-of-the-art. On the experimental side, our ILP algorithm performs poorly, but our heuristic algorithm shows promising results as a pre-processing tool to use prior to running the symbolic algorithm. Notably, this algorithm shows that transition systems are particularly well-suited for handling this learning from positive examples task.

Overall, we believe that a better (theoretical and experimental) understanding of the relationship between the language minimality of DFAs and their number of accepted words would provide a valuable insight in the design of efficient learning algorithms.

Acknowledgements

This work has been financially supported by Deutsche Forschungsgemeinschaft, DFG Project number 434592664.

References

Angluin, D. 1980. Finding Patterns Common to a Set of Strings. *J. Comput. Syst. Sci.*, 21(1): 46–62.

Angluin, D. 1987. Learning Regular Sets from Queries and Counterexamples. *Inf. Comput.*, 75(2): 87–106.

Avellaneda, F.; and Petrenko, A. 2018. Inferring DFA without Negative Examples. In Unold, O.; Dyrka, W.; and Wieczorek, W., eds., *Proceedings of the 14th International Conference on Grammatical Inference, ICGI 2018, Wrocław, Poland, September 5-7, 2018*, volume 93 of *Proceedings of Machine Learning Research*, 17–29. PMLR.

Bharadiya, J. P. 2023. Artificial intelligence in transportation systems a critical review. *American Journal of Computing and Engineering*, 6(1): 35–45.

Bollig, B.; Habermehl, P.; Kern, C.; and Leucker, M. 2009. Angluin-Style Learning of NFA. In Boutilier, C., ed., *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, 1004–1009.

Bordais, B.; and Neider, D. 2025. Learning DFAs from Positive Examples Only via Word Counting. arXiv:2511.08431.

Camacho, A.; and McIlraith, S. A. 2019. Learning Interpretable Models Expressed in Linear Temporal Logic. In Benton, J.; Lipovetzky, N.; Onaindia, E.; Smith, D. E.; and Srivastava, S., eds., *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2019, Berkeley, CA, USA, July 11-15, 2019*, 621–630. AAAI Press.

Carrasco, R. C.; and Oncina, J. 1999. Learning deterministic regular grammars from stochastic samples in polynomial time. *RAIRO Theor. Informatics Appl.*, 33(1): 1–20.

Chen, X.-Q.; Ma, C.-Q.; Ren, Y.-S.; Lei, Y.-T.; Huynh, N. Q. A.; and Narayan, S. 2023. Explainable artificial intelligence in finance: A bibliometric review. *Finance Research Letters*, 56: 104145.

Gold, E. M. 1967. Language Identification in the Limit. *Inf. Control.*, 10(5): 447–474.

Gold, E. M. 1978. Complexity of Automaton Identification from Given Data. *Inf. Control.*, 37(3): 302–320.

Grinchtein, O.; Leucker, M.; and Piterman, N. 2006. Inferring Network Invariants Automatically. In Furbach, U.; and Shankar, N., eds., *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4130 of *Lecture Notes in Computer Science*, 483–497. Springer.

Griva, I.; Nash, S. G.; and Sofer, A. 2008. *Linear and nonlinear optimization 2nd edition*. SIAM.

Gurobi Optimization, LLC. 2024. Gurobi Optimizer Reference Manual.

Heule, M.; and Verwer, S. 2010. Exact DFA Identification Using SAT Solvers. In Sempere, J. M.; and García, P., eds., *Grammatical Inference: Theoretical Results and Applications, 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings*, volume 6339 of *Lecture Notes in Computer Science*, 66–79. Springer.

Ielo, A.; Law, M.; Fionda, V.; Ricca, F.; Giacomo, G. D.; and Russo, A. 2023. Towards ILP-Based LTL f Passive Learning. In Bellodi, E.; Lisi, F. A.; and Zese, R., eds., *Inductive Logic Programming - 32nd International Conference, ILP 2023, Bari, Italy, November 13-15, 2023, Proceedings*, volume 14363 of *Lecture Notes in Computer Science*, 30–45. Springer.

Lingg, J.; de Oliveira Oliveira, M.; and Wolf, P. 2024. Learning from positive and negative examples: New proof for binary alphabets. *Information Processing Letters*, 183: 106427.

Rabin, M. O.; and Scott, D. S. 1959. Finite Automata and Their Decision Problems. *IBM J. Res. Dev.*, 3(2): 114–125.

Roy, R.; Gaglione, J.; Baharisangari, N.; Neider, D.; Xu, Z.; and Topcu, U. 2023. Learning Interpretable Temporal Properties from Positive Examples Only. In Williams, B.; Chen, Y.; and Neville, J., eds., *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, 6507–6515. AAAI Press.

Shahbaz, M.; and Groz, R. 2009. Inferring Mealy Machines. In Cavalcanti, A.; and Dams, D., eds., *FM 2009: Formal Methods, Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings*, volume 5850 of *Lecture Notes in Computer Science*, 207–222. Springer.

Shvo, M.; Li, A. C.; Icarte, R. T.; and McIlraith, S. A. 2021. Interpretable Sequence Classification via Discrete Optimization. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, 9647–9656. AAAI Press.

Stolcke, A.; and Omohundro, S. 1992. Hidden Markov model induction by Bayesian model merging. *Advances in neural information processing systems*, 5.

Weiss, G.; Goldberg, Y.; and Yahav, E. 2024. Extracting automata from recurrent neural networks using queries and counterexamples (extended version). *Machine Learning*, 113(5): 2877–2919.

Zhang, Y.; Weng, Y.; and Lund, J. 2022. Applications of explainable artificial intelligence in diagnosis and surgery. *Diagnostics*, 12(2): 237.