


Efficient Decomposition Identification of Deterministic Finite Automata from Examples

Junjie Meng¹, Jie An², Yong Li³, Andrea Turrini³, Fanjiang Xu²,
Naijun Zhan⁴, and Miaomiao Zhang¹

¹ School of Computer Science and Technology, Tongji University, Shanghai, China

² National Key Lab. of Space Integrated Information System, Institute of Software, Chinese Academy of Sciences, Beijing, China

³ Key Lab. of System Software (Chinese Academy of Sciences) and State Key Lab. of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China

⁴ School of Computer Science & MOE Key Laboratory of High Confidence Software Technologies, Peking University, China
`anjie@iscas.ac.cn`, `liyong@ios.ac.cn`

Abstract. The identification of deterministic finite automata (DFAs) from labeled examples is a cornerstone of automata learning, yet traditional methods focus on learning monolithic DFAs, which often yield a large DFA lacking simplicity and interoperability. Recent work addresses these limitations by exploring DFA decomposition identification problems (DFA-DIPs), which model system behavior as intersections of multiple DFAs, offering modularity for complex tasks. However, existing DFA-DIP approaches depend on SAT encodings derived from Augmented Prefix Tree Acceptors (APTAs), incurring scalability limitations due to their inherent redundancy.

In this work, we advance DFA-DIP research through studying two variants: the traditional Pareto-optimal DIP and the novel states-optimal DIP, which prioritizes a minimal number of states. We propose a novel framework that bridges DFA decomposition with recent advancements in automata representation. One of our key innovations replaces APTA with 3-valued DFA (3DFA) derived directly from labeled examples. This compact representation eliminates redundancies of APTA, thus drastically reducing variables in the improved SAT encoding. Experimental results demonstrate that our 3DFA-based approach achieves significant efficiency gains for the Pareto-optimal DIP while enabling a scalable solution for the states-optimal DIP.

Keywords: DFA decomposition · DFA identification · Model learning · Passive learning · SAT solving · Grammatical inference.

1 Introduction

The identification of Deterministic Finite Automata (DFAs) from labeled examples is a fundamental problem in computer science, with applications in inference of network invariants [7], grammatical inference [10], model checking [14],

reinforcement learning [13], etc. Known as passive model learning [18], classical methods focus on inferring a single DFA from examples. The generated DFA may have a large size and thus suffers from a lack of simplicity and interpretability, as a single DFA representing complex system behaviors can have a very intricate structure.

To address this issue, recent research [13] has moved toward the *DFA decomposition identification problem* (DFA-DIP), i.e., inferring a group of DFAs from examples, where their conjunction language includes all positive examples and excludes all negative examples. This approach allows for capturing sub-tasks performed by the system, with system behavior being described as the intersection of the languages from several DFAs, thereby improving interpretability.

Existing DFA identifying approaches [9, 19, 20] typically first employ a data structure called the Augmented Prefix Tree Acceptor (APTA) [3] to represent the labeled examples and then encode the identification of the minimal DFA from the APTA as a satisfiability (SAT) problem for Boolean formulas. The number of Boolean variables required in the SAT problem grows polynomially with the size of the constructed APTA. However, the size of the APTA increases dramatically with both the number and the length of the examples, since every prefix of the examples corresponds to a unique state in the APTA. This results in a corresponding increase in the number of Boolean variables and, consequently, the size of the SAT problem, posing a significant challenge to SAT solvers.

To alleviate this challenge, a recent work [5] extends the technique presented in [4] to construct the minimal 3-valued DFA (3DFA) consistent with the given examples, i.e., the 3DFA accepts all positive examples and rejects all negative examples. According to its definition, APTA can be viewed as a specific kind of 3DFA. The constructed minimal 3DFA is dramatically smaller than the original APTA. Hence, the proposed method via minimal 3DFAs in [5] significantly improves the DFA identification.

Since the state-of-the-art algorithm for identifying DFA decompositions [13] still relies on the basic APTA construction, a natural improvement would be to apply this minimal 3DFA construction to further reduce the number of Boolean variables required. However, our findings suggest that the minimal 3DFA construction from [5] *cannot* be directly applied to the current DFA decomposition learning framework [13]. This is because with a minimal 3DFA, its structural characteristics are different from those of the prefix tree, which makes the original encoding no longer applicable (cf. Section 4).

Contributions. To advance the DFA decomposition identification research, we make several contributions in this paper, as summarized below:

- We review the Pareto-optimal DIP studied in [13] and introduce a novel DFA-DIP, named the states-optimal DIP, that prioritizes decompositions with smaller state spaces. (Section 3)
- We propose a new method for constructing a succinct 3DFA consistent with the given examples and an improved SAT encoding via 3DFA, both tailored for DFA-DIPs. (Section 4)
- We propose a method for solving the novel states-optimal DIP. (Section 5)

- We have implemented the improved method for solving the Pareto-optimal DIP by replacing the original encoding via APTA in [13] with our improved encoding via 3DFA, and have also implemented our method for solving the states-optimal DIP. The experimental results show that our new 3DFA construction significantly reduces the number of states compared to using APTA and dramatically improves the efficiency in solving the Pareto-optimal DIP. Additionally, we present preliminary experiments on the novel states-optimal DIP, highlighting the scalability of our method. (Section 6)

Related work. We review the most related work on DFA identification from examples. The most common approach is the evidence driven state-merging (EDSM) algorithm [12]. It first constructs an APTA consistent with the given examples, and then iteratively applies a state-merging procedure until no valid merges are left. However, the issue of this algorithm is that it terminates at a local optimum. Current SAT-based methods typically construct an APTA first, and then reduce the problem to a SAT solving problem [9], with its help. These approaches return a minimal DFA. Several works [16, 17, 19, 20] have improved the second step by proposing symmetry-breaking techniques and compact SAT encoding. More recently, the first step, which is an APTA construction, has been improved in [5] by extending the technique in [4] to construct a minimal 3DFA consistent with the given examples, instead of an APTA. The minimal 3DFA requires fewer states than APTA. However, all these works only consider learning a *single* DFA from the examples. In contrast to them, [13] introduces DFA-DIP and extends the SAT encoding via APTA to infer DFA decompositions under their so-called Pareto-optimal partial order. As the minimal 3DFA cannot directly be used for DFA-DIP, our work proposes a method to construct a non-minimal 3DFA with fewer states than APTA and an improved SAT encoding for DFA-DIP. All the works above, including ours, focus on learning *unknown* DFA decompositions from examples. When the finite-state automata to decompose are known, specific approaches [1, 6, 11] can be used.

Outline. After reviewing the basic definitions in Section 2, we give the formal definitions of the two DFA-DIPs in Section 3. We then introduce our improved SAT encoding via 3DFA in Section 4 and our method for solving the states-optimal DIP and the Pareto-optimal DIP in Section 5. We present our experimental evaluation in Section 6 and some concluding remarks in Section 7.

In what follows, due to the limited space available, the omitted proofs of lemmas and theorems are given in Appendix A.

2 Preliminaries

In this paper, given $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, 2, \dots, n\}$. We fix a finite alphabet Σ of letters. A *word* u is a finite sequence of letters in Σ . We denote by ε the empty word and with Σ^* the set of all finite words, and let $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. Given a word u , we denote by $|u|$ the *length* of u ($|\varepsilon| = 0$) and by $u[i]$ the i -th letter of u for $0 \leq i < |u|$. Given two words u and v , we denote by $u \cdot v$ (for short

uv) their concatenation. We say that a word u is a *prefix* of a word w if $w = u \cdot v$ for some word $v \in \Sigma^*$. We denote by $\text{prefixes}(u)$ the set of all prefixes of u and we extend it to a set of words S in the usual way, i.e., $\text{prefixes}(S) = \bigcup_{u \in S} \text{prefixes}(u)$.

Transition system. A *deterministic* transition system (TS) is a tuple $\mathcal{T} = (Q, \iota, \delta)$, where Q is a finite set of states, $\iota \in Q$ is the initial state, and $\delta: Q \times \Sigma \rightarrow Q$ is a transition function. We also extend δ from letters to words in the usual way, by letting $\delta(q, \varepsilon) = q$ and $\delta(q, a \cdot u) = \delta(\delta(q, a), u)$, where $u \in \Sigma^*$ and $a \in \Sigma$. The *run* of a TS \mathcal{T} on a finite word u of length n is the sequence of states $\rho = q_0 q_1 \cdots q_n \in Q^+$ such that, for every $0 \leq i < n$, $q_{i+1} = \delta(q_i, u[i])$.

Definition 1 (Deterministic finite automata). A *deterministic finite automaton* (DFA) is a tuple $\mathcal{A} = (\mathcal{T}, A)$, where \mathcal{T} is a deterministic TS and $A \subseteq Q$ is a set of accepting states.

It is easy to extend DFAs to process languages with “don’t-care” words.

Definition 2 (3-valued DFAs [5]). A *3-valued DFA* (3DFA) is a triple $\mathcal{A} = (\mathcal{T}, A, R)$, where \mathcal{T} is a deterministic TS and A, R , and $D = Q \setminus (A \cup R)$ partition the set of states Q , where $A \subseteq Q$ is the set of accepting states, $R \subseteq Q$ is the set of rejecting states, and the remaining states D are called don’t-care states.

A run is *accepting* (respectively, *rejecting*) if it ends in an accepting (resp. rejecting) state. A finite word $u \in \Sigma^*$ is *accepted* (resp. *rejected*) by \mathcal{A} if it has an accepting (resp. rejecting) run on u . 3DFAs map all words in Σ^* to *three* values: accepting (+), rejecting (−), and don’t-care (?), where they are accepting if they have an accepting run, rejecting if they have a rejecting run, and don’t-care otherwise. Note that DFAs are a special type of 3DFAs with only accepting and rejecting states. We denote the language of \mathcal{A} by $\mathcal{L}(\mathcal{A})$, i.e., the set of words accepted by \mathcal{A} .

3 Problem Formalization

In this work we consider two specific DFA decomposition identification problems (DIPs). One is the Pareto-optimal DIP established in [13], and the other is the states-optimal DIP we introduce in this paper. Before formally presenting the two DIPs, we recall the definition of DFA decompositions.

Definition 3 (DFA decomposition [13]). Let \mathcal{H} be the set of all DFAs over Σ . A (m_1, \dots, m_n) -DFA decomposition is a tuple of n DFAs $(\mathcal{A}_1, \dots, \mathcal{A}_n) \in \mathcal{H}^n$ such that each DFA \mathcal{A}_i has m_i states and $m_1 \leq m_2 \leq \dots \leq m_n$.

A decomposition $(\mathcal{A}_1, \dots, \mathcal{A}_n)$ accepts a word u if and only if all DFAs accept u , i.e., $u \in \mathcal{L}(\mathcal{A}_i)$ for all $1 \leq i \leq n$; a word that is not accepted is rejected. Therefore its language $\mathcal{L}(\mathcal{A}_1, \dots, \mathcal{A}_n)$ is the intersection of the languages of the individual DFAs, i.e., $\mathcal{L}(\mathcal{A}_1, \dots, \mathcal{A}_n) = \bigcap_{1 \leq i \leq n} \mathcal{L}(\mathcal{A}_i)$. We say (m_1, \dots, m_n) is the *states allocation* of the DFA decomposition $(\mathcal{A}_1, \dots, \mathcal{A}_n)$.

DFA decomposition identification problem (DFA-DIP). Given a set of labeled examples $S = (S^+, S^-)$ where S^+ contains positive examples and S^- contains negative examples, respectively, a general DFA-DIP asks to identify a DFA decomposition consistent with the example set S ; formally, it asks to find a DFA decomposition $(\mathcal{A}_1, \dots, \mathcal{A}_n)$ for a given integer $n \in \mathbb{N}$ that satisfies:

C1 Consistency: $S^+ \subseteq \mathcal{L}(\mathcal{A}_1, \dots, \mathcal{A}_n)$ and $S^- \subseteq \Sigma^* \setminus \mathcal{L}(\mathcal{A}_1, \dots, \mathcal{A}_n)$.

Obviously, the identification problem of monolithic DFAs is a special case of DFA-DIP with $n = 1$, i.e., learning a single DFA from examples. We say a DFA decomposition $(\mathcal{A}_1, \dots, \mathcal{A}_n)$ is *consistent* with S if it satisfies **C1**.

To compare decompositions, a *Pareto-optimal partial order* \prec has been introduced in [13]. Formally, given a (m_1, \dots, m_n) -DFA decomposition $(\mathcal{A}_1, \dots, \mathcal{A}_n)$ and a (m'_1, \dots, m'_n) -DFA decomposition $(\mathcal{A}'_1, \dots, \mathcal{A}'_n)$, we say $(\mathcal{A}_1, \dots, \mathcal{A}_n) \prec (\mathcal{A}'_1, \dots, \mathcal{A}'_n)$ if $m_i \leq m'_i$ for all $1 \leq i \leq n$ and there exists $1 \leq j \leq n$ such that $m_j < m'_j$. If so, we say that $(\mathcal{A}_1, \dots, \mathcal{A}_n)$ *dominates* $(\mathcal{A}'_1, \dots, \mathcal{A}'_n)$.

As incomparable decompositions exist, they can form a Pareto frontier of solutions to the Pareto-optimal DIP defined as follows:

Pareto-Optimal DIP [13]. Given a set of labelled examples $S = (S^+, S^-)$, and an integer $n \in \mathbb{N}$, find a (m_1, \dots, m_n) -DFA decomposition $(\mathcal{A}_1, \dots, \mathcal{A}_n)$ such that (i) it satisfies **C1**, and (ii) there does not exist a decomposition satisfying **C1** dominating it under the Pareto-optimal partial order \prec .

In this paper, we also introduce a novel DFA-DIP — states-optimal DIP — which depends on the number of states and the *entropy* of a decomposition defined as follows.

Definition 4 (Entropy of DFA decomposition). Given a (m_1, \dots, m_n) -DFA decomposition $(\mathcal{A}_1, \dots, \mathcal{A}_n)$, its entropy is defined as

$$\mathcal{E}(\mathcal{A}_1, \dots, \mathcal{A}_n) = - \sum_{i=1}^n P(i) \log_2 P(i)$$

where $P(i) = m_i / \sum_{j=1}^n m_j$ for $1 \leq i \leq n$.

Based on Definition 4, we present a *states-optimal preorder* as follows. Given a (m_1, \dots, m_n) -DFA decomposition $(\mathcal{A}_1, \dots, \mathcal{A}_n)$ and a (m'_1, \dots, m'_l) -DFA decomposition $(\mathcal{A}'_1, \dots, \mathcal{A}'_l)$, we say $(\mathcal{A}_1, \dots, \mathcal{A}_n) \preceq (\mathcal{A}'_1, \dots, \mathcal{A}'_l)$ if $\sum_{i=1}^n m_i < \sum_{j=1}^l m'_j$, or $\mathcal{E}(\mathcal{A}_1, \dots, \mathcal{A}_n) \geq \mathcal{E}(\mathcal{A}'_1, \dots, \mathcal{A}'_l)$ if $\sum_{i=1}^n m_i = \sum_{j=1}^l m'_j$. The preorder reflects that we prefer a DFA decomposition that has the minimal number of states or, for the same number of states, it contains more individual DFAs with similar (smaller) size.

A decomposition with a higher entropy value indicates a more evenly distributed state allocation among DFAs, leading to a more flexible and potentially more generalizable representation of the input data. In contrast, lower entropy values suggest that the state allocation is concentrated in fewer DFAs, which

may reduce the system’s ability to capture diverse structural variations in the data. Note that the states-optimal preorder does not require two DFA decompositions having the same number of DFAs, as does the Pareto-optimal partial order. Therefore, the states-optimal preorder is a total preorder.

States-Optimal DIP. Given a set of labelled examples $S = (S^+, S^-)$, find (m_1, \dots, m_n) -DFA decomposition $(\mathcal{A}_1, \dots, \mathcal{A}_n)$ for some $n \in \mathbb{N}$ such that it is a minimal decomposition w.r.t. the states-optimal preorder $<$ satisfying [C1](#).

Note that the states-optimal DIP does not require a given input n to restrict the number of DFAs in the resulted DFA decomposition. Of course, a specific variant is to find a states-optimal decomposition with a given input n .

4 An Improved SAT Encoding of DFA-DIP via 3DFAs

In this section, we first briefly review the existing encoding via APTA proposed in [\[13\]](#), then present our improved SAT encoding of DFA-DIP via 3DFA. We describe a challenge in utilizing minimal 3DFAs [\[5\]](#) within our encoding at last.

4.1 Existing encoding via APTAs

In [\[13\]](#), Lauffer *et al.* proposed a SAT encoding method for DFA-DIP by extending the SAT encoding for identifying a single DFA from examples [\[9, 16, 17, 19\]](#). Their encoding reduces DFA-DIP into a graph coloring problem where the graph is in fact an APTA, a tree-based 3DFA. In the graph coloring problem, the state of each node of the tree structure, each edge of the tree, and for each state, each transition of the DFA, can be represented by a different color variable [\[9\]](#).

In an APTA, each state corresponds to a unique prefix of a word in the set of examples $S = (S^+, S^-)$ that we also write as $S = S^+ \cup S^-$. First, one can define a function $f: \text{prefixes}(S) \rightarrow \mathbb{N}_S$ where $\mathbb{N}_S = \{0, \dots, |\text{prefixes}(S)| - 1\}$. Intuitively, f maps each prefix $u \in \text{prefixes}(S)$ to a state in the APTA represented by a unique number in \mathbb{N}_S . Formally, an APTA \mathcal{P} of S is a 3DFA (\mathcal{T}, A, R) where the TS \mathcal{T} consists of the set of states \mathbb{N}_S , the initial state $f(\varepsilon)$, and the transition function δ defined as $\delta(i, a) = j$ if $f(u) = i$ and $f(ua) = j$ where $u, ua \in \text{prefixes}(S)$ and $a \in \Sigma$; we define $A = \{i \in \mathbb{N}_S \mid f(u) = i \text{ for some } u \in S^+\}$ and $R = \{i \in \mathbb{N}_S \mid f(u) = i \text{ for some } u \in S^-\}$. For example, [Fig. 1\(a\)](#) shows the APTA generated from the set of examples $S = (S^+, S^-)$ where $S^+ = \{aab, aaa, ab\}$, $S^- = \{b, aba\}$, and $f = \{\varepsilon \mapsto 0, a \mapsto 1, aa \mapsto 2, aab \mapsto 3, aaa \mapsto 4, ab \mapsto 5, aba \mapsto 6, b \mapsto 7\}$. The set of accepting states is $A = \{3, 4, 5\}$ and the set of rejecting states is $R = \{6, 7\}$.

Given a positive integer $n \in \mathbb{N}$, the state-of-the-art encoding method associates the APTA states with the states of each DFA \mathcal{A}_i in the (m_1, \dots, m_n) -DFA decomposition $(\mathcal{A}_1, \dots, \mathcal{A}_n)$. More precisely, given an APTA consistent with the examples S and an allocation of states (m_1, \dots, m_n) , each unknown DFA \mathcal{A}_i gets assigned m_i states and the encoding associates each APTA state with one

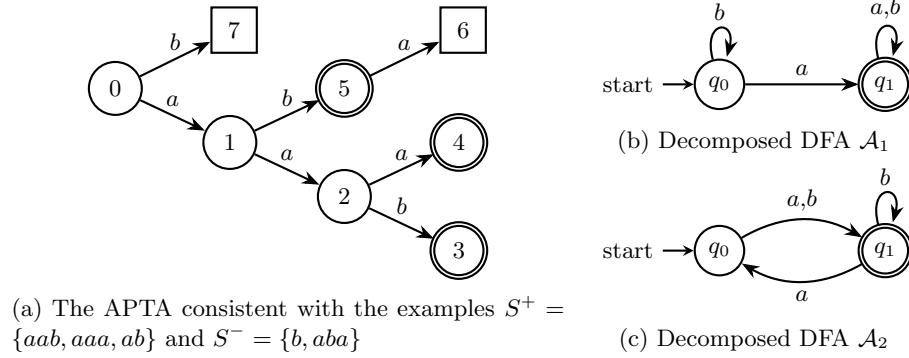


Fig. 1: (a) The generated APTA, where an accepting node is represented by a double circle and a rejecting node is represented by a square. (b) and (c) show the two DFAs in a corresponding $(2, 2)$ -DFA decomposition $(\mathcal{A}_1, \mathcal{A}_2)$ consistent with $S = (S^+, S^-)$.

of the \mathcal{A}_i 's m_i states, subject to a set of constraints, such as each accepting (rejecting, resp.) APTA state should be associated with an accepting (a rejecting, resp.) DFA state. APTA states with the same (DFA-indexed) state variable will be identified as the same state in the corresponding DFA. For instance, for each APTA state $v \in \mathbb{N}_S$, a state variable $x_{v,i}^k$ represents it is associated with the i -th state in the k -th DFA. Considering the APTA in Fig. 1(a), if we would like to generate a $(2, 2)$ -DFA decomposition $(\mathcal{A}_1, \mathcal{A}_2)$, we need 32 state variables $x_{v,i}^k$. Besides these variables, the encoding method requires other variables to represent the transition relations and the accepting conditions. Based on the coloring, several constraints imposed by **C1** – such as a positive example must be accepted by all DFAs and a negative example must be rejected by at least one DFA – will be encoded into a SAT problem. The complete list of Lauffer's encoding [13] can be found in Appendix B. Fig. 1(b) and 1(c) present the identified $(2, 2)$ -DFA decomposition $(\mathcal{A}_1, \mathcal{A}_2)$ consistent with the APTA in Fig. 1(a).

4.2 Our improved encoding via 3DFAs

The existing approaches [9, 13, 16, 17, 19] face a critical challenge: the size of the APTA grows dramatically with both the number and the length of examples S as APTA associates each prefix in $\text{prefixes}(S)$ with a unique node. As mentioned earlier, the increase in the number of nodes in the APTA leads to more encoding variables, which in turn increases the size of the resulting SAT problem [15, 19]. Consequently, the smaller the APTA size is, the easier the SAT problem will be.

In light of this, to obtain smaller APTAs, [5] proposed to construct the *minimal* 3DFA for S by merging equivalent nodes. Moreover, inspired by [4], the minimal 3DFA can even be constructed incrementally from the set S if the examples in S are sorted by the standard lexicographical order. Following [4, 5], we

also propose an incremental construction of 3DFAs by merging equivalent nodes. Differently from [5], our construction, rather than looking for the minimal 3DFA, requires that every rejecting word in S^- must reach a unique state in the constructed 3DFA. That is, we only merge equivalent accepting or “don’t-care” states. In what follows, we first introduce our construction of 3DFAs starting from APTAs and then the improved encoding via 3DFAs. Afterwards, we reveal the reason why it is important to associate every rejecting word with a unique state in the 3DFA.

3DFA Construction. For simplicity, we assume that the full APTA \mathcal{P} of S is given. Note that our 3DFA can also be constructed on-the-fly from S using the same techniques as [4, 5]. Since our goal is to associate each rejecting word with a unique state and merge as many other states as possible, our reduction process works in a backward manner as follows.

Initially, we collapse all accepting nodes without outgoing transitions into one representative state and store it in a hash map called **Register**. Since all rejecting nodes are inequivalent to other states, each rejecting node will be stored as its own representative in the **Register**.

Our reduction process then iteratively traverses the APTA nodes in a backward manner from leaves towards the root and process the 3DFA as follows. In each iteration, we first collect the states whose all successors are representative states in **Register**, and then identify equivalent states with the following two conditions:

- both states must be either accepting or don’t-care states, and
- for every input letter $a \in \Sigma$, either they both have no successors or both have the same successor in **Register**.

We thus create a state representative for each equivalent class, i.e., a set of equivalent states, and store the representative in **Register**. Further, all states that have a representative in **Register** will be replaced by their representative in the updated 3DFA.

Our construction repeats the reduction process until all states, including the initial one, are processed, resulting in a 3DFA consistent with S .

Theorem 1. *Given a set of examples $S = (S^+, S^-)$, the 3DFA construction produces a 3DFA consistent with S .*

Note that our 3DFA can also be constructed *on the fly* from S in the same manner as in [5] if the example words are taken out from S in the standard lexicographical order.

Observe that every rejecting word in S^- leads the constructed 3DFA to a unique rejecting state by construction. In fact, we can get the following stronger result, that we will use later in our SAT encoding phase.

Lemma 1. *Let $\mathcal{A} = (\mathcal{T}, A, R)$ be the outcome of the 3DFA construction, where $\mathcal{T} = (Q, \iota, \delta)$. For two different prefixes $u, u' \in \text{prefixes}(S)$, we have $\delta(\iota, u) \neq \delta(\iota, u')$ if either $u \in \text{prefixes}(S^-)$ or $u' \in \text{prefixes}(S^-)$.*

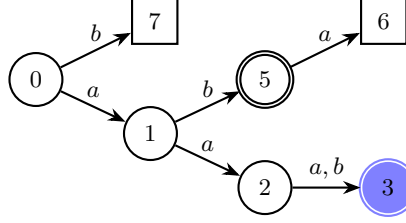


Fig. 2: The 3DFA $\mathcal{A} = (\mathcal{T}, A, R)$ constructed from the APTA in Fig. 1(a), where $A = \{3, 5\}$ and $R = \{6, 7\}$.

Table 1: Size comparison between 3DFAs and APTAs. “Length” indicates the length of each word.

$ \Sigma $	Length	Automata Size	
		3DFA	APTA
5	5	1,255	3,214
5	6	4,449	13,634
5	7	13,787	53,277
5	8	43,064	209,721
5	9	136,019	835,954
5	10	443,763	3,369,694

For instance, as shown in Fig. 2, this construction process can merge the states $\{3, 4\}$ of the APTA in Fig. 1(a) reached by positive examples into a representative state 3. Although this simple example does not clearly show the advantage of the construction, Table 1 presents the size comparison between APTAs and our constructed 3DFAs in terms of the number of states on a number of cases from parity game solving [5]. We can see that the resulting 3DFAs exhibit significantly smaller sizes than the original APTAs. Therefore, using 3DFAs instead of APTAs requires dramatically fewer variables and fewer constraints in our improved encoding given below, yielding easier SAT problems and thus faster solving speed in general than those via APTAs, as confirmed by the experiments in Section 6.

SAT Encoding via 3DFAs. We now present the standard SAT encoding for solving the DFA-DIP problem [13] adapted to our 3DFAs. Note that we also use the symmetry breaking techniques proposed in [16, 17] to further improve our SAT encoding, just like in [13]. In what follows, we focus on our major encoding. Given the 3DFA $\mathcal{A} = (\mathcal{T}, A, R)$ with $\mathcal{T} = (Q, \iota, \delta)$ consistent with S , we are looking for a consistent DFA decomposition $(\mathcal{A}_1, \dots, \mathcal{A}_n)$ of S where the state allocation is (m_1, \dots, m_n) for a given $n \in \mathbb{N}$. To encode the DFA-DIP, we use the following three types of variables:

1. **State variables** $x_{v,i}^k$, where $k \in [n]$, $v \in Q$, and $i \in [m_k]$. $x_{v,i}^k \equiv 1$ iff the state v of the 3DFA and the state i in the DFA \mathcal{A}_k can both be reached on some word $u \in \Sigma^*$ from their initial states.
2. **Transition relation variables** $e_{l,i,j}^k$, where $k \in [n]$, $l \in \Sigma$, and $i, j \in [m_k]$. $e_{l,i,j}^k \equiv 1$ iff DFA \mathcal{A}_k has a transition from state i to state j over the letter l .
3. **Acceptance variables** z_i^k , where $k \in [n]$ and $i \in [m_k]$. $z_i^k \equiv 1$ iff the state i of DFA \mathcal{A}_k is an accepting state.

Recall that A (resp., R) is the set of accepting (resp., rejecting) states in the 3DFA \mathcal{A} . We now give the list of constraints for the SAT encoding. First of all, we require that each individual automaton \mathcal{A}_i should be a DFA. That is, \mathcal{A}_i must be deterministic and also complete.

D1 Determinism. For a state i and a letter l in \mathcal{A}_k , there is at most one successor:

$$\bigwedge_{l \in \Sigma} \bigwedge_{k \in [n]} \bigwedge_{\substack{i, j, t \in [m_k] \\ j < t}} e_{l, i, j}^k \implies \neg e_{l, i, t}^k.$$

D2 Completeness. For a state i and a letter l in \mathcal{A}_k , there must be a successor:

$$\bigwedge_{l \in \Sigma} \bigwedge_{k \in [n]} \bigwedge_{i \in [m_k]} \bigvee_{j \in [m_k]} e_{l, i, j}^k.$$

Second, we require that the DFA decomposition $(\mathcal{A}_1, \dots, \mathcal{A}_n)$ should be consistent with S , so to satisfy **C1**.

R1 Positive Consistency. Every positive example in S^+ must be accepted by all individual DFAs, i.e., by each DFA \mathcal{A}_k where $k \in [n]$. It follows that, if $x_{v, i}^k \equiv 1$, then all positive examples leading the 3DFA \mathcal{A} to an accepting state v must also make \mathcal{A}_k reach an accepting state, i.e., state i must also be an accepting state.

$$\bigwedge_{v \in A} \bigwedge_{k \in [n]} \bigwedge_{i \in [m_k]} x_{v, i}^k \implies z_i^k.$$

R2 Negative Consistency. Each negative example in S^- should be rejected by at least one individual DFA \mathcal{A}_k . That is, if $x_{v, i}^k \equiv 1$, then the example leading the 3DFA \mathcal{A} to a rejecting state v must also make \mathcal{A}_k reach a rejecting state as well, i.e., state i of \mathcal{A}_k should also be rejecting.

$$\bigwedge_{v \in R} \bigvee_{k \in [n]} \left(\bigwedge_{i \in [m_k]} (x_{v, i}^k \implies \neg z_i^k) \right).$$

To further enforce the DFA decomposition to be consistent with S , we also need to perform the product of the 3DFA \mathcal{A} and each individual DFA \mathcal{A}_k in order to build the transition system of each \mathcal{A}_k , where $k \in [n]$. That is, $x_{v, i}^k \equiv 1$ also means that in the product automaton of \mathcal{A} and \mathcal{A}_k , the pair of states (v, i) is seen as a product state and it is reached from the initial product state $(r, 0)$ over some word $u \in \Sigma^*$. The constraints are listed below:

T1 Initialization. The initial state r of 3DFA \mathcal{A} should always be associated with the initial state 0 for each DFA \mathcal{A}_k :

AT: $x_{r, 0}^k$: no variable has 0 as subscript; shall it be 1 instead?

$$\bigwedge_{k \in [n]} x_{r, 0}^k$$

T2 State Correspondence. Each state v of 3DFA \mathcal{A} must be associated with one state in each DFA \mathcal{A}_k , i.e., $x_{v, i}^k \equiv 1$ for some state i in \mathcal{A}_k :

$$\bigwedge_{v \in Q} \bigwedge_{k \in [n]} \bigvee_{i \in [m_k]} x_{v, i}^k.$$

T3 Transition Relation. For each $k \in [n]$, in the product automaton of \mathcal{A} and \mathcal{A}_k , if the product state (v, i) is reachable from the initial product state $(r, 0)$, and there is a transition from state i to j over letter a in \mathcal{A}_k , then the product state $(\delta(v, a), j)$ is also reachable.

$$\bigwedge_{v \in Q} \bigwedge_{k \in [n]} \bigwedge_{i, j \in [m_k]} \bigwedge_{a \in l(v)} (x_{v,i}^k \wedge e_{a,i,j}^k) \implies x_{\delta(v,a),j}^k$$

where $l(v)$ is the set of letters on the outgoing transitions from state v .

In addition to the above constraints, Lauffer *et al.* [13] also proposed an optimization as follows.

O1 Each state v in the 3DFA can only be associated with at most one state i in each individual DFA \mathcal{A}_k .

$$\bigwedge_{v \in Q} \bigwedge_{k \in [n]} \bigwedge_{i, j \in [m_k], i < j} (\neg x_{v,i}^k \vee \neg x_{v,j}^k).$$

This constraint holds in the APTA because each state v corresponds to a unique prefix $u \in \text{prefixes}(S)$. Since both \mathcal{A} and \mathcal{A}_k are deterministic, the product state (v, i) reached from the initial product state $(r, 0)$ over u must be unique. That is, there will be a unique i in \mathcal{A}_k to make $x_{v,i}^k$ be true.

However, this constraint might not be true when we use a 3DFA for our encoding. For instance, consider the 3DFA \mathcal{A} in Fig. 2: we can see that there are two words that can make \mathcal{A} reach state 3, namely aaa and aab . In our target DFA \mathcal{A}_k , these two words might lead to two different states, say i and j . It then follows that we have both $x_{3,i}^k \equiv 1$ and $x_{3,j}^k \equiv 1$, which obviously violates the constraint **O1**. This will prevent us from finding a consistent DFA decomposition $(\mathcal{A}_1, \dots, \mathcal{A}_n)$ with the state allocation (m_1, \dots, m_n) .

Nonetheless, we observe that during the construction of the 3DFA, it is easy to identify those states that correspond to multiple different prefix words in $\text{prefixes}(S)$. Indeed, we can just record the representative states which correspond to multiple equivalent states, denoted by M . That is, for each state $v \in M$, there must be two different prefixes $u, u' \in \text{prefixes}(S)$ such that $v = \delta(r, u) = \delta(r, u')$. In fact, M must not contain the states corresponding to a prefix in $\text{prefixes}(S^-)$ as guaranteed by Lemma 1. The set of states in \mathcal{A} that is associated with a unique prefix word in $\text{prefixes}(S)$ is $Q \setminus M$. Therefore, we can replace the constraint **O1** with the following constraint:

O1' Each state v in the 3DFA that does not correspond to multiple equivalent states can only be associated with at most one state i in each individual DFA \mathcal{A}_k .

$$\bigwedge_{v \in Q \setminus M} \bigwedge_{k \in [n]} \bigwedge_{i, j \in [m_k], i < j} (\neg x_{v,i}^k \vee \neg x_{v,j}^k).$$

Clearly, by Lemma 1, all states having a path to a rejecting state in \mathcal{A} belong to $Q \setminus M$.

Let $\varphi_{(m_1, \dots, m_n)}^{\mathcal{A}, n}$ be the conjunction of the constraints **D1-2**, **R1-2**, **T1-3**, and **O1'**. Then, we get the following result.

Theorem 2. *Let \mathcal{A} be the 3DFA consistent with the examples S and $n \in \mathbb{N}$. $\varphi_{(m_1, \dots, m_n)}^{\mathcal{A}, n}$ is satisfiable if, and only if, there exists a (m_1, \dots, m_n) -DFA decomposition $(\mathcal{A}_1, \dots, \mathcal{A}_n)$ consistent with S .*

Encoding size. The size of the formula $\varphi_{(m_1, \dots, m_n)}^{\mathcal{A}, n}$ is determined by the conjunction of each constraint above. Let m be the maximal number in (m_1, \dots, m_n) . The contributions of each constraint to the formula size are as follows: (D1-2): $O(|\Sigma| \cdot n \cdot m^3) + O(|\Sigma| \cdot n \cdot m^2)$; (R1-2): $O(|Q^+| \cdot n \cdot m) + O(|Q^-| \cdot n \cdot m)$; (T1-3): $O(n) + O(|Q| \cdot n \cdot m) + O(|Q| \cdot n \cdot m^2)$; and (O1'): $O(|Q \setminus M| \cdot n \cdot m^2)$. The overall size is thus $O(|\Sigma| \cdot n \cdot m^3) + O(|Q| \cdot n \cdot m^2)$. \triangleleft

4.3 Why not use minimal 3DFAs

Our 3DFA construction is inspired by the one of **DFAMiner** [5], with the main difference being that we do not generate the minimal 3DFA recognizing S . In the minimal 3DFA generated by **DFAMiner**, all rejecting states that cannot reach a rejecting state are merged together. For instance, in Fig. 2, both rejecting states 6 and 7 do not reach another rejecting state, so in the minimal 3DFA, these two states will be merged. This also means that the two rejecting words b and aba will correspond to the same state, say v , in the minimal 3DFA \mathcal{A} . However, this would make the meaning of constraint **R2** imprecise: by constraint **R2**, we want every word to be rejected by at least one individual DFA, but the constraint actually says that we have at least one individual DFA that rejects a word associated with the rejecting state v . This means that it is *not* guaranteed that every word in S^- reaching v will be rejected. This means, for instance, that for b and aba in Fig. 2, it may happen that only b is rejected by some individual DFA but aba is totally ignored and thus not rejected by any DFA. This kind of issues can lead to inconsistent DFA decompositions from the SAT solver and we indeed observed them in the experiments when we directly used the minimal 3DFA in our encoding.

In light of this, it is clear why we require in our construction that every rejecting word must be associated with a unique state in the constructed 3DFA or, equivalently, that every rejecting state in the APTA must not be equivalent to any other states. We are now ready to introduce in the next section the overall algorithm for finding consistent DFA decompositions.

5 Identification Algorithms for DFA Decomposition

In this section, we present identification algorithms for two DFA decomposition problems: the Pareto-optimal DIP and the states-optimal DIP.

For the Pareto-optimal DIP, our algorithm builds on top of the approach of [13], where we replace their SAT encoding via APTA with our enhanced 3DFA-based encoding. The detailed algorithm can be found in Appendix B. As the experimental results in Section 6 show, this modification yields significant

Algorithm 1: State-optimal DIP Solving

Input: The labeled examples $S = \{S^+, S^-\}$
Output: A DFA decomposition \mathcal{D} for the states-optimal DIP

```

1  $\mathcal{A} \leftarrow \text{3DFAConstruction}(S);$   $\triangleright$  Construct a 3DFA from the examples
2  $N \leftarrow 2;$   $\triangleright$  Initial total number of states in possible decompositions
3 while true do
4    $\mathcal{M} \leftarrow \text{ComputeStatesAllocations}(N, 2);$   $\triangleright$  Get all possible states
     allocations under  $N$ 
5   foreach  $(m_1, \dots, m_n) \in \mathcal{M}$  do
6      $SAT, \mathcal{D} \leftarrow \text{Solve}((m_1, \dots, m_n), \mathcal{A});$ 
7     if  $SAT$  then
8       return  $\mathcal{D};$ 
9     end
10  end
11   $N \leftarrow N + 1;$ 
12 end

```

improvement. Therefore, in what follows, we focus on our method for solving the new states-optimal DIP, which is summarized as Algorithm 1.

Given a set of labeled examples $S = (S^+, S^-)$, Algorithm 1 first builds a 3DFA consistent with S (Line 1) according to Section 4.2 and then looks for the decomposition with the minimal total number of states, starting with $N = 2$ (Line 2) and incrementing it until a suitable decomposition is found. This is achieved by computing all possible states allocations \mathcal{M} (Line 4) having N total states by calling Algorithm 2, which ensures that every states allocation $(m_1, \dots, m_n) \in \mathcal{M}$ satisfies: (1) the total number of states is N (i.e., $\sum_{i=1}^n m_i = N$), (2) every corresponding DFA has at least 2 states¹ (i.e., $m_i \geq 2$ for all $1 \leq i \leq n$), and (3) allocated states are in ascending order (i.e., $m_i \leq m_{i+1}$ for all $1 \leq i < n$). As a result, Algorithm 2 enumerates all possible combinations where the number of DFAs n will range from 1 to $\lfloor \frac{N}{2} \rfloor$. For instance, for $N = 10$, Algorithm 2 returns the following states allocations:

$$\{(2, 2, 2, 2, 2), (2, 2, 3, 3), (2, 2, 2, 4), (3, 3, 4), (2, 4, 4), (2, 3, 5), \\ (2, 2, 6), (5, 5), (4, 6), (3, 7), (2, 8), (10)\}.$$

After calling Algorithm 2, for every possible states allocation $(m_1, \dots, m_n) \in \mathcal{M}$, Algorithm 1 applies our encoding method and calls a SAT solver to determine whether there is a (m_1, \dots, m_n) -DFA decomposition \mathcal{D} for the state-optimal DIP (Line 6). If so, it returns it (Line 8). Otherwise, it means that there is no solution under the current total number of states N , so we increase it (Line 11).

Note that, in Algorithm 1, we increase the total number of states N by 1 in each round. For every fixed N , Algorithm 2 sorts the states allocations in

¹ As the constraint **D2** in Section 4.2 requires that every state in the generated DFAs should have a transition for every action in Σ , the generated DFAs are complete DFAs. Since a complete DFA with a single state will either accept or reject all words, we skip such naïve DFAs and thus every DFA should have at least 2 states.

Algorithm 2: COMPUTESTATESALLOCATIONS

Input: N : the total number of states in unknown decompositions
 k : the minimum number of states in each DFA
Output: \mathcal{M} : all possible states allocations of unknown decompositions

```

1  $\mathcal{M} \leftarrow \{N\};$ 
2 for  $m \leftarrow k$  to  $N$  do                                 $\triangleright$  Current DFA has  $m$  states
3   if  $m \leq N - m$  then                                 $\triangleright$  Still enough states left for the next DFA
4      $\mathcal{M}' \leftarrow \text{COMPUTESTATESALLOCATIONS}(N - m, m);$ 
5      $\mathcal{M} \leftarrow \mathcal{M} \cup (\{m\} \times \mathcal{M}');$   $\triangleright$  Add the new states allocations to  $\mathcal{M}$ 
6   end
7 end
8 Sort  $\mathcal{M}$  in descending order by entropy (see Definition 4);
9 return  $\mathcal{M};$ 

```

descending order based on their entropy values (Line 8). Therefore, they work together to keep the states-optimal preorder defined in Section 3. Theorem 3 shows the termination and the correctness of Algorithm 1.

Theorem 3 (Termination and Correctness). *Let $S = (S^+, S^-)$ be a given set of labeled examples. Algorithm 1 terminates and returns a correct DFA decomposition $\mathcal{D} = (\mathcal{A}_1, \dots, \mathcal{A}_n)$ for the states-optimal DIP.*

Complexity analysis. It can be observed that there is always a correct decomposition $\mathcal{D} = (\mathcal{A}_1, \dots, \mathcal{A}_n)$ when $N = 2 + \sum_{u \in S^-} (|u| + 2)$, where a 2-states DFA \mathcal{A}_1 accepts all words and every other DFA rejects one negative example $u \in S^-$. Therefore, the loop iterations in Algorithm 1 are bounded by $1 + \sum_{u \in S^-} (|u| + 2)$, as N starts with 2 and it is increased by 1 every round. In every loop, the recursive generation of all possible states allocations under a given N in Algorithm 2 is dominated by the integer partition problem. According to the Hardy-Ramanujan formula [8], the asymptotic estimation of the number of states allocations is $|\mathcal{M}| \sim \frac{e^{\pi\sqrt{2N/3}}}{4N\sqrt{3}}$, which shows that the growth of $|\mathcal{M}|$ is subexponential in N . \triangleleft

It is not hard to have a simple variant of Algorithm 1 with a given integer n restricting the number of DFAs in the decomposition. It can be done by checking if a combination has at most n DFAs before adding it to \mathcal{M} in Algorithm 2.

6 Experimental Evaluation

We have implemented our approach² on top of the tool developed in [13] and we used as benchmarks pairs of sets of examples representing partially-ordered tasks (cf. [13, Section III.C]), where we vary the numbers of tasks, the maximum length of each sequence of tasks, and the number of samples in each positive and negative sets; for each combination, we randomly generate 10 instances, provided they can be generated, by following the generation strategy³ in [13]. Let

² Our implementation is available at: [let us make it available](#).

³ See <https://github.com/mvcisback/dfa-identify>

Table 2: Overview of the outcomes of the encoding experiments

Tool	Result	# DFAs ($ \Sigma = 2$)					# DFA ($ \Sigma = 4$)				
		2	3	4	5	6	2	3	4	5	6
PARETOAPTA [13]	Success	460	460	460	456	92	1058	129	85	72	72
	Memoryout	0	0	0	4	397	21	467	981	989	978
	Timeout	0	0	0	0	1	11	3	24	29	40
PARETO3DFA (ours)	Success	460	460	460	460	460	1090	1090	1090	1090	1089
	Timeout	0	0	0	0	0	0	0	0	0	1

PARETOAPTA, PARETO3DFA, and STATESOPTIMALDIP denote the original tool from [13], its version using our 3DFA encoding (cf. Section 4), and by our method for solving states-optimal DIP (cf. Section 5), respectively. We ran the tools on a desktop machine with an i7-4790 CPU and 16 GB of memory running Ubuntu Server 24.04.2 and we used BENCHEXEC [2] to trace and constrain the tools’ executions: we allowed each benchmark to use 15 GB of memory and imposed a time limit of 10 minutes of wall-clock time.

6.1 Comparison between ParetoAPTA and Pareto3DFA on solving Pareto-optimal DIP

In Table 2 we report on how PARETOAPTA and PARETO3DFA performed on 460 benchmarks with 2 tasks ($|\Sigma| = 2$) and 1090 benchmarks with 4 tasks ($|\Sigma| = 4$) when changing the number of DFAs in the decomposition. As we can see from the table, PARETOAPTA can scale to 5 DFAs for the $|\Sigma| = 2$ benchmarks, but it already struggles at 3 DFAs when $|\Sigma| = 4$. By just replacing the original APTA encoding with our 3DFA-based one, PARETO3DFA has been able to solve all benchmarks except for one case up to 6 DFAs. We have also run PARETO3DFA up to 10 DFAs, obtaining success everywhere except for 2 and 3 timeouts on $|\Sigma| = 4$ for 9 and 10 DFAs, respectively.

In Fig. 3 we show the runtime comparison between two methods PARETOAPTA and PARETO3DFA on all 1550 experiments considered in Table 2. The scatter plots in the figure have logarithmic axes and marks above the dotted diagonal line mean that PARETOAPTA took more time than PARETO3DFA to solve the same benchmark; the solid line at 600 seconds represents the timeout we imposed to the experiments while marks on the dashed line at 1000 seconds stand for to experiments where the corresponding tool went memoryout.

As we can see from the plots, except for the cases taking very limited time, our PARETO3DFA always significantly outperforms PARETOAPTA in the running time, while producing DFAs with the same number of states as PARETOAPTA on the commonly solved cases. For benchmarks requiring at least 5 seconds to be computed by both tools, PARETO3DFA is 1.4–80.3 times faster than PARETOAPTA; for at least 10 seconds, the speedup lies in 2.8–40.5.

More detailed comparison on runtime can be found in Appendix C.

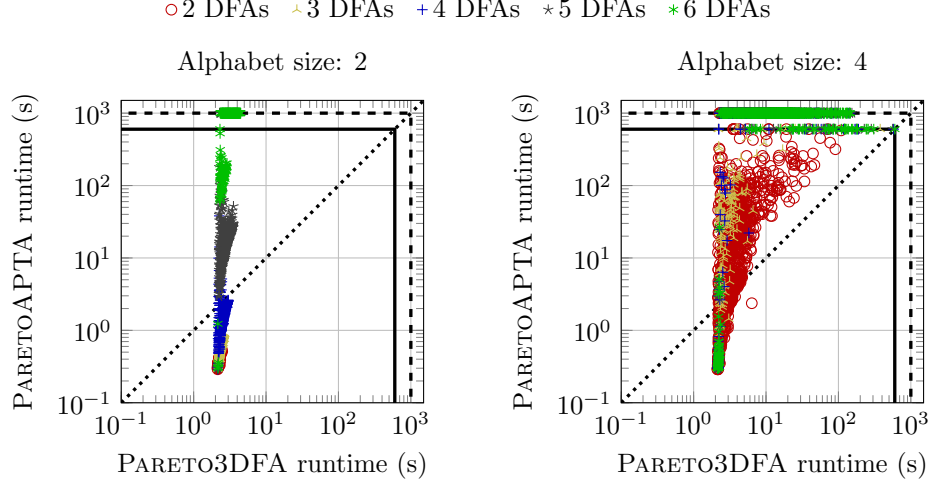


Fig. 3: Running time comparison between PARETOAPTA and PARETO3DFA

6.2 The scalability for our method on solving states-optimal DIP

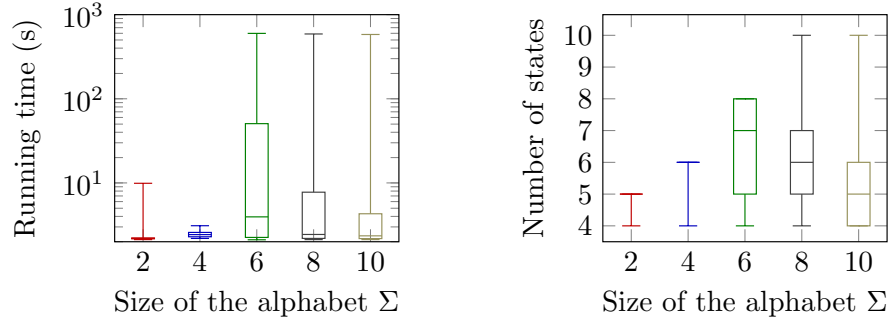


Fig. 4: Box plots for the STATESOPTIMALDIP experiments

To evaluate how STATESOPTIMALDIP is able to scale with more challenging benchmarks, we ran it on the same benchmarks used for Table 2 as well as on 1150, 1190, and 1210 benchmarks with $|\Sigma| = 6$, $|\Sigma| = 8$, and $|\Sigma| = 10$, respectively. STATESOPTIMALDIP has completed successfully all experiments for $|\Sigma| \in \{2, 4\}$; for $|\Sigma| = 6$ it solved 919 cases and went timeout on 231, while for $|\Sigma| = 8$ it solved 689 cases and went timeout on 501, and for $|\Sigma| = 10$ it solved 783 cases and went timeout on 427; no failure by memoryout happened.

The box plots in Fig. 4 show the distribution of the running time and of the number of states relative to the successfully solved benchmarks. As we can see from the plots, STATESOPTIMALDIP is really fast for the simpler benchmarks


with $|\Sigma| \in \{2, 4\}$, taking less than 1 second and needing between 4 and 6 states to solve each of them. For the more demanding benchmarks ($|\Sigma| \in \{6, 8, 10\}$), more states are necessary (at most 10), so by Algorithm 1 more cycles in the loop, more decompositions, and larger encoding formulas are generated and evaluated, as reflected by the higher running times shown in the plot on the left of Fig. 4. Appendix C presents more analysis on the behavior of STATESOPTIMALDIP.

7 Conclusion and Future Work

In this paper we considered two DFA decomposition identification problems: the Pareto-optimal DIP, studied by Lauffer *et al.* in [13], and the states-optimal DIP that we introduced in this paper. To solve the former problem, we proposed an improved SAT encoding via 3DFA; compared to the encoding via APTA [13], our method reduces the number of required encoding variables, thus significantly improving the efficiency, as confirmed by the experimental results, showing that our method is dramatically faster than the state-of-the-art method from [13]. We also proposed a solution method for the novel states-optimal DIP, and the experimental results on a large set of benchmarks demonstrate its scalability.

For future work, we consider further improving the practical efficiency in running time. One possibility is to find a novel encoding method where the input is a fixed total number of states N of the decomposition, but not a states allocation. In this way, if the answer is UNSAT, we can just increase the total number of states N without computing all possible states allocations suitable for the current N .

Acknowledgments. We thank the anonymous reviewers for their useful remarks that helped us improve the quality of the paper. Work supported in part by ...

 This project is part of the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant no. 101008233.

References

1. Ashar, P., Devadas, S., Newton, A.R.: Finite State Machine Decomposition, pp. 117–168. Springer US, Boston, MA (1992)
2. Beyer, D., Löwe, S., Wendler, P.: Reliable benchmarking: requirements and solutions. *Int. J. Softw. Tools Technol. Transf.* **21**(1), 1–29 (2019)
3. Coste, F., Nicolas, J.: How considering incompatible state mergings may reduce the DFA induction search tree. In: *ICGI 1998. Lecture Notes in Computer Science*, vol. 1433, pp. 199–210. Springer (1998)
4. Daciuk, J., Mihov, S., Watson, B.W., Watson, R.E.: Incremental construction of minimal acyclic finite state automata. *Comput. Linguistics* **26**(1), 3–16 (2000)
5. Dell’Erba, D., Li, Y., Schewe, S.: DFAMiner: Mining minimal separating DFAs from labelled samples. In: *FM 2024 (2). Lecture Notes in Computer Science*, vol. 14934, pp. 48–66. Springer (2024)

6. Egri-Nagy, A.: Applications of automata theory and algebra via the mathematical theory of complexity to biology, physics, psychology, philosophy, and games. *Artif. Life* **17**(2), 141–143 (2011)
7. Grinchtein, O., Leucker, M., Piterman, N.: Inferring network invariants automatically. In: *IJCAR 2006. Lecture Notes in Computer Science*, vol. 4130, pp. 483–497. Springer (2006)
8. Hardy, G.H., Ramanujan, S.: Asymptotic formulae in combinatory analysis. *Proceedings of the London Mathematical Society* **s2-17**(1), 75–115 (1918)
9. Heule, M., Verwer, S.: Exact DFA identification using SAT solvers. In: *ICGI 2010. Lecture Notes in Computer Science*, vol. 6339, pp. 66–79. Springer (2010)
10. de la Higuera, C.: A bibliographical study of grammatical inference. *Pattern Recognit.* **38**(9), 1332–1348 (2005)
11. Kupferman, O., Mosheiff, J.: Prime languages. *Inf. Comput.* **240**, 90–107 (2015)
12. Lang, K.J., Pearlmutter, B.A., Price, R.A.: Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In: *ICGI 1998. Lecture Notes in Computer Science*, vol. 1433, pp. 1–12. Springer (1998)
13. Lauffer, N., Yalcinkaya, B., Vazquez-Chanlatte, M., Shah, A., Seshia, S.A.: Learning deterministic finite automata decompositions from examples and demonstrations. In: *FMCAD 2022*. pp. 1–6. IEEE (2022)
14. Neider, D.: Computing minimal separating DFAs and regular invariants using SAT and SMT solvers. In: *ATVA 2012. Lecture Notes in Computer Science*, vol. 7561, pp. 354–369. Springer (2012)
15. Neider, D.: Applications of automata learning in verification and synthesis. Ph.D. thesis, RWTH Aachen University (2014)
16. Ulyantsev, V., Zakirzyanov, I., Shalyto, A.: BFS-based symmetry breaking predicates for DFA identification. In: *LATA 2015. Lecture Notes in Computer Science*, vol. 8977, pp. 611–622. Springer (2015)
17. Ulyantsev, V., Zakirzyanov, I., Shalyto, A.: Symmetry breaking predicates for SAT-based DFA identification. *CoRR* **abs/1602.05028** (2016)
18. Vaandrager, F.W.: Model learning. *Commun. ACM* **60**(2), 86–95 (2017)
19. Zakirzyanov, I., Morgado, A., Ignatiev, A., Ulyantsev, V., Marques-Silva, J.: Efficient symmetry breaking for SAT-based minimum DFA inference. In: *LATA 2019. Lecture Notes in Computer Science*, vol. 11417, pp. 159–173. Springer (2019)
20. Zakirzyanov, I., Shalyto, A., Ulyantsev, V.: Finding all minimum-size DFA consistent with given examples: SAT-based approach. In: *SEFM 2017 Workshops. Lecture Notes in Computer Science*, vol. 10729, pp. 117–131. Springer (2017)

A Proofs of Theorems and Lemmas

Theorem 1. *Given a set of examples $S = (S^+, S^-)$, the 3DFA construction produces a 3DFA consistent with S .*

Proof. The proof is by induction on the 3DFA construction steps that the construction preserves the following invariant: the language of each node mapped by the **Register** to the same representative state is the same, that is, for all nodes v, v' , if $\text{Register}(v) = \text{Register}(v')$, then $\mathcal{L}(v) = \mathcal{L}(v')$.

For the base case, at the end of the initialization phase, **Register** contains two types of representative states: one single “accepting” representative state for all accepting nodes without outgoing transitions and one different “rejecting” representative state for each rejecting node. Obviously, the invariant is satisfied after the initialization phase, since all accepting nodes mapped to the only “accepting” representative state have $\{\varepsilon\}$ as language, given that they have no outgoing transitions; each rejecting node is mapped to a different “rejecting” representative state, so $\text{Register}(v) = \text{Register}(v')$ holds only when $v' = v$ for each rejecting node v .

For the induction case, assume that the invariant holds for the current **Register**; we need to prove that it holds also after **Register** has been updated to **Register'** by one iteration of the reduction process. By the reduction step, **Register'** is **Register** extended with new mappings, namely, for each equivalence class C identified by the reduction step, all nodes in C are mapped to the same newly created representative state. To show that the invariant is preserved it suffices to consider pairs of equivalent nodes (i.e., nodes belonging to the same equivalence class), since nodes in different classes are mapped to different newly created representative states. Thus, consider an equivalence class and two nodes v and v' in it. By definition of the reduction step, we have that

- $\varepsilon \in \mathcal{L}(v) \iff \varepsilon \in \mathcal{L}(v')$, since either both v and v' are accepting (thus $\varepsilon \in \mathcal{L}(v)$ and $\varepsilon \in \mathcal{L}(v')$), or both are don't-care (thus $\varepsilon \notin \mathcal{L}(v)$ and $\varepsilon \notin \mathcal{L}(v')$);
- for every letter $a \in \Sigma$, we have two cases: either both v and v' have no a -successor, or both have an a -successor (denoted v_a and v'_a , respectively). In the former case, for each word $u \in \Sigma^*$, we have $a \cdot u \notin \mathcal{L}(v)$ and $a \cdot u \notin \mathcal{L}(v')$. In the latter case, the reduction step enforces $\text{Register}(v_a) = \text{Register}(v'_a)$ and the inductive hypothesis ensures $\mathcal{L}(v_a) = \mathcal{L}(v'_a)$. This implies that for each $u \in \mathcal{L}(v_a) = \mathcal{L}(v'_a)$, we have $a \cdot u \in \mathcal{L}(v)$ and $a \cdot u \in \mathcal{L}(v')$ and for each $u \in \Sigma^* \setminus \mathcal{L}(v_a) = \Sigma^* \setminus \mathcal{L}(v'_a)$, we have $a \cdot u \notin \mathcal{L}(v)$ and $a \cdot u \notin \mathcal{L}(v')$.

Since the above cases cover all possible words in Σ^* , it follows that $\mathcal{L}(v) = \mathcal{L}(v')$ as required. \square

Lemma 1. *Let $\mathcal{A} = (\mathcal{T}, A, R)$ be the outcome of the 3DFA construction, where $\mathcal{T} = (Q, \iota, \delta)$. For two different prefixes $u, u' \in \text{prefixes}(S)$, we have $\delta(\iota, u) \neq \delta(\iota, u')$ if either $u \in \text{prefixes}(S^-)$ or $u' \in \text{prefixes}(S^-)$.*

Proof. Let $u, u' \in \text{prefixes}(S)$ be two different prefixes and assume that $u \in \text{prefixes}(S^-)$; the case $u' \in \text{prefixes}(S^-)$ is analogous. We prove the statement of

the lemma by contradiction: assume that $\delta(\iota, u) = \delta(\iota, u')$. Since $u \in \text{prefixes}(S^-)$ by assumption, this implies that there exists a rejecting word $uy \in S^-$ with $y \in \Sigma^*$, that is, $r = \delta(\iota, uy)$ is a rejecting state. It then follows that r is also reached by $u'v$, $\delta(\iota, u'y) = \delta(\delta(\iota, u'), y) = \delta(\delta(\iota, u), y) = r$, since \mathcal{A} is deterministic and $\delta(\iota, u) = \delta(\iota, u')$. This then entails that $u'y$ is also a rejecting word in S^- ; however, since $uy, u'y \in S^-$ and both reach r , we get a contradiction with Theorem 1, which guarantees that two different rejecting words are associated with different states by our construction. \square

Theorem 2. *Let \mathcal{A} be the 3DFA consistent with the examples S and $n \in \mathbb{N}$. $\varphi_{(m_1, \dots, m_n)}^{\mathcal{A}, n}$ is satisfiable if, and only if, there exists a (m_1, \dots, m_n) -DFA decomposition $(\mathcal{A}_1, \dots, \mathcal{A}_n)$ consistent with S .*

Proof. We assume that the SAT encoding via APTA in [13] is correct. Compared to it, there are two main differences in our improved encoding: (1) replacing APTA with 3DFA, and (2) replacing Constraint **O1** with Constraint **O1'**. For the first improvement, Theorem 1 guarantees that the constructed 3DFA is consistent with the examples S as the original APTA does. For the second one, Constraint **O1'** allows a merged representative state to be associated with several states of each individual DFA in the DFA decomposition and ensures every rejecting state reached by a negative example in the 3DFA can be associated with at most one state i of each individual DFA in the DFA decomposition. The correctness of Constraint **O1'** is guaranteed by Lemma 1. Therefore, the correctness of Theorem 2 is proved. \square

Theorem 3 (Termination and Correctness). *Let $S = (S^+, S^-)$ be a given set of labeled examples. Algorithm 1 terminates and returns a correct DFA decomposition $\mathcal{D} = (\mathcal{A}_1, \dots, \mathcal{A}_n)$ for the states-optimal DIP.*

Proof. We first consider the termination of Algorithm 1. It is not hard to find that a correct decomposition $\mathcal{D} = \mathcal{A}_1, \dots, \mathcal{A}_n$ can always be found when $N = 2 + \sum_{u \in S^-} (|u| + 2)$ where a 2-states DFA accepting all words and every other DFA rejecting a different negative example $u \in S^-$. Therefore, the loop iterations in Algorithm 1 are bounded by $1 + \sum_{u \in S^-} (|u| + 2)$, as N starts with 2 and it is increased by 1 every round. Then we have the termination of Algorithm 1. As Algorithm 2 enumerates all possible states allocations in the states-optimal order, the correctness of Algorithm 1 is straightforward from Theorem 2 and the correctness of the SAT solver. \square

B SAT Encoding of DFA-DIP and Algorithm for Solving Pareto-optimal DIP in [13]

Below, we list the complete SAT encoding via APTA utilized in [13]. The encoding extends the SAT encoding for monolithic DFA identification. We refer to the constraints as follows:

1. A positive example must be accepted by all DFAs:

$$\bigwedge_{v \in V^+} \bigwedge_{k \in [n]} \bigwedge_{i \in [m_k]} x_{v,i}^k \implies z_i^k.$$

2. A negative example must be rejected by at least one DFA:

$$\bigwedge_{v \in V^-} \bigwedge_{k \in [n]} \bigwedge_{i \in [m_k]} x_{v,i}^k \implies \neg z_i^k.$$

3. Each state of APTA has at least one color for each DFA:⁴

$$\bigwedge_{v \in V} \bigwedge_{k \in [n]} \bigvee_{i \in [m_k]} x_{v,i}^k.$$

4. A transition of a DFA is set when a state and its parent are both colored:

$$\bigwedge_{v \in V \setminus \{v_r\}} \bigwedge_{k \in [n]} \bigwedge_{i,j \in [m_k]} (x_{p(v),i}^k \wedge x_{v,j}^k) \implies y_{l(v),i,j}^k.$$

5. A transition of a DFA targets at most one state:

$$\bigwedge_{l \in \Sigma} \bigwedge_{k \in [n]} \bigwedge_{\substack{i,j,t \in [m_k] \\ j < t}} y_{l,i,j}^k \implies \neg y_{l,i,t}^k.$$

6. Each state of APTA has at most one color for each DFA:

$$\bigwedge_{v \in V} \bigwedge_{k \in [n]} \bigwedge_{i,j \in [m_k]} \neg x_{v,i}^k \vee \neg x_{v,j}^k.$$

7. A transition of a DFA targets at least one state:

$$\bigwedge_{l \in \Sigma} \bigwedge_{k \in [n]} \bigwedge_{i,j \in [m_k]} y_{l,i,j}^k.$$

8. For each DFA, a node color is set when the color of the parent node and the transition between them are set:

$$\bigwedge_{v \in V \setminus \{v_r\}} \bigwedge_{k \in [n]} \bigwedge_{i,j \in [m_k]} (x_{p(v),i}^k \wedge y_{l(v),i,j}^k) \implies x_{v,j}^k.$$

9. Accepting-rejecting nodes of APTA cannot be merged:

$$\bigwedge_{v^- \in V^-} \bigwedge_{v^+ \in V^+} \bigwedge_{k \in [n]} \bigwedge_{i \in [m_k]} (x_{v^-,i}^k \wedge \neg z_i^k) \implies \neg x_{v^+,i}^k.$$

⁴ In the CoRR version of [13], the formula is written with the conjunction $\bigwedge_{i \in [m_k]}$ instead of the disjunction $\bigvee_{i \in [m_k]}$. However, to make the constraint satisfiable, this would cause all variables $x_{v,i}^k$ to be set to true, which is not what the constraint should require. The tool implementation indeed uses $\bigvee_{i \in [m_k]}$, so here we provide the corrected version of the constraint.

The next set of constraints encode the symmetry breaking clauses introduced in [16, 17] to avoid consideration of isomorphic DFAs. The main idea is to enforce individual DFA states to be enumerated in a depth-first search (DFS) order. Let $\Sigma = \{l_1, \dots, l_L\}$.

1. Each state must have a smaller parent in the DFS order:

$$\bigwedge_{k \in [n]} \bigwedge_{i \in [2, m_k]} (p_{i,1}^k \vee \dots \vee p_{i,i-1}^k).$$

2. Define $p_{j,i}^k$ in terms of auxiliary variable $t_{i,j}^k$:

$$\bigwedge_{k \in [n]} \bigwedge_{\substack{i,j \in [m_k] \\ i < j}} (p_{j,i}^k \iff t_{i,j}^k \wedge t_{i+1,j}^k \wedge \dots \wedge t_{j-1,j}^k).$$

3. Define $t_{i,j}^k$ in terms of $y_{l,i,j}^k$:

$$\bigwedge_{k \in [n]} \bigwedge_{\substack{i,j \in [m_k] \\ i < j}} (t_{i,j}^k \iff y_{l_1,i,j}^k \vee \dots \vee y_{l_L,i,j}^k).$$

4. The parent relationship follows the DFS order:

$$\bigwedge_{k \in [n]} \bigwedge_{\substack{i,j,p,q \in [m_k] \\ i < p < j < q}} (p_{j,i}^k \implies \neg t_{p,q}^k).$$

5. Define $m_{l,i,j}^k$ in terms of $y_{l,i,j}^k$:

$$\bigwedge_{k \in [n]} \bigwedge_{\substack{i,j \in [m_k] \\ i < j}} \bigwedge_{l_r \in \Sigma} (m_{l_r,i,j}^k \iff y_{l_r,i,j}^k \wedge \dots \wedge y_{l_1,i,j}^k).$$

6. Enforce DFAs to be DFS-enumerated in the order of symbols on transitions:

$$\bigwedge_{k \in [n]} \bigwedge_{\substack{i,j,q \in [m_k] \\ i < j < q}} \bigwedge_{\substack{l_r, l_s \in \Sigma \\ r < s}} (p_{j,i}^k \wedge p_{q,i}^k \wedge m_{l_r,i,j}^k \implies \neg m_{l_s,i,k}^k).$$

Based on the above encoding, Laufer *et al.* provided Algorithm 3 in [13]. Compared to it, our method for solving Pareto-optimal DIP replaces the SAT encoding via APTA by our improved encoding via 3DFA in Line 5 of Algorithm 3.

Algorithm 3: Pareto-optimal DIP Solving [13]

Input: The labeled examples $S = \{S^+, S^-\}$; the number of DFAs n .
Output: Pareto frontier P^* .

```

1  $(P^*, Q) \leftarrow \{(2, \dots, 2)\};$   $\triangleright$  Initial Pareto frontier and queue
2 while  $Q \neq \emptyset$  do
3    $m \leftarrow Q.dequeue();$ 
4   if  $\nexists \hat{m} \in P^*$  such that  $\hat{m} \prec m$  then
5      $SAT, \mathcal{A} \leftarrow \text{SOLVE}(n, m, S_+, S_-);$ 
6     if  $SAT$  then
7        $P^* \leftarrow P^* \cup \mathcal{A};$   $\triangleright$  Update Pareto frontier
8     end
9   else
10    for  $k \leftarrow 1$  to  $n$  do
11       $(m', m'_k) \leftarrow (m, m'_k + 1);$ 
12      if  $\text{ordered}(m')$  then
13         $Q.enqueue(m');$ 
14      end
15    end
16  end
17 end
18 end
19 return  $P^*$ ;

```

C More Detailed Experimental Results

In this appendix, we provide more detailed plots and analyses for the experiments we presented in Section 6.

C.1 Distribution of the benchmark files

Table 3 shows how benchmarks are distributed among the different choices of number of examples, maximum length of each example, and alphabet size. Each entry in the table says where there are 10 benchmarks of the corresponding combination: for example, entry “-/4/6/8” at “#Examples” 10 and “Max length” 3 means that there are 10 benchmarks are available for all alphabet sizes except for $|\Sigma| = 2$. This is because with 2 letters and words of length at most 3, we can have at most $1 + 2 + 4 + 8 = 15$ words, and these are not enough to have 10 positive examples and 10 negative examples in the benchmarks; with larger alphabets, instead, we have enough words to choose randomly 10 accepted and 10 rejected words to populate the example sets. This is why we have a different total number of benchmarks for the different alphabet sizes. Note that we also consider alphabet size 10 but we don not report it in the table since all entries would have it; alphabet of size 10 also enables to have 10 benchmarks of maximum length 2 and 5 examples in each of the positive and negative example sets.

Table 3: Distribution of the benchmarks with respect to the number of examples, the maximum length of each example, and the alphabet size; entries in the table show for which alphabet size there are 10 benchmarks

#Examples	Max length							
	3	4	5	6	7	8	9	10
5	2/4/6/8	2/4/6/8	2/4/6/8	2/4/6/8	2/4/6/8	2/4/6/8	2/4/6/8	2/4/6/8
10	-/4/6/8	2/4/6/8	2/4/6/8	2/4/6/8	2/4/6/8	2/4/6/8	2/4/6/8	2/4/6/8
15	-/4/6/8	-/4/6/8	2/4/6/8	2/4/6/8	2/4/6/8	2/4/6/8	2/4/6/8	2/4/6/8
20	-/4/6/8	-/4/6/8	-/4/6/8	2/4/6/8	2/4/6/8	2/4/6/8	2/4/6/8	2/4/6/8
25	-/-/6/8	-/4/6/8	-/4/6/8	2/4/6/8	2/4/6/8	2/4/6/8	2/4/6/8	2/4/6/8
30	-/-/6/8	-/4/6/8	-/4/6/8	-/4/6/8	2/4/6/8	2/4/6/8	2/4/6/8	2/4/6/8
35	-/-/6/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8	2/4/6/8	2/4/6/8	2/4/6/8
40	-/-/6/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8	2/4/6/8	2/4/6/8	2/4/6/8
45	-/-/6/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8	2/4/6/8	2/4/6/8
50	-/-/6/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8	2/4/6/8	2/4/6/8
60	-/-/-/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8	2/4/6/8
70	-/-/-/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8
80	-/-/-/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8
90	-/-/-/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8
100	-/-/-/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8	-/4/6/8

Since no other combination of alphabet size and number of examples allows us to get benchmarks with maximum length 2, we omit the corresponding column from the table.

C.2 Runtime comparison between ParetoAPTA and Pareto3DFA

Fig. 5 shows the experiments on running PARETOAPTA and PARETO3DFA on the 460 benchmarks with $|\Sigma| = 2$ and 1050 benchmarks with $|\Sigma| = 4$ when using a decomposition with two DFAs. As we can see from the plots, for $|\Sigma| = 2$, PARETOAPTA solved all cases by taking between 0.3 and 0.6 seconds; PARETO3DFA instead took between 2.1 and 2.5 seconds. These larger running times can be explained by the additional computation needed in the reduction of the 3DFA obtained by merging nodes that are equivalent (cf. Section 4.2); the time spent in this reduction is not compensated by the time saved by the SAT solver to decide the satisfiability of the smaller encoding formulas. The situation however changes when we consider the benchmarks with $|\Sigma| = 4$: here, the more demanding a benchmark is, the more effective the 3DFA encoding becomes, as shown by the cloud of points above the diagonal.

Fig. 6 is similar to Fig. 5, but in this case we set the number of DFAs in the decomposition to be three. Here, the running time of PARETO3DFA for $|\Sigma| = 2$ is essentially the same (between 2.1 and 2.7 seconds); PARETOAPTA instead starts requiring more time (between 0.3 and 1.7 seconds). The increased demand by PARETOAPTA is more evident from the plot relative to $|\Sigma| = 4$, where the

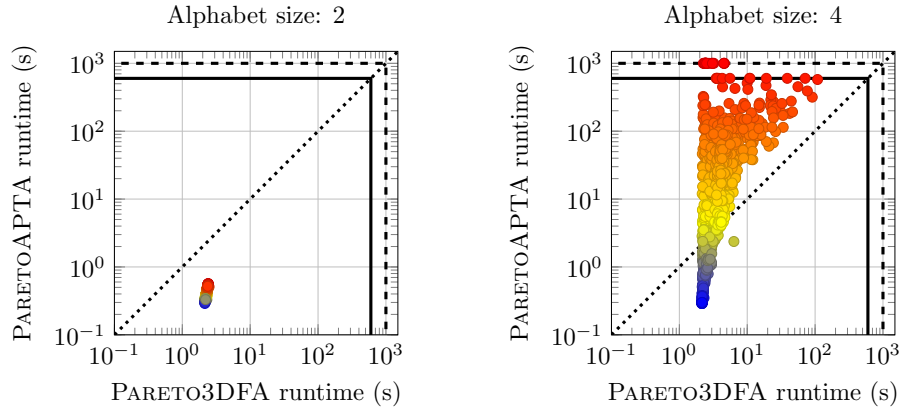


Fig. 5: Running time comparison between PARETOAPTA and PARETO3DFA using two DFAs

cloud of points is narrower and several points occur on the timeout/memoryout lines.

The sensibility of PARETOAPTA to the number of DFAs in the decomposition is more and more clear when we increase them to four, five, and six, as shown by the plots in Fig. 7, 8, and 9, respectively. In these plots, more and more points approach the timeout/memoryout lines for PARETOAPTA, also when $|\Sigma| = 2$; PARETO3DFA instead is less affected by the number of DFAs.

C.3 Behavior of Pareto3DFA when varying the number of DFAs

In Fig. 10 and 11 we show the cactus plots for the solved cases vs. time by PARETO3DFA on benchmarks with $|\Sigma| = 2$ and $|\Sigma| = 4$, respectively, where we ran PARETO3DFA with the number of DFAs in the decomposition ranging between two and ten. A point (x, y) in the plot means that there have been y instances that have taken at most x seconds each to be analyzed successfully. We also add a marker corresponding to the instance taking the longest time for a given number of DFAs in the decomposition.

As one would expect, increasing the number of DFAs causes PARETO3DFA to run for longer time before solving the problem. This is due to two main factors: with more DFAs, there might be more decompositions that need to be checked before finding the minimal one under the Pareto-optimal partial order; with more DFAs, the formula obtained from encoding the 3DFA is larger and with more variables, thus the SAT solver is likely to require more time before deciding its satisfiability. This is evident in the plot in Fig. 10 about the 460 benchmarks with $|\Sigma| = 2$: the point corresponding to the instance that took the highest amount of time to be solved moves steadily to the right of the plot when we increase the number of DFAs from two to ten. This is slightly different for

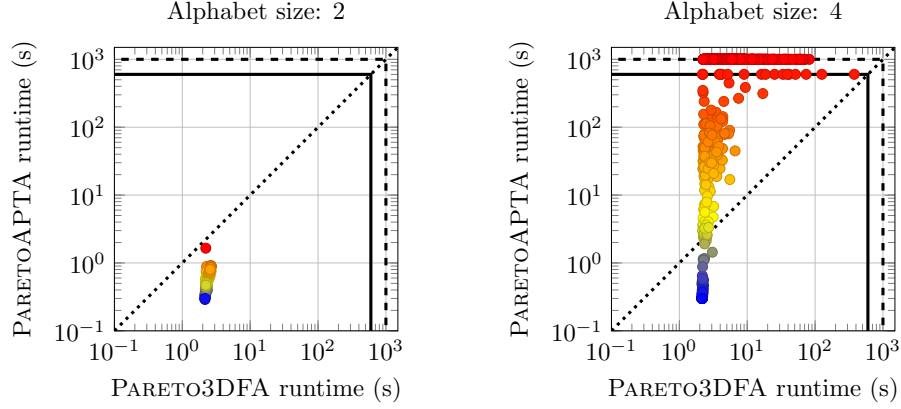


Fig. 6: Running time comparison between PARETOAPTA and PARETO3DFA using three DFAs

the plot in Fig. 11 about the 460 benchmarks with $|\Sigma| = 4$: for instance, the highlighted point for six DFAs is on the left of the points for three, four, and five DFAs. This is caused by the fact that on six DFAs, PARETO3DFA went timeout in one case, the one taking the longest for three, four, and five DFAs.

C.4 Behavior of StatesOptimalDIP on varying the size of the alphabet

We now focus our attention on the behavior of STATESOPTIMALDIP when we change the size of the alphabet. In Fig. 12 we show the cactus plots for the solved cases vs. the running time while in Fig. 13 we consider the states; these two plots present in a different format the same outcomes shown by the box plots in Fig. 4.

As already reported in Section 6.2, STATESOPTIMALDIP solved successfully all 460 benchmarks for $|\Sigma| = 2$ and 1090 benchmarks for $|\Sigma| = 4$; there have been experiments that did not complete for the other alphabet sizes: out of 1150 benchmarks with $|\Sigma| = 6$, there have been 231 timeouts while out of 1190 benchmarks with $|\Sigma| = 8$, there have been 501 timeouts; lastly, for the 1210 benchmarks with $|\Sigma| = 10$, there have been 427 timeouts. No failure by memoryout occurred in any of the benchmarks.

As we can see from the plots, the larger the alphabet is, the longer STATESOPTIMALDIP takes to give a successful decomposition. This is related to the fact that with more letters in the alphabet, it is more difficult to find equivalent states in the 3DFA-based encoding, so the acceptor has more states in it. Moreover, with a larger alphabet, we need more variables to encode the transitions, so the encoding formula is larger and the SAT solver is likely to require more time to decide its satisfiability.

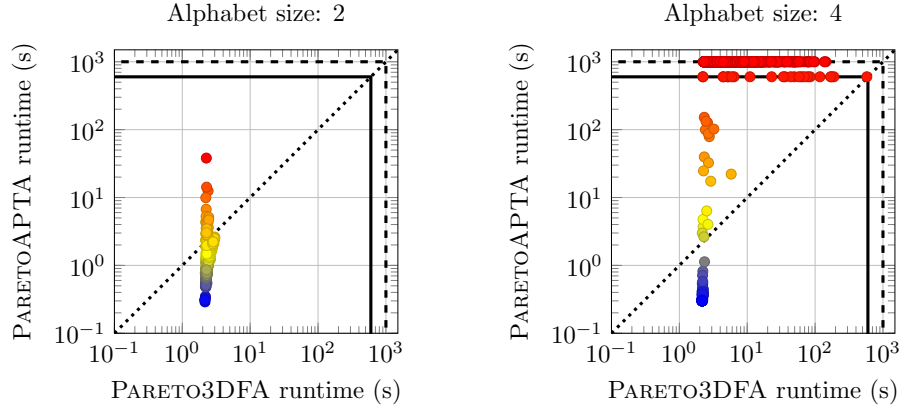


Fig. 7: Running time comparison between PARETOAPTA and PARETO3DFA using four DFAs

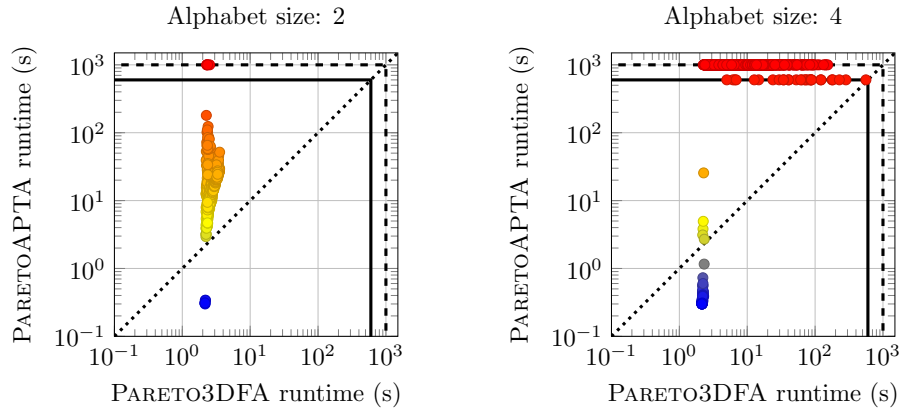


Fig. 8: Running time comparison between PARETOAPTA and PARETO3DFA using five DFAs

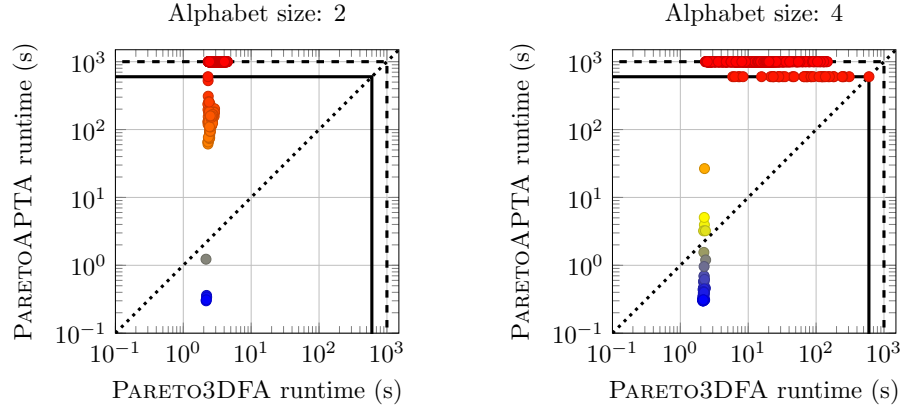


Fig. 9: Running time comparison between PARETOAPTA and PARETO3DFA using six DFAs

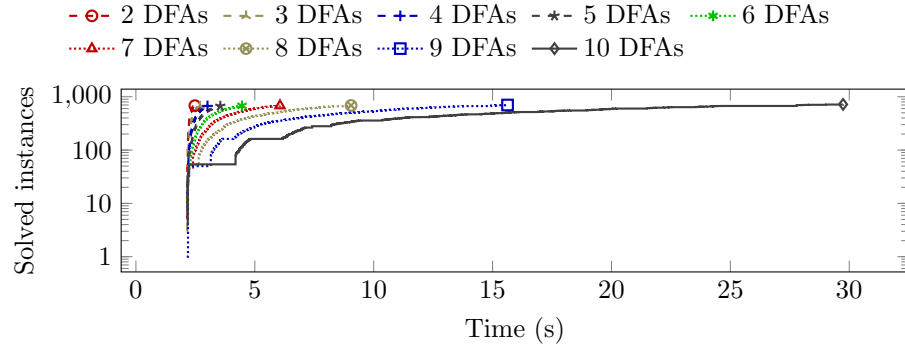


Fig. 10: Cactus plot for the solved cases vs. time by PARETO3DFA on benchmarks with $|\Sigma| = 2$

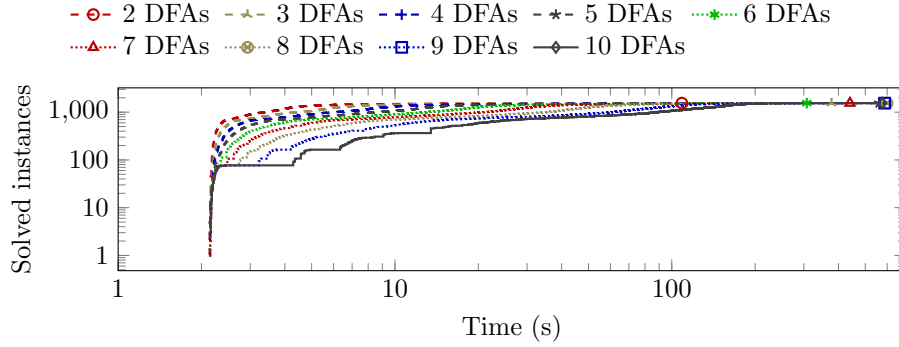


Fig. 11: Cactus plot for the solved cases vs. time by PARETO3DFA on benchmarks with $|\Sigma| = 4$

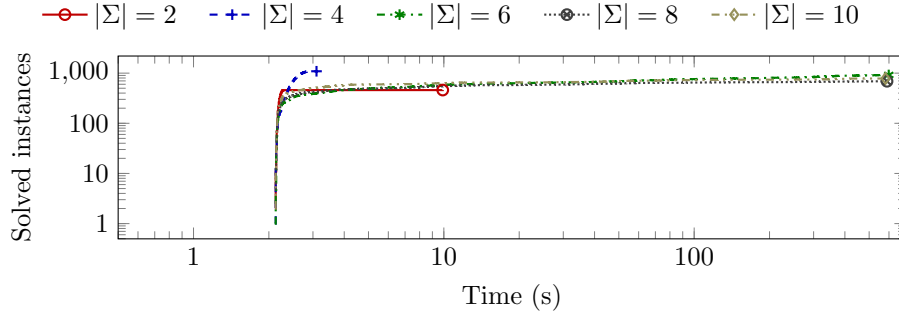


Fig. 12: Cactus plot for the solved cases vs. time by STATESOPTIMALDIP

All these things also reflect on the number of states, since we likely need more states in the DFAs in the decomposition to capture the different languages of the states in the 3DFA-based acceptor.

In Table 4 we report the number of benchmarks solved by STATESOPTIMALDIP with the computed number of states, split by alphabet size. From the table we can see that the larger the alphabet is, the more states are needed. This is also caused by the fact that with more letters available, it is possible to have more examples in the positive and negative example sets forming the benchmark $S = (S^+, S^-)$ (cf. Table 3). Table 5 is similar to Table 4 except for the fact that we report the number of states against the maximum length of the examples; in Table 6 we instead consider the number of positive and negative examples.

Table 4: Distribution of the minimal states computed by STATESOPTIMALDIP vs. alphabet size

$ \Sigma $	#Benchmarks	Number of states N								Failures
		4	5	6	7	8	9	10		
2	460	5	455	0	0	0	0	0	0	
4	1090	66	81	943	0	0	0	0	0	
6	1150	107	129	162	255	266	0	0	231	
8	1190	160	159	195	113	35	25	2	501	
10	1210	210	196	250	85	18	21	3	427	
Total	5100	548	1020	1550	453	319	45	5	1159	

Table 5: Distribution of the minimal states computed by STATESOPTIMALDIP vs. maximum length of the examples

Max length	#Benchmarks	Number of states N								Failures
		4	5	6	7	8	9	10		
2	10	10	0	0	0	0	0	0	0	
3	440	63	78	101	84	60	46	5	3	
4	620	70	83	215	84	72	0	0	96	
5	630	65	108	199	46	45	0	0	167	
6	650	66	125	203	30	1	0	0	225	
7	660	69	133	203	43	1	0	0	211	
8	680	68	151	205	31	42	0	0	183	
9	700	72	162	214	61	58	0	0	133	
10	710	65	180	210	75	40	0	0	141	
Total	5100	548	1020	1550	453	319	45	5	1159	

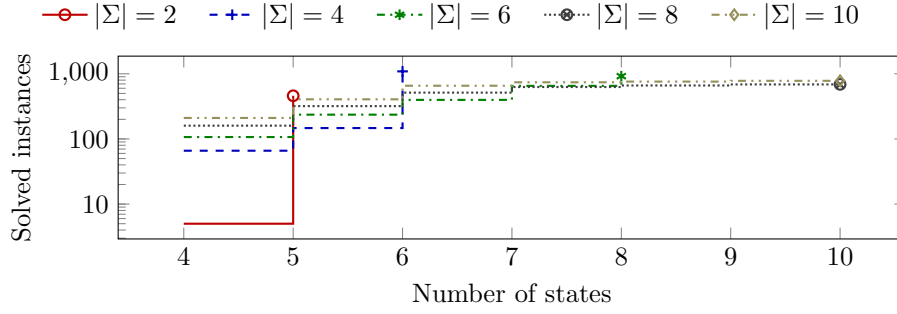


Fig. 13: Cactus plot for the solved cases vs. states by STATESOPTIMALDIP

Table 6: Distribution of the minimal states computed by STATESOPTIMALDIP vs. number of examples

$ S^+ = S^- $	#Benchmarks	Number of states N								Failures
		4	5	6	7	8	9	10		
5	410	318	92	0	0	0	0	0	0	
10	390	181	188	21	0	0	0	0	0	
15	380	47	248	85	0	0	0	0	0	
20	370	2	209	159	0	0	0	0	0	
25	360	0	120	225	15	0	0	0	0	
30	350	0	53	229	67	0	0	0	1	
35	340	0	30	183	97	1	0	0	29	
40	340	0	30	142	90	16	0	0	62	
45	330	0	20	85	86	26	0	0	113	
50	330	0	20	71	71	45	0	0	123	
60	310	0	10	70	19	50	1	0	160	
70	300	0	0	70	7	52	6	0	165	
80	300	0	0	70	0	39	17	1	173	
90	300	0	0	70	1	43	18	1	167	
100	290	0	0	70	0	47	4	3	166	
Total	5100	548	1020	1550	453	319	45	5	1159	