Learning Deterministic Finite Automata Decompositions from Examples and Demonstrations

Niklas Lauffer*, Beyazit Yalcinkaya*, Marcell Vazquez-Chanlatte, Ameesh Shah, and Sanjit A. Seshia
University of California, Berkeley, CA, USA
{nlauffer, beyazit, marcell.vc, ameesh, sseshia}@berkeley.edu

***Abstract*—The identification of a *deterministic finite automaton* (DFA) from labeled examples is a well-studied problem in the literature; however, prior work focuses on the identification of monolithic DFAs. Although monolithic DFAs provide accurate descriptions of systems' behavior, they lack simplicity and interpretability; moreover, they fail to capture sub-tasks realized by the system and introduce inductive biases away from the inherent decomposition of the overall task. In this paper, we present an algorithm for learning conjunctions of DFAs from labeled examples. Our approach extends an existing SAT-based method to systematically enumerate Pareto-optimal candidate solutions. We highlight the utility of our approach by integrating it with a state-of-the-art algorithm for learning DFAs from demonstrations. Our experiments show that the algorithm learns sub-tasks realized by the labeled examples, and it is scalable in the domains of interest.**

I. Introduction

Grammatical inference is a mature and well-studied field with many application domains ranging from machine learning to computational biology [1]. The identification of a minimum size *deterministic finite automaton* (DFA) from labeled examples is one of the most well-investigated problems in this field. Furthermore, with the increase in computational power in recent years, the problem can be solved efficiently by various tools available in the literature (e.g., [2], [3]).

Existing work on DFA identification primarily focuses on the monolithic case, i.e., learning a single DFA from examples. Although such DFAs capture a language consistent with the examples, they may lack simplicity and interpretability. Furthermore, complex tasks often decompose into independent sub-tasks. However, monolithic DFA identification fails to capture the natural decomposition of the system behavior, introducing an inductive bias away from the inherent decomposition of the overall task. In this paper, we present an algorithm for learning *DFA decompositions* from examples by reducing the problem to graph coloring in SAT and a Pareto-optimal solution search over candidate solutions. A DFA decomposition is a set of DFAs such that the *intersection* of their languages is the language of the system, which implicitly defines a conjunction of simpler specifications realized by the overall system.[1] We present an application of our algorithm to a state-of-the-art method for learning task specifications from unlabeled demonstrations [4] to showcase a domain of interest for DFA decompositions.

Related Work. Existing work considers the problem of minimal DFA identification from labeled examples [1]. It is shown that the DFA identification problem with a given upper bound on the number of states is an NP-complete problem [5]. Another work shows that this problem cannot be efficiently approximated [6]. Fortunately, practical methods exist in the literature. A common approach is to apply the evidence driven state-merging algorithm [7], [8], [9], which is a greedy algorithm that aims to find a good local optimum. Other works for learning DFAs use evolutionary computation [10], [11], later improved by multi-start random hill climbing [12].

A different approach to the monolithic DFA identification is to leverage highly-optimized modern SAT solvers by encoding the problem in SAT [13]. In follow up works, several symmetry breaking predicates are proposed for the SAT encoding to reduce the search space [3], [14], [15], [16]. However, to the best of our knowledge, no work considers directly learning DFA decompositions from examples and demonstrations.

This work also relates to the problem of decomposing a known automaton. Ashar et al. [17] explore computing cascade and general decomposition of finite state machines. The Krohn–Rhodes theorem [18] reduces a finite automaton into a cascade of irreducible automata. Kupferman & Mosheiff [19] present various complexity results for DFA decomposability.

Finally, the problem of learning objectives from demonstrations of an expert dates back to the problem of Inverse Optimal Control [20] and, more recently in the artificial intelligence community, the problem of Inverse Reinforcement Learning (IRL) [21]. The goal in IRL is to recover the unknown reward function that an expert agent is trying to maximize based on observations of that expert. Recently, several works have considered a version of the IRL problem in which the expert agent is trying to maximize the satisfaction of a Boolean task specification [22], [23], [4]. However, no work considers learning *decompositions* of specifications from demonstrations.

II. Problem Formulation

Let $\mathcal{D}$ denote the set of DFAs over some fixed alphabet Σ. An $(m_1, \ldots, m_n)$-*DFA decomposition* is a tuple of n DFAs $(\mathcal{A}_1, \ldots, \mathcal{A}_n) \in \mathcal{D}^n$ where $\mathcal{A}_i$ has m_i states and

*Equal contribution

This work was partially supported by NSF grants 1545126 (VeHICaL) and 1837132, by the DARPA contracts FA8750-18-C-0101 (Assured Autonomy) and FA8750-20-C-0156 (SDCPS), by Berkeley Deep Drive, by Toyota under the iCyPhy center, and by Toyota Research Institute.

[1] Our algorithm and SAT encoding can easily be generalized to unions or even arbitrary Boolean combinations of DFAs.

https://doi.org/

$m_1 \leq m_2 \leq \cdots \leq m_n$. We associate a partial order $\prec$ on DFA decompositions using the standard product order on the number of states. That is, $(\mathcal{A}'_1, \ldots, \mathcal{A}'_n) \prec (\mathcal{A}_1, \ldots, \mathcal{A}_n)$, if $m'_i \leq m_i$ for all $i \in [n]$ and $m'_j < m_j$ for some $j \in [n]$. In this case, we say $(\mathcal{A}'_1, \ldots, \mathcal{A}'_n)$ *dominates* $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$. A DFA decomposition $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ *accepts* a string w iff all $\mathcal{A}_i$ accept w. A string that is not accepted is *rejected*. The *language* of a decomposition, $\mathcal{L}(\mathcal{A}_1, \ldots, \mathcal{A}_n)$, is the set of accepting strings, i.e., the intersection of all DFA languages.

In order to bias towards "simpler" solutions, we further extend the partial order $\prec$ over equally sized (i.e., if $m'_i = m_i$ for all $i \in [n]$) decompositions by letting $(\mathcal{A}'_1, \ldots, \mathcal{A}'_n) \prec (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ if $(\mathcal{A}'_1, \ldots, \mathcal{A}'_n)$ has fewer total non-stuttering edges than $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$.

We study the problem of finding a DFA decomposition from a set of positive and negative labeled examples such that the decomposition accepts the positive examples and rejects the negative examples. We start by formally defining *the DFA decomposition identification problem* (DFA-DIP), and then presenting an overview of the proposed approach.

The Deterministic Finite Automaton Decomposition Identification Problem (DFA-DIP). Given positive examples, D_+ and negative examples, D_-, and a natural number $n \in \mathbb{N}$, find a $(m_1, \ldots, m_n)$-DFA decomposition $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ satisfying the following conditions.

(C1) The decomposition is consistent with (D_+, D_-):

$$D_+ \subseteq \mathcal{L}(\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n),$$
$$D_- \subseteq \Sigma^* \setminus \mathcal{L}(\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n).$$

(C2) There does not exist a DFA decomposition that *dominates* $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ and satisfies **(C1)**.

We refer to the set of DFA decompositions that solve an instance of DFA-DIP as the Pareto-optimal frontier of solutions. Note that for $n = 1$, DFA-DIP reduces to monolithic DFA identification. We propose finding the set of DFA decompositions that solve DFA-DIP by reduction to graph coloring in SAT and a breadth first search in solution space. Specifically, we extend the existing work on SAT-based monolithic DFA identification [13], [15] to finding n DFAs with $m_1, \ldots, m_n$ states and q non-stuttering edges such that the intersection of their languages is consistent with the given examples. On top of this SAT-based approach, we develop a search strategy over the numbers of states and edges passed to the SAT solver as these values are not known a priori.

III. LEARNING DFAS FROM EXAMPLES[2]

In this section, we present the proposed approach. We start with the SAT encoding of the DFA decomposition problem and continue with the Pareto frontier search in the solution space. We then showcase an example of learning conjunctions of DFAs from labeled examples. Finally, we present experimental results and evaluate the scalability of our method.

[2]Our MIT licensed code is freely available at [24].

A. Encoding DFA-DIP in SAT

We extend the SAT encoding for monolithic DFA identification presented in [13], [15], which solves a graph coloring problem, to finding n DFAs with $m_1, m_2, \ldots, m_n$ states. The extension relies on the observation that for conjunctions of DFAs, we need to enforce that a positive example must be accepted by *all* DFAs, and a negative example must be rejected by *at least* one of the DFAs. Due to space limitations, we only present the modified clauses of the encoding, and invite reader to Appendix A of the extended version of the paper [25] for further details.

The encoding works on an *augmented prefix tree acceptor* (APTA), a tree-shaped automaton with nodes corresponding to prefixes and edges to appending letters, constructed from given examples, which has paths for each example leading to accepting or rejecting states based on the example's label; therefore, an APTA defines D_+ and D_- which then constrains the accepting states, rejecting states, and the transition function of the unknown DFAs. For each DFA, $\mathcal{A}_i$, the encoding will associate the APTA states with one of the m_i colors for DFA $\mathcal{A}_i$, subject to the constraints imposed by D_+ and D_-. APTA states with the same (DFA-indexed) color will be the same state in the corresponding DFA. We refer to states of an APTA as V, its accepting states as V_+, and its rejecting states as V_-. Given n for the number of DFAs, $m_1, \ldots, m_n$ for the number of states of DFAs, and q for the number of non-stuttering edges, the SAT encoding uses three types of variables:

1) *color* variables $x^k_{v,i} \equiv 1$ ($k \in [n]$; $v \in V$; $i \in [m_k]$) iff APTA state v has color i in DFA k,
2) *parent relation* variables $y^k_{l,i,j} \equiv 1$ ($k \in [n]$; $l \in \Sigma$, where Σ is the alphabet; $i, j \in [m_k]$) iff DFA k transitions with symbol l from state i to state j, and
3) *accepting color* variables $z^k_i \equiv 1$ ($k \in [n]$; $i \in [m_k]$) iff state i of DFA k is an accepting state.

The encoding for the monolithic DFA identification also uses the same variable types; however, in our encoding, we also index variables over n DFAs instead of a single DFA. With this extension, one can trivially instantiate the encoding presented in [13], [15]. Below, we list the new rules we define for our problem. For the complete list of rules, see Appendix A of the extended version of the paper [25].

(R1) A negative example must be rejected by *at least* one DFA:

$$\bigwedge_{v \in V_-} \bigvee_{k \in [n]} \bigwedge_{i \in [m_k]} x^k_{v,i} \implies \neg z^k_i.$$

(R2) Accepting and rejecting states of APTA cannot be merged:

$$\bigwedge_{v_- \in V_-} \bigwedge_{v_+ \in V_+} \bigwedge_{k \in [n]} \bigwedge_{i \in [m_k]} (x^k_{v_-,i} \wedge \neg z^k_i) \implies \neg x^k_{v_+,i}.$$

(R3) Upperbound on the number of non-stuttering edges:

$$\sum_{k \in [n]} \sum_{l \in \Sigma} \sum_{i,j \in [m_k], i \neq j} y^k_{l,i,j} \leq q.$$

In the encoding of [13], [15], we replace the rule stating that the resulting DFA must reject all negative examples with **(R1)**, and **(R2)** is used instead of the original rule stating that accepting and rejecting states of APTA cannot be merged. Notice that since a rejecting state of APTA is not necessarily a rejecting state of a DFA k, we need to use the new rule **(R2)**. Finally, **(R3)** enables controlling the maximum number of non-stuttering transitions. As we shall see, this will enable us to satisfy **(C2)**.

Theorem 1. *Given labeled examples, n for the number of DFAs, $m_1, \ldots, m_n$ for the number of states of DFAs, and q for the number non-stuttering edges, a solution to the above SAT encoding satisfies **(C1)** of* `DFA-DIP`.

Proof: We assume that the SAT-based reduction to graph coloring for monolithic DFA identification given in [13] is correct. Next, observe that **(R3)** can only remove solutions and thus does not effect **(C1)**. Constraint **(R1)** and **(R2)** replace similar constraint in the monolithic encoding given in [13]:

(R1′) a negative example must be rejected by the DFA:

$$\bigwedge_{v \in V_-} \bigwedge_{i \in [m_k]} x_{v,i} \implies \neg z_i, \text{ and}$$

(R2′) accepting and rejecting states of the APTA cannot be merged:

$$\bigwedge_{v_- \in V_-} \bigwedge_{v_+ \in V_+} \bigwedge_{i \in [m_k]} x_{v_-,i} \implies \neg x_{v_+,i}.$$

In the monolithic DFA case, there is only a single DFA so for ease of notation, we drop the index k. First notice that constraints **(R1′)** and **(R2′)** have no bearing on whether the DFA accepts each positive example. Therefore, our encoding automatically requires that each DFA in the DFA decomposition accepts all of the positive examples and is not constrained to unecessarily accept any unspecified examples.

Constraint **(R1′)** ensures that the resulting monolithic DFA rejects every negative example by making the color of the node in the APTA associated with the negative example rejecting. Constraint **(R1)** replaces this and ensures that at least one of the DFAs in the DFA decomposition rejects a negative example by making the color of the node in the APTA associated with the negative example rejecting in at least one of the n DFAs in the decomposition. Thus, the language intersection of the resulting decomposition correctly rejects negative examples.

Constraint **(R2′)** ensures that all pairs of rejecting and accepting nodes of the APTA cannot be assigned the same color (i.e., merged) in the resulting DFAs. Constraint **(R2)**, which replaces **(R2′)**, ensures that for each DFA in the decomposition, the pair $(x^k_{v_-,i}, x^k_{v_+,i})$ of accepting and rejecting nodes of the APTA cannot be assigned the same color only if DFA k is rejecting the negative example associated with $x^k_{v_-,i}$ (which is handled by constraint **(R1)**). This allows all but one DFA in the DFA decomposition to accept negative examples. Therefore, the language of the decomposition is not constrained to reject any unspecified examples. ■

Algorithm 1 Pareto frontier enumeration algorithm.

```
Require: Positive $D_+$ and negative $D_-$ labeled examples and positive integer $n$.
1: $(P^\star, Q) \leftarrow \{(1, \ldots, 1)\}$                ▷ Initial Pareto front and queue.
2: while $Q \neq \emptyset$ do
3:     $m \leftarrow Q.dequeue()$
4:     if $\nexists \hat{m} \in P^\star$ s.t. $\hat{m} \prec m$ then
5:         $SAT, \mathcal{A} \leftarrow$ SOLVE$(n, m, D_+, D_-)$    ▷ Omits (R3).
6:         if $SAT$ then
7:             $P^\star = P^\star \cup \mathcal{A}$              ▷ Add to the Pareto frontier.
8:         else
9:             for $k = 1, \ldots, n$ do
10:                $(m', m'_k) \leftarrow (m, m'_k + 1)$
11:                if ordered$(m')$ then $Q.enqueue(m')$
12: return minimize_stutter$(P^\star)$          ▷ Binary search using (R3).
```

B. Pareto Frontier Search

The SAT encoding detailed in section III-A produces a DFA decomposition that satisfies **(C1)**, but not necessarily **(C2)**. In this section, we provide the details of the Pareto frontier enumeration algorithm that uses the SAT encoding as an inner loop to find a DFA decomposition that solves `DFA-DIP`.

Our proposed Pareto frontier enumeration algorithm is a breadth first search (BFS) over DFA decomposition size tuples that skips tuples that are dominated by an existing solution. This BFS is over a directed acyclic graph $G = (V, E)$ formed in the following way. There is a vertex in the graph for every ordered tuple of states sizes. There is an edge from $(m_1, m_2, \ldots, m_n)$ to $(m'_1, m'_2, \ldots, m'_n)$ if there exists some $j \in [n]$ such that:

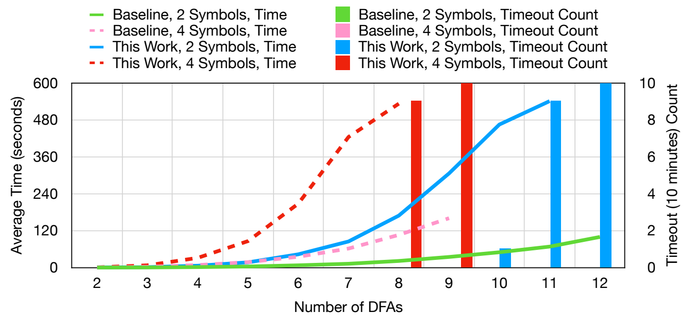
$$m'_i = \begin{cases} m_i + 1 & \text{if } i = j; \\ m_i & \text{otherwise.} \end{cases}$$

A size tuple $(m_1, \ldots, m_n)$ is a sink, i.e., the search does not continue past this vertex, if there exists a $(m_1, \ldots, m_n)$-decomposition that solves `DFA-DIP` or the size tuple is dominated by a previously traversed solution. In the prior case, the associated DFA decomposition is also returned as a solution on the Pareto-optimal frontier. The BFS starts from $m_1 = m_2 = \cdots = m_n = 1$, and performs the search as explained. Algorithm 1 presents the details of the BFS performed in the solution space for finding the Pareto frontier.

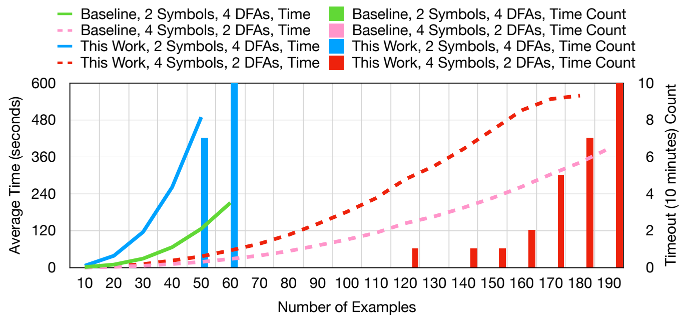
After finding a minimal number of states $m_1, m_2, \ldots, m_n$ that solve the problem, there still might exist multiple DFA decompositions of that size that solve **(C1)**. These ties are broken in favor of DFA decompositions that have the fewest total non-stuttering edges, q. For each minimal dfa this is done by a binary search over q and denoted: minimize_stutter$(\bullet)$.

Theorem 2. *Algorithm 1 is sound and complete; it outputs the full Pareto-optimal frontier of solutions without returning any dominated solutions, therefore satisfying **(C2)** of* `DFA-DIP`.

Proof: See the extended version of the paper [25]. ■



(a) Experiment results answering **(Q1)**, where we vary number of DFAs.

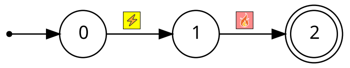


(b) Experiment results answering **(Q2)**, where we vary number of examples.

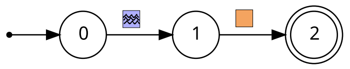
Fig. 1. Experiment results evaluating the scalability of our algorithm w.r.t. (a) number of DFAs implied by the examples and (b) number of labeled examples.

C. *Example: Learning Partially-Ordered Tasks*

We continue with a toy example showcasing the capabilities of the proposed approach. Later, we use the same class of decompositions to evaluate the scalability of our algorithm. Inspired from the multi-task reinforcement learning literature [26], our example focuses on partially-ordered temporal tasks executed in parallel. Specifically, consider a case where an agent is performing two ordering tasks in parallel: (i) observe ⚡ before 🔥, and (ii) observe ≋ before 🟧. A positive example of such behavior is simply any sequence of observations ensuring both of the given orderings, e.g. ⚡🔥≋🟧, and a negative example is any sequence that fails to satisfy both orderings, e.g. ⚡🔥🟧≋. We generate such positive and negative examples and feed them to our algorithm. Figure 2 presents the learned DFAs recognizing ordering subtasks of the example. The intersection of their languages is consistent with the given observations, and their conjunction is the overall task realized by the system generating the traces. The monolithic DFA recognizing the same language has nine states, and is more complicated (see Figure 4 in Appendix C of the extended version of the paper [25]).



(a) Learned DFA recognizing the ordering between ⚡ and 🔥.



(b) Learned DFA recognizing the ordering between ≋ and 🟧.

Fig. 2. Learned DFA decomposition.

D. *Experimental Evaluation*

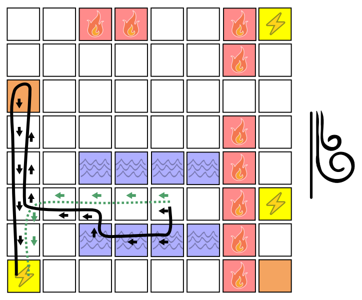
We evaluate the scalability of our algorithm through experiments with changing sizes of partially-ordered tasks introduced in Section III-C. In our evaluation, we aim to answer two questions: **(Q1)** "How does solving time scale with the number of ordering tasks?", and **(Q2)** "How does solving time scale with the number of labeled examples?". We implement our algorithm in Python with PySAT [27], and we use Glucose4 [28] as the SAT solver. Our baseline is an implementation of the monolithic DFA identification encoding from [13], [15] with the same software as our implementation. Experiments are performed on a Quad-Core Intel i7 processor clocked at 2.3 GHz and a 32 GB main memory.

To evaluate the scalability, we randomly generate positive and negative examples with varying problem sizes. For **(Q1)**, we generate 10 (half of which are positive and half of which are negative) partially-ordered task examples with (i) 2 symbols, and (ii) 4 symbols, and we vary the number of DFAs from 2 to 12. For **(Q2)**, we generate 10 to 20 partially-ordered task examples with (i) 2 symbols and 4 DFAs, and (ii) 4 symbols and 2 DFAs. Half of these examples are positive and the other half is negative. Since the examples are generated randomly, we run the experiments for 10 different random seeds and report the average. We set the timeout limit to 10 minutes, and stop when our algorithm timeouts for all random seeds.

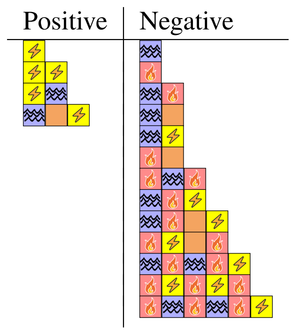
Figure 1a presents the experiment results answering **(Q1)**, where we vary the number of DFAs implied by the given examples. For partially-ordered tasks with 2 symbols, green solid line is the (monolithic DFA) baseline and the blue solid is our algorithm. Similarly, for partially-ordered tasks with 4 symbols, pink dashed line is the baseline and the red dashed line is our algorithm. Figure 1b presents the experiment results answering **(Q2)**, where we vary the number of examples. For partially-ordered tasks with 2 symbols and 4 DFAs, green solid line is the baseline and the blue solid is our algorithm; for partially-ordered tasks with 4 symbols and 2 DFAs, pink dashed line is the baseline and the red dashed line is our algorithm. As expected, the baseline scales better than our algorithm as we also search for the Pareto frontier and solve an inherently harder problem. Notice that given 10 examples, our algorithm is able to scale up to 11 DFAs for tasks with 2 symbols, and 8 DFAs for tasks with 4 symbols; for 2 symbols and 4 DFAs, it is able to scale up to 60 examples, and for 4 symbols and 2 DFAs, it is able to scale up to 190 examples. As we demonstrate in the next section, these limits for scalability are practically useful in certain domains.

IV. Learning DFAs from Demonstrations

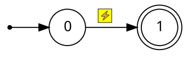
Next, we show how our algorithm can be incorporated into Demonstration Informed Specification Search (DISS) - a framework for learning languages from expert demonstra-



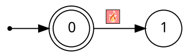
(a) A stochastic grid world environment with expert demonstrations of an agent trying to accomplish a task.



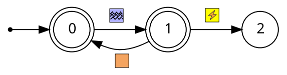
(b) Labeled examples conjectured by DISS.



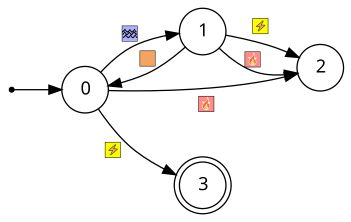
(c) Go to 🟨.



(d) Avoid 🟥.



(e) After 🟦, go to 🟧 before 🟨.



(f) Monolithic DFA for the example presented in Section IV.

Fig. 3. Figure 3a shows the stochastic grid world environment. Figure 3b shows the positive and negative examples of the expert's behavior conjectured by DISS and Figures 3c to 3e showcases the associated DFA decomposition identified by our algorithm. Figure 3f shows the monolithic DFA learned in [4].

tions [4]. For our purposes a *demonstration* is an unlabeled path through a workspace that maps to a string and is biased towards being accepting by some unknown language. For example, we ran our implementation of DISS using demonstrations produced by an expert attempting to accomplish a task in a stochastic grid world environment, the same example used in [4] and shown in Figure 3a. At each step, the agent can move in any of the four cardinal directions, but because of wind blowing from the north to the south, with some probability, the agent will transition to the space south of it in spite of its chosen action. Two demonstrations of the task "Reach 🟨 while avoiding 🟥. If it ever touches 🟦, it must then touch 🟧 before reaching 🟨." are shown in Figure 3a.

In order to efficiently search for tasks, DISS reduces the learning from demonstrations problem into a series of identification problems to be solved by a black-box identification algorithm. The goal of DISS is to find a task that minimizes the joint description length, called the energy, of the task and the demonstrations assuming the agent were performing said task. The energy is measured in bits to encode an object.

Below, we reproduce the results from [4], but using our algorithm as the task identifier rather than the monolithic DFA identifier provided[3]. The use of DFA decompositions biases DISS to conjecture concepts that are *simpler* to express in terms of a DFA decomposition. To define the description length of DFA decompositions, we adapt the DFA encoding used in [4] by expressing a decomposition as the concatenation of the encodings of the individual DFAs. To remove unnecessary redundancy two optimizations were performed. First common headers, e.g. indicating the alphabet size, were combined. Second, as the DFAs in a decomposition are ordered by size, we expressed changes in size rather than absolute size, see Appendix B in the extended version of the paper [25] for details.

[3]To allow exploring more decompositions, with some probability, the number of DFAs in the decomposition was randomly incremented or decremented during identification.

A. Experimental Evaluation

In Figures 3c to 3e we present the learned DFA decomposition along with the corresponding Figure 3b labeled examples conjectured by DISS to explain the expert behavior. Importantly, this decomposition exactly captures the demonstrated task. We note that this is in contrast to the DFA learned in [4], shown in Figure 3f, which allows visiting 🟥 after visiting 🟨. Further, we remark that the time required to learn the monolithic and decomposed DFAs was comparable. In particular, the number of labeled examples was less than 60 and as with the monolithic baseline, most of the time is not spent in task identification, but instead conjecturing the labeled examples. As we saw with in Section III-D, this number of examples is easily handled by our SAT-based identification algorithm. Finally, the number of labeled examples that needed to be conjectured to find low energy tasks was similar for both implementations (see Figures 5 and 6 in Appendix C of the extended versoin of the paper [25]). Thus, our variant of DISS performed similar to the monolithic variant, while finding DFAs that exactly represented the task.

V. Conclusion

To the best of our knowledge, this work presents the first approach for solving `DFA-DIP`. Our algorithm works by reducing the problem to a Pareto-optimal search of the space of the number of states in a DFA decomposition with a SAT call in the inner loop. The SAT-based encoding is based on an efficient reduction to graph coloring. We demonstrated the scalability of our algorithm on a class of problems inspired by the multi-task reinforcement learning literature and show that the additional computational cost for identifying DFA decompositions over monolithic DFAs is not prohibitive. Finally, we showed how identifying DFA decompositions can provide a useful inductive bias while learning from demonstrations.

References

[1] C. De La Higuera, "A bibliographical study of grammatical inference," *Pattern recognition*, vol. 38, no. 9, pp. 1332–1348, 2005.

[2] S. Verwer and C. A. Hammerschmidt, "Flexfringe: a passive automaton learning package," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2017, pp. 638–642.

[3] I. Zakirzyanov, A. Morgado, A. Ignatiev, V. Ulyantsev, and J. Marques-Silva, "Efficient symmetry breaking for sat-based minimum dfa inference," in *International Conference on Language and Automata Theory and Applications*. Springer, 2019, pp. 159–173.

[4] M. Vazquez-Chanlatte, A. Shah, G. Lederman, and S. A. Seshia, "Demonstration informed specification search," *CoRR*, vol. abs/2112.10807, 2021. [Online]. Available: https://arxiv.org/abs/2112.10807

[5] E. M. Gold, "Complexity of automaton identification from given data," *Information and control*, vol. 37, no. 3, pp. 302–320, 1978.

[6] L. Pitt and M. K. Warmuth, "The minimum consistent dfa problem cannot be approximated within any polynomial," *Journal of the ACM (JACM)*, vol. 40, no. 1, pp. 95–142, 1993.

[7] K. J. Lang, B. A. Pearlmutter, and R. A. Price, "Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm," in *International Colloquium on Grammatical Inference*. Springer, 1998, pp. 1–12.

[8] K. J. Lang, "Faster algorithms for finding minimal consistent dfas," *NEC Research Institute, Tech. Rep*, 1999.

[9] M. Bugalho and A. L. Oliveira, "Inference of regular languages using state merging algorithms with search," *Pattern Recognition*, vol. 38, no. 9, pp. 1457–1467, 2005.

[10] P. Dupont, "Regular grammatical inference from positive and negative samples by genetic search: the gig method," in *International Colloquium on Grammatical Inference*. Springer, 1994, pp. 236–245.

[11] S. Luke, S. Hamahashi, and H. Kitano, "" genetic" programming," in *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 2*, 1999, pp. 1098–1105.

[12] S. M. Lucas and T. J. Reynolds, "Learning dfa: evolution versus evidence driven state merging," in *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.*, vol. 1. IEEE, 2003, pp. 351–358.

[13] M. J. Heule and S. Verwer, "Exact dfa identification using sat solvers," in *International Colloquium on Grammatical Inference*. Springer, 2010, pp. 66–79.

[14] V. Ulyantsev, I. Zakirzyanov, and A. Shalyto, "Bfs-based symmetry breaking predicates for dfa identification," in *International Conference on Language and Automata Theory and Applications*. Springer, 2015, pp. 611–622.

[15] ——, "Symmetry breaking predicates for sat-based dfa identification," *arXiv preprint arXiv:1602.05028*, 2016.

[16] I. Zakirzyanov, A. Shalyto, and V. Ulyantsev, "Finding all minimum-size dfa consistent with given examples: Sat-based approach," in *International Conference on Software Engineering and Formal Methods*. Springer, 2017, pp. 117–131.

[17] P. Ashar, S. Devadas, and A. R. Newton, "Finite state machine decomposition," in *Sequential Logic Synthesis*. Springer, 1992, pp. 117–168.

[18] J. Rhodes, *Applications of automata theory and algebra : via the mathematical theory of complexity to biology, physics, psychology, philosophy, and games*. Singapore Hackensack, NJ: World Scientific, 2010.

[19] O. Kupferman and J. Mosheiff, "Prime languages," *Information and Computation*, vol. 240, pp. 90–107, 2015.

[20] R. E. Kalman, "When is a linear control system optimal," 1964.

[21] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *ICML*. Morgan Kaufmann, 2000, pp. 663–670.

[22] D. Kasenberg and M. Scheutz, "Interpretable apprenticeship learning with temporal logic specifications," in *CDC*. IEEE, 2017, pp. 4914–4921.

[23] G. Chou, N. Ozay, and D. Berenson, "Explaining multi-stage tasks by learning temporal logic formulas from suboptimal demonstrations," in *Robotics: Science and Systems*, 2020.

[24] M. Vazquez-Chanlatte, V. Lee, A. Shah, N. Lauffer, and B. Yalcinkaya, 2022. [Online]. Available: https://github.com/mvcisback/dfa-identify/tree/decomposition

[25] N. Lauffer, B. Yalcinkaya, M. Vazquez-Chanlatte, A. Shah, and S. A. Seshia, "Learning deterministic finite automata decompositions from examples and demonstrations," 2022. [Online]. Available: https://arxiv.org/abs/2205.13013

[26] P. Vaezipoor, A. C. Li, R. A. T. Icarte, and S. A. Mcilraith, "Ltl2action: Generalizing ltl instructions for multi-task rl," in *International Conference on Machine Learning*. PMLR, 2021, pp. 10 497–10 508.

[27] A. Ignatiev, A. Morgado, and J. Marques-Silva, "PySAT: A Python toolkit for prototyping with SAT oracles," in *SAT*, 2018, pp. 428–437. [Online]. Available: https://doi.org/10.1007/978-3-319-94144-8_26

[28] N. Eén and N. Sörensson, "An extensible sat-solver," in *International conference on theory and applications of satisfiability testing*. Springer, 2003, pp. 502–518.