Minimization of Limit-Average Automata

Jakub Michaliszyn and **Jan Otop**
University of Wrocław
{jmi, jotop}@cs.uni.wroc.pl

Abstract

LimAvg-automata are weighted automata over infinite words that aggregate weights along runs with the limit-average value function. In this paper, we study the minimization problem for (deterministic) LimAvg-automata. Our main contribution is an equivalence relation on words characterizing LimAvg-automata, i.e., the equivalence classes of this relation correspond to states of an equivalent LimAvg-automaton. In contrast to relations characterizing DFA, our relation depends not only on the function defined by the target automaton, but also on its structure.

We show two applications of this relation. First, we present a minimization algorithm for LimAvg-automata, which returns a minimal LimAvg-automaton among those equivalent and structurally similar to the input one. Second, we present an extension of Angluin's L^*-algorithm with syntactic queries, which learns in polynomial time a LimAvg-automaton equivalent to the target one.

1 Introduction

Automata have a wide range of AI-related applications, such as natural language processing, verification, compiler construction and others [Rich, 2008]. In many of these applications, the size of the constructed automaton has a far greater impact on performance than the time spent on construction of the automaton. Therefore, it is desirable to develop tools that reduce the number of automata states.

There are various minimization algorithms for (deterministic) finite-state automata over finite words [Nerode, 1958], deterministic weighted automata over finite words [Beimel *et al.*, 1999] or deterministic finite tree automata [Brainerd, 1968]. They are typically based on the right congruence relation of a given language, which characterises the minimal automaton, i.e., the minimal automaton can be constructed over the equivalence classes of the right congruence relation. This approach does not extend to infinite-word automata as the right congruence relation does not characterize deterministic Büchi automata [Maler and Staiger, 1997]. Furthermore, the minimization problem for deterministic Büchi automata is NP-complete [Schewe, 2010], thus establishing a simple relation characterizing languages of these automata is elusive.

In this paper, we study the minimization problem for (deterministic) LIMAVG-automata [Chatterjee *et al.*, 2010], which are weighted automata over infinite words. In these automata, each transition has a rational weight. The run over an infinite word w produces an infinite sequence of weights and the value of w returned by the automaton is the limit of the partial averages of this sequence of weights.

While typical infinite-word automata (e.g. Büchi automata) express properties regarding finite or infinite occurrences of specified events, LIMAVG-automata return more precise answers regarding the long-run frequency of specified events. This makes LIMAVG-automata an attractive specification formalism capable of expressing quantitative system properties [Cerný *et al.*, 2013; Henzinger and Otop, 2017], stream properties [Alur *et al.*, 2017], or even population dynamics in evolutionary games [Miekisz, 2008]. In all these applications, the size of the automaton has a far greater impact on the performance than the time spent on its construction. Therefore, implementations involving LIMAVG-automata would greatly benefit from employing minimization of these automata. Yet, to the best of our knowledge, there is no work on the minimization of LIMAVG-automata.

1.1 Plan and Contributions

This paper is the first work on minimization of LIMAVG-automata. We begin with basic definitions (Section 2) followed by the discussion on various sources of difficulty in minimization of LIMAVG-automata (Section 3). Next, we present our main contributions.

In Section 4, for a given LIMAVG-automaton $\mathcal{T}$, we define the relation $\cong^{\mathcal{T}}$ on words that characterizes $\mathcal{T}$, i.e., we construct a LIMAVG-automaton equivalent to $\mathcal{T}$, whose states are equivalence classes of $\cong^{\mathcal{T}}$. Unlike the right-congruence relation, which is defined for the language of an automaton, the relation $\cong^{\mathcal{T}}$ depends on the structure of $\mathcal{T}$. Therefore, the automaton obtained by minimization with respect to $\cong^{\mathcal{T}}$ is not only semantically equivalent to $\mathcal{T}$, but also has similar structural properties as $\mathcal{T}$. The latter property facilitates explainability of the minimized automaton.

Then, we show two applications of $\cong^{\mathcal{T}}$. In Section 5, we employ $\cong^{\mathcal{T}}$ to minimize an LIMAVG-automata in polynomial time. In Section 6, we propose an active learning framework

for LIMAVG-automata and develop an active learning algorithm LIMAVG-automata utilizing $\cong^{\mathcal{T}}$.

1.2 Related Work

As mentioned above, this is the first paper on minimizing LIMAVG-automata. The most closely-related work to this is [Michaliszyn and Otop, 2020], where an algorithm for learning deterministic Büchi automata is presented. The common ingredient is the syntactic approach: the learning algorithm of [Michaliszyn and Otop, 2020] is based on queries regarding a syntactic function called *loop-index*. This is similar in principle to the algorithm we present in Section 6. However, the key property of deterministic Büchi automata that makes the algorithm of [Michaliszyn and Otop, 2020] work is that a word that does not belong to the language is a proof that all the states it visits infinitely often are non-accepting. We discuss in Section 3 that this kind of argument fails in the LIMAVG case, and thus the algorithm of [Michaliszyn and Otop, 2020] cannot be simply adapted.

For other types of infinite-word automata, it has been shown that deciding, given k and a Deterministic Büchi automaton (DBA) $\mathcal{A}$, whether $\mathcal{A}$ admits a language-equivalent DBA $\mathcal{A}'$ of at most k states, is NP-complete [Schewe, 2010]. It follows that there is no polynomial-time minimization algorithm for DBA, and no learning algorithm for DBA that returns a minimal-size DBA. To bypass this hardness, various techniques have been employed. To learn an ω-language $\mathcal{L}$, [Calbrix *et al.*, 1993] considers learning languages of finite words of the form $\mathcal{L}_\$ = \{u\$v \mid uv^\omega \in \mathcal{L}\}$, where the word $u\$v$ is intended to represent the word uv^ω. However, the size of the constructed deterministic finite automaton can be exponential in the size of the minimal DBA. In [Calbrix *et al.*, 1993], it has been shown that ω-regular languages that can be recognized with a DBA and with a deterministic co-Büchi automaton can be learned in polynomial time. Another approach [Angluin *et al.*, 2018] is to learn DBA into some different formalism, called Families of DFA (FDFA) [Angluin and Fisman, 2016]. Such FDFA can be transformed to deterministic parity automata, but with an exponential blow-up [Angluin *et al.*, 2018].

2 Preliminaries

A *word* w over a finite alphabet Σ of letters is a finite or infinite sequence of letters. An *ultimately periodic* word is a word of the form wv^ω. The sets of all finite and infinite words over Σ by denoted by Σ^* and Σ^ω respectively.

For any sequence w, including words, we define $w[i]$ as the i-th element (letter) of w, and we define $w[i,j]$ as the subsequence (subword) $w[i]w[i{+}1]\dots w[j]$ of w. We assume that sequences start with 0 index.

Deterministic finite-state and ω-automata. A *deterministic finite-state automaton (DFA)* is a tuple $\langle \Sigma, Q, q_0, F, \delta\rangle$ consisting of the alphabet Σ, a finite set of states Q, the initial state $q_0 \in Q$, a subset $F \subseteq Q$ of accepting states, and a transition function $\delta\colon Q \times \Sigma \to Q$.

An (deterministic) ω-automaton is a DFA such that all states are accepting. DFA and deterministic ω-automata differ in semantics. In this paper, we consider only deterministic automata and hence we omit the word "deterministic".

The size of an automaton $\mathcal{A}$, denoted by $|\mathcal{A}|$, is its number of states.

Semantics of automata. We extend δ to $\hat{\delta}\colon Q \times \Sigma^* \to Q$ inductively: for each q, we set $\hat{\delta}(q,\epsilon) = q$, and for all $w \in \Sigma^*, a \in \Sigma$, we set $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q,w), a)$. Given a finite word w, the *run* π of a DFA $\mathcal{A}$ on w is the sequence of states $\hat{\delta}(q_0,\epsilon)\hat{\delta}(q_0, w[0])\hat{\delta}(q_0, w[0,1])\dots$. The run is accepting if the last state belongs to F. The language of $\mathcal{A}$, dented by $\mathcal{L}(\mathcal{A})$, is the set of words having an accepting run.

An ω-automaton processes infinite words and a run over an infinite word is defined as for DFA.

Deterministic LIMAVG-automata. A (deterministic) LIMAVG-automaton $\mathcal{A}$ is a pair $\langle \mathcal{B}, \mathbf{wt}\rangle$ such that $\mathcal{B}$ is an ω-automaton and $\mathbf{wt}\colon Q \times \Sigma \to \mathbb{Q}$ is a labeling of transitions of $\mathcal{A}$ with rationals, called weights. As for ω-automata, we consider only deterministic LIMAVG-automata and hence we omit the word "deterministic".

A *run* π of a LIMAVG-automaton $\mathcal{A}$ on a word w is the run of $\mathcal{B}$ on w. Every run π of $\mathcal{A}$ on an infinite word w defines a sequence of weights $\mathbf{wt}(\pi)$ of successive transitions of $\mathcal{A}$, i.e., $\mathbf{wt}(\pi)[i] = \mathbf{wt}(\pi[i-1], w[i])$. The *value* of the run π is then defined as $\textsc{LimAvg}(\pi) = \limsup_{k\to\infty} \textsc{Avg}(\pi[0,k])$, where for finite runs π we have $\textsc{Avg}(\pi) = \textsc{Sum}(\pi)/|\pi|$, where SUM if the sum of the weights of the run. The value of a word w assigned by the automaton $\mathcal{A}$, denoted by $\mathcal{A}(w)$, is the value of the run of $\mathcal{A}$ on w.

A LIMAVG-automaton $\mathcal{A}$ defines the function $\mathcal{L}(\mathcal{A})\colon \Sigma^\omega \to \mathbb{R}$ such that $\mathcal{L}(\mathcal{A})(w) = \mathcal{A}(w)$. We say that LIMAVG-automata $\mathcal{A}_1, \mathcal{A}_2$ are equivalent if and only if they define the same function, i.e., $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$. Checking equivalence can be done in polynomial time [Chatterjee *et al.*, 2010], and the if two deterministic automata are not equivalent, then there is an ultimately-periodic word wv^ω, where $|w|, |v|$ are of polynomial length, distinguishing them.

ω-generators and factorizations. A run of an ω-automaton on an ultimately periodic word is an ultimately periodic sequence, but the cycle in the run can start in a different position than the cycle in the word. We define *factorization* of a given word to capture that.

An *ω-generator* is a pair $(w, v) \in \Sigma^* \times (\Sigma^* \setminus \{\epsilon\})$. For an ω-generator (w, v), we say that (u_0, u_1) is a factorization of (w, v) with respect to an ω-automaton $\mathcal{A}$ if and only if (i) $wv^\omega = u_0 u_1^\omega$, and (ii) there is a number $c > 0$ such that in the run π on wv^ω for all $i > |u_0|$ we have $\pi[i] = \pi[i+c]$.

Note that in the definition of a factorization we do not require minimality and hence every ω-generator has infinitely many factorizations. Furthermore, if (u_0, u_1) is a factorization w.r.t. $\mathcal{A}$, the automaton $\mathcal{A}$ does not necessarily have a cycle over u_1; it can be over some power u_1^k.

3 Challenges

We discuss various sources of difficulty that appear in attempting to minimize LIMAVG-automata.

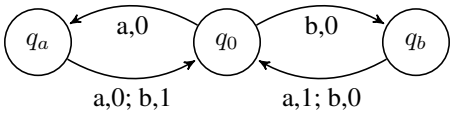

Figure 1: The automaton $\mathcal{A}_{ab}$

3.1 The Right Congruence Relation Is Insufficient

The minimization of DFA is based on the right congruence relation $\sim_{\mathcal{L}} \subseteq \Sigma^* \times \Sigma^*$ of $\mathcal{L}$. For $v, u \in \Sigma^*$ we have $u \sim_{\mathcal{L}} v$ if and only if for all $w \in \Sigma^*$ it holds that $uv \in \mathcal{L} \iff vw \in \mathcal{L}$. The Myhill-Nerode theorem [Nerode, 1958] states that $\mathcal{L}$ is regular exactly when $\sim_{\mathcal{L}}$ has finitely many equivalence classes. Moreover, the relation $\sim_{\mathcal{L}}$ defines the minimal DFA recognizing $\mathcal{L}$; equivalence classes of $\sim_{\mathcal{L}}$ correspond to states of the minimal DFA, while transitions can be defined in a natural way. The Myhill-Nerode theorem has its counterparts for various types of automata such as tree automata [Brainerd, 1968] or weighted automata [Beimel *et al.*, 1999].

We define the counterpart of $\sim_{\mathcal{L}}$ for LIMAVG-automata. For a LIMAVG-automaton $\mathcal{T}$ we define $\sim^{\mathcal{T}}$ over Σ^* in the following way: for all $u, v \in \Sigma^*$ we have $u \sim^{\mathcal{T}} v$ if and only if for all $w \in \Sigma^\omega$ it holds that $\mathcal{T}(uw) = \mathcal{T}(vw)$.

However, the Myhill-Nerode theorem does not hold for LIMAVG-automata. Consider the automaton $\mathcal{A}_{ab}$ presented in Figure 1 with q_0 being the initial state. It has two equivalence classes of $\sim^{\mathcal{T}}$: the class $[\epsilon]_{\sim^{\mathcal{T}}}$ containing all the words with even length and $[a]_{\sim^{\mathcal{T}}}$ containing all the remaining words. To see that the classes are different, consider words u of odd length and v of even length. Then $\mathcal{A}_{ab}(u(aabb)^\omega) = \frac{1}{2}$ and $\mathcal{A}_{ab}(v(aabb)^\omega) = 0$. All the words of even length are in the same equivalence class because after reading them, the automaton is in the state q_0. The most interesting part is that all odd length words are in the same equivalence class: this is because after reading such a word, the automaton can be in q_a or q_b, but, after reading any letter, it will be in q_0. Since LIMAVG does not depend on finite prefixes, for any v, v' of odd length and $w \in \Sigma^\omega$ we have $\mathcal{A}_{ab}(vw) = \mathcal{A}_{ab}(v'w)$.

Observe that there is no equivalent LIMAVG-automaton based on these equivalence classes. Such an automaton would require two states, q'_0 and q'_a, such that $\delta(q'_0, a) = \delta(q'_0, b) = q'_a$ and $\delta(q'_a, a) = \delta(q'_a, b) = q'_0$. To define the weights $\mathbf{wt}$, observe that the following equations need to be satisfied:

$$\mathbf{wt}(q'_0, a) + \mathbf{wt}(q'_a, a) = 0 \quad \mathbf{wt}(q'_0, b) + \mathbf{wt}(q'_a, b) = 0$$
$$\mathbf{wt}(q'_0, b) + \mathbf{wt}(q'_a, a) = 1 \quad \mathbf{wt}(q'_0, a) + \mathbf{wt}(q'_a, b) = 1$$

By adding top two equations and subtracting the bottom two, we obtain $0 = -2$, a contradiction.

In Section 4 we present a relation refining the right congruence relation characterizing LIMAVG-automata.

3.2 Minimal Automata Are Not Unique

We present two examples of ambiguity, which we generalise later on.

Consider automata $\mathcal{A}_a, \mathcal{A}_b$ resulting from changing the initial state in the automaton $\mathcal{A}_{ab}$ from Figure 1 to q_a and q_b respectively. Observe that for every infinite word w, the runs in $\mathcal{A}_a$ and $\mathcal{A}_b$ over w differ only in the first transition to q_0, while the whole infinite suffix is the same. Therefore, the word w has the same value in both automata and hence $\mathcal{A}_a$ and $\mathcal{A}_b$ are equivalent.

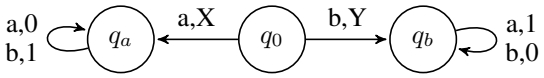

Figure 2: The automaton $\mathcal{A}_{X,Y}$ parameterized by X, Y

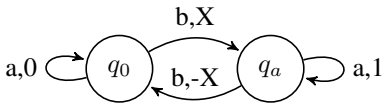

Figure 3: The automaton $\mathcal{A}^{\pm X}$ parameterized by X

Consider automata $\mathcal{A}_{ab,L}$ and $\mathcal{A}_{ab,R}$ defined as an extension of $\mathcal{A}_{ab}$ in the following way. Both automata have an additional state s_0, which is initial and has a self-loop over a of the weight 0. The automaton $\mathcal{A}_{ab,L}$ moves with the weight 0 to q_a from $\mathcal{A}_{ab}$, while $\mathcal{A}_{ab,R}$ moves on b with the weight 0 to q_a from $\mathcal{A}_{ab}$. Observe that both automata are equivalent

More generally, in any LIMAVG-automaton $\mathcal{T}$, changing the initial state q_0 to another state q equivalent w.r.t. the right congruence (i.e., the state q such that for every u if $\hat{\delta}(q_0, u) = q$, then $\epsilon \sim^{\mathcal{T}} u$) preserves equivalence of automata. Furthermore, it follows that transitions between strongly-connected components are not unique as long as the target states are equivalent w.r.t. the right congruence.

3.3 Weights Are Ambiguous

Now, we discuss why the weights in LIMAVG-automata are not uniquely defined. First, consider automata $\mathcal{A}_{X,Y}$ from Fig. 2. These automata are minimal and all pairwise equivalent, since LIMAVG does not depend on finite prefixes.

We further argue that even the values of edges that are on some cycles are not uniquely defined. Consider any $X \in \mathbb{Q}$ and the automaton $\mathcal{A}^{\pm X}$ presented in Figure 3. Then, this automaton is equivalent to $\mathcal{A}^{\pm 0}$. This is because for any word w, the difference between the partial sum for any prefix of the runs of $\mathcal{A}^{\pm X}$ and $\mathcal{A}^{\pm 0}$ on w is bounded by X, and thus the limits are the same. The same thing can happen in virtually every automaton with more than one state: if we take an automaton $\mathcal{A}$, a state q and a number $X \in \mathbb{Q}$, and define $\mathcal{A}^X$ as a copy of $\mathcal{A}$, expect that all the incoming edges (except for self-loops) to q have weights increased by X, and all the outgoing edges (except for self-loops) from q have weights decreased by X, then $\mathcal{A}$ and $\mathcal{A}^X$ are equivalent. We call this "pushing weights". It demonstrates that every single weight of an automaton, except self-loops, can be set arbitrarily.

A more general issue regarding weights is called **non-locality**. In the deterministic Büchi automata case, a word that does not belong to the language is a witness that all the states it visits infinitely often are rejecting. This is the fundamental idea of [Michaliszyn and Otop, 2020]. In the LIMAVG case, a single ultimately-periodic word only defines the average value of its ultimately-periodic cycle, but the values of the particular edges can be "pushed", as showed above. Thus, computing weights for the automaton has to be done globally.

4 Characterization of Automata

In this section we introduce a relation $\cong^{\mathcal{T}}$ on words, which characterizes a given LIMAVG-automaton $\mathcal{T}$. This relation is an intersection of two relations: $\sim^{\mathcal{T}}$, which is a counterpart of the classical right-congruence relation (as discussed in the previous section), and $\equiv^{\mathcal{T}}$, which is a syntactical relation. This in turn, can be used to construct a LIMAVG-automaton equivalent to $\mathcal{T}$, whose states correspond to equivalence classes of $\cong^{\mathcal{T}}$.

Definition 1. *Given a* LIMAVG*-automaton $\mathcal{T}$, we define the right-congruence relation $\sim^{\mathcal{T}}$, the cycle-equivalence relation $\equiv^{\mathcal{T}}$ and the mixed relation $\cong^{\mathcal{T}}$ on Σ^* as follows:*

$u \sim^{\mathcal{T}} v \iff \mathcal{T}(uw) = \mathcal{T}(vw)$ *for all* $w \in \Sigma^\omega$,

$u \equiv^{\mathcal{T}} v \iff$ *for all* $c, d \in \Sigma^*$, (uc, d) *factorizes to* (uc, d) *iff* (vc, d) *factorizes to* (vc, d),

$u \cong^{\mathcal{T}} v \iff u \sim^{\mathcal{T}} v$ *and* $u \equiv^{\mathcal{T}} v$.

Note that $\sim^{\mathcal{T}}$, $\equiv^{\mathcal{T}}$ and $\cong^{\mathcal{T}}$ are equivalence relations.

Observe that relation $\sim^{\mathcal{T}}$ is semantic, i.e., it depends only on the function defined by $\mathcal{T}$. In particular, it is invariant under equivalence, i.e., swapping $\mathcal{T}$ with an equivalent LIMAVG-automaton does not change this relation. In contrast, relation $\equiv^{\mathcal{T}}$ depends on the structure of $\mathcal{T}$, while it does not depend on weights of $\mathcal{T}$. In other words, $u \equiv^{\mathcal{T}} v$ holds if for all words $c, d \in \Sigma^*$, either from both states $\hat{\delta}^{\mathcal{T}}(q_0^{\mathcal{T}}, uc)$ and $\hat{\delta}^{\mathcal{T}}(q_0^{\mathcal{T}}, vc)$, the automaton $\mathcal{T}$ has cycles labeled by some powers of d, or from none of them.

Definition 2. *For the relation $\cong^{\mathcal{T}}$ of $\mathcal{T}$, we say that an ω-automaton $\mathcal{B} = \langle \Sigma, Q, q_0, \delta \rangle$ is* consistent *with $\cong^{\mathcal{T}}$ if it satisfies the following conditions: (1) Q is the set of equivalence classes of $\cong^{\mathcal{T}}$, (2) $q_0 = [\epsilon]_{\cong^{\mathcal{T}}}$ is the equivalence class of the empty word ϵ, and (3) for all $q \in Q$ and $a \in \Sigma$, there exists u s.t. $[u]_{\cong^{\mathcal{T}}} = q$ such that $\delta(q, a) = [ua]_{\cong^{\mathcal{T}}}$.*

A consistent automaton clearly exists. We show that it is unique and can be extended to an automaton equivalent to $\mathcal{T}$.

Theorem 3. *For every* LIMAVG*-automaton $\mathcal{T}$, there exists a consistent ω-automaton $\mathcal{B}$, which is (a) unique, and (b) there exists labeling $\mathbf{wt}$ of transitions of $\mathcal{B}$ with rational numbers such that the resulting* LIMAVG*-automaton $(\mathcal{B}, \mathbf{wt})$ is equivalent to $\mathcal{T}$. Such a labeling $\mathbf{wt}$ can be computed in polynomial time in $|\mathcal{T}|$.*

The proof of (a) follows from the fact that $\cong^{\mathcal{T}}$ is a congruence. Based on this, we create $\mathcal{B}$ in the straightforward way. To show (b), we pick an automaton $\mathcal{T}$, an equivalence class $[u]_{\cong^{\mathcal{T}}}$ and define S_u as the set of states s such that if $\delta^{\mathcal{T}}(q_0^{\mathcal{T}}, v) = s$ then $u \cong^{\mathcal{T}} v$. States S_u can be contracted to a single state q_S and the resulting LIMAVG-automaton $\mathcal{A}_u$ is equivalent to $\mathcal{T}$. Relations $\cong^{\mathcal{T}}$ and $\cong^{\mathcal{A}_u}$ coincide. We iterate this process until we obtain a LIMAVG-automaton $\mathcal{A}^*$ such that each state of $\mathcal{A}^*$ corresponds to a single equivalence class of $\cong^{\mathcal{T}}$. The automaton $\mathcal{A}^*$ is equivalence to all its predecessors and in particular $\mathcal{T}$ and it is isomorphic to $\mathcal{B}$. Therefore, the weights $\mathbf{wt}$ of $\mathcal{A}^*$ can be transferred to $\mathcal{B}$.

To compute $\mathbf{wt}$, we can create in polynomial time a system of linear equations whose solution defines $\mathbf{wt}$. The details can be found in the extended version of this paper.

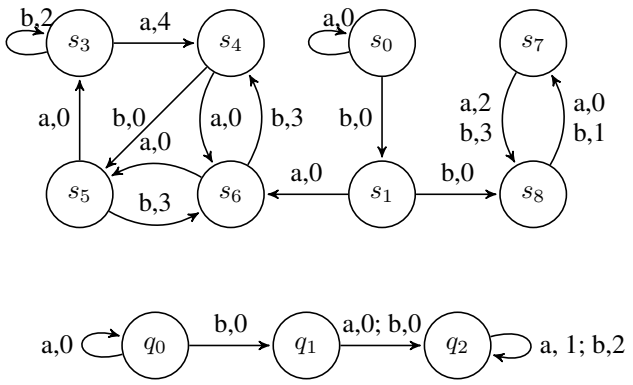

Figure 4: An example of an input and an output of the minimization procedure. Notice that the minimal semantically equivalent automaton has 2 states (q_1 can be removed).

5 Minimization Algorithm

In this section we present a minimization algorithm for LIMAVG-automata. For a given LIMAVG-automaton $\mathcal{T}$ the presented algorithm constructs an automaton whose states are the equivalence classes of $\cong^{\mathcal{T}}$. This automaton has at most as many states as $\mathcal{T}$. However, it does not need to be minimal among all LIMAVG-automata equivalent to $\mathcal{T}$. Indeed, if $\mathcal{T}$ has a complex structure, its cycle-equivalence is likely non-trivial. However, if it has only 0 weights, then it is equivalent to a single-state LIMAVG-automaton. Figure 4 is an example of how the algorithm works: it reduces an 8-states automaton to a 3-states automaton, while the minimal semantically equivalent automaton has 2 states.

The relation $\cong^{\mathcal{T}}$ is defined over words. This is a typical choice for such relations, and this will prove handy in the automata learning process (Section 6), where the automaton $\mathcal{T}$ is concealed and can be only queried. However, in this section, we assume the full access to the automaton $\mathcal{T}$. In such a scenario, it is convenient to define relations on states rather than words. We present a canonical way to lift an equivalence relation from words to states.

Let $\bowtie^{\mathcal{T}}$ be one of the symbols among $\sim^{\mathcal{T}}$, $\equiv^{\mathcal{T}}$ and $\cong^{\mathcal{T}}$. We define the relation $\bowtie^{Q}$, which is a counterpart of $\bowtie^{\mathcal{T}}$, as follows. For each state q of $\mathcal{T}$, let W_q be the set of words w that lead to q, i.e., $\hat{\delta}^{\mathcal{T}}(q_0^{\mathcal{T}}, w) = q$. For all $q, s \in Q^{\mathcal{T}}$ it holds that $q \bowtie^{Q} s$ if and only if for some words $u \in W_q$ and $v \in W_s$ we have $u \bowtie^{\mathcal{T}} v$. Observe that if $u, u' \in W_q$ (resp., W_s), then $u \bowtie^{\mathcal{T}} u'$. Therefore, the choice of words $u \in W_q$ and $v \in W_s$ is irrelevant.

The main step of the minimization algorithm is to compute the relation $\cong^{Q}$. Observe that for states $q, s \in Q^{\mathcal{T}}$ we have $q \cong^{Q} s$ if and only if $q \sim^{Q} s$ and $q \equiv^{Q} s$. The algorithm works in two steps: first it constructs $\sim^{Q}$, and then for each equivalence class of $\sim^{Q}$, it refines the partition w.r.t. $\equiv^{Q}$. We discuss these two steps below.

Computing relation $\sim^{Q}$. Consider $q, s \in Q^{\mathcal{T}}$. To decide whether $q \sim^{Q} s$, we construct the product automaton $\mathcal{A}_\times = \mathcal{T} \times \mathcal{T}$ with the initial state (q, s) and the weight of each transition being the difference between weights of transitions in the components, i.e., for all $q_1, s_1 \in \mathcal{T}$ and $a \in \Sigma$ we

define $\mathbf{wt}^{\mathcal{A}_\times}((q_1, s_1), a) = \mathbf{wt}^{\mathcal{T}}(q_1, a) - \mathbf{wt}^{\mathcal{T}}(s_1, a)$. Then, we check whether $\mathcal{A}_\times$ has a reachable cycle of non-zero sum of weights, which can be done in polynomial time [Cormen *et al.*, 2009, Chapter 24]. If there is no such cycle, then the limit-average of weights along every run in $\mathcal{A}_\times$ is 0, and hence for every word w, we have $\mathcal{T}_q(w) = \mathcal{T}_s(w)$, i.e., the automaton $\mathcal{T}$ starting from q on w returns the same value as starting from s. Thus, $q \sim^Q s$. Otherwise, the cycle of a non-zero sum of weights corresponds to a word w such that $\mathcal{T}_q(w) \neq \mathcal{T}_s(w)$ and hence $q \not\sim^Q s$.

Imporved complexity. The above procedure can be optimized two work in $O(|Q|^4|\Sigma|)$. The idea is to compute $\sim^Q$ for all pairs of states at once. We construct the automaton $\mathcal{A}_\times$ as previously, but we drop the initial state and labels of letters. We obtain a weighted graph G over vertexes $Q^{\mathcal{T}} \times Q^{\mathcal{T}}$. We partition G into strongly connected components (SCCs). Next, for each SCC we check whether it has a cycle of a non-zero sum of weights [Cormen *et al.*, 2009, Chapter 24]. We label all pairs of states (q, s) in SCCs with a non-zero sum cycle as non-$\sim^Q$-equivalent, and all pairs of states (q, s) from which non-$\sim^Q$-equivalent SCCs are reachable as non-$\sim^Q$-equivalent as well. Finally, from any pair (q, s) of states, which are not labeled as non-$\sim^Q$-equivalent, only cycles of total sum 0 are reachable, and hence $q \sim^Q s$. Careful analysis shows that the obtained complexity of this approach is dominated by the cost of finding non-zero cycles, which is quadratic in the number of states of the product automaton (and the number of such states is quadratic), resulting in the above-mentioned complexity.

Computing relation $\equiv^Q$. We say that q and s and not *immediately cycle-equivalent* if there is a word d such that for some words $v_q, v_s \in \Sigma^*$ that lead to q and s respectively we have (v_q, d) factorizes to (v_q, d), while (v_s, d) does not factorize to (v_s, d) (or the same with q and s swapped). Observe that $q \not\equiv^Q s$ if and only if there is a word c such that $\hat{\delta}(q, c)$ and $\hat{\delta}(s, c)$ are not immediately cycle-equivalent. We argue that the following conditions are equivalent:

(i) q and s are not immediately cycle-equivalent,

(ii) there is a word $u \in \Sigma^*$ such that $\mathcal{T}$ has a cycle over some u^k from q, but for every $n > 0$, there is no cycle over u^n from s (or the same with q and s swapped), and

(iii) there is a word $u \in \Sigma^*$ and a state $s' \neq s$ such that $\mathcal{T}$ has a cycle over some u from q ($\hat{\delta}(q, u) = q$), $\hat{\delta}(s, u) = s'$, and $\hat{\delta}(s', u) = s'$ (or the same with q and s swapped).

The equivalence between (i) and (ii) follows from the definition. Condition (iii) clearly implies condition (ii). For the converse, consider a word d satisfying (ii). Then, $d' = d^k$ satisfies $\hat{\delta}(q, d') = q$ and for all $n > 0$ we have $\hat{\delta}(s, d') \neq s$. Then, there are $a, b > 0$ such that $\hat{\delta}(s, (d')^a) = \hat{\delta}(s, (d')^{a+b})$, and hence $\hat{\delta}(s, (d')^{ab}) = \hat{\delta}(s, (d')^{ab+b})$. Therefore, $\hat{\delta}(s, (d')^{ab}) = \hat{\delta}(s, (d')^{ab+ab})$ and d^{kab} satisfies condition (iii).

Condition (iii) can be checked in polynomial time. We can try all states as s'. For a fixed s' we construct the product automaton consisting of three copies of $\mathcal{T}$ starting from the state (q, s, s') and check whether it is possible to reach the state (q, s', s'). This can be done in time $O(|V|^4|\Sigma|)$.

We mark all pairs q and s that are not immediately cycle-equivalent as not $\equiv^Q$-equivalent and back propagate, i.e., if q', s' are such that $\delta(q', a) = q$ and $\delta(s', a) = s$ then we mark q' and s' as not $\equiv^Q$-equivalent. The back-propagation can be done in time $O(|V|^4|\Sigma|)$.

The minimization. The intersection of $\equiv^Q$ and $\sim^Q$ yields $\cong^Q$. We construct an automaton $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, Q, \mathbf{wt} \rangle$ in the following way. The set $Q \subseteq Q^{\mathcal{T}}$ is any selector of equivalence classes of $\cong^Q$, and q_0 is the representative of the equivalence class of $q_0^{\mathcal{T}}$. The transition relation is defined as follows: for $q \in Q$ and $a \in \Sigma$ we define $\delta(q, a)$ as the representative from Q of the equivalence class of $\delta^{\mathcal{T}}(q, a)$. Observe that the ω-automaton $\langle \Sigma, Q, q_0, \delta, Q \rangle$ is consistent with $\cong^{\mathcal{T}}$. Therefore, by Theorem 3 we can compute $\mathbf{wt} : Q \times \Sigma \to \mathbb{Q}$ in polynomial time such that the LIMAVG-automaton $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, Q, \mathbf{wt} \rangle$ is equivalent with $\mathcal{T}$.

Theorem 4 (Minimization w.r.t. $\cong^{\mathcal{T}}$)**.** *Given a* LIMAVG*-automaton $\mathcal{T}$, we can compute in polynomial time in $|\mathcal{T}|$ a* LIMAVG*-automaton $\mathcal{A}$, which is equivalent to $\mathcal{T}$, and each state of $\mathcal{A}$ corresponds to a different equivalence class of $\cong^{\mathcal{T}}$.*

6 Learning Algorithm

We present an active learning algorithm for (deterministic) LIMAVG-automata base on L^*-algorithm [Angluin, 1987].

We construct the relation $\cong^{\mathcal{T}}$ iteratively. We start with defining a relation $\cong^{\mathcal{T}}_C$, which is a version of the relation $\cong^{\mathcal{T}}$ parametrized by a set of ω-generators C. The relation $\cong^{\mathcal{T}}_C$ approximates its non-parameterized counterpart. We show that there is a parameter C of polynomial size in $\mathcal{T}$ such that $\cong^{\mathcal{T}}_C$ and $\cong^{\mathcal{T}}$ coincide. The learning algorithm will learn such a C.

The equivalence classes of $\cong^{\mathcal{T}}$ can be characterized by means of the following function $\mathrm{TLV}^{\mathcal{T}}_u$. Let $\mathrm{TLV}^{\mathcal{T}}_u(c, d)$ be defined as the pair $(\mathcal{T}(ucd^\omega), e)$, where e is the boolean value representing whether (uc, d) factorizes to (uc, d). It can be easily shown that for all words u, v we have $\mathrm{TLV}^{\mathcal{T}}_u = \mathrm{TLV}^{\mathcal{T}}_v$ if and only if $u \cong^{\mathcal{T}} v$.

A natural approach would be to consider queries regarding $\mathrm{TLV}^{\mathcal{T}}$. Here, we show that even a simpler function suffices for learning: $\mathrm{LV}^{\mathcal{T}}_u$. Let $\mathrm{LV}^{\mathcal{T}}_u(c, d)$ be defined as $\mathcal{T}(ucd^\omega)$ if (uc, d) factorizes to (uc, d), and $\perp$ otherwise.

Definition 5 (Parameterized word relations)**.** *Given a* LIMAVG*-automaton $\mathcal{T}$ and a set of ω-generators C, we define the relation $\cong^{\mathcal{T}}_C$ on Σ^ω as follows: $u \cong^{\mathcal{T}}_C v$ iff for all $(c, d) \in C$ we have $LV_u(c, d) = LV_v(c, d)$.*

Observe that for a LIMAVG-automaton $\mathcal{T}$ and any two different equivalence classes $[u]_{\cong^{\mathcal{T}}}, [v]_{\cong^{\mathcal{T}}}$ of $\cong^{\mathcal{T}}$ either there is an ultimately periodic word w witnessing that $u \not\sim^{\mathcal{T}} v$, or finite words c, d witnessing that $u \not\equiv^{\mathcal{T}} v$. We can pick the word w to factorize in at least one of the words u, v. Therefore, there is C with at most $|\mathcal{T}|^2$ elements such that relations $\cong^{\mathcal{T}}_C$ and $\cong^{\mathcal{T}}$ coincide. Furthermore, a pumping argument shows that we can pick C whose elements are of length at most $|\mathcal{T}|^3$.

Lemma 6. *For every* LIMAVG*-automaton $\mathcal{T}$, there is a finite set of ω-generators C such that $\cong^{\mathcal{T}}_C$ coincides with $\cong^{\mathcal{T}}$.*

An ω-automaton $\mathcal{B} = \langle \Sigma, Q, q_0, \delta \rangle$ is *partially consistent* with $\cong_C^{\mathcal{T}}$ if it satisfies the following conditions: (1) Q is a subset of the equivalence classes of $\cong^{\mathcal{T}}$, (2) $q_0 = [\epsilon]_{\cong^{\mathcal{T}}}$ is the equivalence class of the empty word ϵ, and (3) for all $q \in Q$ and $a \in \Sigma$, there exists $u \in q$ such that $\delta(q, a) = [ua]_{\cong^{\mathcal{T}}}$.

The framework. For the remainder of this section we fix a target LIMAVG-automaton $\mathcal{T}$. We consider the active learning setting, in which the learning algorithm asks queries to an oracle called *the teacher*. The teacher answers queries about the target automaton $\mathcal{T}$, which is concealed from the learning algorithm. We consider the following types of queries:

LV-value query	
Input	word $u \in \Sigma^*$ and ω-generator (c, d)
Output	$\mathrm{LV}_u^{\mathcal{T}}(c, d)$

LV-equivalence query	
Input	ω-automaton $\mathcal{A}$
Output	**Consistent** if for all u, v such that $\hat{\delta}^{\mathcal{A}}(q_0^{\mathcal{A}}, u) = \hat{\delta}^{\mathcal{A}}(q_0^{\mathcal{A}}, v)$ we have $\mathrm{LV}_u^{\mathcal{T}} = \mathrm{LV}_v^{\mathcal{T}}$, a counterexample otherwise

We assume that a counterexample to an LV-equivalence query is a tuple $\langle u, v, c, d \rangle$ of words such that $\hat{\delta}^{\mathcal{A}}(q_0, u) = \hat{\delta}^{\mathcal{A}}(q_0, v)$ and $\mathrm{LV}_u^{\mathcal{T}}(c, d) \neq \mathrm{LV}_v^{\mathcal{T}}(c, d)$.

The algorithm. Let $\mathcal{T}$ be the target automaton. The learning algorithm works in two stages. First, it learns an ω-automaton $\mathcal{A}$ consistent with $\cong^{\mathcal{T}}$. Having the automaton $\mathcal{B}$ it finds the labeling $\mathbf{wt}$ such that $\langle \mathcal{B}, \mathbf{wt} \rangle$ is equivalent to the target automaton $\mathcal{T}$. We discuss these two stages below.

Learning a consistent automaton. To learn an ω-automaton consistent with $\cong^{\mathcal{T}}$, we adapt the L^*-algorithm [Angluin, 1987]. The algorithm maintains two sets: a set of finite words S, called *selectors*, and a set of ω-generators C, called *test words*. It works by saturating S and C until (i) $\cong_C^{\mathcal{T}}$ coincides with $\cong^{\mathcal{T}}$, and (ii) every equivalence $[u]_{\cong^{\mathcal{T}}}$ contains exactly one word from S. The main body of the algorithm is presented as Algorithm 1, where T.LVEQUIVALENCE(A) is the LV-equivalence query.

For S, C, $s \in S$ and $a \in \Sigma$, let $\delta_{S,C}^{MAY}(s, a) = \{s' \in S \mid sa \cong_C^{\mathcal{T}} s'\}$ be the set of possible values of $\delta(s, a)$. Observe that testing the membership of $\delta_{S,C}^{MAY}(s, a)$ can be done effectively: $s' \in \delta_{S,C}^{MAY}(s, a)$ if and only if for all $(c, d) \in C$ we have T.LVVALUE$(s', (c, d))$ = T.LVVALUE$(sa, (c, d))$.

The procedure EXTEND$(S, C, \langle u, v, c, d \rangle)$, defined in Algorithm 1, extends S, C to S', C' via a fixed-point computation that makes sure that for every selector $s \in S$ and every letter $a \in \Sigma$ there is a selector $s' \in S$ such that $sa \cong_{C'}^{\mathcal{T}} s'$. If such a selector is not in S, then it extends S with sa. It terminates in polynomial time; $|S|$ is bounded by the number of equivalence classes of $\cong_{C'}^{\mathcal{T}}$, which is bounded by $|\mathcal{T}|$.

Lemma 7. *The procedure* EXTEND$(S, C, \langle u, v, c, d \rangle)$ *works in polynomial time and returns S', C', A such that A is partially consistent with $\cong_C^{\mathcal{T}}$ and either $S' \supset S$ or $S' = S$ and $\delta_{S',C'}^{MAY} \subset \delta_{S,C}^{MAY}$.*

Algorithm 1 The learning algorithm.

```
1:  procedure LEARN-STRUCTURE(T)
2:      S := {ε}, C := ∅
3:      A := the one state automaton
4:      while true do
5:          e := T.LVEQUIVALENCE(A)
6:          if e = Consistent then return A
7:          S, C, A := EXTEND(S, C, e)
8:  procedure EXTEND(S, C, ⟨u, v, c, d⟩)
9:      C := C ∪ {(xc, d) | x is a suffix of u or v}
10:     δ = ∅, Done = ∅
11:     while S ≠ Done do
12:         for all s ∈ S \ Done, a ∈ Σ do
13:             if there is s' ∈ δ^MAY_{S,C}(s, a)
14:             then δ(s, a) = s'
15:             else S := S ∪ {sa}, δ(s, a) = sa
16:             Done := Done ∪ {s}
17:     return S, C, ⟨Σ, S, ε, δ⟩
```

The algorithm LEARN-STRUCTURE(T) iterates the main loop at most polynomially many times and each iteration takes polynomial time. Therefore, it works in polynomial time. The termination condition implies that the returned automaton is consistent with $\cong^{\mathcal{T}}$.

Learning weights. Consider an ω-automaton $\mathcal{B}$ consistent with $\cong^{\mathcal{T}}$. We assign with each transition e_i a variable x_i corresponding to its weight. Suppose that transition e_1 occurs in a cycle $e_1e_2e_3$ labeled with aba and u leads to e_1 cycle from q_0. Then, the LV-query (u, aba) returns α, which is the mean of weights in the cycle $e_1e_2e_3$. We express this with the equation: $x_1 + x_2 + x_3 = 3 \cdot \alpha$.

Based on the structure of $\mathcal{B}$, we can construct in polynomial time a maximal linearly independent system of linear equation characterizing weights of transitions $e_1, \ldots, e_k$. It follows that every solution to that system gives rise to the labeling $\mathbf{wt}$ defined by $\mathbf{wt}(e_i) = x_i$ such that $\langle \mathcal{B}, \mathbf{wt} \rangle$ is equivalent to $\mathcal{T}$.

Lemma 8. *Given an ω-automaton $\mathcal{B}$, we can compute a system of linear equations $A\vec{x} = \vec{b}$ in polynomial time such that any solution $\vec{x}$ defines $\mathbf{wt}$ by $\mathbf{wt}(e_i) = x_i$ such that $\langle \mathcal{B}, \mathbf{wt} \rangle$ is equivalent to $\mathcal{T}$. Furthermore, the matrix A is defined based only on $\mathcal{B}$ and the vector $\vec{b}$ is computed as the result of LV-value queries.*

Linear equations over the rational numbers can be solved in polynomial time, thus we can effectively learn LIMAVG-automata:

Theorem 9. *Active learning deterministic* LIMAVG*-automata with teacher answering LV-value queries and LV-equivalence queries can be done in time polynomial in the size of the target automaton and the size of teachers' responses.*

Acknowledgments

This work was supported by the National Science Centre (NCN), Poland under grant 2017/27/B/ST6/00299.

References

[Alur *et al.*, 2017] Rajeev Alur, Konstantinos Mamouras, and Caleb Stanford. Automata-based stream processing. In *ICALP 2017*, volume 80 of *LIPIcs*, pages 112:1–112:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[Angluin and Fisman, 2016] Dana Angluin and Dana Fisman. Learning regular omega languages. *Theor. Comput. Sci.*, 650:57–72, 2016.

[Angluin *et al.*, 2018] Dana Angluin, Udi Boker, and Dana Fisman. Families of DFAs as acceptors of ω-regular languages. *LMCS*, 14(1), 2018.

[Angluin, 1987] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

[Beimel *et al.*, 1999] Amos Beimel, Francesco Bergadano, Nader Bshouty, Eyal Kushilevitz, and Stefano Varricchio. Learning functions represented as multiplicity automata. *Journal of the ACM*, 47, 10 1999.

[Brainerd, 1968] Walter S Brainerd. The minimalization of tree automata. *Information and Control*, 13(5):484–491, 1968.

[Calbrix *et al.*, 1993] Hugues Calbrix, Maurice Nivat, and Andreas Podelski. Ultimately periodic words of rational w-languages. In *MFPS 1993*, pages 554–566, 1993.

[Cerný *et al.*, 2013] Pavol Cerný, Thomas A. Henzinger, and Arjun Radhakrishna. Quantitative abstraction refinement. In *POPL 2013*, pages 115–128. ACM, 2013.

[Chatterjee *et al.*, 2010] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Trans. Comput. Log.*, 11(4):23:1–23:38, 2010.

[Cormen *et al.*, 2009] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[Henzinger and Otop, 2017] Thomas A Henzinger and Jan Otop. Model measuring for discrete and hybrid systems. *Nonlinear Analysis: Hybrid Systems*, 23:166–190, 2017.

[Maler and Staiger, 1997] Oded Maler and Ludwig Staiger. On syntactic congruences for omega-languages. *Theor. Comput. Sci.*, 183(1):93–112, 1997.

[Michaliszyn and Otop, 2020] Jakub Michaliszyn and Jan Otop. Learning deterministic automata on infinite words. In *ECAI 2020 - 24th European Conference on Artificial Intelligence*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 2370–2377. IOS Press, 2020.

[Miekisz, 2008] Jacek Miekisz. Evolutionary game theory and population dynamics. In *Multiscale problems in the life sciences*, pages 269–316. Springer, 2008.

[Nerode, 1958] Anil Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958.

[Rich, 2008] Elaine Rich. *Automata, computability and complexity: theory and applications*. Pearson Prentice Hall Upper Saddle River, 2008.

[Schewe, 2010] Sven Schewe. Beyond hyper-minimisation—minimising DBAs and DPAs is NP-complete. In *FSTTCS 2010*, pages 400–411, 2010.