Learning to reuse: Adaptive model learning for evolving systems

Carlos Diego N. Damasceno[1,2][0000−0001−8492−7484], Mohammad Reza Mousavi[2][0000−0002−4869−6794], and Adenilso da Silva Simao[1][0000−0002−1454−2607]

[1] University of Sao Paulo (ICMC-USP), São Carlos-SP, BR
damascenodiego@usp.br, adenilso@icmc.usp.br
[2] University of Leicester, Leicester, UK
mm789@leicester.ac.uk

Abstract. Software systems undergo several changes along their life-cycle and hence, their models may become outdated. To tackle this issue, we propose an efficient algorithm for adaptive learning, called `partial-Dynamic` $\mathtt{L}^*_\mathtt{M}$ ($\partial\mathtt{L}^*_\mathtt{M}$) that improves upon the state of the art by exploring observation tables *on-the-fly* to discard *redundant* prefixes and *deprecated* suffixes. Using 18 versions of the OpenSSL toolkit, we compare our proposed algorithm along with three adaptive algorithms. For the existing algorithms in the literature, our experiments indicate a strong positive correlation between number of membership queries and temporal distance between versions and; for our algorithm, we found a weak positive correlation between membership queries and temporal distance, as well, a significantly lower number of membership queries. These findings indicate that, compared to the state-of-the-art algorithms, our $\partial\mathtt{L}^*_\mathtt{M}$ algorithm is less sensitive to software evolution and more efficient than the current approaches for adaptive learning.

Keywords: Active learning · Mealy machines · Software evolution · Software reuse · Reactive systems

1 Introduction

According to Binder [3], software analysis is necessarily a *model-based* activity, whether models are in engineers' minds, informally sketched on papers or formally denoted as an explicit model [4]. Nevertheless, as requirements change and systems evolve, model maintenance is often neglected due to its high cost and models are rendered outdated [33]. To tackle this issue, active model learning [1] has been increasingly used to automatically derive behavioral models [15,21,30]. Active model learning aims at building a hypothesis $\mathcal{H}$ about the "language" (i.e., the set of possible behaviors) of a system under learning (SUL) by iteratively providing input sequences and observing outputs [30]. To formulate a hypothesis model $\mathcal{H}$, the learning algorithm searches for specific pairs of sequences to reach and distinguish states, i.e., called *transfer* and *separating* sequences, respectively.

Applying model learning to industrial systems is hampered by scalability issues [8] as well as the constant changes along their life-cycle [5] that may require *learning from scratch*. Adaptive learning [10] is an approach that attempts to speed up learning by reusing the knowledge from existing models. Studies [5,10,35,13] have shown that pre-existing models can steer learning by reusing the sequences applied in the past queries and hence, reduce the cost for re-inferring models from different versions. However, after several changes, old separating sequences may no longer be able to distinguish states, and lead to *deprecated* queries. Similarly, transfer sequences may not lead to different states anymore, and become *redundant*. These are known to be major threats to adaptive learning [13].

In this paper, we address the issues of redundant and deprecated sequence. To mitigate the aforementioned issues, we introduce `partial-Dynamic` $\mathtt{L}^*_\mathtt{M}$ ($\partial\mathtt{L}^*_\mathtt{M}$), an adaptive algorithm where the cost for active model learning is reduced by exploring observation tables *on-the-fly* for discarding *redundant* and *deprecated* sequences. Moreover, we present an experiment to compare our technique against three state-of-the-art adaptive algorithms and evaluate how these sequences can hamper learning. By this experiment, we answer the following questions:

(RQ1) Is our on-the-fly adaptive technique more efficient than the state-of-the-art adaptive learning techniques?
(RQ2) Is the effectiveness of adaptive learning strongly affected by the temporal distance between versions?

In our experiment, we reused 18 Mealy machines from a large-scale analysis of several versions of the OpenSSL project [26], an open-source and commercial-grade cryptographic toolkit [23]. To answer RQ1, we used the number of membership queries (`MQ`) to learn models with a fixed level of accuracy as a measure of effectiveness. To answer RQ2, we used the temporal distance between versions, denoted by the difference between their release dates as measure of software evolution because structural changes (e.g., changed transitions) in black-box setting are often unknown and behavioral metrics (e.g., percentage of failed tests) may mislead minor modifications closer to initial states compared to major changes.

Based on our experiments, we find that our $\partial\mathtt{L}^*_\mathtt{M}$ algorithm presented weak positive correlation between `MQ`s and temporal distance. These findings support the answer to **RQ1** that our algorithm is less sensitive to model evolution and more efficient than the state-of-the-art adaptive algorithms. On the other hand, we found that existing adaptive algorithms can have a strong positive correlation between `MQ`s and the temporal distance between the SUL and reused version. These findings affirmatively answer **RQ2** for existing techniques; and corroborate previous studies [13] where the quality of the reused models is shown as a factor that affects adaptive learning.

Our contributions: (1) We introduce the $\partial\mathtt{L}^*_\mathtt{M}$ algorithm to mitigate the cost for re-inferring Mealy machines from evolving systems; and (2) We present an experiment comparing our proposal to three state-of-the-art adaptive learning algorithms [5,13] to show that the side-effects of *redundant* and *deprecated* sequences can be mitigated by exploring reused observation tables on-the-fly. To

date, there is a lack of studies about the pros and cons of adaptive learning and how much extra (and irrelevant) effort it may take if low-quality models are re-used, our second contribution provides essential insights to fill in this gap.

The remaining of this paper is organized as follows: In section 2, we introduce the concepts of finite state machines, model learning and adaptive learning; In section 3, we present the $\partial \mathtt{L}^*_{\mathtt{M}}$ algorithm; In section 4, we design an experiment to evaluate and analyze the effect of reuse and its correlation with model quality; and, In section 5, we draw some conclusions and pointout some avenues for future work. For the sake of reproducibility and repeatability, a lab package is available at https://github.com/damascenodiego/DynamicLstarM.

2 Background

In this section, we focus on model learning based on complete deterministic Mealy machines [4,27], hereafter called finite state machines (FSM). FSMs have been successfully used to learn models of hardware [8], software [13] and communication protocols at an abstract level [26].

2.1 Finite state machines

Definition 1. *(Complete Deterministic FSM) An FSM* $\mathcal{M} = \langle S, s_0, I, O, \delta, \lambda \rangle$ *is a 7-tuple where* S *is the finite set of states,* $s_0 \in S$ *is the initial state,* I *is the set of inputs,* O *is the set of outputs,* $\delta : S \times I \to S$ *is the transition function, and* $\lambda : S \times I \to O$ *is the output function.*

Initially, an FSM is in the initial state s_0. Given a current state $s_i \in S$, when a defined input $x \in I$, such that $(s_i, x) \in S \times I$, is applied, the FSM responds by moving to state $s_j = \delta(s_i, x)$ and producing output $y = \lambda(s_i, x)$. The concatenation of two inputs α and ω is denoted by $\alpha \cdot \omega$. An input sequence $\alpha = x_1 \cdot x_2 \cdot ... \cdot x_n \in I^*$ is defined in state $s \in S$ if there are states $s_1, s_2, ..., s_{n+1}$ such that $s = s_1$ and $\delta(s_i, x_i) = s_{i+1}$, for all $1 \leq i \leq n$. Transition and output functions are lifted to input sequences, as usual. For the empty input sequence ϵ, $\delta(s, \epsilon) = s$ and $\lambda(s, \epsilon) = \epsilon$. For a non-empty input sequence $\alpha \cdot x$ defined in state s, we have $\delta(s, \alpha \cdot x) = \delta(\delta(s, \alpha), x)$ and $\lambda(s, \alpha \cdot x) = \lambda(s, \alpha)\lambda(\delta(s, \alpha), x)$. An input sequence α is a prefix of β, denoted by $\alpha \leqslant \beta$, when $\beta = \alpha \cdot \omega$, for some sequence ω. An input sequence α is a proper prefix of β, denoted by $\alpha < \beta$, when $\beta = \alpha \cdot \omega$, for $\omega \neq \epsilon$. The prefixes of a set T are denoted by $pref(T) = \{\alpha | \exists \beta \in T, \alpha \leq \beta\}$. If $T = pref(T)$, it is *prefix-closed*.

An input sequence $\alpha \in I^*$ is a transfer sequence from s to s', if $\delta(s, \alpha) = s'$. An input sequence γ is a separating sequence for $s_i, s_j \in S$ if $\lambda(s_i, \gamma) \neq \lambda(s_j, \gamma)$. Two states $s_i, s_j \in S$ are equivalent if, for all $\alpha \in I^*$, $\lambda(s_i, \alpha) = \lambda'(s_j, \alpha)$, otherwise they are distinguishable. An FSM is *deterministic* if, for each state s_i and input x, there is at most one possible state $s_j = \delta(s_i, x)$ and output $y = \lambda(s_i, x)$. Notice that our definition only allows for complete deterministic FSMs, which are the focus of this paper. If all states of an FSM are pairwise distinguishable, then the FSM is *minimal*.

A set Q of input sequences is a *state cover* for $\mathcal{M}$ if $\epsilon \in Q$ and, for all $s_i \in S$, there is an $\alpha \in Q$ such that $\delta(s_0, \alpha) = s_i$. A set P of input sequences is a *transition cover* for $\mathcal{M}$ if $\epsilon \in P$ and, for all $(s, x) \in S \times I$, there are $\alpha, \alpha x \in P$, such that $\delta(s_0, \alpha) = s$, $x \in I$. A set W of input sequences is *characterization set* for $\mathcal{M}$ if for all $s_i, s_j \in S$ such that, with $s_i \neq s_j$, there is an $\alpha \in W$ where $\lambda(s_i, \alpha) \neq \lambda(s_j, \alpha)$.

Example 1. (The windscreen wiper FSM) Figure 1 depicts a windscreen wiper system supporting intervaled and fast wiping, if any raindrop is sensed, such that $S = \{off, itv, rain\}$, $I = \{rain, swItv\}$ and $O = \{0, 1\}$. Transition and output functions are represented by directed edges labeled with input/output symbols.

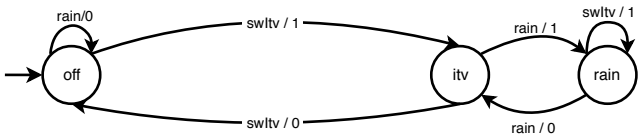

Fig. 1: The windscreen wiper FSM

2.2 Model learning

Coined by Dana Angluin [1], active model learning was originally designed to formulate a hypothesis $\mathcal{H}$ about the behavior of a SUL as a deterministic finite automaton (DFA). Model learning has been adapted to several notations [30] and is often described in terms of the Minimally Adequate Teacher (MAT).

In the MAT framework [30], there are two phases: (i) *hypothesis construction*, where a learning algorithm poses Membership Queries (`MQ`) to gain knowledge about the SUL using `reset` operations and input sequences; and (ii) *hypothesis validation*, where based on the model learnt so far a hypothesis $\mathcal{H}$ about the "language" of the SUL is formed and tested against the SUL using Equivalence Queries (`EQ`). The results of the queries are organized in an observation table that is iteratively refined and used to formulate $\mathcal{H}$.

Definition 2. *(Observation Table) An observation table* $\mathtt{OT} = (S_M, E_M, T_M)$ *is a triple, where* $S_M \subseteq I^*$ *is a prefix-closed set of transfer sequences (i.e., prefixes);* $E_M \subseteq I^+$ *is a set of separating sequences (i.e., suffixes); and* T_M *is a table where rows are labeled by elements from* $S_M \cup (S_M \cdot I)$*, columns are labeled by elements from* E_M*, such that for all* $pre \in S_M \cup (S_M \cdot I)$ *and* $suf \in E_M$*,* $T_M(pre, suf) = \lambda(\delta(q_0, pre), suf)$ *where* q_0 *is the initial state.*

Two rows $pre_1, pre_2 \in S_M \cup (S_M \cdot I)$ are equivalent, denoted by $pre_1 \cong pre_2$, when for all $suf \in E_M$ it holds that $T_M(pre_1, suf) = T_M(pre_2, suf)$. The equivalence class of a row r is denoted by $[r]$.

The $\mathtt{L}^*_\mathtt{M}$ algorithm Traditionally, the $\mathtt{L}^*_\mathtt{M}$ algorithm [27] starts with the sets of prefixes $S_M = \{\epsilon\}$ and suffixes $E_M = I$ so that it can reach the initial state and observe the outputs of the outgoing transitions, respectively. Afterwards, it poses `MQ`s until the properties of closedness and consistency hold:

Definition 3. *(Closedness property) An observation table* OT *is closed if for all* $pre_1 \in (S_M \cdot I)$*, there is a* $pre_2 \in S_M$ *where* $pre_1 \cong pre_2$*.*

Definition 4. *(Consistency property) An observation table* OT *is consistent if for all* $pre_1, pre_2 \in S_M$*, such that* $pre_1 \cong pre_2$*, it holds that* $pre_1 \cdot \alpha \cong pre_2 \cdot \alpha$*, for all* $\alpha \in I$*.*

If an observation table is not closed, the algorithm finds a row $s_1 \in S_M \cdot I$, such that $s_1 \ncong s_2$ for all $s_2 \in S_M$, moves it to S_M, and completes the observation table by asking MQs for the new rows. If the observation table is not consistent, the algorithm finds $s_1, s_2 \in S_M, e \in E_M, i \in I$, such that $s_1 \cong s_2$ but $T_M(s_1 \cdot i, e) \neq T_M(s_2 \cdot i, e)$, adds $i \cdot e$ to E_M, and completes the observation table by asking MQs for the new column. Given a *closed* and *consistent* observation table, the $\mathtt{L}^*_\mathtt{M}$ formulates a hypothesis $\mathcal{H} = (Q_M, q_{0_M}, I, O, \delta_M, \lambda_M)$ where $Q_M = \{[pre] | pre \in S_M\}$, $q_{0_M} = [\epsilon]$ and, for all $pre \in S_M$, $i \in I$, $\delta_M([pre], i) = [pre \cdot i]$ and $\lambda_M([pre], i) = T_M(pre, i)$.

After formulating $\mathcal{H}$, $\mathtt{L}^*_\mathtt{M}$ works under the assumption that an EQ can return either a counterexample (CE) exposing the non-conformance, or yes, if $\mathcal{H}$ is indeed equivalent to the SUL. When a CE is found, a CE processing method adds prefixes and/or suffixes to the OT and hence refines $\mathcal{H}$. The aforementioned steps are repeated until EQ = yes. For black-box systems, EQs are often approximated using random walks [1,12], conformance testing [6,32,9], or both [17,22].

Example 2. (OT from the windscreen wiper FSM) In Table 1, we show an observation table built using $\mathtt{L}^*_\mathtt{M}$, a CE = swItv · rain · rain · rain and the Rivest and Schapire method [25], that uses binary search to find the shortest suffix from CE that refines a hypothesis. The cost to build this OT is 24 MQs and 1 EQ.

Table 1: OT extracted from the windscreen wiper FSM

		rain	*swItv*	*rain · rain*
S_r	ϵ	0	1	0 · 0
	swItv	1	0	1 · 0
	swItv · rain	0	1	0 · 1
$S_r \cdot I_r$	*rain*	0	1	0 · 0
	swItv · swItv	0	1	0 · 0
	swItv · rain · rain	1	0	1 · 0
	swItv · rain · swItv	0	1	0 · 1

The worst-case complexity of the $\mathtt{L}^*_\mathtt{M}$ algorithm for the number of MQs is $\mathcal{O}(|I|^2 mn + |I| mn^2)$ parameterized on the size of the the input domain I, the length m of the longest CE and the number of states n of the minimal FSM describing the SUL. Motivated by the impact of CEs on the complexity of $\mathtt{L}^*_\mathtt{M}$, a wide range of processing methods are found in the literature [15]. Another important component for model learning is cache filter which can pre-process queries to eliminate redundancy [20].

2.3 Adaptive learning

Adaptive learning is a variant of model learning which attempts to speed up learning by reusing pre-existing models from previous/alternative versions [10].

In adaptive learning, transfer and/or separating sequences built from pre-existing models are used to initialize learning algorithms with sets of prefixes and suffixes (possibly) better than the traditional sets of sequences to reach the initial state (i.e., $S_M = \{\epsilon\}$) and collect outputs from outgoing transitions (i.e., $E_M = I$). Thus, a reduction on the number of MQs and EQs may be obtained. In this context, we are aware of four studies that address adaptive learning [10,35,13,5].

Groce, Peled & Yannakakis [10] introduce an approach where inaccurate (but not completely irrelevant) models are reused to reduce the time spent on model learning and model checking. They evaluate the benefits of reusing either transfer sequences, or separating sequences, or both; compared to *learning from scratch*. Their results indicate that adaptive learning is useful especially when modifications are minor or when they may have a very limited effect on the correctness of properties checked.

Windmüller, Neubauer, Steffen, Howar & Bauer [35] present an adaptive learning technique which periodically infers automata from evolving complex applications. Moreover, they show that learning algorithms which reuse separating sequences from models of previous versions are capable of finding states maintained in newer versions.

Huistra, Meijer & van de Pol [13] show that the benefits of adaptive learning are influenced by (i) the complexity of the SUL, (ii) differences between the reused version and the SUL, and (iii) the quality of the suffixes. Thus, if a set of reused separating sequences has bad quality (i.e., low ability to re-distinguish states), irrelevant queries may be posed, and hence extra effort will be required. To mitigate that, the authors suggest that calculating a subset of good separating sequence should provide a reduction on the number of deprecated sequences.

Chaki, Clarke, Sharygina & Sinha [5] introduce *DynamicCheck*, an approach to reduce the cost for model checking software upgrades. Central to their approach is Dynamic L*, an adaptive learning algorithm which reuses observation tables from previous versions for inferring DFAs [1]. As result, upgrade checking can succeed in a small fraction of the time to verify its reference version [5]. Next, we briefly present Dynamic L* in terms of Mealy machines.

Dynamic L^* Normally, $\mathtt{L}^*_{\mathtt{M}}$ starts with $S = \{\epsilon\}$, $E = I$ and, if there is any previously learnt model from some reference version v_{ref}, it misses opportunities for optimizing the learning process. To this end, Dynamic L* restores the *agreement* of an outdated table $\mathtt{OT}_o = (S_r, E_r, T_o)$ built from v_{ref} by re-posing MQs to the updated release v_{updt} to build an updated observation table $\mathtt{OT}_r = (S_r, E_r, T_r)$.

Definition 5. *(Agreement) An $\mathtt{OT}_r = (S_r, E_r, T_r)$ agrees with v_{updt} if and only if, for all $s \in (S_r \cup S_r \cdot I_u)$ and $e \in E_r$, it holds that $T_r(s, e) = \lambda_u(s, e)$, such that λ_u is the output function of v_{updt}.*

After restoring the agreement, the observation table $\mathtt{OT}_r$ may have redundant prefixes and deprecated suffixes. To discard them, the Dynamic L* searches for a smaller $S_R \subseteq S_r$ with the same state coverage capability but less prefixes, referred to as *well-formed cover* [5].

Definition 6. *(Well-Formed cover subset) Let S_r be the set of prefixes from an observation table $\mathtt{OT}_r$ in agreement with v_{updt}; a subset $S_R \subseteq S_r$ is well-formed cover, if and only if (i) S_R is prefix-closed, (ii) for all $s1, s2 \in S_R$, it holds that $s1 \not\supseteq s2$, and (iii) S_R is a maximal subset from S_r.*

After finding a $S_R \subseteq S_r$, we use a *Column function* to group prefixes into equivalence classes given a subset of suffixes. Thus, we search for an optimal subset of suffixes $E_R \subseteq E_r$, referred to as the *experiment cover* [5].

Definition 7. *(Column Function) Let S_R be well-formed cover, an observation table $\mathtt{OT}_{R'} = (S_R, E_r, T_{R'})$ derived from $\mathtt{OT}_r$, the input set I_u of v_{updt}, and an $e \in E_r$; the column function is $Col(S_R, e) : S_R \times E_r \rightarrow \{B_1, B_2, ..., B_n\}$ where B_i are non-empty partitions of S_R (i.e., $B_i \subseteq S_R$), $\cap_{i=1}^{n} B_i = \emptyset$, $\cup_{i=1}^{n} B_i = S_R$, $Col(S_R, \epsilon) = \{S_R\}$ and $x, y \in B_i$ if and only if $T(x, e) = T(y, e)$.*

An $E_R \subseteq E_r$ is an *experiment cover subset* iff for all distinct $e_1, e_2 \in E_R$, it holds that $Col(S_R, e_1) \neq Col(S_R, e_2)$ and for all $e' \in E_R$ there is an $e \in E_r$ where $Col(S_R, e) = Col(S_R, e')$. Finally, $\mathtt{L}^*_\mathtt{M}$ is initialized with the subsets S_R and E_R. Thus, the time for upgrade model checking can be reduced to a small fraction of the time needed to verify its reference version from the scratch [5].

3 The `partial-Dynamic` $\mathtt{L}^*_\mathtt{M}$ algorithm

For evolving systems, significant updates can lead adaptive learning to pose several irrelevant queries composed by redundant and deprecated sequences. Thus, calculating a "useful" subset of sequences should mitigate this risk [13]. To achieve this goal, we introduce `partial-Dynamic` $\mathtt{L}^*_\mathtt{M}$ ($\partial\mathtt{L}^*_\mathtt{M}$), an extension of the adaptive algorithm proposed in [5] and briefly described in Section 2.3. Our algorithm improves upon the state-of-the-art by exploring observation tables on-the-fly to avoid irrelevant queries rather than indiscriminately re-asking `MQ`s.

The term *partial* applies as the $\partial\mathtt{L}^*_\mathtt{M}$ algorithm copes with reused observation tables by exploring them using depth-first search (DFS) to find redundant prefixes; and breadth-first search (BFS) to find deprecated suffixes. Our $\partial\mathtt{L}^*_\mathtt{M}$ algorithm comprises three sequential steps which are discussed in the next subsections using two versions of the windscreen wiper as running examples.

Example 3. (Updated windscreen wiper) Let the FSMs in Figures 1 and 2 be the reference version v_{ref} and updated release v_{updt} of an evolving system, respectively. Added elements are shown in dotted lines.

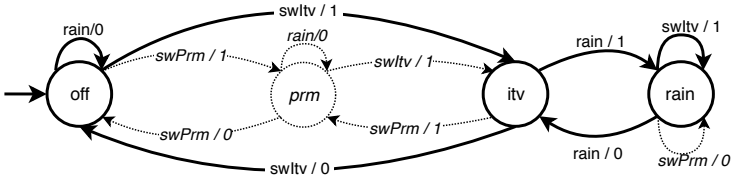

Fig. 2: Windscreen wiper with permanent movement

We refer to their representation as minimal FSMs and counterpart elements as $M_r = \langle Q_r, q_{0_r}, I_r, O_r, \delta_r, \lambda_r \rangle$ and $M_u = \langle Q_u, q_{0_u}, I_u, O_u, \delta_u, \lambda_u \rangle$.

3.1 (Step 1) On-the-fly exploration of the reused table

Let S_r and E_r be prefixes and suffixes of an observation table $\mathtt{OT}_r = (S_r, E_r, T_r)$. Since we do not know how the states may have changed, $\mathtt{OT}_r$ may be outdated, redundant prefixes may emerge from S_r and no longer reach the same states [13], respectively. Thus, an updated observation table $\mathtt{OT}_{R'} = (S_R, E_r, T_{R'})$ has to be built by restoring the agreement of $\mathtt{OT}_r$ to v_{updt}. To achieve this, instead of indiscriminately re-asking queries, we explore the tree representation of $S_r \cdot I_u$ on-the-fly using depth-first search (DFS) to pose `MQ`s and build an updated table $\mathtt{OT}_{R'}$. In this tree representation, paths leading from root to nodes represent elements from $S_r \cdot I_u$ and nodes are annotated using rows of the updated observation table $\mathtt{OT}_{R'}$. Thus, we can identify prefixes leading to states already discovered by E_r and find a well formed cover subset $S_R \subseteq S_r$.

Example 4. (Well-formed cover) Figure 3 shows parts of the tree representation of an $\mathtt{OT}_{R'}$ built from an outdated observation table with prefixes and suffixes $S_r = \{\epsilon,\ swItv,\ swItv \cdot rain,\ swItv \cdot rain \cdot rain,\ wItv \cdot rain \cdot rain \cdot swItv,\ rain\}$ and $E_r = \{rain,\ swItv,\ swPrm,\ rain \cdot rain\}$, respectively.

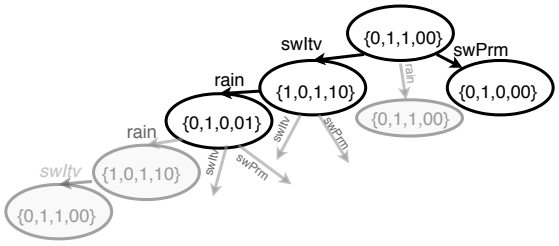

Fig. 3: Well-formed cover subset S_R generated from S_r

The well-formed cover subset is denoted by black arrows and discarded prefixes are in gray. The cost to find this well-formed cover subset is 40 `MQ`s, in contrast to 76 `MQ`s to completely restore the agreement of $\mathtt{OT}_r$ to v_{updt}.

3.2 (Step 2) Building an experiment cover tree

After finding $S_R \subseteq S_r$, we use the upper part from $\mathtt{OT}_{R'} = (S_R, E_r, T_{R'})$ to search for an experiment cover subset. An experiment cover subset $E_R \subseteq E_r$ can be obtained by finding the subsets of equivalent suffixes and picking a representative element from each set [5]. To achieve this, we propose an optimization technique that runs a breadth first search (BFS) on a tree representation of E_r, referred to as *experiment cover tree*, in a similar fashion to homing trees [4].

Definition 8. *(Experiment cover tree) Consider the updated observation table* $\mathtt{OT}_{R'} = (S_R, E_r, T_{R'})$ *and an input domain* I_u*; an experiment cover tree is a rooted tree that satisfies the following constraints:*

1. *The root node is labeled as* $lbl(root) = Col(S_R, \epsilon)$*;*
2. *Each edge* e *is labeled with one suffix* $e \in E_r$*;*
3. *Each node* n *linked to parent* n_p *by edge* e *is labeled as* $lbl(n) = Col(lbl(n_p), e)$*;*
4. *Non-leaf nodes* n *have outgoing edges for all suffixes* $E_r \backslash E_{R(n)}$*, where* $E_{R(n)}$ *is the set of suffixes labeling the edges in the path from root to* n*;*
5. *A node* n *is leaf iff*
   (a) *for all* $B_i \in lbl(n)$*,* $|B_i| = 1$*; or*
   (b) *there is a lower node* n_l *where* $lbl(n) = lbl(n_l)$*.*

The *experiment cover tree* is built using BFS and, if a node d satisfying *5a* is found, the suffixes labeling the path from *root* to d is returned as the experiment cover subset E_R. Otherwise, we traverse the whole experiment cover tree and the first node d found with maximum separating capability is selected as the E_R, i.e., $max(|Col(S_R, E_{R(d)})|)$. Neither `MQ`s nor `EQ`s are posed in this step.

Example 5. (Experiment cover) Figure 4 shows a fragment of the experiment cover tree generated from the subset S_R in Figure 3 and the set of suffixes $E_r = \{rain, swItv, swPrm, rain \cdot rain\}$. The subset E_R is highlighted in black.

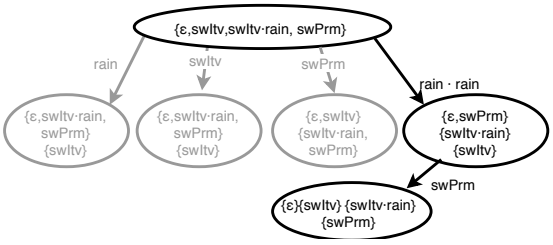

Fig. 4: Experiment cover tree

3.3 (Step 3) Running $\mathtt{L}^*_\mathtt{M}$ using the outcomes of $\partial\mathtt{L}^*_\mathtt{M}$

At this point, redundant prefixes and deprecated suffixes are discarded and we initialize the $\mathtt{L}^*_\mathtt{M}$ algorithm using the well-formed and experiment cover subsets, rather than $S_u = \{\epsilon\}$ and $E_u = I$. As results, we expect to build in the first iteration an observation table with higher state discovery capability at a reduced number of queries, especially if v_{ref} and v_{updt} are not drastically different.

Example 6. In Table 2, we summarize the number of `MQ`s and `EQ`s posed to the SULs in Figures 1 and 2 by five model learning algorithms: $\mathtt{L}^*_\mathtt{M}$; our $\partial\mathtt{L}^*_\mathtt{M}$; an adaptive approach, referred to as `Adp`, where $\mathtt{L}^*_\mathtt{M}$ starts with suffixes from a previous version [13]; and two straightforward implementations of the `Dynamic` $\mathtt{L}^*$ algorithm for Mealy machines, referred to as $\mathtt{DL}^*_\mathtt{M}$ and $\mathtt{DL}^*_{\mathtt{M}+}$. The latter differs by restoring the properties of closedness and consistency to avoid the loss of prefixes $s_1, s_2 \in S_R$ where $s_1 \cong s_2$ and $\exists(i,e) \in (I_u, E_r), T_{R'}(s_1 \cdot i, e) \neq T_{R'}(s_2 \cdot i, e)$. In this example, the $\partial\mathtt{L}^*_\mathtt{M}$ algorithm posed the lowest number of `MQ`s.

Table 2: Reuse approaches and number of queries

Algorithm	Reuse		Restore properties	SUL	OT	Number of	
	Prefixes	Suffixes				MQs	EQs
L_M^* [27]	-	-	-	v_{ref}	-	18	2
				v_{updt}	-	48	2
Adp [13]	No	Complete	No	v_{updt}	$\mathtt{OT}_r$	48	1
$\mathtt{DL}_\mathtt{M}^*$ [5]	Indiscriminate		No	v_{updt}	$\mathtt{OT}_r$	76	2
$\mathtt{DL}_{\mathtt{M}+}^*$			Yes	v_{updt}	$\mathtt{OT}_r$	81	1
$\partial\mathtt{L}_\mathtt{M}^*$	On-the-fly		No	v_{updt}	$\mathtt{OT}_r$	43	1

4 Empirical evaluation

According to Huistra et al. [13], the more the states of a SUL have been changed, the lower is the number of suffixes with good quality. Therefore, we expect a higher number of irrelevant queries from state-of-the-art adaptive learning algorithms when older versions are re-used.

4.1 Research questions

To evaluate adaptive learning in different settings, we extended the LearnLib framework [17] with the algorithms in Table 2. Thus, we investigated if our $\partial\mathtt{L}_\mathtt{M}^*$ algorithm is more efficient than three state-of-the-art adaptive algorithms (RQ1) and the impact of the temporal distance in their effectiveness (RQ2).

As a measure of software evolution, we opted for the temporal distance between the versions in terms of their release dates as structural changes (e.g., changed transitions) in black-box setting are often unknown; and behavioral metrics (e.g., percentage of failed test cases) may mislead minor modifications closer to initial states compared to major changes [34]. As a measure of effectiveness, we used the number of MQs and EQs posed by each adaptive learning algorithm compared to traditional learning using the $\mathtt{L}_\mathtt{M}^*$ algorithm [27].

In Table 3, we formulated hypotheses about the influence of the temporal distance on the number of queries, denoted by $\Delta\mathtt{T}$; and the average difference between the number of MQs and EQs posed by the adaptive learning and $\mathtt{L}_\mathtt{M}^*$, denoted by $\mu\mathtt{MQ}$ and $\mu\mathtt{EQ}$.

Table 3: Hypotheses

Measure	Hypotheses	Description
$\mu\mathtt{MQ}$	$H_0^{\mu\mathtt{MQ}}$	The $\partial\mathtt{L}_\mathtt{M}^*$ requires an equivalent $\mu\mathtt{MQ}$
	$H_1^{\mu\mathtt{MQ}}$	The $\partial\mathtt{L}_\mathtt{M}^*$ requires a higher $\mu\mathtt{MQ}$
	$H_2^{\mu\mathtt{MQ}}$	The $\partial\mathtt{L}_\mathtt{M}^*$ requires a lower $\mu\mathtt{MQ}$
$\mu\mathtt{EQ}$	$H_0^{\mu\mathtt{EQ}}$	The $\partial\mathtt{L}_\mathtt{M}^*$ requires an equivalent $\mu\mathtt{EQ}$
	$H_1^{\mu\mathtt{EQ}}$	The $\partial\mathtt{L}_\mathtt{M}^*$ requires a higher $\mu\mathtt{EQ}$
	$H_2^{\mu\mathtt{EQ}}$	The $\partial\mathtt{L}_\mathtt{M}^*$ requires a lower $\mu\mathtt{EQ}$
$\Delta\mathtt{T}$	$H_0^{\Delta\mathtt{T}}$	The $\partial\mathtt{L}_\mathtt{M}^*$ is influenced by the temporal distance
	$H_1^{\Delta\mathtt{T}}$	The $\partial\mathtt{L}_\mathtt{M}^*$ is not influenced by the temporal distance

For each scenario $\langle v_l, \texttt{OT}_\texttt{r}\rangle$, such that v_l is a SUL and $\texttt{OT}_\texttt{r}$ is an observation table built from a reference version v_r, we used the Mann-Whitney-Wilcoxon (MWW) to check if there was *statistical significance* ($p < 0.01$) between the number of queries posed by the adaptive algorithms. To measure the *scientific significance* of our results [18], we used the Vargha-Delaney's $\hat{A}_{c,t}$ effect size [31,36] to assess the probability of one algorithm outperforming another [2]. If $\hat{A}_{c,t} < 0.5$, then the treatment t poses more queries than, the control c. If $\hat{A}_{c,t} = 0.5$, they are equivalent. To categorize the magnitude, we used the intervals between $\hat{A}_{c,t}$ and 0.5 suggested by [11,29]: $0 \leq negligible < 0.147$, $0.147 \leq small < 0.33$, $0.33 \leq medium < 0.474$ or $0.474 \leq large \leq 0.5$. Finally, we used the Pearson's correlation coefficient to evaluate the relationship between the temporal distance between versions $\langle v_l, v_r\rangle$ to the numbers of MQs and EQs.

4.2 Subject Systems

As evolving system, we reused 18 Mealy machines learned in a large scale analysis of several versions of OpenSSL [26], an open-source and commercial-grade cryptographic toolkit [23]. In Figure 5, we depict over 14 years of development branches from the server-side of OpenSSL. SULs are denoted by white boxes with arrows pointing out to their previous release in the branch, the number of implemented states in parentheses, and behavioral overlaps (i.e., equivalent FSMs) are grouped by dashed areas.

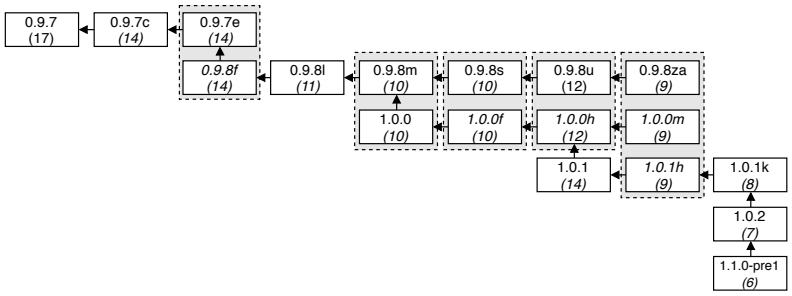

Fig. 5: OpenSSL server-side: 18 FSMs versions used as SUL

4.3 Experiment design

Let $\langle v_l, \texttt{OT}_r\rangle$ be a learning scenario where v_l is the SUL, and $\texttt{OT}_r$ is an observation table built from a reference version v_r. For all 18 versions and their precedents, we measured the difference between the numbers of MQs and EQs posed by each adaptive algorithm and $\texttt{L}^*_\texttt{M}$, such that positive values denote *adaptive learning posing more queries*; and their temporal distance in years. For each version v_r, we built 500 different $\texttt{OT}_\texttt{r} = (S_r, E_r, T_r)$ with prefix-closed state cover sets S_r created using randomized DFS and $E_r = I_u \cup W_r$, such that W_r is a W set for v_r; and calculated the μMQs and μEQs. For processing CE, we used the *Suffix1by1* [14], and the CLOSE_FIRST strategy to close tables [19]. To build EQs, we used the Wp method for conformance testing [9] with an upper bound $m = 2$.

4.4 Analysis of results

In this section, we analyze the μMQs and μEQs posed by the adaptive algorithms and their relationship to the temporal distance within $\langle v_l, v_r \rangle$. For the averages, we calculated the difference between the numbers of MQs and EQs posed by each adaptive algorithm and learning from scratch using the $\mathtt{L}^*_{\mathtt{M}}$ algorithm.

By analyzing the release dates for all versions [24], we found a strong positive correlation ($r = 0.72$) between the temporal distance and difference of numbers of states implemented in each pair of versions. These findings corroborate de Ruiter [26] findings that OpenSSL improved over time and recent versions were more succinct (i.e., had fewer states). Moreover, they also indicate that the OpenSSL server-side represents an evolving system that can pose interesting challenges to adaptive learning algorithms, e.g., the reuse of larger and older versions may impact the benefits of adaptive learning.

Average difference of EQs In Figure 6, we depict boxplots for the μEQs as a function of the temporal distance within $\langle v_l, v_r \rangle$. To keep the figure uncluttered, we calculated the boxplots for time windows of one year. Outliers are depicted as red dots.

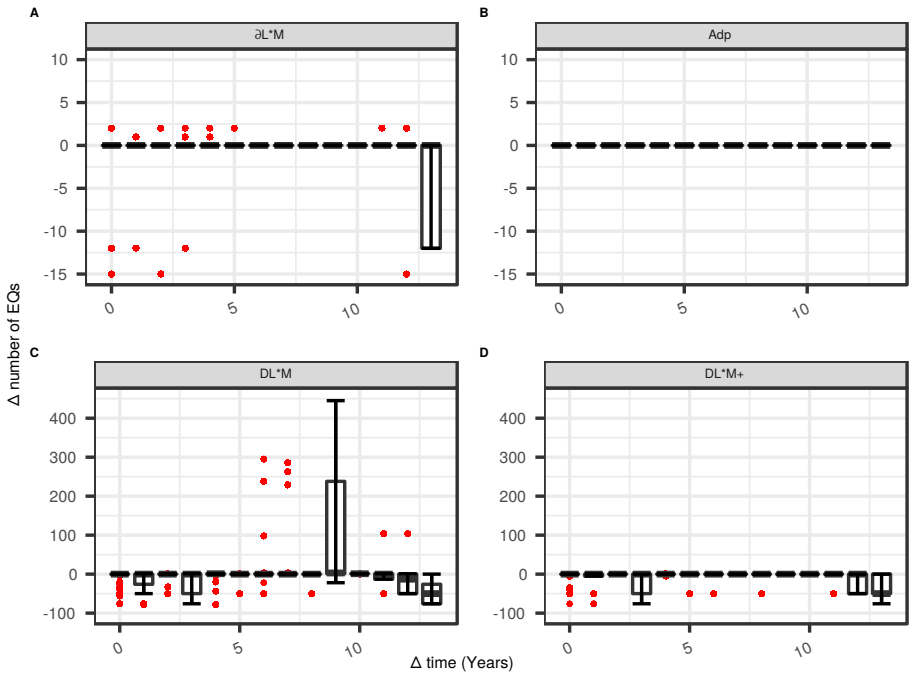

Fig. 6: Boxplots of the μEQs posed by adaptive and traditional learning

By analyzing the μEQs and the learnt observation tables, we found that I_u turned out to be a good set of separating sequences. The I_u set allowed to discover most of the states from OpenSSL and, since the reused observation tables included I_u, the resulting μEQs happened to be quite similar, and whiskers turned out to be very close to their boxes.

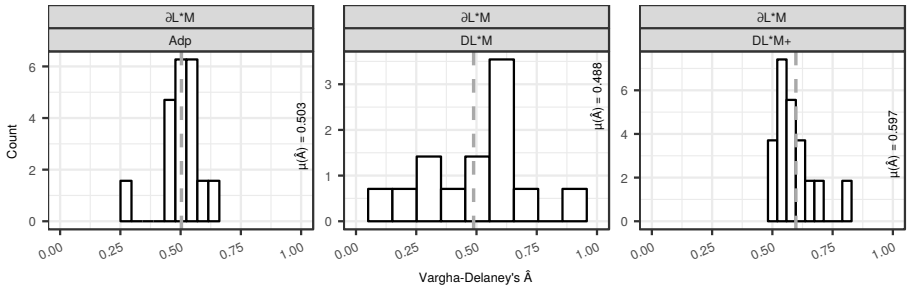

Fig. 7: Histograms of the effect sizes for EQs posed by the adaptive algorithms

In Figure 7, we show histograms to the effect size for EQs, where $\partial \mathtt{L}^*_{\mathtt{M}}$ is the control method, i.e., $\hat{A} < 0.5$ means that the $\partial \mathtt{L}^*_{\mathtt{M}}$ algorithm posed less queries than the other adaptive algorithm. The MWW test indicated a statistically significant difference ($p < 0.01$) between $\partial \mathtt{L}^*_{\mathtt{M}}$ and the other adaptive algorithms; however, as the histograms indicate, the effect sizes were mostly categorized as *negligible*.

Added to this, the Pearson's correlation coefficient indicated a *very weak* to *no correlation* between μEQs and temporal distance. The number of rounds also happened to be approximately the same, i.e., *one* round for all versions with less than 14 states and *two to five* for all other versions. These findings support the hypothesis $H_0^{\mu\mathtt{EQ}}$ that our adaptive learning algorithm $\partial \mathtt{L}^*_{\mathtt{M}}$ required an μEQs similar to traditional learning.

Average difference of MQs In Figure 8, we depict boxplots for the μMQs as a function of the temporal distance between v_l and v_r. To keep the figure uncluttered, we calculated one boxplot for each time window of a year.

By analyzing the μMQs, we found that Adp posed around 50 additional MQs when versions older than four years were reused. An increment on the μMQs also emerged for the $\mathtt{DL}^*_{\mathtt{M}}$ and $\mathtt{DL}^*_{\mathtt{M+}}$ where up to 800 extra MQs occurred to be required. Our results indicated more significant increments on the number of MQs when the temporal distance was maximum (i.e., 14 years).

For the existing adaptive learning algorithms, we found a *strong* to *very strong* correlation between μMQs and the temporal distance within $\langle v_l, v_r \rangle$. Thus, our findings corroborate to Huistra et al. [13] indicative that the quality of the reused sequences is a factor that can underpower adaptive learning. Consequently, the existing adaptive learning algorithms posed a large number of MQs composed by *redundant* prefixes and *deprecated* suffixes.

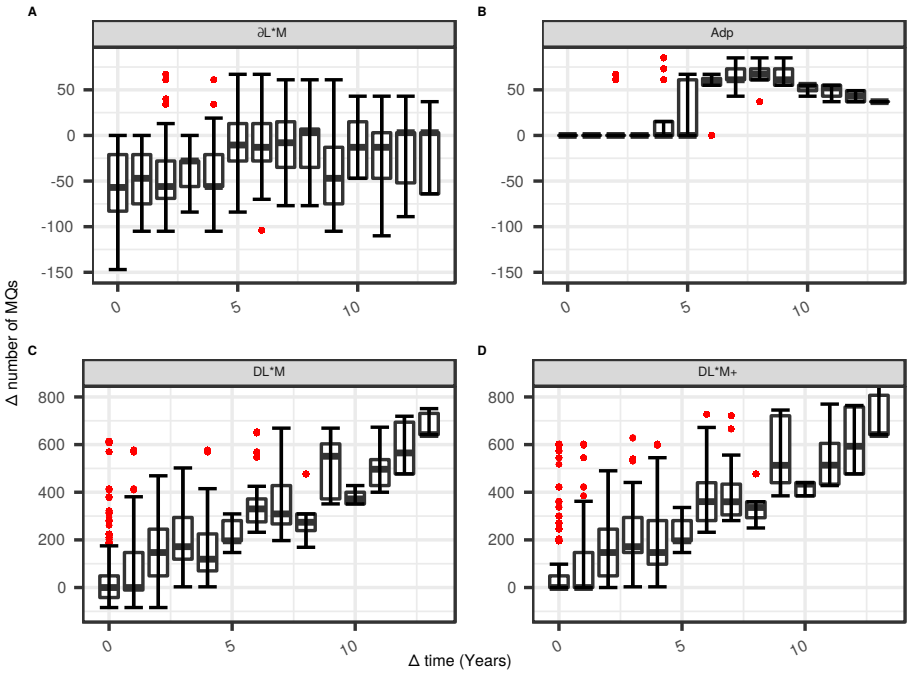

Fig. 8: Boxplots of the μMQs posed by adaptive and traditional learning

Differently from the state-of-the-art adaptive learning algorithms, our $\partial \mathrm{L}^*_{\mathrm{M}}$ algorithm turned out to be more robust than the other algorithms. In Figure 9, we show histograms for the effect sizes of the $\partial \mathrm{L}^*_{\mathrm{M}}$ algorithm compared to the three other adaptive techniques.

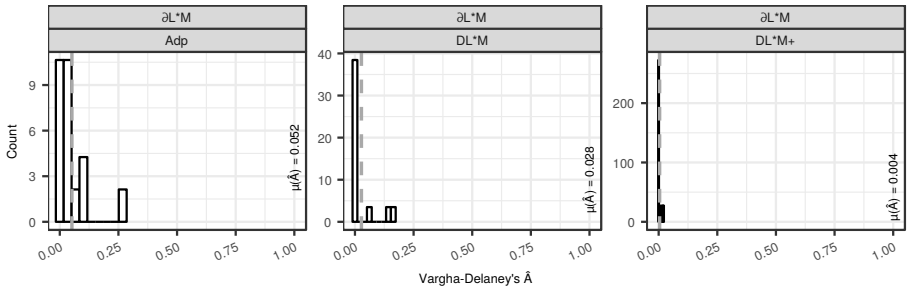

Fig. 9: Histograms of the effect sizes for MQs posed by the adaptive algorithms

For the $\partial \mathrm{L}^*_{\mathrm{M}}$ algorithm compared to the other algorithms, we found a significant difference ($p < 0.01$) on the number of MQs, and a *weak positive* correlation between the μMQs and temporal distance, with effect sizes mostly categorized as *large*. Thus, our results favoured the hypothesis $H_2^{\mu\mathrm{MQ}}$ that identifies $\partial \mathrm{L}^*_{\mathrm{M}}$ as the algorithm showing the lowest μEQs compared to the other adaptive techniques, and answered **RQ1** by placing our on-the-fly technique as more efficient than the existing adaptive algorithms in terms of MQs.

Benefits of adaptive learning vs. temporal distance For all the adaptive learning algorithms, the Pearson's correlation coefficients indicated *very weak* positive correlation (≤ 0.20) between number of EQs and temporal distance. These findings corroborate the mostly constant μEQ seen in Figure 6.

For the algorithms Adp, $\mathtt{DL}^*_{\mathtt{M}}$ and $\mathtt{DL}^*_{\mathtt{M}+}$, we observed the Pearson's coefficients indicated strong positive correlations (0.73, 0.78 and 0.80, respectively) between number of MQs and temporal distance. Alternatively, for our $\partial\mathtt{L}^*_{\mathtt{M}}$ algorithm, we observed weak positive correlation (0.33) between number of MQs and temporal distance. Thus, we confirm the hypothesis $H_1^{\Delta\tau}$ that the $\partial\mathtt{L}^*_{\mathtt{M}}$ algorithm is not influenced by the temporal distance between versions and affirmatively answer **RQ2** by showing that the effectiveness of existing adaptive learning techniques are indeed affected by the temporal distance between versions.

4.5 Threats to validity

Internal validity: Threats to internal validity concern with the influences that can affect the casual relationship between the treatment and outcomes. One element that forms a threat to internal validity is the temporal distance between versions as a measure of software evolution. We found a strong positive correlation ($r = 0.72$) between temporal distance and difference in the numbers of states of the underlying FSMs. Hence, we found the temporal distance as a reasonable measure, at least for this particular case. Failed test-cases may not be good measures, as they do not reflect the point of failure in the SUL semantics, e.g., minor changes close to initial states against major changes on the language.

External validity: Threats to external validity concern with generalization. To guarantee the reliability of our experiment results, we relied on the LearnLib framework [17] to implement adaptive learning algorithms. Our study is based essentially on the OpenSSL toolkit; and this poses a threat to external validity. However, since the OpenSSL has realistic size and structure, we believe that our findings are generalizable to other real systems. We plan to mitigate this by extending our study to other systems, such as software product lines [7].

5 Conclusions and future work

Real systems pass through many changes along their life-cycle and, as we often do not know how states may have changed, behavioral models tend to become outdated, incomplete, or even deprecated. To deal with these issues, several studies have proposed the application of active model learning to automatically derive behavioral models.

Particularly, adaptive model learning is a variant of active learning which has been studied to speed up model inference by reusing transfer and separating sequences from previously learnt observation tables. However, software evolution may degrade the quality of existing artifacts so that they may no longer distinguish states and compose *redundant* and *deprecated* sequences. To date, there is a limited number of studies about the pros and cons of adaptive model learning and how much extra effort it may pose if low-quality models are re-used.

To mitigate these problems, we introduced a novel adaptive algorithm that explores observation tables *on-the-fly* to avoid irrelevant MQs, called $\partial \mathtt{L}^*_{\mathtt{M}}$. Using 18 versions of the OpenSSL toolkit, we showed that state-of-the-art adaptive algorithms mostly show a strong positive correlation between the number of MQs and temporal distance between the reused and learnt versions.

Alternatively, our $\partial \mathtt{L}^*_{\mathtt{M}}$ algorithm presented a weak positive correlation between temporal distance and MQs. Thus, our algorithm turned out to be less sensitive to software evolution and more efficient than current approaches for adaptive learning. Also, $\partial \mathtt{L}^*_{\mathtt{M}}$ posed fewer MQs compared to three state-of-the-art adaptive learning algorithms. In this study, we move towards solving this issue by evaluating adaptive learning algorithms for evolving systems and how *redundant* and *deprecated* sequences can undermine the benefits of these algorithms.

As future work, we plan to investigate the problem of learning behavioral models of software product lines (SPL) [7]. Recently, we have investigated the problem of combining models from product families into a single representation, referred to as *family model* [7]. Family models differs from traditional notations by including propositional logic formulae to express the combination of features involved in the concerned states/transitions of the model [28]. In our investigations, we have found that although traditional model learning can be applied to SPLs in an exhaustive way, it can be prohibitive due to redundant queries performed over products sharing behavior and non-trivial due to differences in the input domains of their products. Thus, we believe that $\partial \mathtt{L}^*_{\mathtt{M}}$ can be improved for learning models of SPLs in a efficient and effective way. Another branch of future research forms to extending adaptive learning to *discrimination tree*-based algorithms, such as TTT [16]. These algorithms compose an interesting domain due to their redundancy-free nature and improved space complexity.

Acknowledgment

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, and University of Leicester, College of Science & Engineering. Research carried out using the computational resources of the Center for Mathematical Sciences Applied to Industry (CeMEAI) funded by FAPESP (grant 2013/07375-0). The authors are also grateful to the anonymous reviewers; the *VALidation and Verification* (VALVE) research group at the University of Leicester; and *Laboratory of Software Engineering* (LabES) at the University of Sao Paulo (ICMC-USP) for their insightful comments and suggestions.

References

1. Angluin, D.: Learning regular sets from queries and counterexamples. vol. 75, pp. 87 – 106 (1987)
2. Arcuri, A., Briand, L.: A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: Proceedings of the 33rd International Conference on Software Engineering. pp. 1–10. ICSE '11, ACM, NY, USA (2011)

3. Binder, R.V.: Testing Object-oriented Systems: Models, Patterns, and Tools. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1999)
4. Broy, M., Jonsson, B., Katoen, J.P., Leucker, M., Pretschner, A.: Part I. Testing of Finite State Machines. Springer, Berlin, Heidelberg (2005)
5. Chaki, S., Clarke, E., Sharygina, N., Sinha, N.: Verification of evolving software via component substitutability analysis. Formal Methods in System Design **32**(3), 235–266 (Jun 2008)
6. Chow, T.S.: Testing software design modeled by finite-state machines. IEEE Transactions on Software Engineering **SE-4**(3), 178–187 (May 1978)
7. Damasceno, C.D.N., Mousavi, M.R., da Silva Simao, A.: Learning from difference: An automated approach for learning family models from software product lines. In: Proceeedings of the 23rd International Systems and Software Product Line Conference - Volume 1, SPLC 2019. ACM Press, Paris, France (2019)
8. al Duhaiby, O., Mooij, A., van Wezep, H., Groote, J.F.: Pitfalls in applying model learning to industrial legacy software. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice. pp. 121–138. Springer International Publishing (2018)
9. Fujiwara, S., v. Bochmann, G., Khendek, F., Amalou, M., Ghedamsi, A.: Test selection based on finite state models. IEEE Transactions on Software Engineering **17**(6), 591–603 (Jun 1991)
10. Groce, A., Peled, D., Yannakakis, M.: Adaptive model checking. In: Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 357–370. TACAS '02, Springer-Verlag, Berlin, Heidelberg (2002)
11. Hess, M.R., Kromrey, J.D.: Robust confidence intervals for effect sizes: A comparative study of cohen's d and cliffs delta under non-normality and heterogeneous variances. In: Annual meeting - American Educational Research Association (2004)
12. Howar, F., Steffen, B., Merten, M.: From ZULU to RERS, pp. 687–704. Springer, Berlin, Heidelberg (2010)
13. Huistra, D., Meijer, J., Pol, J.: Adaptive learning for learn-based regression testing. In: Howar, F., Barnat, J. (eds.) Formal Methods for Industrial Critical Systems. pp. 162–177. Lecture Notes in Computer Science, Springer, Switzerland (9 2018)
14. Irfan, M.N., Oriat, C., Groz, R.: Angluin style finite state machine inference with non-optimal counterexamples. In: Proceedings of the First International Workshop on Model Inference In Testing. pp. 11–19. MIIT '10, ACM, New York, NY, USA (2010)
15. Irfan, M.N., Oriat, C., Groz, R.: Chapter 3 - model inference and testing. Advances in Computers, vol. 89, pp. 89 – 139. Elsevier (2013)
16. Isberner, M., Howar, F., Steffen, B.: The TTT algorithm: A redundancy-free approach to active automata learning. In: Bonakdarpour, B., Smolka, S.A. (eds.) Runtime Verification. pp. 307–322. Springer International Publishing (2014)
17. Isberner, M., Howar, F., Steffen, B.: The Open-Source LearnLib, pp. 487–495. Springer International Publishing (2015)
18. Kampenes, V.B., Dyb, T., Hannay, J.E., Sjberg, D.I.: A systematic review of effect size in software engineering experiments. Information and Software Technology **49**(11), 1073 – 1086 (2007)
19. LearnLib: LearnLib 0.13 - Javadoc. http://learnlib.github.io/learnlib/maven-site/0.13.0/apidocs/ (2018), [Online; accessed 06-Aug-2018]
20. Margaria, T., Raffelt, H., Steffen, B.: Knowledge-based relevance filtering for efficient system-level test-based model generation. Innovations in Systems and Software Engineering **1**(2), 147–156 (Sep 2005)

21. Mariani, L., Pezz, M., Zuddas, D.: Chapter four - recent advances in automatic black-box testing. Advances in Computers, vol. 99, pp. 157 – 193. Elsevier (2015)
22. Meinke, K., Sindhu, M.A.: Incremental learning-based testing for reactive systems. In: Gogolla, M., Wolff, B. (eds.) Tests and Proofs. pp. 134–151. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
23. OpenSSL Foundation, Inc.: OpenSSL - Cryptography and SSL/TLS Toolkit. https://www.openssl.org/ (2018), [Online; accessed 21-Ago-2018]
24. OpenSSL Foundation, Inc.: OpenSSL Releases at Github. https://github.com/openssl/openssl/releases (2018), [Online; accessed 26-Ago-2018]
25. Rivest, R.L., Schapire, R.E.: Inference of finite automata using homing sequences. Information and Computation **103**(2), 299 – 347 (1993)
26. de Ruiter, J.: A tale of the openssl state machine: A large-scale black-box analysis. In: Secure IT Systems - 21st Nordic Conference, NordSec 2016, Oulu, Finland, November 2-4, 2016, Proceedings. pp. 169–184 (2016)
27. Shahbaz, M., Groz, R.: Inferring mealy machines. In: Cavalcanti, A., Dams, D.R. (eds.) FM 2009: Formal Methods. pp. 207–222. Springer, Berlin, Heidelberg (2009)
28. Thüm, T., Apel, S., Kästner, C., Schaefer, I., Saake, G.: A classification and survey of analysis strategies for software product lines. ACM Comput. Surv. **47** (Jun 2014)
29. Torchiano, M.: effsize: Efficient Effect Size Computation (v. 0.7.1). CRAN package repository (3 2017), https://cran.r-project.org/web/packages/effsize/effsize.pdf [Online; accessed 20-November-2017]
30. Vaandrager, F.: Model learning. Commun. ACM **60**(2), 86–95 (Jan 2017)
31. Vargha, A., Delaney, H.D.: A critique and improvement of the CL common language effect size statistics of McGraw and Wong. Journal of Educational and Behavioral Statistics **25**(2), 101–132 (2000)
32. Vasilevskii, M.P.: Failure diagnosis of automata. Cybernetics **9**(4), 653–665 (Jul 1973)
33. Walkinshaw, N.: Chapter 1 - reverse-engineering software behavior. In: Memon, A. (ed.) Advances in Computers, Advances in Computers, vol. 91. Elsevier (2013)
34. Walkinshaw, N., Bogdanov, K.: Automated comparison of state-based software models in terms of their language and structure. ACM Transactions on Software Engineering and Methodology **22**(2), 1–37 (Mar 2013)
35. Windmüller, S., Neubauer, J., Steffen, B., Howar, F., Bauer, O.: Active continuous quality control. In: Proceedings of the 16th International ACM Sigsoft Symposium on Component-based Software Engineering. pp. 111–120. CBSE '13, ACM, New York, NY, USA (2013)
36. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Systematic Literature Reviews, pp. 45–54. Springer, Berlin, Heidelberg (2012)