Adaptive Learning for Learn-Based Regression Testing

David Huistra*, Jeroen Meijer*, and Jaco van de Pol†

Formal Methods and Tools, University of Twente, the Netherlands
{d.j.huistra, j.j.g.meijer, j.c.vandepol}@utwente.nl

Abstract. Regression testing is an important activity to prevent the introduction of regressions into software updates. Learn-based testing can be used to automatically check new versions of a system for regressions on a system level. This is done by learning a model of the system and model checking this model for system property violations.
Learning the model of a large system can take an unpractical amount of time however. In this work we investigate if the concept of adaptive learning can improve the learning speed of a model in a regression testing scenario.
We have performed several experiments with this technique on two systems: ToDoMVC and SSH. We find that there can be a large benefit to using adaptive learning. In addition we find three main factors that influence the benefit of adaptive learning. There are however also some shortcomings to adaptive learning that should be investigated further.

1 Introduction

Successful software systems are often continuously updated throughout their life cycle [1]. Updates to the system often extend or alter the functionality. These changes occasionally unintentionally alter the behavior of existing functionality. This is what we call a regression.

In order to detect regressions, it is important to test from them [2]. Regressions can occur at many different levels of functionality, such as unit or system level.

In practice, regression testing is mostly performed on unit level. Here each code unit is tested independently. Unit testing techniques enjoy a lot of popularity, as it has proven to be an efficient way to identify regressions and it can be automated to test each version of a system [3].

In this work we focus on a testing technique for system level testing called Learn-Based Testing (LBT) [4]. The LBT testing technique is based on model checking and capable of identifying different type of regressions than unit testing. In addition it can also be automated to test each software version for system level regressions.

Importance of automated testing. To understand why automated testing is important for regression testing, it is important to understand the nature of regression testing. In regression testing there are often only a small number of regressions to

* Supported by STW SUMBAT grant: 13859
† Supported by the 3TU.BSR project

be found compared to the amount of functionality that is being tested. Therefore, regression testing often requires a big effort to find only a few regressions. In practice, this means that for many testing techniques, the effort required to apply them is not considered worth the possible reward.

This is however where automated testing stands out. Another aspect of regression testing is that it tends to happen periodically. A lot of versions of the system will all need to be tested for the same regressions. Unit testing can be setup to take advantage of this fact and automatically test all versions in the same manner. The initial setup of writing all unit tests will still require quite a bit of effort, but after this it can be used to test each version with minimal manual effort.

Our hypothesis is that LBT has the same advantages as unit testing, but enables regression testing at the system level. The purpose of this paper is to investigate the use of LBT in the context of regression testing.

How LBT works The core concept of LBT is to learn a behavioural model of the system. Such a model describes how the system reacts to sequences of inputs. Using such a model the system can then be tested for regressions.

Identifying regressions is done by determining if (the model of) a system adheres to a set of predefined system properties. This can be performed automatically by giving a set of properties and a model to a model checker such as LTSmin [5].

Interaction bottleneck Learning the behavioural model of a system can be performed automatically. A learning algorithm will interact with the system by performing sequences of actions and observing the outputs. Given a set of input-output combinations a model hypothesis can be constructed.

However, depending on the size of a system (i.e. the amount of interaction required) and how fast interaction with the system is, learning a model can take a significant amount of time. For larger systems the learning time can make the approach unpractical.

To combat this issue, there is an active area of research on the topic of reducing the amount of interaction required with the system. There have been a number of techniques proposed that can be used to reduce the amount of interaction required, such as better learning algorithms or caching mechanisms.

Adaptive learning When learning a model in the context of regression testing however, there is a specific technique that we believe can aid in reducing the time required to learn a model. We call this technique adaptive learning [6].

When performing regression testing on a system, in all but the first testing of the system, there is a previous regression test of the same system. In the previous regression testing of the system, the model of a previous version of the system was already learned. In most cases, the previous system is very similar to the updated system in terms of behavior. Therefore, the models of these systems will likely also be very similar.

With adaptive learning we want to reuse information about the system learned during the previous test to speed up the new test. Conceptually this is done by 'adapting' the existing model to the updated system.

Our contribution There is however little known about the effectiveness of this technique. Therefore we wanted to study how much benefit can be gained from using adaptive learning when learning the model of a system in a regression testing context.

In this work we setup an experiment to determine the benefit of adaptive learning when learning a system in a regression testing context. We also discuss several factors we found that influence the benefit of adaptive learning.

We find that in the right situations there can be a large benefit to using adaptive learning. There are however also still some shortcomings that should be investigated further.

Outline In Section 2 we first give more background information about the adaptive learning technique and learning the model of a system in general. In Section 3 we then discuss the experiments we performed with adaptive learning and show the outcome. We discuss the main factors that influence the benefit of adaptive learning that we identified in Section 4. In the discussion Section 5 we discuss the shortcomings of adaptive learning and the experiments and propose what should be done to improve upon this work. Finally in Section 6 we conclude this work by summarizing our findings.

2 Background

In this section we want to further explain the technique of adaptive learning. Before we do that however, we first introduce the reader with the concept of automatically learning a behavioral model of a system called active automata learning.

2.1 Active automata learning

In active automata learning, a learning algorithm is given a set of actions it can perform and asked to produce a model that describes the behaviour of a system [7]. It does this by interacting with the system through the set of actions it has been given and observing the outputs. Based on this interaction it will try to determine what states there are in the system and what the result is of applying each action in each state. With this information it will then construct a model hypothesis.

It is difficult for the learner to determine if it has identified all possible states or not. In principle it can keep on trying all possible sequences of actions, but this does not scale very well. Therefore the learning algorithm is designed to interact with the system until it has found a consistent set of observations and then produce a minimal model hypothesis.

To determine if the learner has identified all possible states, the model hypothesis is then given to a so called teacher. The teacher will determine if the hypothesis is correct or not. If the hypothesis is not correct, it will return its findings to the learner so the leaner can improve the model.

There are different ways to implement the learner and the teacher. These different implementations also influence the benefit that can be gained from using adaptive learning. In the next paragraph we discuss the different implementation's

of a learning algorithm. In the paragraph following that, we discuss the different teacher implementations.

Learning algorithms In general the learning algorithms work by constructing an observation table while interacting with the system. The rows of an observation table are (possible) access sequences to the different states of the system that have been discovered. The columns of an observation table are separating sequences that are used to distinguish states from each other. The leaner will fill this observation table by interacting with the system.

When the learner believes it is necessary to make the set of observations consistent, the learner will add access and/or separating sequences to the table. When the observations in the table are considered consistent, the learner will than construct a hypothesis model. If the learner receives a counterexample back from the teacher it will add this observation to the table and extend the observation table to make it consistent for all observations.

When and how separating sequences are added to the observation table depends on the specific learner implementation. We look at two implementations: L* and R&S.

- The idea of L* [8] is that it will try to learn as much from a counterexample as possible. It will also add all prefixes of the counterexample to the observation. By doing this it may find more new states and avoid work of the teacher, but it will require more interaction with the system to fill the observation table.
- R&S [9] works differently because it will only add a minimal version of the counterexample to the observation table. This keeps the observation table small but it may result in more work for the teacher in the future.

In addition to observation table based learners, there are also discrimination-tree based learners. These learners are however not yet compatible with adaptive learning, as is discussed in the next section.

Teacher Algorithms The teacher algorithm is given a model hypothesis and asked to determine if this hypothesis is correct. It does this by attempting to find a counterexample, a sequence of actions that produces a different result in the system compared to in the model. It will try a large set of sequences to see if they are a counterexample. If it cannot find a counterexample, it will determine that the hypothesis is correct.

What sequences and how many sequences teacher will try depends on the specific implementation. We distinguish between two: the WMethod and RandomWord method.

- The WMethod [10] is an FSM testing method which requires that an upper bound on the number of states is known and systematically tries to find a difference between an hypothesis and a system.
- RandomWord method. The RandomWord algorithm will generate a random set of sequences that it will try out on the system. The amount and length of the

sequences is given by the user. If the output of the system deviates from the system for one of these sequences, a counterexample is found. Otherwise the model is finalized.

2.2 Adaptive Learning

A learning algorithm will iteratively try discover all states of a system by extending the access and separating sequences. Once it is able to distinguish between all states using those sequences, it can fill the observation table and construct a model hypothesis.

A large amount of the learning effort goes into discovering all the states of a system. But in a regression testing scenario, an updated version of a system will generally still have most of the states of the previous version. Adaptive learning attempts to reuse knowledge about the states of a system from a previously learned model. This should reduce the amount of effort that goes into the discovery of the states.

We do not know for certain however if all states still exist. The question is therefore; how can we give a learner information about the possible states of a system, even though these state might not exist anymore?

In related work there have been two techniques proposed to steer the learning using an older model.

The first is the approach called Adaptive Model Checking by Groce et al. [6]. Their approach is based on calculating the access and separating sequences from an existing model. This information is then added to the observation table before the learner starts interacting with the system. After this, the learner will then proceed as normal by fulling the observation table until the observations are consistent and then generate a model hypothesis and giving this to the teacher.

The second approach is part of the Active Continuous Quality Control approach by Windmüller et al. [11]. They key idea of this approach is to extract the set of separating sequences from the old observation table and add these to the table of the new learner, and then proceed as normal.

They have found that this approach works well for the R&S learning algorithm suggested by Rivest & Schapire [9]. In this learning algorithm each counterexample is used to extend the set of separating sequences with exactly one element. Therefore, this approach in essence reuses all counterexamples found during the learning of the previous model.

Windmüller et al. also describe why the separating sequences discovered while learning the previous model can be reused to learn the new model. The separating sequences are used by a learner to distinguish between states. A learner will initially start with a minimal set of separating sequences and add sequences to this set if it discovers it can otherwise not distinguish between two states. If a new learner reuses these separating sequences, it will directly be capable of distinguishing between states. Even if the system has been changed and a sequence no longer helps to distinguish two states, the new observations will show this and a correct model will be constructed.

The second approach seems more suitable for regression testing, as it can directly extract information from the observation table of a learner.

2.3 The role of separating sequences.

In order to understand how much adaptive learning can help to reduce the interaction need to learn a model, it is important to better understand the role of separating sequences when learning a model. In this section we give more insight into separating sequences.

The role of separating sequences is to steer the observations the learner makes when learning a model. Initially a learning algorithm does not know what observations to make. It is only given a set of actions it can perform.

Learning algorithms such as R&S will therefore try to develop a minimal viable hypothesis. They will perform a minimal amount of interaction such as performing each action once. If the observations are consistent with each other, it will immediately produce a model hypothesis, otherwise it will keep adding observations until they are consistent. If it cannot distinguish possible states from one another with the observed outputs, the learner will merge these states.

The learner will then ask the teacher for a counterexample. When the learner receives a counterexample, the learner can learn what sequence of actions distinguishes two states from one another. It will add this sequence to the set of separating sequences and perform this sequence in all possible discovered states to determine if it can distinguish two states from one another.

With each separating sequences, the learner learns what observations it should make in order to identify more unique states. And the more states it discovers, the more accurate the model becomes.

Therefore, the set of separating sequences tells the learning algorithm what sequences it should try in the possible states it has discovered to determine if the states can be distinguished from one another.

2.4 Example

In this section we attempt to illustrate the background information through an example of learning a model. In this example we learn a simple system with just two actions: a and b. The system is shown in figure 1. Performing action a and receiving output z is denoted as a/z.

We use the R&S learner and RandomWord teacher in this example.

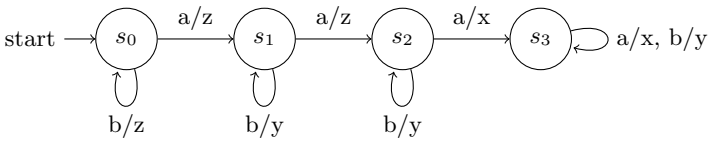

Fig. 1: Model of the system learned in the example.

Iteration 0 Input: The user has to specify a system and alphabet of the system that should be learned.

Step 1: The first step of the learner is to process the input. This is the alphabet of the system. In our example *a* and *b*. Based on this the learner initializes *access sequences* ← *[a,b]* and *separating sequences* ← *[]*.

Step 2: Then the learner starts filling the observation table with each combination of a access sequence and a separating sequence element. In the first iteration it makes only two observations: *a/z* & *b/z*.

Step 3: Given this filled consistent table, the learner then constructs a model hypothesis. The hypothesis is shown in Figure 2. Based on these observations alone, it can only identify one unique state.

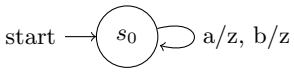

Fig. 2: Model hypothesis 1

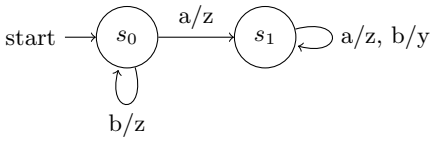

Fig. 3: Model hypothesis 2

Iteration 1 Input: The model hypothesis of iteration 0 is processed by a teacher that attempts to find a counterexample. In our example the teacher finds the following counterexample sequence: *a/z,b/y*.

Step 1: The learner starts with processing the given counterexample. From this counterexample it determines that it should make more observations. It adds action *b* to the set of separating sequences, as performing *b* separates two possible states from one another.

Step 2: The learner will extend the observation table using the updated separating sequence set. It will fill the observation tables with combinations of access sequence and separating sequence elements. It identifies that the sequence of action *a* transitions the system into a unique state. Therefore, the sequences *a,a* and *a,b* are added to the set of access sequences of possible states.

Step 3: Once the observation table is filled it constructs the hypothesis shown in Figure 3

Iteration 2 Input: The teacher will find another counterexample in the model hypothesis. This time *a/z,a/z,a/x*

Step 1: The learner will process this counterexample and identify that the sequence *a,a* can be used to identify inconsistent behavior. Therefore *a,a* will be added to the set of separating sequences.

Step 2 & 3: We skip the details, but the observation table is extended, new states are identified and new access sequences are added. The resulting hypothesis matches the model shown in Figure 1.

Output: This time the teacher does not find a counterexample. Therefore the leaner will return the model hypothesis as the final hypothesis to the user.

Adaptive learning Now we learn the same system once again, but this time we use adaptive learning.

Input: This time the input is the alphabet of the system and a set of separating sequences from the previous learner: *alphabet* ← *[a,b]* and *old separating sequences* ← *[b,(a,a)]*.

Step 1: Once again the learner will use the alphabet to initialize the access sequence. But this time the learner will set the separating sequences to the old separating sequences.

Step 2: When the learner starts filling the observation table, it needs to makes a lot more observations. Using these observations however it can iteratively identify new unique states and add the corresponding access sequences while filling the observation table. While filling the observation table it identifies all possible states in the same iteration.

Step 3: The constructed model hypothesis in the first iteration is equivalent to the system shown in in Figure 1.

3 Experiments

As stated previously, the main practical bottleneck for using learn-based testing is the learning time of a model. Our goal is to determine how much adaptive learning can help to reduce the amount of interaction required to learn a model. In other words, when learning the model of a system, is it more efficient to adapt a (similar) model or to learn a model from scratch?

There is no definitive answer to this question. It depends on the situation. For example, how similar the model to adapt is to the system that is being learnt.

We are however specifically interested to determine the benefit of adaptive learning in the context of regression testing. Here we assume that iterative versions of a system will all need to be learnt to be checked for regressions. This means that for each version of the system that needs to be learnt, the model of a relatively similar previous version was already learnt and that model can be adapted. The result is that models to adapt from are often very similar to the model that is being learnt, which is an optimal scenario for adaptive learning.

To determine the benefit of adaptive learning in a regression testing context in practice, we set up two experiments to compare the performance of adaptive learning to regular learning. In the experiment we learn multiple versions of a system through both adaptive and regular learning and compare the interaction required to learn those versions.

In this section we discuss the setup, procedure and results of this experiment.

3.1 The setup

When studying the concept of adaptive learning and during the experimentation we found several factors that influence the benefit of adaptive learning. These factors

will be discussed further in the next section. We however wanted to take these factors into account in the experiment in order to get a good estimation of the benefits of adaptive learning in general. In this section we discuss how the experiment was setup to produce a good estimation of the benefit of adaptive learning in a general regression setting context.

Chosen systems The specific systems that are learnt have a large impact on the interaction required to learn its model. Therefore experiments were performed on two different systems. We chose systems already learnt in related work to build upon those efforts and show the benefit of adaptive learning. In addition, the learnt models of these systems are publicly available. This allowed us to perform the experiments on a simulated version of the real system. We created a simulator that simulates a system's behavior based on a given model. This simulator made it much faster to perform experiments compared to working with a real system. In addition, this allowed us to create several versions of the implementation by creating different models that where given to the simulator.

The chosen systems are as follows:

- **ToDoMVC**: ToDoMVC[1] is a project that contains a large number of implementations of a standardized set of functionality but implemented using different frameworks/libraries. The main goal is to compare these frameworks/libraries with one another. Bainczyk and Schieweck [12] have learnt the model of a large number of the implementations and shown that they do not all produce the same functionality.
- **SSH**: Models of SSH implementations where previously learnt by Fiterău-Broştean et al. [13] in order to verify these systems using a list of system properties. These system properties were also available and provides a nice template that can be used for model checking different versions after their model is learnt, and gives a good indication as to what type of changes between versions should be detected. We focused our experiments on the DropBear[2] implementation.

Learning parameters Two account for and determine the influence of learning parameters on the benefit of adaptive learning, the experiments where performed with different combination of learning parameters. L* and R&S were used as the learning algorithms, as these are the two main observation table based learners and our adaptive learning approach is developed for those. For the teacher algorithms WMethod and RandomWord where used. See the background section for more information. While there are many variations of these learners and teachers, we found that these four where a good representation of the different behaviour we saw during experimentation.

Multiple versions The difference between two versions of a system can vary. A new version can be a code-refactoring where only the underlying code is changed but the functionality remains the same, or a new version can change a large part of

[1] http://todomvc.com/
[2] https://matt.ucc.asn.au/dropbear/dropbear.html

the functionality. When the difference between two versions varies, the benefit of adaptive learning also varies.

In order to take this into account, as well as to determine how much the difference between two versions influences the benefit of adaptive learning, the experiment is based on learning multiple versions of a system with varying degrees of difference to one another. The details are discussed in the next sections.

Measurements The experiments focus only on the interaction required to learn a model in different situations. The models are not actually checked for regressions, as this is not relevant for measuring the benefit of adaptive learning.

The interaction required to learn a model is measured by the number of queries that have to be processed by the system. We measure both the learning and equivalence queries.

We do not count the queries used to test the final hypothesis for counterexamples. This is a fixed number for each learning experiment and is not relevant when comparing the two approaches. In addition, this number depends on user settings and a hypothesis can already be checked for regressions while the hypothesis is still searched for counterexamples.

3.2 Learning ToDoMVC

In the ToDoMVC experiment we wanted to determine the benefit of adaptive learning in the optimal situation. In the optimal situation, adaptive learning is used to learn a model that is unchanged from the previously learnt model. In this experiment we therefore learn the same system twice, once with regular learning and once with adaptive learning.

We also look at the influence of the different learner and teacher algorithms. We look at the benefit of adaptive learning for all combinations of the L* and R&S learner and the WMethod and RandomWord teacher algorithms. The results of the experiment can be found in Table 1. We discuss these results in Section 3.4

	Regular Learning		Adaptive Learning	
Parameters	Learner	Teacher	Learner	Teacher
L* + WMethod	2.534	1.944	1.634	0
L* + RandomWord	19.215	3	1.743	0
R&S + WMethod	549	2.037	544	0
R&S + RandomWord	337	2	326	0

Table 1: Interaction queries needed to learn ToDoMVC with different learning parameters

3.3 Learning SSH

For this experiment we manually created several versions of the SSH program with varying degrees of differences between those versions. The experiment is based on

learning these different versions of the program by adapting a model of the base system and comparing this to learning from start.

We performed the experiments with the L* and R&S learning algorithms and the WMethod equivalence oracle. The RandomWord oracle was not able to find sufficient counterexamples within 10 million attempts.

In the following we discuss the versions of the program we created and how we performed the experiment on that version.

Base system The base system is the system learnt by BroStein *et al*[13]. This system is used to learn the initial model without adaptive learning.

Version 1: The first version of SSH that was created is functionally equivalent to the base system. This is for example the case when non-functional changes have been introduced, such as code refactoring or styling adjustments. Even with such changes a system should be tested for regressions, to make sure that the functionality did not change. This is an optimal situation for adaptive learning, as the model will not need to be adapted at all. The model only needs to be verified as correct.

Version 2: The second version of DropBear is a system that introduces a regression into the system. We created a version that contains a property violation according to the LTL formulae specified by BroStein at al.

Version 3: The third version of SSH is a system that introduces a special type of new functionality to the system. Here an action needs to be performed twice in order to proceed with a key-reset, which should require an additional separating sequence to identify the new state.

The results of this experiment can be seen in Table 2.

		Regular Learning		Adaptive Learning	
System	Parameters	Leaner	Teacher	Learner	Teacher
Version 1	L*	15.311	605.534	9.071	0
	R&S	5.310	618.868	5.291	0
Version 2	L*	15.623	503.978	9.071	0
	R&S	5.309	5666.240	5.291	0
Version 3	L*	15.911	604.617	10.749	42.356
	R&S	6.081	1.290.732	6.061	42.356

Table 2: Interaction queries needed to learn SSH with different learning parameters

3.4 Discussion

In the ToDoMVC experiment we see that there is a benefit to using adaptive learning with all combinations of learning parameters. However, the benefit reduces when the required teacher queries using traditional learning is reduced. We can summarize the findings as following:

1. When using RandomWord, the effort required to find all separating sequences for ToDoMVC is very small. Therefore little effort can be saved by using adaptive learning.

2. RandomWord produces very long counterexamples. This results in L* creating a large observation table. L* benefits from the shorter counterexamples produced by WMethod, while R&S is better capable of processing large counterexamples.

When learning the first version of SSH with regular learning, we saw that L* and R&S performed similar. Both require around 620.000 queries to learn the base system, although R&S required significantly less learning queries. In both cases adapting a model requires significantly less queries than learning a model from scratch.

L* however requires almost twice as much learning queries as R&S. We believe this comes from the fact that the L* learner produces more distinguishing suffixes and thus larger observation tables. Simply filling the observation table of a L* learner requires significantly more queries.

For the learning of version 2 we see the same results as for learning version 1. This indicates that even though a bug has been introduced in version 2, this version of the system can be learnt with the same distinguishing suffixes as the base system. Therefore the learner only needs to fill the observation table to learn the model of this version.

When learning version 3 we see that the learner needs to find additional separating sequences. The effort required to identify the additional sequences is however significantly smaller than finding all of them.

4 A theory of reuse

As discussed in Section 2, a learning algorithm needs a set of separating sequences to determine what observations it should make to identify and distinguish the states of a system. By reusing an existing set of separating sequences discovered while learning a similar model, adaptive learning aims to reduce the interaction needed to discover the set of separating sequences.

The goal of this research effort is to determine how much interaction can be avoided by using adaptive learning. To this end we performed an experiment to compare adaptive learning to regular learning. We however also identified three main factors that determine the benefit of adaptive learning. In this section we discuss those factors.

4.1 Discovery

The main factor that determines the benefit of adaptive learning is the amount of interaction required to discover a set of separating sequences in the model. The difficulty of discovering a set of separating sequences depends mainly on the behaviour of a system that is being learned and partially on the method used to discover separating sequences.

We can see this when looking at the differences between the experiments on ToDoMVC and SSH. SSH requires a lot more interaction to discover the set of separating sequence, while ToDoMVC showed the difference the teacher algorithm can have on the effort required to discover a set of separating sequences.

The effort that goes into the discovery of a set of separating sequences is a combination of the following two aspects:

1. The number of suffixes that need to be discovered
2. The effort to discover a suffix

The amount of suffixes required A separating suffix is used to distinguish two states from one another. The number of separating sequences required is therefore the amount of pairs of states that need to be distinguished between, subtracting the pairs that can reuse the same suffix.

The number of separating sequences is therefore related to the specific behaviour in a system.

Discovering a suffix The discovery of suffixes is performed by a teacher algorithm. Given a hypothesis, the teacher will attempt to find two states that should be distinguished from each other. It does this by finding a sequence of actions that shows these states have a different behavior/output, i.e. a counterexample.

The teacher algorithm tries to find such a sequences by simply trying (random) sequences of actions on the system. The amount of sequences that can be tried and the percentage of sequences that result in a counterexample however depend on the system.

The amount of possible sequences of a certain length is simply the number of actions to the power of the length of the sequence. Therefore the amount of possible actions significantly increases the average effort required to find a counterexample. The required length of counterexamples and the percentage of sequences that produce a counterexample depend on the behaviour of the system.

For example, a system that resets to the initial state when a wrong action is performed requires a precise set of sequence of actions to reach certain states, thus the percentage of sequences that are a counterexample is reduced. In the experiments we saw for example that SSH resets back to the initial state when a wrong actions is performed, therefore it required very specific sequences of actions to reach certain states. This is also the reason that WMethod did not perform well for SSH.

4.2 Reuse

Depending on the change between two versions of a system, the amount of distinguishing suffixes that can be reused and the amount of new suffixes that need to be discovered varies on how much the behaviour of the system changed.

Generally, the more the states of a system have been altered, the amount of suffixes that can be reused is reduced.

However, we believe that generally the difference in behaviour between two versions of a system is minor. Therefore in most situations there should be a high amount of distinguishing suffixes that can be reused.

When creating versions of SSH we noticed that many small changes did not require the discovery of additional separating sequences and we had to purposefully make changes that would trigger this need.

4.3 Quality

The third factor that influences the benefit of adaptive learning is the quality of the set of separating sequences. The separating sequences guide the learner in what observations it should make. With a bad quality of separating sequences however, the learner can make a large amount of observations that does not assist in identifying new states.

Generally a learner will attempt to find a sufficient set of separating sequences. It will generally however not find the minimal set of separating sequences.

An example of this is the L* learner. This learner will add a large number of sequences to the set. Not all of these sequences are required. Therefore, this learner will generally make more observations than required.

This can be seen in the ToDoMVC experiment. With adaptive learning, the L* learner required almost twice as much queries compared to the R&S learner. The reason for this is the large set of separating sequences that L* creates.

5 Discussion & Future Work

Our experiments confirm that adaptive learning improves the LBT approach for regression testing: the number of queries needed to learn the adapted system is significantly lower than the number of queries needed to learn a system from scratch.

We also explained the factors that influence this gain. The reusability of the learnt distinguishing suffixes depends on the complexity of the base system, the difference with the updated system, and the quality of the set of suffixes.

These observations lead to two potential improvements that can be studied in future work:

- **Discrimination Tree based learners**: The approach for adaptive learning used in our work is based on observation tree-based learners such as L* and R&S. More recent learners are based on a discrimination tree and have shown to be more efficient in the learning queries they require to create a hypothesis. Therefore, we believe that an adaptive learning approach should be developed for discrimination tree-based learners.
- **Calculate optimal set of separating sequences**: The quality of the set of separating sequences identified while learning a model can vary. Instead of using a set that is discovered during the learning of a model we can also calculate a set of separating sequences on a given model. An approach to do this was proposed by Smetsers et al. [14]. This operation can be performed in between learning two models. It should provide a better-quality set and also remove sequences if they are no longer required.

6 Conclusion

In the experiments we have seen that adaptive learning can reduce the interaction required to learn the model compared to regular learning. This is especially the case when changes between models are small, such as a regression testing context.

The benefits of adaptive learning can vary a great deal however. We have identified three main factors that influence the benefit of adaptive learning. The first two of these are the specific behavior of the system that is being learned and the amount of change between two versions of a system. These two factors can be used to determine if adaptive learning should be applied when learn-based regression testing a specific system. If the system needs a lot of difficult to find separating sequences and the changes between versions are small, then adaptive learning can provide a large benefit.

The third factor is the quality of the separating sequences and how they are used. We have seen that the learning parameters have a large impact on this. They determine what separating sequences are identified and how they are used to make observations. We have also discussed two ways in which the quality of the separating sequences can be improved.

References

1. Marvin V. Zelkowitz. Perspectives in Software Engineering. *ACM Comput. Surv.*, 10(2):197–216, 1978.
2. W. Eric Wong, Joseph R. Horgan, et al. A study of effective regression testing in practice. In *ISSRE, Albuquerque, NM, USA, November 2-5*, pages 264–274, 1997.
3. Michael Olan. Unit testing: test early, test often. *Journal of Computing Sciences in Colleges*, 19(2):319–328, 2003.
4. Karl Meinke. Automated black-box testing of functional correctness using function approximation. In *ISSTA, Boston, MA, USA, July 11-14*, pages 143–153, 2004.
5. Gijs Kant, Alfons Laarman, et al. LTSmin: High-Performance Language-Independent Model Checking. In *TACAS, London, UK, April 11-18.*, pages 692–707, 2015.
6. Alex Groce, Doron A. Peled, and Mihalis Yannakakis. Adaptive Model Checking. *Logic Journal of the IGPL*, 14(5):729–744, 2006.
7. Bernhard Steffen et al. Introduction to Active Automata Learning from a Practical Perspective. In *SFM, Bertinoro, Italy, June 13-18. Adv. Lect.*, pages 256–296, 2011.
8. Dana Angluin. Learning Regular Sets from Queries and Counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.
9. Ronald L. Rivest and Robert E. Schapire. Inference of Finite Automata Using Homing Sequences. In *Machine Learning: From Theory to Applications - Cooperative Research at Siemens and MIT*, pages 51–73, 1993.
10. Tsun S. Chow. Testing Software Design Modeled by Finite-State Machines. *IEEE Trans. Software Eng.*, 4(3):178–187, 1978.
11. Stephan Windmüller, Johannes Neubauer, et al. Active continuous quality control. In *CBSE, Vancouver, BC, Canada, June 17-21*, pages 111–120, 2013.
12. Alexander Bainczyk, Alexander Schieweck, et al. Model-Based Testing Without Models: The TodoMVC Case Study. In *ModelEd, TestEd, TrustEd - Essays Dedicated to Ed Brinksma on the Occasion of His 60th Birthday*, pages 125–144, 2017.
13. Paul Fiterau-Brostean et al. Model learning and model checking of SSH implementations. In *SPIN, Santa Barbara, CA, USA, July 10-14*, pages 142–151, 2017.
14. Rick Smetsers, Joshua Moerman, et al. Minimal Separating Sequences for All Pairs of States. In *LATA, Prague, Czech Republic, March 14-18*, pages 181–193, 2016.