Ensemble techniques for (P)DFA learning

Effect of changing the sequence orders on DFA ensembles learned via EDSM

Wiktor Cupiał[1]

Supervisors: Sicco Verwer[1]**, Simon Dieck**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Wiktor Cupiał
Final project course: CSE3000 Research Project
Thesis committee: Sicco Verwer, Simon Dieck, Merve Gürel

Abstract. Learning a Deterministic Finite Automaton (DFA) from a language sample is an essential problem in grammatical inference, with applications in various fields, such as modeling and analyzing software systems. In this work, we propose approaches to create an ensemble of DFAs learned with the Evidence Driven State Merging algorithm. To produce varying models from the given data, we introduce two algorithms for manipulating the sequence orders. Additionally, we propose a similarity metric that allows for reducing the ensemble size by discarding similar models. The proposed approaches were analyzed and empirically evaluated using the dataset used during the StaMinA competition. Experimental results demonstrate that the methods for obtaining ensembles of DFAs presented in this work provide a number of advantages over the single DFA learned from the classical prefix tree acceptor using EDSM.

1 Introduction

Deterministic Finite Automaton (DFA) is a fundamental concept in computer science. DFA serves as a powerful tool for modelling complex computer systems [1], attack graphs [2], and much more. Another strength of this model is the fact that it not only enables formal reasoning over the system, but it can also be a great source of visualization. These traits combined make it a powerful and desirable representation. However, it is often impractical to derive and maintain such models [1]. That is why learning the automaton from a given language sample is a problem of great importance. However, as stated by Gold [3], producing the minimal DFA consistent with a language sample is an NP-hard problem. To address this issue, there exist heuristics that aim to learn the model by employing the state merging technique [4]. Those methods start by constructing a structure called a prefix tree acceptor. Then, nodes in the tree are iteratively merged, aiming to produce the smallest automaton consistent with the given language sample. However, performing an incorrect merge can result in a much inferior model, since each merge creates constraints for future merges [5]. One of the methods commonly used for selecting the order of merges is the Evidence Driven State Merging (EDSM) algorithm introduced by Lang et al. [5]. It operates under the assumption that the merges towards which there is the most evidence should be performed first. Evidence in this context is the number of states merged with the same label. However, it does not solve the issue of inferior merges completely.

To address this issue, the ensemble method [6] could be employed. In short, the ensemble technique aims to create a collection of hypotheses. The combination of their results is used to achieve better results. Although ensembles are a well-established method in the field of machine learning, they have not yet been explored in the context of DFA learning. This work aims to explore the use of ensemble methods in combination with the EDSM algorithm. Given the multitude of methods suitable for creating ensembles of DFAs, this work specifically focuses on approaches that modify the sequence order. The goal of modifying the sequence order before running the EDSM algorithm is to potentially highlight different aspects of data and thus inject some variety into the models produced by it.

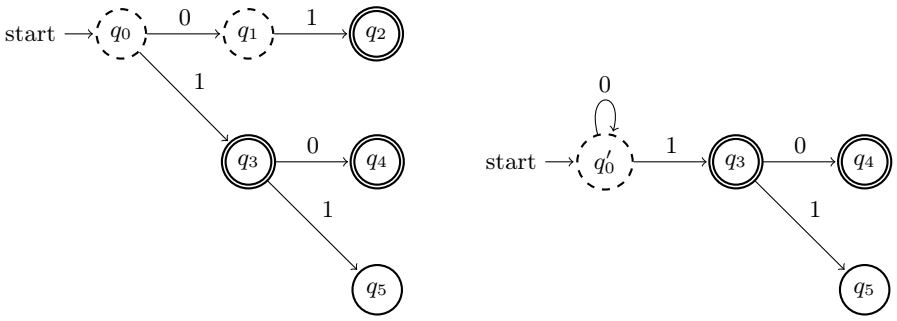
In this paper, we analyze two different approaches to manipulating the sequence order. Additionally, we provide a metric for measuring how different two DFAs are, which is used to control the diversity inside the ensemble. All these methods are evaluated against a DFA learned from a prefix tree acceptor using the EDSM algorithm. The dataset that was used in those experiments comes from the StaMinA competition [7]. We show that constructing an ensemble of DFAs can offer a better classification of unseen traces than a single DFA.

Additionally, we show how the number of models in the ensemble can be greatly reduced by using a similarity metric.

The paper is organized as follows. In section 2 we provide the necessary background, together with the motivation for this work. Section 3 showcases the evaluation metric that we used and the baseline approach. Additionally, it present the main contribution in the form of two methods for obtaining the ensembles of DFA, following with the showcase of the similarity metric used. Section 4 discusses the dataset and software package used for the testing. Following with section 5, which showcases the results obtained during the work. Section 6 discusses the ethical considerations surrounding the work, finishing with the conclusions and future research topics in section 7.

2 Background

In general, the problem of finding a minimal DFA consistent with the language sample is proven to be NP-hard [3]. The typical heuristic used to approximate a DFA from a language sample is state merging. State merging starts by building the prefix tree acceptor from the available traces. Then the nodes are iteratively merged, as long as there are merges available. To better explain the process of merging, follow the example. Given a pair of sets: $S_+ = \{01, 1, 10\}$, set of positive samples, and $S_- = \{11\}$, set of negative samples, we construct the prefix tree acceptor showcased in the figure 1a.



(a) Initial prefix tree acceptor. (b) Automaton after the merge.

Fig. 1: Example of performing a merge.

To merge two states p and q, we construct a new state r such that all transitions previously targeting p or q are redirected to r, and all transitions originating from p or q now originate from r. If at least one of the nodes p or q had a label (rejecting or accepting), we set the same label to r, otherwise, it remains unlabeled. The merge is only possible if no accepting node is merged with a rejecting one. As an example of such an operation, the automaton pictured in the figure 1b is the result of merging nodes q_0 and q_1.

Since merges introduce new constraints on future ones, we must use a specific ordering scheme to determine which ones should be performed first. One of the common heuristics to select the merges is the Evidence Driven State Merging (EDSM), introduced by Lang et al. [5]. It scores the merges by counting the number of nodes with the same label

merged. However, this greedy process relies mostly on local evidence and thus can overlook some deeper patterns in the data by following a suboptimal merge.

Suffix-based probabilistic finite automaton (S-PDFA) introduced by Nadeem et al. [2] works in a similar way to the previously mentioned models, but instead of constructing a prefix tree and then learning from it, they start by constructing a suffix tree. This approach has one crucial advantage over the classical prefix tree: it brings the suffix patterns into the spotlight of the state merging algorithm [2]. This information is crucial, since it allows us to formulate the hypothesis that by modifying the sequence orders we can indeed generate different, and possibly more accurate models.

To combat the risk of following the bad merge and consequently producing a poor-performing model, we need a better strategy. Ensemble is a well-known and established technique in the machine learning community [6]. It works by taking multiple models that span the hypothesis space. These models are then combined by means of, for example, majority voting or weighted majority voting to obtain the final answer. As long as the error rates of the individual models are less than $\frac{1}{2}$, and the errors are uncorrelated, such an algorithm yields better accuracy [6]. However, despite all of their advantages, ensemble methods were not studied in depth in the context of DFA learning.

The objective of this study is to present and evaluate two methods to obtain ensembles of DFAs by changing the sequence orders. The motivation for changing the sequence orders comes from the previous success in replacing the traditional prefix tree with a suffix tree. However, the problem with this approach is that if we do not have deeper insight into the data, it is difficult to decide which model would perform better. The idea is to create a multitude of models, where each highlights different parts of the data and constructs an ensemble out of them.

3 Methodology

This section outlines the approach adopted to investigate and validate our proposed solution. We begin by presenting the dataset used to measure the effectiveness of our method. Next, we introduce the baseline model. Following this, we present our approach for measuring the inter-model variety. In the last two subsections, we detail the main contribution of this paper.

3.1 Evaluation

The StaMinA dataset consists of 100 different sets. Unfortunately, the official test sets were never published. To address this issue, we decided to perform an 85-15 split, where 15% of the dataset is the test data. The dataset was obtained from the GitHub repository of the FlexFringe library, since it is already converted to the Abbadingo One format used by it. The dataset consists of 100 independent language samples with varying alphabet sizes and sparsity levels. Furthermore, the dataset can be divided into 5 parts distinguished by their alphabet size: 2, 5, 10, 20, 50. Inside each part, the sets follow the rule that the sparsity increases, which often means that they are more difficult to infer.

Instead of measuring the accuracy by the ratio of correctly classified labels, we decided to use the harmonic Balanced Classification Rate (BCR) measure [7]. This is done for two major reasons. Firstly, as stated in the StaMinA paper, if a model happens to be biased in the same way as the test data is, the standard accuracy measure is unreliable [7]. The second reason is that we decided to evaluate the proposed algorithms on the StaMinA dataset, thus,

it is fitting to use the same measure that was used during the competition. The formula (2) contains the exact BCR formula used. In this context, *FP* stands for *False Positive*, *TN* stands for *True Negative*, and so on.

$$SE = \frac{|TP|}{|TP \cup FN|}, \; SP = \frac{|TN|}{|TN \cup FP|} \tag{1}$$

$$BCR = \frac{2 \cdot SE \cdot SP}{SE + SP} \tag{2}$$

3.2 Baseline

To evaluate proposed approaches, we chose the dataset from the StaMinA competition [7]. As stated by Walkinshaw et al. [7], the competition was aimed at producing a reliable method for the evaluation of DFA learning techniques, which perfectly suits our application. As the baseline approach, we also chose EDSM starting with a prefix tree acceptor. This was done for two reasons. First, it is the same method that was used during the competition. Additionally, since the models in the ensemble are inferred using EDSM, it provides a good measure of whether the proposed approaches improve on a single method learned with the same method.

3.3 Changing the Reading Direction

As highlighted by Nadeem et al. [2], certain applications greatly benefit from the choice of a suffix tree instead of a prefix tree. To extend on that idea, we propose the following approach. Instead of reading the sequence from one direction from start to finish, we propose switching the direction after some predefined number of steps. The motivation for this approach is the following. If the underlying structure of the data is unknown, it becomes challenging to make assumptions that would favor the prefix tree over the suffix tree, or vice versa.

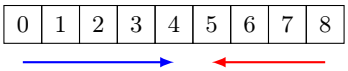


Fig. 2: An example reading order achieved with this method.

The figure 2 pictures the example order achieved on a trace of length 9. The trace is read by following the blue arrow first, and then switching to the red one. This results in an order starting from a prefix and reversing after $k = 5$ steps. Note that we could also start by following the red arrow, which would result in an order starting at the suffix and reversing the direction after $k = 4$ steps.

To construct an ensemble, we propose sampling without replacement from all the possible transformations of this form. If we were to create more models than there are in the space, we would just allow for the duplication and continue with random sampling. However, certain transformations applied to the traces may confuse the model, leading to decreased performance. We suggest using a validation set to select the most appropriate models for the final ensemble. Following that, the chosen models are retained on the complete training set.

3.4 Changing Substring Reading Direction

To improve the method mentioned in subsection 3.3 in some areas, we propose the following changes. First, it is possible that the state merging algorithm does not benefit from highlighting the prefix or suffix of the trace. However, it might be possible that it contains an important control flow mechanism in the middle. In that case, the model would likely benefit from highlighting it or its connections.

To approach this problem, we propose reversing the substring of the trace. Specifically, we select indices i and j, such that $i < j$, and then read the trace in order, with the substring being read in the opposite direction.

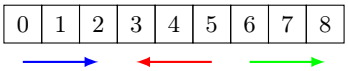


Fig. 3: An example processing order achieved with this method.

The figure 2 shows the example order achieved on a trace of length 9. The indices chosen for this transformation are 3 and 5. The trace is processed in the following order based on the arrow colors: blue, red, and green. Notice that the substring spanning the indices 3 to 5 is read in the opposite direction.

Similarly to the subsection 3.3, to construct the ensemble, we do random sampling without replacement on the transformation space. We apply the selected transformation to the data. Following that, we use the validation set to pick the fixed number of best models for the ensemble.

3.5 Similarity Metric

Since in the model we presented, each automaton has exactly one vote, we want to control how diverse the ensemble is. To maintain the diversity inside the ensemble, we would want a metric that could judge how similar two models are. To achieve that, we propose a method that tries to compute a Jaccard-like index. Given two DFAs p and q, that recognize languages $L(p)$ and $L(q)$, we want to compute $\frac{L(p) \cap L(q)}{L(p) \cup L(q)}$. We start with approximating the intersection by randomly sampling from the languages via means of a random walk. During the random walk, we randomly choose between all transitions with equal probability. While in an accepting state v, we end the walk with the probability: $\frac{1}{1+2 \cdot \text{outdegree}(v)}$. The formula used to terminate the walk is the same as that used to generate samples for the StaMinA competition [7]. Having sampled from the $S \subseteq L(p)$ and $T \subseteq L(q)$, to now compute the result, we need to unapply the transformation made to them. Remember that both automata are constructed on modified samples. Then, we apply transformations and we use q to predict from S and p to predict from T. With these predictions, we can now calculate the score, defined as the fraction of labels that they agree upon, referred to as $Sim(p, q)$. Notice that if the models agree upon a label, the word is in the $L(p) \cap L(q)$, and the set of all words collected with the random walk is the $L(p) \cup L(q)$. Ultimately, we obtained a metric that closely resembles the Jaccard index.

To now use the obtained metric for ensemble construction, we propose the following algorithm. Instead of picking a fixed number of the best-performing models, we pick them one by one and add them to the ensemble. However, we only add model p if there is no

model q in the ensemble, such that: $Sim(p, q) \geq 0.9$. This way, the ensemble will not have duplicate models, substantially lowering the average ensemble size. However, this method greatly increases the computation cost of finding the ensemble. Not only does one have to perform nw random walks, where n is the number of models in the ensemble, and w is the number of walks per model. Each time the model is being tested, we have to compare it to the $\mathcal{O}(n)$ different models that are already in the ensemble. This means that the similarity metric is computed $\mathcal{O}(n^2)$ times.

4 Experimental Setup

In this section, we present the experimental setup for the work, starting with an overview of the datasets used and subsequently introducing the software package that was used to obtain the results.

The dataset that we decided to use was obtained from the StaMinA competition [7]. The competition was held to test and compare numerous approaches for DFA learning. It was aimed as an empirically sound basis for comparison of grammatical inference techniques [7]. The test data that was used in the competition was crafted to resemble real-life systems. Unfortunately, since only training data was used during the competition was published, to evaluate proposed approaches, a train-test split on the published data was performed. We opted for an 85-15 split, with the additional condition that none of the sequences appeared in both the training and test data. To obtain the validation set needed for the two proposed methods, we performed an additional split to obtain 70-15-15 validation split. Important insight about the StaMinA data set is that it consists of the 5 different blocks. Each of them has a different alphabet size, and inside a block, the sparsity increases with increasing test number.

FlexFringe [1] is a software library that contains many different methods for DFA learning. Most notably, it implements the EDSM algorithm, which is used throughout this work. To obtain models for different reading strategies, we preprocess the training data and then follow with a FlexFringe run of the EDSM algorithm. To later get the prediction from the model built on the modified prefix tree acceptor, we apply the same preprocessing to the input data, followed by a FlexFringe run to obtain the predicted labels.

5 Results

This section presents the results collected during numerous experiments. It contains a detailed explanation of how we tweaked the hyperparameters and approached the problem in general. As mentioned in the section 4, to evaluate our methods, we used the StaMinA dataset.

5.1 Baseline

The figure 4 presents the BCR of the DFA learned with the EDSM algorithm, starting with a standard prefix tree acceptor. The segments divided by the red dashed lines picture the separate packages mentioned in the subsection 3.1.

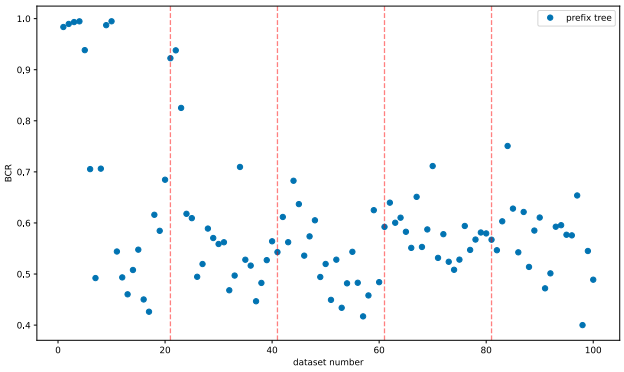


Fig. 4: BCR of the reference approach.

5.2 Changing the Reading Direction

To begin the evaluation, we need to determine sensible values for two hyperparameters: the ensemble size and the percentage of votes required to accept a trace. To achieve this, we performed a grid search over those two parameters using the StaMinA dataset. We started by analyzing the acceptance threshold with the ensemble of size 80. We then used the best-performing threshold to determine the appropriate ensemble size. All the performed experiments in this subsection sampled 150 models. This exact number was chosen since there is a maximum of $2S$ models, where S is the length of the longest trace. Most traces in the StaMinA dataset have a length smaller than 75.

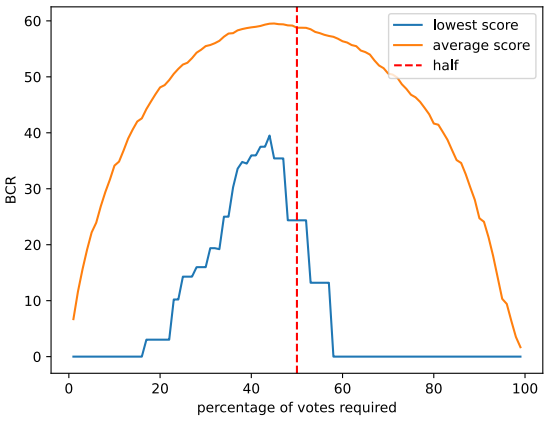
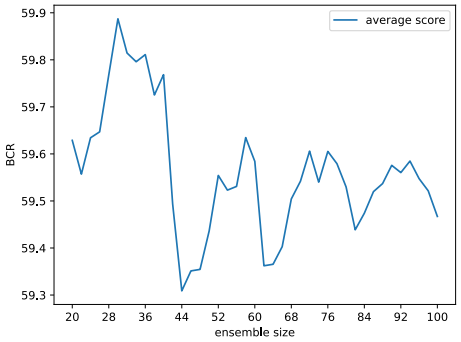
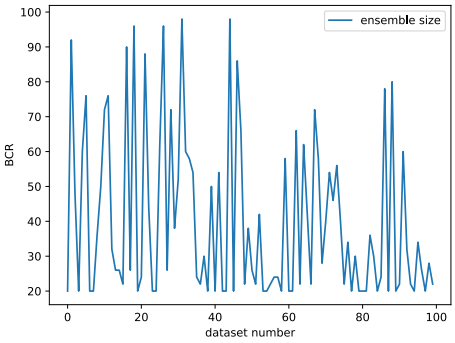


Fig. 5: The average and minimal BCR corresponding to the percentage of votes required.

From the figure 5, we concluded that 45% is a well-performing acceptance threshold, scoring an average of 59.53. This threshold was later used to analyze the ensemble size. Subsequently, we examined how the score varied with the increasing ensemble size.



(a) The average BCR corresponding to the ensemble size.



(b) First ensemble achieving maximum score.

Fig. 6: Results of ensemble size tuning experiment for the first method.

Unfortunately, the experiment to determine the ensemble size was inconclusive. Looking at the figure 6, we can notice that the average score, as well as the ensemble size that resulted in the best result, varied a lot during the experiment. We decided to use 90 models in the ensemble, since several of the models benefited from picking that amount. It is worth noting that, for some reason, the best average score was obtained while using the 30 models in the ensemble. However, we consider this an outlier, seeing how many test cases benefited from a bigger ensemble.

Having decided on the hyperparameters, we decided to perform a test of the *reverse after k steps* method, with the acceptance threshold of 45% and the ensemble size of 90.

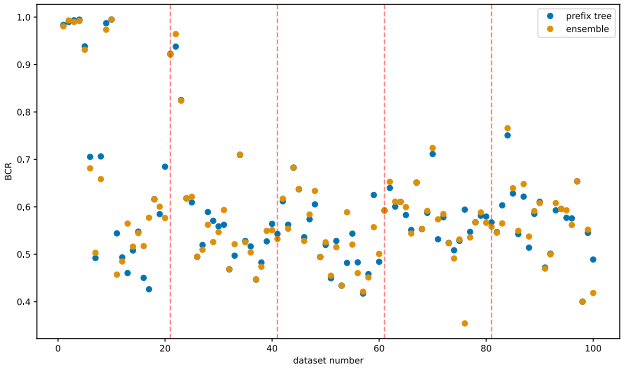


Fig. 7: Comparison of the ensemble method to the prefix tree.

To quantify the results, we compute two metrics: the number of times one method outperformed the other, and Root Mean Square (RMS), calculate with the following formula: $RMS = \sqrt{\frac{1}{n}\sum_i^n (x_i - y_i)^2}$. We calculate two separate RMS values for the cases where the

proposed solution outperformed the baseline, and vice versa. The second metric is used to determine the significance of the difference, there may be many small improvements and some big regressions, which are hard to notice from the first metric.

Alphabet size	2	5	10	20	50	combined
Score distribution	+7 =2 -11	+6 =4 -10	+9 =3 -8	+10 =4 -6	+9 =3 -8	+41 =16 -43
RMS, baseline higher	0.045	0.018	0.028	0.098	0.029	0.047
RMS, proposed higher	0.074	0.022	0.038	0.016	0.016	0.038

Table 1: The score distribution and RMS for the *reverse after k steps* method.

Analyzing the results, we can notice that the proposed approach beats the baseline in 41% test cases, and is worse in 43% of test cases. It is difficult to quantify the results to some wider trend. However, we can notice that in one test case, numbered 75, it performed much worse compared to the single model. We can also notice that it tends to perform better for bigger alphabet sizes, however, we do not believe there is a significant difference that would favor this method of ensembling. Furthermore, it is difficult to derive why in some test cases the ensemble method performed much better or much worse than the baseline approach.

5.3 Changing Substring Reading Direction

To evaluate the method proposed in the subsection 3.4, we start with the same two experiments as with the previous method. They are needed to determine the sensible ensemble size, as well as the acceptance threshold. These two values cannot be reused, since the ensemble is considerably different. To perform the experiments, we decided to sample 300 transformations per test case. This amount is double compared to the previous approach, since this time the transformation space is much bigger, consisting of $\mathcal{O}(S^2)$ models, where S is the length of the longest trace.

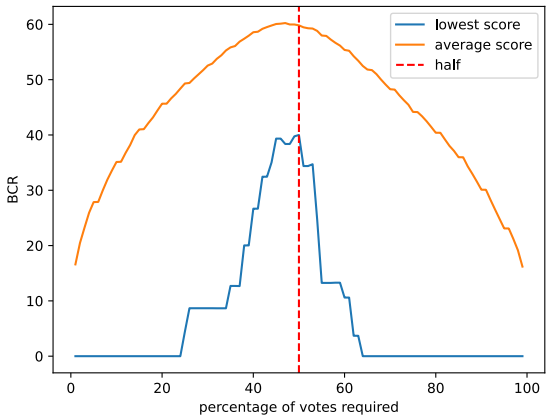
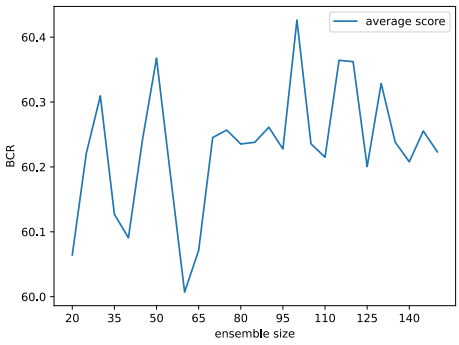
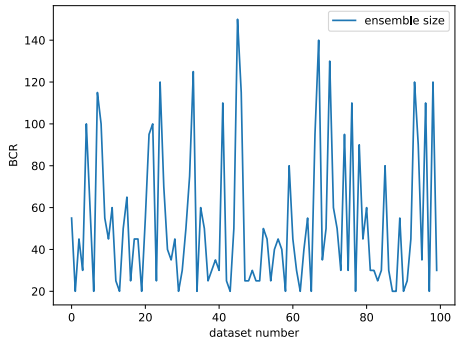


Fig. 8: The average and minimal BCR corresponding to the percentage of votes required.

The figure 8 presents the average BCR score over all test cases for the given acceptance threshold. From this experiment, we determined that the best-suited threshold would be 47%, since it scored the highest, 60.24 BCR.



(a) The average BCR corresponding to the ensemble size.



(b) First ensemble achieving maximum score.

Fig. 9: Results of ensemble size tuning experiment for the substring method.

Looking at the figure 9, we can derive that 120 is the sensible ensemble size in this scenario. Most models achieve their best score below it, but a considerable number of test cases need more than 100 models to get the best results.

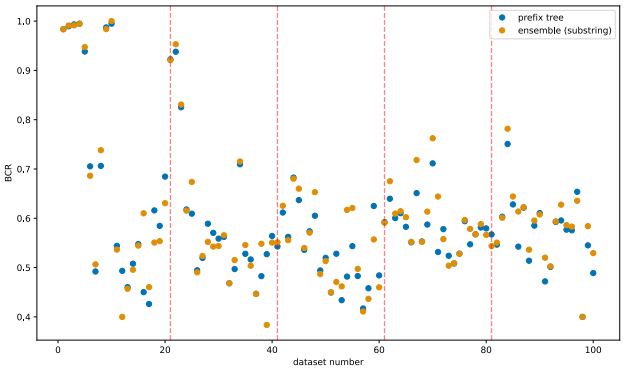


Fig. 10: Comparison of the substring ensemble method to the prefix tree.

From the table 2 and the figure 10, we can observe that the substring method outperforms the baseline approach on the last two test packages with the largest alphabet sizes. From the *RMS* we can notice that despite having a lower score than the baseline approach on the smallest alphabet size, when it achieves a better score, it is considerably better.

Alphabet size	2	5	10	20	50	combined
Score distribution	+7 =2 -11	+9 =2 -9	+10 =0 -10	+13 =2 -5	+15 =1 -4	+54 =7 -39
RMS, baseline higher	0.040	0.051	0.030	0.014	0.015	0.036
RMS, proposed higher	0.063	0.032	0.053	0.042	0.030	0.043

Table 2: The score distribution and RMS of the substring method .

5.4 Similarity Metric

To further enhance the given methods, we can employ the similarity metric mentioned in the subsection 3.5. However, since we changed the criteria for picking the models into the ensemble, we need to reevaluate the acceptance thresholds.

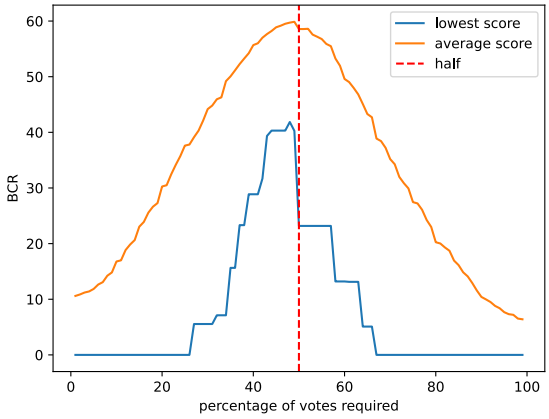


Fig. 11: The average and minimal BCR corresponding to the percentage of votes required, while using the reduced ensemble.

Analyzing the figure 11, we decided to pick the 49% as the threshold, as it scored the highest BCR, being 59.87. Notice that this slope is much steeper, which means that this method is much more sensitive to the regulation of the acceptance threshold.

The figure 12 and the table 3 show how the ensemble obtained by pruning the similar models according to the metric described in section 3.5 performs. We can observe that it has trouble solving the less difficult test cases, for which the baseline approach gets above 98%. However, for the test cases of increased difficulty, the reduced ensemble manages the outperform the baseline approach. Interesting things happen when we compare both the complete ensemble and the reduced one. The number of times one outperforms the other is around 50%, but the interesting part is the *RMS*. Since the ensembles are composed of the same models, we would expect similar results, meaning low *RMS*, but this is not the case. We offer two possible explanations for this. First, the reduced ensemble may pick up the models scoring lower, which provided additional context, but since we only allow a fixed number of models in the complete ensemble, they were not selected. Naturally, this context could improve predictions or mislead the model. The second explanation is that, since we

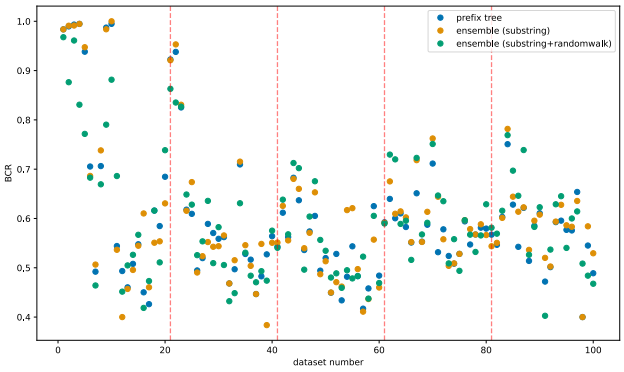


Fig. 12: Comparison of baseline approach and the reverse substring with and without the model pruning using the similarity metrics.

Alphabet size	2	5	10	20	50	combined
Distribution (vs baseline)	+6 =1 -13	+11 =0 -9	+13 =0 -7	+13 =0 -7	+14 =0 -6	+57 =1 -42
RMS, baseline higher	0.100	0.062	0.035	0.026	0.045	0.067
RMS, reduced higher	0.068	0.025	0.047	0.063	0.059	0.053
Distribution (vs complete)	+8 =0 -12	+8 =0 -12	+14 =0 -6	+9 =1 -10	+12 =0 -8	+51 =1 -48
RMS, complete metric higher	0.119	0.058	0.079	0.024	0.062	0.077
RMS, reduced metric higher	0.075	0.051	0.043	0.055	0.053	0.055

Table 3: The score distribution and RMS of the ensemble with reduced size.

use unbiased voting and at the same time we discard similar models, we naturally disrupt the voting process, causing much different results than in the complete method.

The figure 13 shows that the similarity metric greatly reduces the ensemble sizes. Coupling this with the information from table 3, we see that the reduced ensembles perform reasonably well. This means that opting for reduced ensembles does not have a great negative impact on the ensemble performance in most cases, while smaller ensembles are much easier to analyze.

Additionally, it is worth investigating the runtime of this method, since the random walks and picking method involve a lot more calculations. The testing was done on a CPU at 5.4 GHz, running under *Linux 6.12.30*. Each test was run on a single core, but the whole pipeline, consisting of 100 test cases was run on 20 different threads.

Method (num. models)	Substring (150)	Random walk (150)	Substring (300)	Random walk (300)
Runtime	372s	729s	565s	1734s

Table 4: Runtime of both methods compared.

As shown in table 4, we observe a significant change in runtime while using the random walk similarity metric. Furthermore, from section 3.5 we know that this method is quadratic, which means the gap between picking a fixed number of models will only increase.

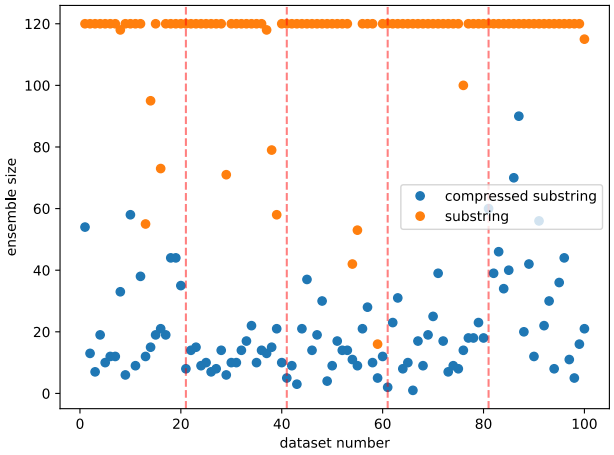


Fig. 13: Sizes of the ensembles with and without using the similarity metric.

6 Responsible Research

This section analyzes the ethical considerations associated with the presented work, together with the reproducibility of the results presented in the work.

Despite the many advantages of ensembles discussed in the paper, there are potential drawbacks to them. Since creating an ensemble involves training multiple models rather than a single one, there is a risk of consuming a significant amount of electricity, which can have a negative impact on the environment. However, all training can be efficiently done on a single-core CPU. In contrast to the other popular techniques in machine learning nowadays, learning EDSM ensembles does not benefit from using heaps of electricity. We believe that the methods presented in this paper are efficient and result in a minimal increase in environmental footprint compared to the well-established state merging techniques. Therefore, we believe that the discussed approaches are aligned with the goals for sustainable growth.

To ensure that all the results obtained during this research are reproducible, we published the testing code and the datasets on a publicly available GitHub repository[1]. Given that the presented methods involve randomization, we set up all the experiments in a way that controls the supply of the seed value. All parameters related to the randomization are also included in the repository. We believe that this data, together with the precise steps described in sections 3, 4, and 5 should be sufficient to reproduce our results.

7 Conclusions and Future Work

In this work, we presented approaches for constructing an ensemble of DFAs by manipulating the sequence orders. We showed how reversing the reading direction on particular segments of the data can influence the variety and results of models learned with the EDSM algorithm. We showed a simple metric that could be used to measure the similarity between two models, along with the algorithm that employs it for ensemble construction.

[1] https://github.com/mooshroom4422/FlexFringe

We empirically demonstrated that, on easier examples from the StaMinA competition [7], the single model learned with EDSM tends to perform better than ensemble methods. However, with the increased difficulty of languages, meaning increasing sparsity and alphabet size, the ensembles provide better results. We showed that using the similarity metric to discard models from the ensemble greatly reduces its size. The reduction of the ensemble size also influences the results substantially, but sometimes it makes predictions worse. To summarize, these results show that ensemble methods can be efficiently employed to derive software models from given traces.

While this work showcased the potential of using sequence ordering for constructing DFA ensembles, several directions are left unexplored. First, to evaluate proposed approaches, we used only the StaMinA dataset. While the competition was designed to provide an empirically sound benchmark, it would be beneficial to test these approaches in different scenarios. Another limitation of the proposed approaches comes from the fact that, since the models that we try to infer can have loops, relying on different indices for different models could make the ensemble even better. This could potentially be improved by applying the transformation after identifying a specific combination of traces.

References

[1] Sicco Verwer and Christian Hammerschmidt. *FlexFringe: Modeling Software Behavior by Learning Probabilistic Automata*. 2024. arXiv: 2203.16331 [cs.LG].

[2] Azqa Nadeem, Sicco Verwer, and Shanchieh Jay Yang. "Sage: Intrusion alert-driven attack graph extractor". In: *2021 IEEE symposium on visualization for cyber security (VizSec)*. IEEE. 2021, pp. 36–41.

[3] E Mark Gold. "Complexity of automaton identification from given data". In: *Information and control* 37.3 (1978), pp. 302–320.

[4] José Oncina and Pedro Garcia. "Identifying regular languages in polynomial time". In: *Advances in structural and syntactic pattern recognition*. World Scientific, 1992, pp. 99–108.

[5] Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. "Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm". In: *Grammatical Inference*. Ed. by Vasant Honavar and Giora Slutzki. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 1–12.

[6] Thomas G Dietterich. "Ensemble methods in machine learning". In: *International workshop on multiple classifier systems*. Springer. 2000, pp. 1–15.

[7] Neil Walkinshaw et al. "STAMINA: a competition to encourage the development and assessment of software model inference techniques". In: *Empirical software engineering* 18.4 (2013), pp. 791–824.