Adapting the EDSM Algorithm for Ensemble Learning

A Machine Learning Approach to DFA Inference

Radu-Cosmin Dumitru[1]
Supervisor(s): 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: Radu-Cosmin Dumitru
Final project course: CSE3000 Research Project
Thesis committee: Sicco Verwer, Simon Dieck, Merve Gürel

Abstract

Learning Deterministic Finite Automata (DFA) from given input data has been a central task in the field of Grammatical Inference, and progress in this area is of great interest from both theoretical and practical points of view. To address this challenge, several algorithms have been proposed and evaluated using established benchmarks. One such competition-winning algorithm, Evidence Driven State Merging (EDSM), uses a heuristic to learn a DFA from given data. However, improvements leveraging ensemble techniques from machine learning have yet to be explored. In this paper, we investigate ways to adapt the EDSM algorithm to fit into the ensemble learning framework and analyze the performance of such obtained models when applied to unseen data. To this end, we compare the performance of the ensembles to that of a standard EDSM-learned model, evaluating both their output quality and the diversity within each ensemble. The results indicate significant improvements in scenarios where the data is sparse.

1 Introduction

Analyzing abstract models of computation is essential in order to determine what is feasible using computers and other types of machines, the extent of their computational power, and theoretical limitations. The Deterministic Finite Automaton (DFA) is one such type of formalism, often considered to be the simplest one. It serves as the foundation for abstract computational models, and its study provides useful insight into the workings of all others that build upon it. From a practical perspective, DFA inference algorithms aim to construct models that capture and generalize the patterns observed in labeled data, making them valuable for understanding complex or previously unknown systems. Inferred automata provide compact and explainable representations that have applications in domains such as cybersecurity, compiler design, software analysis, and information retrieval [1, 2, 3, 4].

The topic of learning or inferring languages has been introduced with Gold's seminal paper [5] and has since evolved into its own subfield of Grammatical Inference. DFA learning has been studied in various forms, such as active [6] and passive [7] learning. The main idea of passive learning is that an algorithm receives as input a set of strings, or traces, that the target automaton should accept or reject, and outputs an automaton that is consistent with the given data. There is no general consensus on which algorithm is the best, as the inputs and goals vary depending on the application, but there have been attempts to find the winner in some categories, such as the Abbadingo One DFA learning competition [8], which sought to promote highly scalable algorithms, with a focus on sparse data. The winning algorithm of this competition, EDSM, or evidence-driven state merging, is the one that will be the focus of this paper.

As a machine learning technique, ensemble learning has been introduced as a way to use existing models to obtain a result better than all other individual ones [9]. Ensembles generally work by combining individually strong yet distinct models so that they can capture as much information as possible. Therefore, diversity across models is encouraged, as it is thought to lead to more generalizable results. This technique has proven to be highly effective in the case of decision trees, an interpretable model of machine learning [10]. However, ensembles have not been implemented in conjunction with DFA learning so far. While providing interpretable models for sequential data, learning algorithms often struggle with limited generalizability when confronted with sparse data. DFAs learned from sparse datasets are prone to overfitting, making them less effective at classifying new inputs. Using

ensemble techniques, there is reason to believe that we will obtain models that are more accurate and robust in real-world applications, where data may be sparse or incomplete.

This work aims to investigate the effectiveness of applying ensemble techniques on the competition-winning EDSM algorithm for DFA learning, by modifying its state merging heuristic and creating multiple candidate automata. The main contributions of this paper are the adaptation of the EDSM algorithm for use within ensemble learning frameworks, the analysis of diversity among models produced by the ensemble, and performance comparisons of different types of ensembles against a single DFA learned using state-of-the-art methods.

To address these research questions, the paper is structured as follows. In Section 2, the necessary background for the EDSM algorithm is given, along with an overview of the ensemble techniques. Next, in Section 3, the modifications made to the algorithm are presented, as well as the implementation of the ensemble and its evaluation. Section 4 details the experimental results, and their corresponding discussion is presented in Section 5. Finally, Section 6 addresses responsible research practices, and Section 7 provides concluding remarks and an outlook for the future directions of this research.

2 Background

This section presents the preliminary concepts required for the remainder of the paper by detailing the EDSM algorithm for learning a DFA, as well as common ensemble techniques.

2.1 DFA Learning

It has been proven that the general task of inferring a minimal DFA consistent with a given dataset S is NP-complete [11]. Nevertheless, algorithms that identify DFAs under certain conditions have been developed, such as RPNI (Regular Positive and Negative Inference), a polynomial time algorithm which is based on characteristic sets [7].

In the context of passive learning, the EDSM algorithm operates on two finite sets: S_+ and S_-, containing positive and negative traces extracted from a target formal language, respectively.

2.1.1 Initial Hypothesis

The Prefix Tree Acceptor is the starting point of most inference algorithms. It accepts all positive traces and rejects all negative ones in the dataset by organizing them into a rooted tree based on their shared prefixes. As such, it recognizes the input data perfectly, and the paths from the root to any state are unique. Each path from the root to a leaf corresponds to a trace, with accepting or rejecting states marking the ends of the recognized strings. The Augmented Prefix Tree Acceptor (APTA) serves as the initial hypothesis and includes intermediate or 'free' states that are neither accepting nor rejecting, therefore enabling the possibility of state merges during the inference process. An example of such an automaton is given in Figure 1.

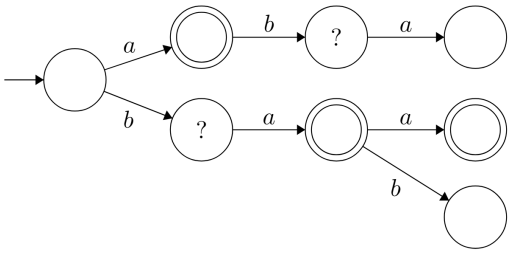

Figure 1: Augmented Prefix Tree Acceptor for $S_+ = \{a, ba, baa\}, S_- = \{\epsilon, aba, bab\}$, where ϵ denotes the empty string. The initial state is marked by the transition with no source. Accepting states are marked by a double circle, rejecting states by a single circle, and states of unknown type by '?'.

2.1.2 State Merging

The next step is to refine the initial hypothesis, which is fully tailored to the training data, by merging its states, in order to achieve better generalization. Due to the enormous number of total merges possible at any given point for a DFA, exhaustively checking all of them is infeasible. EDSM uses the red-blue state merging framework as a strategy for exploring merges to perform [8]. An overview of this framework is given in Algorithm 1.

Algorithm 1 Red-blue state merging

Input: $S = \langle S_+, S_- \rangle$: training set
Output: DFA D consistent with S

```
D ← CreateAPTA(S)
Red ← {root}, Blue ← {children of root}
while Blue ≠ ∅ do
    if there are 2 states r ∈ Red and b ∈ Blue such that their merge is consistent then
        Pick the highest scoring pair (r, b)
        Merge(D, r, b)
    else
        random ← a random state drawn uniformly from Blue
        Blue ← Blue \ {random}
        Red ← Red ∪ {random}
    end if
    for each state r ∈ Red do
        for each child c of r do
            if c ∉ Red ∪ Blue then
                Blue ← Blue ∪ {c}
            end if
        end for
    end for
end while
return D
```

The red states serve as the foundation of the hypothesis, while blue states are candidates that try to merge into it. The red states are assumed to represent a correct and consistent

part of the final DFA. Initially, the root of the APTA is marked as red, and its children become blue states, forming the fringe. The algorithm iteratively attempts to merge blue states into red ones, assigning a score to each merge and choosing to execute the best one.

Merge(D, r, b) is a procedure that creates a new state in D and reroutes all inbound and outbound transitions from r and b to it. However, in this process, non-deterministic transitions may appear — the merged state could have transitions with the same symbol to multiple target states. In such situations, a determinization step is required to preserve this property of the DFA. This involves recursively merging the target states of overlapping transitions until there is no non-determinism left. If, at any point, a merge is attempted between an accepting state and a rejecting one, it is deemed *inconsistent* and the entire merge is canceled. Figure 2 shows an example of non-determinism.

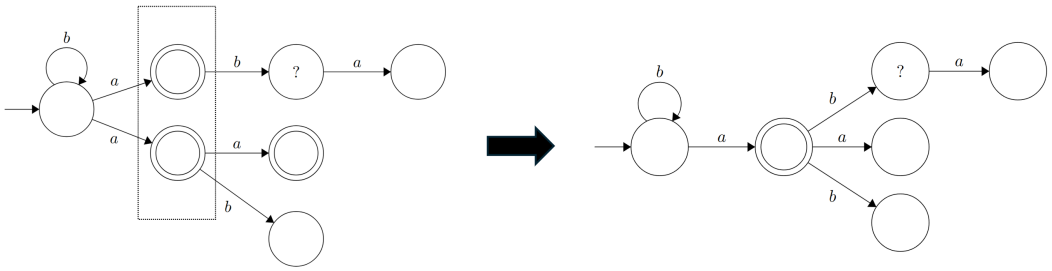

Figure 2: A merge between 2 states during determinization, highlighted by the dashed rectangle. The result is still non-deterministic.

In the case that there are no consistent merges available, the framework randomly picks a blue state and promotes it to red, thus continuing the merging process. At the end of each iteration, the blue fringe is updated such that all uncolored children of the red states become blue.

2.1.3 EDSM Heuristic

The score of a red-blue candidate merge is determined based on *evidence*. In principle, we want to avoid suboptimal merges that lead to bigger and worse-performing DFAs later on. Therefore, the heuristic for evidence tries to steer the merges towards a more accurate DFA that fits the data well.

The evidence score s is computed as the sum of the number of merges between accepting states and the number of merges between rejecting states, before and after performing the merge, as part of the determinization process: $s = P + N$, where P is the number of positive-positive merges and N is the number of negative-negative merges that were executed. In the original algorithm, the candidate merge with the highest score is greedily chosen as the next one to be executed. This gradually builds a DFA that is consistent with all traces in the dataset, with the objective of generalizing effectively to unseen data.

2.2 Ensemble Techniques

While a multitude of ensemble types exists, each with varying assumptions and use cases, we focus on two approaches that are particularly well-suited for our setting: randomization and

boosting. These techniques were chosen for their compatibility with heuristic-based algorithms like EDSM and for their potential to combat imbalanced scenarios often encountered in automaton inference.

2.2.1 Randomized Models

In learning tasks, algorithms that employ a greedy strategy where, at each step, a locally optimum choice is made based on a predefined evaluation function or heuristic, may run into the problem of converging to a local optimum and missing better solutions elsewhere in a vast search space.

To mitigate this issue, a common technique involves introducing a controlled element of randomness or noise into the decision-making process of the underlying algorithm. Instead of strictly adhering to the highest scoring action at all times, a slight perturbation is applied to the decision criteria. This makes it so that the resulting collection of models might capture aspects of the input data that were previously missed.

2.2.2 Boosting

Boosting is a type of ensemble that is based on the intuition that combining multiple weak models should result in a stronger one [12]. It is a sequential training process in which a weak learner is given adjusted training data at every step. Models are *boosted* iteratively, where they are trained to correct the errors made by the previous ones. Boosting assigns higher weights to properties of misclassified samples, forcing subsequent models to focus on those cases. It is appreciated for performing well by achieving high final accuracies, not overfitting, and reducing bias in the models, properties which are also desirable for the problem at hand.

It has been shown to be successful in both classification [13] and regression [14] tasks. Since EDSM outputs an automaton that will be used to predict new traces, this is analogous to a binary classification problem, for which boosting is appropriate.

3 Methodology

This section describes the ensembling approaches and the evaluation methods used to test the effectiveness of the ensembles.

3.1 Ensemble Types

3.1.1 Randomized Models

In the context of EDSM, the heuristic computes an evidence score for each candidate merge and then greedily picks the best one. To diversify the resulting hypotheses, we introduce noise into the scoring mechanism. This allows the algorithm to choose merges that appear to be suboptimal, which can be beneficial by enabling the red-blue framework to perform merges that would otherwise be inaccessible. This can potentially reveal DFAs that perform better than the current state-of-the-art.

EDSM Adaptation

We apply a multiplicative random factor to the evaluation score s:

$$s_{new} = s \cdot (1 - R), \text{where } R \sim \mathcal{U}(0, r), \text{with } 0 < r < 1.$$

Here, $\mathcal{U}(0, r)$ denotes the uniform distribution over the interval (0, r). Each member of the ensemble will use a fixed r as a hyperparameter in the training phase.

3.1.2 Boosting

In order to perform boosting, we assign weights to every state in the APTA derived from the training set. All weights are initialized uniformly in the *CreateWeightedAPTA* procedure, and get recalculated at every learning iteration using a validation set. Pseudocode of this process is given in Algorithm 2.

Algorithm 2 Boosting process

Input: S = $\langle S_+, S_- \rangle$: training set, V = $\langle V_+, V_- \rangle$: validation set, n: number of models
Output: Ensemble $\varepsilon = (M_1, M_2, ..., M_n)$

```
D ← CreateWeightedAPTA(S)
for i in 1, 2, ..., n : do
    M_i ← MergeStates(D)
    V_i ← Validation traces misclassified by M_i
    D ← UpdateWeights(D, V_i)
    Add M_i to ε
end for
return ε
```

Each model is merged from the weighted APTA using EDSM in *MergeStates*, taking into account the weights obtained after the previous iteration. The accuracy of the new model on the validation set is used as its weight, determining its contribution to the ensemble's vote. Then, all misclassified traces are simulated on the APTA. The final states of these simulations have their weights doubled in *UpdateWeights*. In cases where a validation trace does not have a complete path in the APTA, the simulation considers the last reachable state as its final state.

EDSM Adaptation

For all candidate merges (r, b), the score calculation is changed to

$$s_{new} = \frac{w_r + w_b}{2} \cdot s$$

Where w_r and w_b are the weights of the red and blue states, respectively, and s is the standard EDSM score. This is motivated by the fact that we want to focus more on the properties of a trace that caused it to be misclassified. States with higher weights correspond to regions of the DFA that require more careful handling, because they represent uncertainty based on a validation set of traces. Averaging the weights of the red and blue states in a candidate merge ensures that both contribute to the adjusted evidence score. If either state is frequently responsible for misclassification of traces, the combined score increases relative to the standard one, making the merge gain priority over others.

3.2 Ensemble Prediction

After an ensemble has been created, we use it to predict traces of unseen data. In the context of ensembles of DFAs, we choose weighted majority voting as the prediction method, due to its ability to account for differences in model quality. In a standard majority voting scheme, each DFA in the ensemble would cast one vote for a prediction, and the final decision would be based on which one occurred more often amongst the models. However, this approach assumes that all models are equally accurate, which is false, due to either randomization or state weighting. Weighted majority voting addresses this by assigning a weight to each model's vote, based on its individual accuracy. In order to compute these weights, we will use validation sets, separated from the training and test sets, to avoid leaking information and skewing results. At prediction time, the votes of the models are scaled by their weights and the final decision corresponds to the class with the higher weighted sum. Formally, given n DFA models $M_1, M_2, ..., M_n$, each with an associated weight $w_i, i \in \overline{1,n}$, the ensemble prediction for a trace t is:

$$\text{prediction}(t) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i \cdot M_i(t) > 0 \\ 0 & \text{otherwise} \end{cases}$$

where $M_i(t) = 1$ if model i accepts trace t and -1 otherwise. This makes the models that were more accurate on the validation data have a proportionally bigger influence on the ensemble prediction, reducing the impact of weaker models.

3.3 Evaluation Methods

The resulting ensembles are analyzed from 2 perspectives: their predictive quality on new data and their internal diversity.

3.3.1 Ensemble Output Metrics

The ensemble outputs are measured using 4 key metrics in order to provide a comprehensive overview of their predictive quality:

- Balanced Accuracy: overall proportion of correctly classified samples out of all samples, adjusted for the class ratios

$$\text{Balanced Accuracy} = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right) \tag{1}$$

- Precision: proportion of true positive predictions out of all samples predicted as positive

$$\text{Precision} = \frac{TP}{TP+FP} \tag{2}$$

- Recall: proportion of true positive predictions out of all positive samples

$$\text{Recall} = \frac{TP}{TP+FN} \tag{3}$$

- F_1-score: harmonic mean of precision and recall

$$F_1\text{-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{4}$$

Where: TP = True Positives, TN = True Negatives, FP = False Positives, FN = False Negatives.

Given that most real-world applications consist of imbalanced data, we use balanced accuracy, which ensures that both the positive and negative classes are weighted equally in the evaluation. This prevents the ensemble from achieving high predictive scores when it favors the majority class.

We also include precision, recall and F_1-score in our analysis. These are metrics that focus on the positive class, which is justified in the context of software inference or protocol analysis. High precision indicates that the ensemble makes few false positive errors, while high recall reflects a low number of false negative errors. False negatives can lead to overlooking valid behaviors of the system, potentially blocking system operations. On the other hand, false positives have the risk of allowing unsafe behavior. The F_1-score provides a balanced view when there is a trade-off between these two types of errors. A high F_1-score means that the ensemble has good overall classification accuracy with respect to positive traces. Negative traces often represent faults or random deviations from normal behavior, so their accurate prediction is not as essential, their role being mainly to prevent oversimplifying the automaton by reducing the number of permitted, consistent merges.

3.3.2 Inter-model Variety Metric

Average Disagreement Rate

For any 2 distinct models M_A and M_B, their disagreement rate $\Delta_{A,B}$ is calculated as the proportion of test samples for which M_A and M_B have different predictions. To obtain the overall disagreement rate of the ensemble Δ_ε, this rate is then averaged over all ordered pairs of models in the ensemble ε. This measure provides insight into an ensemble's diversity: a higher average disagreement rate indicates greater diversity, which leads to robustness and resilience to outliers.

$$\Delta_{A,B} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}_{M_A(t_i) \neq M_B(t_i)} \tag{5}$$

where N is the number of test traces t, and $\mathbf{1}_{condition}$ is the conditional function that returns 1 if the condition is true and 0 otherwise.

4 Experimental Setup and Results

This section covers the experiments that were performed in order to evaluate the effectiveness of our approaches. We first describe the general setup regarding the implementation details, followed by results obtained using the methodology.

4.1 Setup

All experiments were performed using the open-source FlexFringe framework [15] written in C++, which provides implementations of automaton inference algorithms, including EDSM. The proposed ensembles are integrated within its codebase.

We used 10 constituent models for every ensemble configuration. The Randomized Models were tested in 3 instances: a highly randomized ensemble with the hyperparameter $r = 0.7$. Second, a slightly randomized ensemble with $r = 0.4$. The third ensemble has

the lowest amount of randomization, with $r = 0.1$. The boosted ensemble was also benchmarked. Lastly, all ensembles were compared to a single DFA learned using the original EDSM heuristic.

Datasets

The StaMinA competition [16] offers datasets derived from automata that model software systems. These datasets vary in sparsity and alphabet sizes, as both factors are controlled during generation, making them well-suited for evaluating our ensembles. FlexFringe also includes 100 training datasets from the StaMinA competition. We split each of those into training, validation and test sets with a 70-15-15 ratio, such that the sparsity of the original datasets is preserved. The training set is used to infer the DFA. The validation set is used to compute the model's weight in the ensemble, and, in the case of boosting, to update state weights in the APTA after each iteration. The test set is reserved for evaluating the predictive performance of the ensemble. The original datasets contained duplicates, but as these make no difference to the EDSM inference, we made all traces unique, ensuring that each trace appears only once across the training, validation and test sets in every split. This prevents information leakage and preserves the integrity of the evaluation.

4.2 Results

We plotted the evaluation metrics of 50 out of the 100 datasets for ease of visualization. The datasets are sorted in increasing order of their density, measured as the proportion of positive traces. Furthermore, in order to provide objective analysis of the results, we computed the mean and standard deviation of every method. For the ensembles, we additionally performed a t-test against the baseline EDSM to determine whether there are statistically significant improvements. Appendix A provides the complete statistics for interpretation.

Figures 3, 4, 5, 6 show the ensemble output metrics, while Figure 7 compares the inter-model variety of the ensembles.

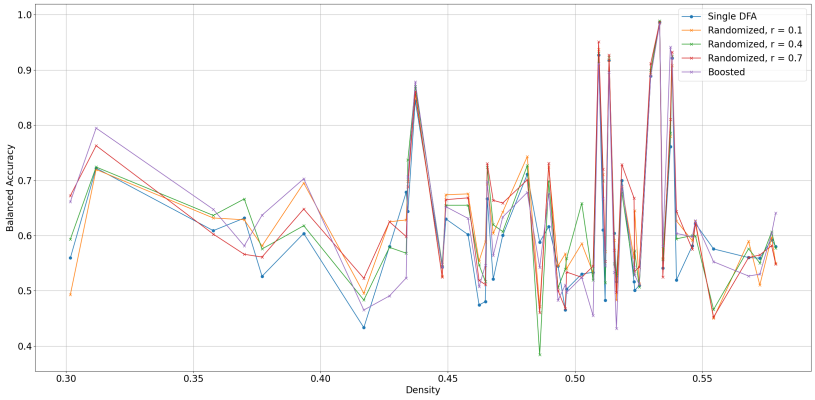

Figure 3: Balanced Accuracy

Balanced Accuracy is similar across all ensembles and almost every type of ensemble performs marginally better than the single DFA on the overall datasets.

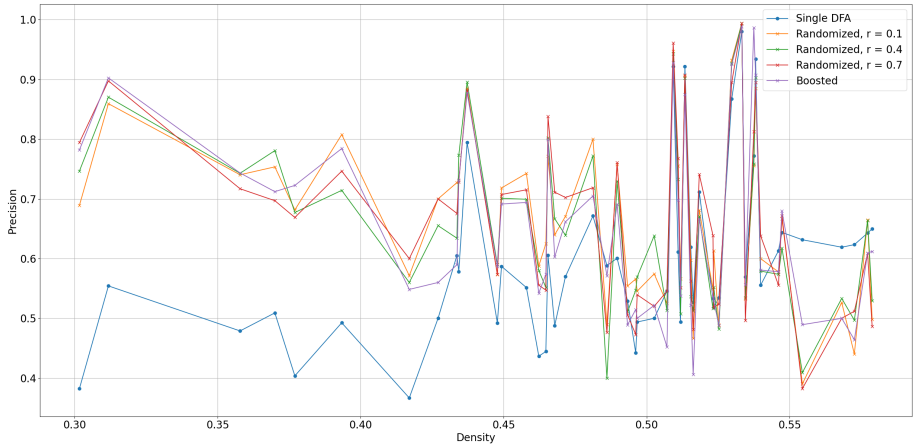

Figure 4: Precision

For the sparsest datasets, the ensembles significantly outperform the single DFA in terms of precision, meaning the traces predicted as positive by the ensembles are much more likely to be truly positive. This improvement is statistically significant, with p-values resulting from the paired t-tests for all ensembles, over all datasets, well below 0.05. The gap closes as the sets become denser, as reflected by the negative t-statistics.

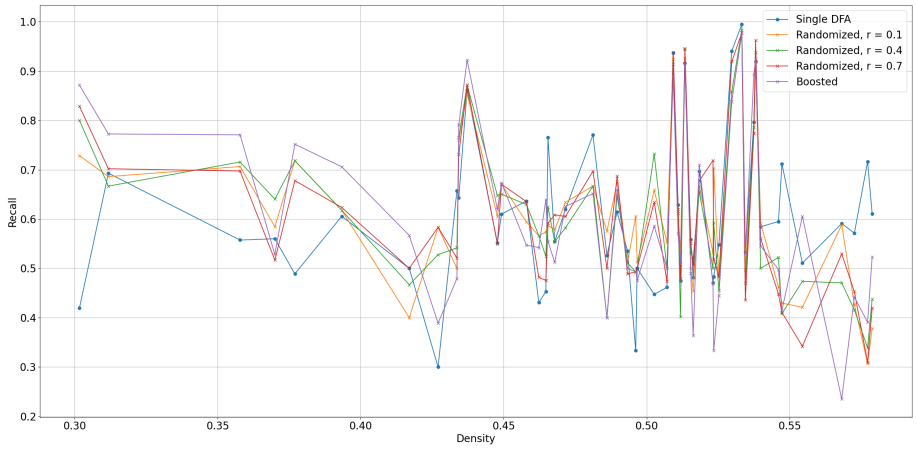

Figure 5: Recall

Ensemble performance regarding recall was not superior to the baseline, with their benefits primarily realized on the less sparse datasets. As the sets begin to contain more positive traces, the single DFA distances itself, but the ensembles maintain comparable performance when all datasets are considered.

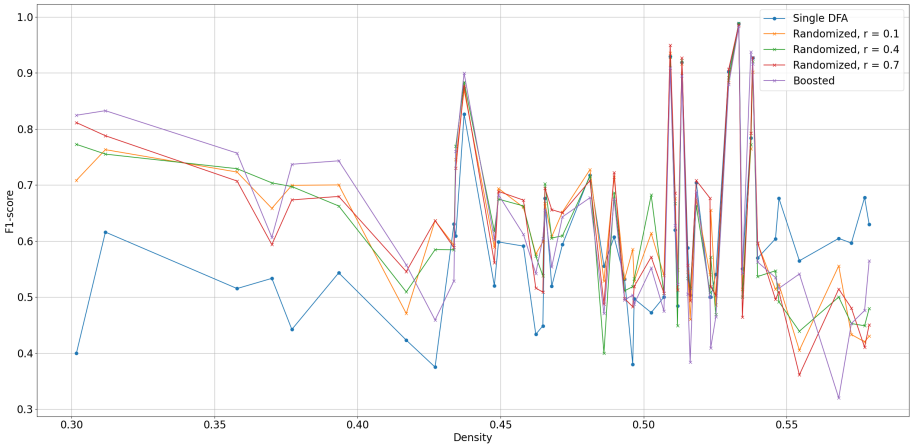

Figure 6: F_1-score

As the harmonic mean of precision and recall, the F_1-score aligns with the trends observed in the previous results. Ensemble methods are significantly better than the baseline when the datasets contain a limited number of positive traces. Conversely, their performance significantly worsens as the datasets gain more positive traces. This behavior is supported by high t-statistic values and exceptionally low p-values in all paired t-tests performed on both the lower and upper halves of the data.

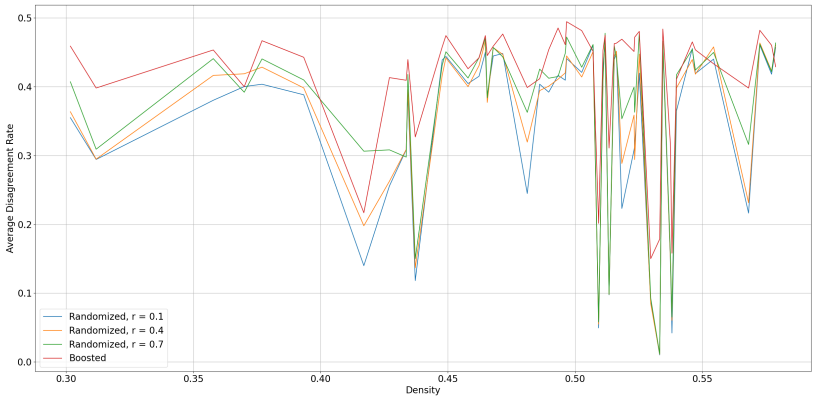

Figure 7: Average Disagreement Rate

We observe the expected pattern of disagreement rates being higher the more randomized an ensemble is. The boosted ensemble consistently has the highest average disagreement rate among all other evaluated methods, regardless of the dataset portion analyzed.

5 Discussion

All metrics point to the fact that the ensemble methods are able to generalize better than a single, greedily-learned DFA when the amount of positive data available is limited. The baseline EDSM model performs better as the datasets increase in the number of positive traces. This is a good indicator of the importance of evidence when merging states of the original APTA—the better EDSM can differentiate between candidate merges, the more it can achieve good predictive performance. However, the ensemble methods were able to maintain comparable output quality, a result which may be attributed to the weighted majority voting, which attempts to compensate for less desirable models present in the ensemble.

Boosting was inconsistent, varying from the best to the worst performing one on seemingly unrelated datasets. However, the F_1-score indicates that it is a middle ground between the single DFA and the Randomized Models. The biggest differences are observed at the extremes of the datasets, in the sparsest and densest ones. This can be attributed to the constant recalculation of the state weights and points to non-convergence. It consistently exhibited the highest average disagreement rate among its models, but instead of enabling a more comprehensive representation of the target data, this appears to have hindered its performance.

6 Responsible Research

This section discusses the ethical implications regarding DFA learning and outlines the responsible research practices for this paper.

While this work focuses on the theoretical learning and evaluation of DFAs, we acknowledge the importance of responsible research. Since no personal, sensitive or real-world user data was used, and the methods do not interact with human subjects, the ethical risks in this context are minimal.

Nevertheless, there are some broader implications to take into account. For example, DFA learning methods could be integrated into real-world systems such as intrusion detection, threat classification, and behavioral modeling. In such applications, ethical concerns might be present, especially related to fairness and misuse. While this paper is not connected in any way to these cases, we encourage mindful application of the presented methods.

The use of generative AI in this paper was limited to non-substantial assistance, such as improving sentence wording and formatting. The research ideas, concepts, experiments and analyses were fully conducted by the authors. The used prompts had the form of "Does this sentence sound natural, and is it easy to understand? [...]".

To ensure reproducibility of the results, the FlexFringe framework[1], which includes the datasets and codebase used for DFA inference and ensemble methods, is publicly available. The code for the implementation of the ensembles is provided, along with the datasets and scripts used to run the experiments.

7 Conclusions and Future Work

This paper presented a novel application of ensemble learning to DFA inference by adapting the EDSM algorithm in such a way that it fits the general ensemble framework. Through

[1] `https://github.com/tudelft-cda-lab/FlexFringe`

the introduction of controlled stochasticity and state weights, we generated diverse models that deviated from the greedy path of learning. This allowed for construction of ensembles of DFAs that achieved similar or better performance and generalization than single models, depending on the sparsity of the data, with ensembles outperforming the DFA learned using a state-of-the-art algorithm due to the fact that they were able to capture a more complete representation of the data at hand. The experiments showed that the impact of the evidence heuristic used in the original algorithm was not negligible—as the datasets got denser, the heuristic had access to more information about merges and could therefore make better decisions.

Future work in this direction may include exploring a wider variety of ensemble methods and investigating alternative modifications to the EDSM heuristic to enhance accuracy and generalizability. Additionally, developing ways to increase model diversity without compromising the predictive performance of the ensemble, together with more comprehensive measures of this quantity, might be useful. Evaluating ensembles on a larger range of hyperparameters could yield practical advantages. For randomized models, considering specific probability distributions beyond the uniform one presents an interesting path to pursue. In the context of boosting, updating the state weights based on the trace path, rather than only the final state that the simulation ends up in, could prove beneficial. Furthermore, establishing theoretical guarantees on the convergence of boosting algorithms that employ robust weight recalibration methods and EDSM score adaptations should lead to better results. One may also draw inspiration from combinatorial optimization approaches to effectively balance the exploration of a large set of merge candidates with the exploitation of promising options.

References

[1] E. Gribkoff. Applications of deterministic finite automata. *UC Davis*, pages 1–9, 2013.

[2] S. Wang, F. Sun, H. Zhang, D. Zhan, S. Li, and J. Wang. EDSM-based binary protocol state machine reversing. *Computers, Materials and Continua*, 69(3):3711–3725, 2021.

[3] M.J.H. Heule and S. Verwer. Software model synthesis using satisfiability solvers. *Empirical Software Engineering*, 18(5):825–856, 2013.

[4] A. Aho and M. Corasick. Efficient string matching: An aid to bibliographic search. *Communications of the ACM*, 18:333–340, 1975.

[5] E. M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

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

[7] J. Oncina and P. García. Inferring regular languages in polynomial update time. *World Scientific*, 1992.

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

[9] D. Opitz and R. Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999.

[10] M. Azad, T.H. Nehal, and M. Moshkov. A novel ensemble learning method using majority based voting of multiple selective decision trees. *Computing*, 107(42), 2025.

[11] E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.

[12] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[13] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory*, pages 23–37. Springer Berlin Heidelberg, 1995.

[14] R. S. Zemel and T. Pitassi. A gradient-based boosting algorithm for regression problems. In *Proceedings of the 14th International Conference on Neural Information Processing Systems*, pages 675–681, 2000.

[15] S. Verwer and C. A. Hammerschmidt. flexfringe: A passive automaton learning package. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 638–642. IEEE, 2017.

[16] N. Walkinshaw, B. Lambeau, C. Damas, P. Dupont, S. Ramesh, J. Clark, J. Hughes, T. Margaria, and J. Krinke. STAMINA: a competition to encourage the development and assessment of software model inference techniques. *Empirical Software Engineering*, 18(5):791–824, 2013.

A Statistical Analysis

Tables 1–4 present the statistics for the 4 output metrics, in order: Balanced Accuracy, Precision, Recall and F_1-score. Table 5 presents the statistics for the average disagreement rate within an ensemble. Every table except Table 5 has three subtables that offer a more detailed breakdown of the results by dataset density. The datasets are sorted in increasing order of density, so the lower half corresponds to sparser datasets and the upper half to denser ones.

Table 1: Balanced Accuracy

Method	**Mean**	**Standard Deviation**	**Paired t-test vs Single** (t, p)
Single	0.6019	0.1183	—
Random, $r = 0.1$	0.6116	0.1153	(1.30, 0.1952)
Random, $r = 0.4$	0.6083	0.1178	(0.87, 0.3846)
Random, $r = 0.7$	0.6121	0.1211	(1.33, 0.1859)
Boosted	0.6092	0.1259	(1.29, 0.1990)

(a) All datasets

Method	**Mean**	**Standard Deviation**	**Paired t-test vs Single** (t, p)
Single	0.5834	0.0735	—
Random, $r = 0.1$	0.6039	0.0825	(2.51, 0.0155)
Random, $r = 0.4$	0.6004	0.0859	(1.98, 0.0528)
Random, $r = 0.7$	0.6048	0.0894	(2.35, 0.0229)
Boosted	0.5943	0.0841	(1.32, 0.1925)

(b) Lower half of datasets

Method	**Mean**	**Standard Deviation**	**Paired t-test vs Single** (t, p)
Single	0.6204	0.1480	—
Random, $r = 0.1$	0.6193	0.1402	(-0.08, 0.9328)
Random, $r = 0.4$	0.6162	0.1422	(-0.35, 0.7286)
Random, $r = 0.7$	0.6194	0.1458	(-0.08, 0.9373)
Boosted	0.6241	0.1556	(0.48, 0.6365)

(c) Upper half of datasets

Table 2: Precision

Method	**Mean**	**Standard Deviation**	**Paired t-test vs Single** (t, p)
Single	0.5790	0.1316	—
Random, $r = 0.1$	0.6330	0.1460	(3.29, 0.0014)
Random, $r = 0.4$	0.6284	0.1466	(3.03, 0.0031)
Random, $r = 0.7$	0.6323	0.1488	(3.24, 0.0016)
Boosted	0.6331	0.1387	(4.38, 0.0001)

(a) All datasets

Method	**Mean**	**Standard Deviation**	**Paired t-test vs Single** (t, p)
Single	0.5186	0.0856	—
Random, $r = 0.1$	0.6689	0.0929	(10.50, 0.0001)
Random, $r = 0.4$	0.6641	0.0976	(9.70, 0.0001)
Random, $r = 0.7$	0.6707	0.0989	(10.48, 0.0001)
Boosted	0.6552	0.0908	(9.26, 0.0001)

(b) Lower half of datasets

Method	**Mean**	**Standard Deviation**	**Paired t-test vs Single** (t, p)
Single	0.6394	0.1414	—
Random, $r = 0.1$	0.5970	0.1772	(-1.89, 0.0648)
Random, $r = 0.4$	0.5927	0.1758	(-2.15, 0.0367)
Random, $r = 0.7$	0.5939	0.1777	(-2.06, 0.0443)
Boosted	0.6109	0.1711	(-2.59, 0.0125)

(c) Upper half of datasets

Table 3: Recall

Method	**Mean**	**Standard Deviation**	**Paired t-test vs Single** (t, p)
Single	0.5900	0.1444	—
Random, $r = 0.1$	0.5766	0.1469	(-0.78, 0.4349)
Random, $r = 0.4$	0.5722	0.1540	(-1.03, 0.3057)
Random, $r = 0.7$	0.5719	0.1573	(-1.06, 0.2911)
Boosted	0.5834	0.1559	(-0.44, 0.6590)

(a) All datasets

Method	**Mean**	**Standard Deviation**	**Paired t-test vs Single** (t, p)
Single	0.5509	0.1164	—
Random, $r = 0.1$	0.5972	0.0909	(2.34, 0.0236)
Random, $r = 0.4$	0.5940	0.1078	(2.05, 0.0459)
Random, $r = 0.7$	0.5884	0.1085	(1.83, 0.0738)
Boosted	0.5985	0.1195	(2.09, 0.0417)

(b) Lower half of datasets

Method	**Mean**	**Standard Deviation**	**Paired t-test vs Single** (t, p)
Single	0.6292	0.1583	—
Random, $r = 0.1$	0.5561	0.1845	(-2.88, 0.0059)
Random, $r = 0.4$	0.5504	0.1866	(-3.17, 0.0026)
Random, $r = 0.7$	0.5554	0.1927	(-2.94, 0.0050)
Boosted	0.5682	0.1840	(-3.69, 0.0006)

(c) Upper half of datasets

Table 4: F_1-score

Method	**Mean**	**Standard Deviation**	**Paired t-test vs Single** (t, p)
Single	0.5825	0.1351	—
Random, $r = 0.1$	0.5996	0.1399	(1.10, 0.2730)
Random, $r = 0.4$	0.5952	0.1443	(0.81, 0.4203)
Random, $r = 0.7$	0.5966	0.1469	(0.91, 0.3674)
Boosted	0.6032	0.1421	(1.66, 0.1009)

(a) All datasets

Method	**Mean**	**Standard Deviation**	**Paired t-test vs Single** (t, p)
Single	0.5318	0.0967	—
Random, $r = 0.1$	0.6276	0.0807	(6.54, 0.0001)
Random, $r = 0.4$	0.6232	0.0926	(5.68, 0.0001)
Random, $r = 0.7$	0.6226	0.0928	(5.89, 0.0001)
Boosted	0.6214	0.0956	(5.25, 0.0001)

(b) Lower half of datasets

Method	**Mean**	**Standard Deviation**	**Paired t-test vs Single** (t, p)
Single	0.6331	0.1484	—
Random, $r = 0.1$	0.5717	0.1762	(-2.71, 0.0092)
Random, $r = 0.4$	0.5671	0.1774	(-3.01, 0.0041)
Random, $r = 0.7$	0.5706	0.1821	(-2.78, 0.0077)
Boosted	0.5849	0.1750	(-4.00, 0.0002)

(c) Upper half of datasets

Table 5: Average Disagreement Rate

Method	**Mean**	**Standard Deviation**
Random, $r = 0.1$	0.3703	0.1119
Random, $r = 0.4$	0.3774	0.1089
Random, $r = 0.7$	0.3902	0.1045
Boosted	0.4245	0.0801

(a) All datasets

Method	**Mean**	**Standard Deviation**
Random, $r = 0.1$	0.3885	0.0751
Random, $r = 0.4$	0.3949	0.0697
Random, $r = 0.7$	0.4080	0.0588
Boosted	0.4381	0.0457

(b) Lower half of datasets

Method	**Mean**	**Standard Deviation**
Random, $r = 0.1$	0.3522	0.1370
Random, $r = 0.4$	0.3600	0.1351
Random, $r = 0.7$	0.3723	0.1332
Boosted	0.4109	0.1019

(c) Upper half of datasets