DFA Learning: Minimal Models from Subsamples vs. Heuristic Models from Full Data

Matei Hristodorescu

Supervisor(s): Sicco Verwer, Simon Dieck

EEMCS, Delft University of Technology, The Netherlands

June 22, 2025

Abstract

Learning deterministic finite automata (DFAs) from labeled traces is a key problem with applications in software analysis and system modeling. SAT-based methods are effective but can be slow when dealing with large datasets. To address this, we propose a sampling method that selects a smaller, but still representative set of traces. Our approach groups traces with similar suffixes and uses edit distance to choose diverse examples. The proposed sampling performs better than random uniform sampling and significantly better than heuristic algorithms.

1 Introduction

A deterministic finite automaton (DFA) is a well-known language model that can be used to recognize a regular language. Identifying the smallest DFA that is consistent with a given set of labeled traces is a well-studied problem in grammatical inference [1]. DFA learning has applications in various fields such as computational linguistics, bioinformatics, speech processing, and verification [2].

Finding the smallest consistent DFA can be a very difficult problem. It has been shown that finding a consistent DFA of fixed size is NP-Complete [3]. While optimal algorithms exist, due to the computational complexity of the problem, heuristic algorithms have also been developed.

According to Heule and Verwer [2], challenging problems, such as those in the Abbadingo challenge problem set [4], are too large for current state-of-the-art SAT solvers. To solve such problems, heuristic methods, such as EDSM, are required. EDSM is a heuristic that tries to find a local optimum. Thus, whilst EDSM is generally faster than an optimal method, it is guaranteed to find the global optimum only given infinite data [4].

The encoding used in [2] needs $O(n^2 * |V|)$ SAT clauses, where n is the size of the identified DFA and V is the training set of labeled traces. This means that the time in which we solve a problem is directly correlated to the size of the training set. Thus, eliminating traces that do not grant information from the training set improves the running time of the algorithm, potentially allowing us to use the SAT solver to solve problems which we could not solve before.

As shown by Smetsers, Moerman, and Jansen [5], a minimal characteristic sample exists, i.e., a minimal set of traces that contains enough information to uniquely identify a DFA. Unfortunately, identifying the minimal characteristic sample requires prior knowledge of the DFA, so we cannot identify this sample and use it to construct the optimal DFA. In this paper, we develop 2 heuristics in which we try to approximate the characteristic sample as closely as possible and test them against each other, against random sampling, and against a model learned with EDSM on the whole dataset. To do this, we use an intuition that can be derived from the Myhill-Nerode theorem, namely the idea that in order to merge two states in a DFA, they must not have any distinguishing extension, i.e. all possible extensions lead to equivalent states (accepting or rejecting). From this, if there are many traces that end with the same suffix in our training set, we remove some of them, trying to keep the most distinguishing ones.

Our experiments show that DFA models minimised from samples extracted by the two heuristics that we propose perform significantly better than models that are created from the whole dataset using the EDSM heuristic. Additionally, they perform marginally better than models obtained from random sampling.

2 Literature Review

In this section, we present the two primary approaches to identifying DFAs: heuristic methods, such as EDSM, and optimal methods based on SAT encodings. We then look at the Myhill-Nerode theorem and some of its consequences that will help us with subsampling. Finally, we introduce the Levenshtein distance which will be used as a metric for quantifying the similarity between traces.

2.1 DFA Identification

A **Deterministic Finite Automaton (DFA)** is formally defined as a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite set of input symbols (the alphabet), $\delta : Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of accepting (final) states [9]. Given a finite set of positive and negative sample strings, referred to as the input sample, the goal of DFA identification is to find the smallest DFA which is compatible with those strings.

A state-merging algorithm works by first constructing a special tree-shaped DFA, called an augmented prefix tree acceptor (APTA), from the input sample, and then merging the states of this APTA. In an APTA, two strings reach the same state only if they share the same prefix until that state, hence the name prefix tree. An APTA is augmented because it contains states for which it is not yet known whether they are accepting or rejecting. In an APTA, merging two states means creating a single state that inherits all the transitions of the original states. A merge is only allowed if the two states are both either accepting or rejecting. In a state-merging algorithm, merges are iteratively performed until no more merges are possible [2].

The evidence-driven state merging (EDSM) algorithm is currently the most successful heuristic algorithm for DFA identification. It uses a simple scoring heuristic to determine which merge it should perform next[4]. More advanced variants of the algorithm exist, where search techniques are used in order to explore other paths than the one given by standard EDSM [11][12].

In [2], Heule and Verwer propose another way in which we can identify a DFA. Based on a reduction of DFA identification to a graph coloring problem [8], they reduce the problem into an instance of the satisfiability problem (SAT). In the graph coloring translation, a distinct color is used for every state of the identified DFA, whilst the nodes in the graph coloring instance represent the labeled traces. Two nodes are connected if merging them would result in an accepting state being merged with a rejecting state. Directly encoding the graph coloring constraints into a SAT instance results in $|Q|^2|V|^2$ clauses, where Q is the set of states of the DFA and V is the set of training labeled traces. Heule and Verwer use auxiliary variables to represent the problem more efficiently, resulting in $|Q|^2|V|$ clauses. They additionally apply symmetry breaking and add redundant clauses to improve the runtime of the algorithm.

2.2 The Myhill-Nerode Theorem

This subsection is based on [6].

Definition Let $L \subseteq \Sigma^*$ be any language over the alphabet Σ. For $x, y \in \Sigma^*$, we call $z \in \Sigma^*$ a *distinguishing extension* of x and y if exactly one of xz and yz is in L (where xz is the concatenation of x and z). If such a z exists, we say that x and y are *distinguishable* by L; otherwise we say x and y are *indistinguishable* by L.

Suppose L is a regular language recognized by a DFA D. Let $Q_D(s)$ be the state D is in after reading s. If $Q_D(x) = Q_D(y)$, we will have $Q_D(xz) = Q_D(yz)$ for any word z, so that D either accepts both xz and yz or rejects both. On the other hand, even if $Q_D(x) \neq Q_D(y)$, it is still possible for x and y to be indistinguishable by L. In this case, the two states are equivalent, and can be combined into one without changing the behaviour of the DFA.

Proposition 1. Let $L \subseteq \Sigma^*$ be any language over the alphabet Σ. Define $\sim_L$ to be the relation on Σ^* where x$\sim_L$y iff x and y are indistinguishable by L. Then $\sim_L$ is reflexive, symmetric, and transitive so that $\sim_L$ is an equivalence relation on Σ^*.

An equivalence relation $\sim$ on a set S induces equivalence classes which partition S into subsets of elements related to each other by $\sim$.

Proposition 2. Let L be regular and let D be its minimal DFA with states $\{q_0, q_1, \ldots, q_n\}$. We can then systematically define the equivalence classes of $\sim_L$ to be the sets $S_i = \{w \in \Sigma^* \mid Q_D(w) = q_i\}$.

It follows from Proposition 2 that if a DFA D has no states which are equivalent to each other, then x and y are in the same equivalence class iff $Q_D(x) = Q_D(y)$ and that D has the least amount of states possible.

We can now state the following theorem:

Theorem Let $L \subseteq \Sigma^*$ be a language. Then L is regular if and only if the number of equivalence classes of $\sim_L$ is finite. Furthermore, if L is regular, then the number of equivalence classes of $\sim_L$ is also the number of states in the minimal DFA for L.

2.3 Levenshtein Distance

This subsection is based on [7].

Definition The Levenshtein distance or the minimum edit distance between two strings is defined as the minimum number of edit operations (insertion, deletion, substitution) needed to transform one string into the other.

For example, the minimum edit distance between the words *kitten* and *sitting* is 3. We can turn the k into an s (sitten), turn the e into an i (sittin) and finally add a g to the end (sitting).

3 Methodology

In this section, we present a key takeaway from the Myhill-Nerode theorem, which forms the basis of our sampling algorithm. We then present two variations of the sampling algorithm, called Binary Search k Sampling and Dynamic k Sampling.

3.1 Sampling Algorithm

From the Myhill-Nerode theorem, we know that in order to merge two states in a DFA, they must have no distinguishing extension, i.e., all possible extensions lead them to equivalent states. From this, we can draw the intuition that traces which end in the same substring and produce the same output are similar. Specifically, if we have two traces xz and yz that produce the same output, it is likely that, in the minimal DFA, x and y lead to the same state, meaning that there is substantial overlap in the information encoded by those two traces. This means that removing one of the two traces from the training sample is unlikely to affect the final DFA.

Based on this intuition, we can derive the following sampling algorithm:

1. Split the data into accepting and rejecting traces.
2. Find an integer value k.
3. Split the traces into groups, where the strings in each group end in the same k characters.
4. Sample from these groups.

The first three steps can be seen in Algorithm 1.

Choosing the right value for k is essential to the sampling algorithm described above. If it is too large, it will result in too many groups, rendering our sampling ineffective. If the value is too small, the resulting groups are not representative enough. We propose two different methods for choosing k, both of which will be evaluated experimentally.

3.2 Dynamic k Sampling

This method starts with a fixed value of k. We take 10% of the required samples, then increment k by 1 and repeat until we have enough samples. We only increment k by 1 in order to avoid creating a small number of groups too quickly. Incrementing by another value might also skip over values that capture important patterns in the data, leading us to miss useful traces during sampling.

3.3 Binary Search k Sampling

In this method, we choose a target number of groups and perform a binary search on k to find the closest value that gives approximately that number of groups. As we want multiple samples from each group, the target number of groups should be lower than the target number of samples. For this paper, we set the number of target groups to be equal to the number of target traces divided by 5. We do this because we want to take five samples from each group. If any of the resulting groups has fewer than five traces, we supplement the sample with diverse traces from other groups. This method can be seen in Algorithm 3.

When selecting samples from each group, we aim to ensure that the traces are as different as possible from each other. To achieve this, we compute the Levenshtein distance between each pair of traces. We then select the five traces with the highest minimum distance to any other trace. This method is shown in Algorithm 2.

Algorithm 1: Group by suffix

```
Input: A set of traces $T$,where each trace is a tuple (label, length, string) and an
       integer $k$
Output: A dictionary mapping (label, suffix) to a list of traces
1 Initialize empty map groups from (label, suffix) to list;
2 foreach $(label, length, string)$ in $T$ do
3     if $k \leq$ length of trace then
4         $key \leftarrow$ last $k$ characters of $trace$;
5     else
6         $key \leftarrow trace$;
7     Append $(label, length, trace)$ to groups[$(label, key)$];
8 return groups;
```

Algorithm 2: Choose Diverse Traces

```
Input: A list of traces $T$
Output: A list of sampled traces $S$
1  if $|T| \leq 5$ then
2      return $T$;
3  Initialize empty list $S$ ;
4  for $i \leftarrow 0$ to $|T| - 1$ do
5      ; Initialize $min\_dist \leftarrow +\infty$;
6      for $j \leftarrow 0$ to $|T| - 1$ do
7          if $j \neq i$ then
8              Compute $d \leftarrow$ Levenshtein.distance($T[i], T[j]$);
9              if $d < min\_dist$ then
10                 $min\_dist \leftarrow d$;
11     Append $(min\_dist, T[i])$ to $S$;
12 Sort $S$ in descending order by $min\_dist$;
13 return the first 5 elements of $S$;
```

Algorithm 3: Binary Search for Optimal k

```
Input: A list of traces $T$, target number of samples $n$
Output: An integer value $k$
1  $low \leftarrow 1$;
2  $high \leftarrow$ max(length of $trace \in T$);
3  $target\_groups \leftarrow n/5$;
4  while $low < high$ do
5      $mid \leftarrow \lfloor (low + high)/2 \rfloor$;
6      if $|group\_by\_suffix(T,\ mid)| > target\_groups$ then
7          $low \leftarrow mid + 1$;
8      else
9          $high \leftarrow mid$;
10 return $low$;
```

4 Experimental Setup

To test the accuracy of different sampling methods, we use a custom dataset of problems, generated using the code[1] created by Max Pieters, which implements the trace generating method from the StaMInA competition [10]. We generated DFAs of size 9 to 13, with 50% of the final states being accepting and with the size of the training set + testing set equal to $2 \cdot |Q|^2$, where Q is the set of states in the DFA. Furthermore, the dataset is balanced such that 50% of the traces are accepting and 50% are rejecting. On these datasets, we perform 5-fold cross-validation. We split a dataset into 5 parts, using 4 of them as the training set with the remaining one as the testing set. We repeat this process 5 times, such that each of the 5 folds is used once as the testing set. The generator we use ensures that a data point

[1]The code can be found at:`https://github.com/Max-Pieters/DFA-GeneratorFunctions`

cannot appear more than once in a set, therefore duplicate traces are not a concern. This yields 25 instances on which we will test the sampling methods.

We will use classification accuracy as the metric for evaluation. Since the dataset is balanced with 50% accepting traces and 50% rejecting traces, we will use regular accuracy rather than balanced accuracy. We will compare the proposed sampling methods against random sampling and an instance of the EDSM algorithm. Sampling will be performed at various percentages of the full dataset, specifically 25%, and 50%. In order to further assess the effectiveness of the models derived from the sampling techniques, we will also test them on the entire training set from which the samples were taken. High accuracy on this set would suggest that most of the eliminated traces were redundant.

The datasets we will use to test these methods are balanced in terms of the distribution of accepting and rejecting states, as well as accepting and rejecting traces. Moreover, since our generator produces traces through random walks across the DFA, each state in the DFA should be represented fairly evenly. For this reason, we expect all sampling methods under evaluation, including random sampling, to perform reasonably well.

All experiments will be conducted using the FlexFringe library, with Glucose as the SAT solver. The tests will be run on an Intel i5-11400H at 2.70 GHz with 16 GB of memory, running Ubuntu 24.04. The standard random number generator from Python was used without setting a fixed seed.

5 Results

Table 1: Average accuracy of EDSM heuristic vs optimal learning with different sampling methods at 25% and 50%, on the testing set.

DFA Size	EDSM	Random25	Binary25	Dynamic25	Random50	Binary50	Dynamic50
9	75.4	60.4	51	54.4	99.2	98.5	100
10	74	64.4	66.8	65.4	93	97	95.6
11	97.6	70.2	57.2	61.6	89	96.8	97.8
12	79.4	75.4	73.4	72.6	90.6	94.8	89.6
13	77	72.2	51.4	72.6	97.4	98	97.4
Average	80.6	68.5	59.9	65.3	93.8	97	96.1

The results of the experiments are presented in Table 1. We can see that, no matter the chosen sampling method, taking 25% of samples is too sparse to get a good result. Sampling at 50% of the dataset seems to do better than EDSM with all three methods, with random sampling being the worst and binary search sampling being the best. To evaluate each method against the others and to make sure that the results are statistically significant, we can perform a p-test on each pair of the 4 methods: EDSM, random, binary and dynamic. Since we cannot assume anything about the distribution of our results, we use a Wilcoxon signed-rank test.

The improvements of all three 50% sampling techniques over EDSM are highly significant ($p < 0.001$), according to the p-values shown in Table 1. Random sampling performs noticeably worse than both binary and dynamic sampling when comparing the 50% methods among themselves ($p = 0.018$ and $p = 0.021$, respectively). Nonetheless, there is no statistically significant difference between dynamic and binary sampling ($p = 0.35$), indicating that both approaches function similarly well. All things considered, these findings demonstrate

Table 2: Pairwise Comparison of Sampling Methods (p-values)

	EDSM	**Random50**	**Binary50**	**Dynamic50**
EDSM	–	0.0002	0.00003	0.00003
Random50	–	–	0.018	0.021
Binary50	–	–	–	0.35
Dynamic50	–	–	–	–

Note: Values represent p-values from Wilcoxon signed-rank tests. Only upper triangle values are shown for clarity.

that employing a more knowledgeable sampling technique, such as binary or dynamic search at 50%, significantly beats EDSM and random sampling.

Another way to evaluate these sampling methods is by testing the models they generate on the entire training set from which the samples were drawn. A high accuracy should indicate that the traces that were removed were mostly redundant or less informative, meaning that the sampled subset effectively captures the patterns of the full dataset. We can see the results of this test in Table 3. All 3 sampling methods perform very well, with Binary Search Sampling getting the highest accuracy at 98.5%.

Table 3: Accuracy on Full Training Set for Different Sampling Methods

Sampling Method	**Accuracy (%)**
Random Sampling (50%)	96.9
Binary Search Sampling (50%)	98.5
Dynamic Sampling (50%)	97.6

6 Ethical Analysis

6.1 Reproducibility of Experiments

The DFAs used in our experiments were self-generated. To ensure transparency, we provide both the generator code and the resulting DFAs. The DFAs are available in both DOT and JSON formats, allowing for easy visualization as diagrams and parsing by validation tools. Additionally, the code used for data sampling and validation is published alongside this paper. The code and data can be found at `https://github.com/mateih/DFASampling`.

6.2 Ethical Considerations

Like any data-driven model, DFA learning can be misused. It could be used for harmful purposes such as user surveillance, behavioral manipulation, privacy invasion, or even helping attackers reverse-engineer systems for cyberattacks. Although DFA models are interpretable, the ethical responsibility depends on how they are used and the kind of data they are trained on.

7 Conclusions and Future Work

We presented two efficient ways in which we can reduce the training set of traces used for learning DFAs, whilst maintaining high accuracy. Using an implication of the Myhill-Nerode theorem, we can find groups of traces that provide similar information, allowing us to remove some of them from the training set. By doing this, we can speed up the learning process without losing too much accuracy. Our experimental results show that models minimized using our two proposed methods outperform those obtained through random sampling and the EDSM heuristic.

The datasets used in our experiments were balanced, which helped random sampling perform well. A possible next step is to try the experiments again using unbalanced datasets to check if the three sampling methods maintain their performance. We expect that in this environment, random sampling would perform significantly worse. Another experiment could be to measure how much time sampling actually saves in practice.

Although our methods show promise, our experiments also indicate that sampling too much can be very detrimental to the accuracy of the learned model. Thus, sampling is best used when we know that a dataset is dense, allowing us to remove some traces whilst still maintaining high accuracy. One potential improvement to our approach is to adapt the algorithm so that, instead of sampling at a fixed rate, it selectively removes samples it deems redundant with a certain confidence level. This is left as future work. This adaptation would make the algorithm more flexible, as it could also be used on sparser datasets.

References

[1] de la Higuera, C.: A bibliographical study of grammatical inference. *Pattern Recognition*, **38**(9), 1332–1348 (2005).

[2] Heule, M. J. H., Verwer, S.: Exact DFA identification using SAT solvers. In: *Lecture Notes in Computer Science*, pp. 66–79 (2010). DOI: 10.1007/978-3-642-15488-1_7

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

[4] Lang, K. J., Pearlmutter, B. A., Price, R. A.: Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In: *Lecture Notes in Computer Science*, pp. 1–12 (1998). DOI: 10.1007/bfb0054059

[5] Smetsers, R., Moerman, J., Jansen, D. N.: Minimal separating sequences for all pairs of states. In: *Lecture Notes in Computer Science*, pp. 181–193 (2016). DOI: 10.1007/978-3-319-30000-9_14

[6] Malkin, T.: *Handout 4: The Myhill-Nerode Theorem and Implications.* COMS W3261: Computer Science Theory, Columbia University (2022). Available at: `https://www.cs.columbia.edu/~tal/3261/fall22/handouts/4_MyhillNerodePlus.pdf`

[7] Jurafsky, D., Martin, J. H., Weizenbaum: *Speech and Language Processing.* (2025). Available at: `https://web.stanford.edu/~jurafsky/slp3/2.pdf`

[8] Coste, F., Nicolas, J.: *Regular inference as a graph coloring problem.* Workshop on Grammatical Inference, Automata Induction, and Language Acquisition, ICML 1997 (1997).

[9] Hopcroft, J.E., Motwani, R., Ullman, J.D.: *Introduction to Automata Theory, Languages, and Computation.* 3rd edn. Pearson (2006).

[10] Walkinshaw, N., Bogdanov, K., Holcombe, M., SalaÃŒn, G.: *STAMINA: a competition to encourage the development and assessment of software model inference techniques.* Empirical Software Engineering **17**, 605-638 (2012). https://doi.org/10.1007/s10664-012-9210-3.

[11] Oliveira, A.L., Marques-Silva, J.P.: Efficient search techniques for the inference of minimum sized finite state machines. In: *Lecture Notes in Computer Science*, pp. 81–89 (1998).

[12] Abela, J., Coste, F., Spina, S.: Mutually compatible and incompatible merges for the search of the smallest consistent DFA. In: Paliouras, G., Sakakibara, Y. (eds.) *ICGI 2004, Lecture Notes in Artificial Intelligence*, vol. 3264, pp. 28–39. Springer, Heidelberg (2004).