Testing The Performance Of Minimal Models Trained On Sparse Data

Using Optimal Methods

Max Pieters
Supervisor(s): Sicco Verwer, Simon Dieck
[1]EEMCS, Delft University of Technology, The Netherlands

June 24, 2025

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 26, 2025

Name of the student: Max Pieters
Final project course: CSE3000 Research Project
Thesis committee: Sicco Verwer, Simon Verwer, Soham Chakraborty
An electronic version of this thesis is available at http://repository.tudelft.nl/.

Abstract. Deterministic Finite Automata (DFA) learning is the problem of reconstructing a DFA from its traces. For the development of methods for this problem, randomly sampled data is often used to train and test the performance of models. The choice of sampling technique can result in data sets with unforseen properties. The technique used in the STAMINA competition is such that that the number of final states and the size of alphabet were thought to potentially effect the test performance of resultant models. This was tested experimentally, by comparing the test performances of minimal models identified on traces from differently constructed DFAs. It was found that, although an increase in alphabet size results in overall longer traces that vary more with length, test performance still struggled. This shows that a DFA with a larger alphabet will need more traces than an equivalent smaller DFA. Additionally, it was found that the number of final states had a significant effect in the resulting test performance, and had a significant effect on the length of sampled traces. It was found that increased node count did not have an effect on sampled word length, and resulted in worse test performance.

1 Introduction

1.1 General Background

Deterministic Finite Automata (DFA) are defined by a set of states Q, an alphabet of used symbols $s \in \Sigma$, starting state $q_0 \in Q$, final states $F \subseteq Q$, and transition function $q, s \to q$. Every DFA accepts a regular language, and in turn, every regular language can be described by at least one DFA. A minimal model is the smallest possible DFA for a given regular language. As such models can describe real world systems, there are many desirable traits to a minimal model, namely simplicity and efficiency. The problem of converting DFAs into equivalent smaller counterparts is known as DFA minimization. For each regular language, there exists a unique minimal DFA[1].

The problem of learning a DFA from a set of traces is known as DFA identification. There are numerable practical applications for this, such as the modeling and of software systems [4]. Another example is the generation of attack graphs for networks to identify possible avenues of attack for malicious actors[2]. DFA identification is an NP-hard problem. Thus, it is often very difficult to learn a correct model in a timely manner[3]. As such, several methods have been developed to handle this problem, with varying strengths and specializations.

1.2 DFA Learning Methods

Several competitions have been held to develop DFA identification meethods, namely the Abbadingo One and STAMINA competitions. From these the EDSM and DFASAT methods emerged respectively [5] [4].

Broadly speaking, there are two categories of DFA learning methods. Optimal methods are guaranteed to learn the smallest model that is entirely consistent with the training data. Thus, given enough representative data, an optimal

method is guaranteed to find the correct model. Heuristic methods, on the other hand, estimate the likelihood that a state merge is correct. As such, it is prone to making incorrect merges if given sparse data, and can sometimes lead to a larger model than one trained heuristically. Applying the principal of Occam's Razor, a smaller, simpler model is generally expected to perform better than a bigger model. Additionally, heuristic methods tend to be significantly faster than optimal methods in practice.

All methods used in this research were from the software framework, FlexFringe, a state-of-the-art automaton learning package. It consists of both optimal and heuristic methods [8]. This should allow for easy comparison between methods.

The SATSOLVER mode of FlexFringe relies on reducing a DFA identification problem into a vertex colouring problem. This is run on a SAT Solver, and then translated back to a solution for DFA identification[11]. FlexFringe requires an external SAT solver to be provided. For this, Glucose was used, as it is internally considered the default choice of SAT solver [12].

EDSM is used for the heuristic method, to offer a point of comparison for models learned using the SAT method. To briefly summarize, it first generates an abstract syntax tree based off of the training data, then performs merging using a probabilistic estimation of which merges are likely to be valid[5]. It performs well on current benchmarks[6].

2 Hypothesis

STAMINA uses a simple procedure for sampling traces from some generating DFA $D_g = \{Q, \Sigma, \delta, q_0, F\}$. It takes from the one used for the Abbadingo One competition[5][4]. The sampling algorithm is relatively simple. Beginning from q_0, a random walk is performed, meaning that edges are uniformly selected at random. When in a terminal state $q_f \in F$, the walk is terminated with a probability of $1/(1 + 2 \cdot deg^+(q)$, where $deg^+(q)$ is the number of outgoing states from state $q \in Q$. This is repeated until the desired amount of positive traces is reached. Negative traces are created by performing modifications uniformly on positive traces at n different positions, where n is randomly sampled from the Poisson distribution with $\lambda = 3$. These operations include inserting, deleting, and replacing characters at some position. If the resulting trace, when simulated on D_g, ends up in an accepting state, the trace is not used.

There are some quirks to this procedure. Duplicate traces are allowed, which results in large data sets not necessarily being more representative than an equivalent set without duplicates. Another particular quirk of the STAMINA procedure is the probability of terminating a walk. Consider DFAs with $|\Sigma| = 1$. Due to this subset of DFAs describing simple languages, with no branching behavior, one would expect them to be easy to derive representative traces for. Consider the DFA in figure1. It has a relatively high proportion of final states. In order for all states and transitions to be captured by the data set, the sampling method needs to produce at least 1 trace with length $l > 10$.

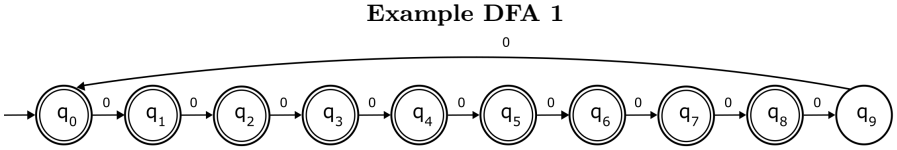

Fig. 1. DFA that rejects all words of length 10, $|\Sigma| = 1$

Due to the relatively simple nature of this DFA, we can calculate the exact likelihood that a trace will have this length. Given that this DFA is complete, $deg^+(q) = |\Sigma| \ \forall \ q \in Q$. As this is the case for D_1, and $|\Sigma| = 1$, the probability p of terminating in some final state is $1/(1 + 2 * 1) = 1/3$. This effectively makes the likelihood of this procedure terminating after some amount of final states follow a the geometric distribution. Using the geometric cumulative distribution function $1 - (1 - p)^x$ [13], we can expect that approximately 1.27% of sampled traces will have lengths longer than 10. Although this DFA should require very few traces to be learned correctly, the STAMINA sampling procedure would likely need to produce significantly more traces in order to be representative.

Additionally, the inverse language provides some hurdles for producing a representative sample. As the STAMINA procedure applies insertion, selection, and modifications of symbols in a word uniformly, words can be expected to vary binomially in length. Non-terminal states that are very far from any final state would thus be less likely to be included when sampled.

Due to the sampling procedure relying on final states to produce positive traces, and on positive traces to produce negative traces, the number of final states could have a significant impact on how likely a set of traces is to be representative.

Hypothesis 1: For two minimal models m_1 and m_2, trained on n traces sampled from DFAs D_1, D_2 respectively, where D_2 is identical to D_1 except for $|F_1| < |F_2|$, m_1 is more likely to have a higher test performance than $m2$.

DFAs with larger alphabets may be expected to behave differently from those with $|\Sigma| = 1$. As this allows for branching behavior, there are multiple potential paths for reaching some final states. Additionally, the sampling procedure has a lower probability of terminating in any one states. We could expect average word length to be longer in DFAs with larger alphabets. This could, counter intuitively, make these DFAs generate more representative samples, and thus be more likely to result in a correctly identified model using fewer traces. It is difficult to say whether these observations are applicable based on reasoning alone for DFAs with larger alphabets and with different final state distributions. To check wether there is a significant statistical difference in correctly identifying a target model, an experimental methodology could be used. Specifically, we could check if certain properties of generating DFAs has a statistically significant effect on the properties of sampled traces and the test performance of their

resulting models. To compare between test performances, a two tailed paired T-test can be performed between the test performances of groups of models that differ by certain properties.

Hypothesis 2: For two minimal models m_1 and m_2, trained on n traces sampled from DFAs D_1, D_2 respectively, where D_1 is identical to D_2 except for $|\Sigma_3| = |\Sigma_4| - 1$, m_1 is likely to have comparable test performance to m_2. It can be expected that traces from D_2 will be longer than those from D_1, and will therefor be more useful in identifying the target DFA. It can also be reasoned that D_2 would require required more information to be identified correctly, as it has $|Q|$ additional edges that need to be learned.

Furthermore, we can expect that an increase in the number of states will have an adverse effect on the resulting test performance. We can expect the distributions of word lengths to be similar as DFA size increases, assuming the proportion of final states remains consistent.

Hypothesis 3: It can be expected that the test performance of models identified from traces more states DFAs will be lower than equivalent models attempting to identify DFAS with fewer states. This is assuming amount of traces provided remains constant.

Finally, it can be expected that heuristically trained models will perform worse than equivalent models trained using optimal methods, due to optimally trained models likely being smaller.

Hypothesis 4: Models trained using optimal methods will on average have a better test performs than equivalent models trained using heuristic methods.

3 Experimental Methodology

3.1 DFA Generation

For the purposes of this paper, it was paramount that the method used to generate DFAs be versatile and configurable, and scale to any desired size of alphabet and number of states. A desired quality was the ability to produce models of an exact size. Additionally, the number of final states should be flexibly configurable. Furthermore, a complete DFA should be produced. Neither the STAMINA DFA generating procedure, nor the one described in Abbadingo fit these requirements [5] [4]. As such, the choice was made to use a procedure devised in Almeida et al [7]. The procedure to generate the skeleton of a random DFA D with N states with alphabet Σ_{DFA} is as follows:

1. Create a starting state q_0, and add to Q.
2. Uniformly sample some state $q \in Q$.
3. If $|OutgoingEdges(c)| = |\Sigma|$, go to step 2.
4. Select a symbol $s \in \Sigma \setminus deg^+(q)$.
5. Create new state $\overline{q}$, and add to Q.
6. Add transition $q \times s \to \overline{q}$ to δ.
7. If $|Q| < N$, repeat from step 2.

Due to this construction, every states is reachable from q_0. In order to convert it to a complete DFA, the following procedure is applied to the previously generated incomplete DFA D:

1. Repeat bellow steps for each state $q \in Q$
2. For each $s \in \Sigma \setminus OutgoingEdges(q)$, uniformly sample state $\overline{q} \in Q \setminus \{q\}$.
3. Add transition $q \times s \to \overline{q}$ to δ.

Lastly, the following procedure is used to select n_F accepting states for D. A particular strength of this procedure is that is can be applied to existing DFAs to generate near identical automata with additional final states.

1. Uniformly selected some state $q \in Q \setminus F$
2. Add q to F.
3. If $|F| < n_F$, repeat from step 1

A weakness of this procedure is that the resultant DFA is not guaranteed to be minimal. In practice, DFA size among randomly sampled DFAs strongly correlates with state complexity, or the number of states in the equivalent DFA **??**. For the purposes of this paper, the size of the produced DFA is considered to be a good proxy for it's state complexity.

3.2 Sampling Training/Testing Data

To create training and testing data, the generated DFAs are sampled from. The procedure for this is based off of the one described in Stamina, with minor modifications [4] The procedure sampling accepting words W_+ from some generating DFA D_g is as follows:

1. Create set $W_+ = \{\}$ for words that D_g accepts.
2. Create state variable q as the currently selected state of the DFA. Let word $w = \lambda$, representing the path taken so far while traversing D. λ entails an empty word.
3. $q = q_0$ and $w = \{\}$
4. If $q \in F$, terminate the walk with probability $1/(1 + 2 \cdot deg^+(q))$ and add w to W. If $|W| = n^+$, stop.
5. Uniformly sample $s \in OutgoingEdges(q)$, and update $q = \delta(q, s)$, and append s to the end of w. If no states are selectable, go to step 3.
6. Repeat from step 4.

A modified version of the STAMINA procedure is used to generate negative traces. To create a set of negative traces W_-, the following procedure is applied:

1. Cycle through words $w \in W_+$
2. Sample $p = Poisson(3)$.
3. Uniformly select p positions in w
4. For each position, randomly uniformly select between the removing, adding, or replacing operations. Apply these at all positions. For the later two of these, uniformly select the new symbol $a \in \Sigma$.
5. Simulate this newly modified word w_- on D_g. If rejecting, add to negative trace set.
6. Repeat from step 1 until n^- traces have been sampled.

The original procedure described in STAMINA did not provide control over the amount of these traces generated. It would only iterate through positive traces once. As such, the number of negative traces would be $0 \leq n^- \leq n^+$. As the focus of this research is on what words are sampled using the STAMINA procedure, producing little to no traces was not deemed to be insightful. As such, this modified procedures allows for generating any number of positive traces by setting n^+, n^-.

3.3 Generated Data Sets

For each claim stated in section 1.2, a data collection was generated to test whether this claim holds. They are:

1. Data collection G, consisting of 100 sets. A set consists of three DFAs with $|Q| = 10, |\Sigma| = 2$, that are identical in construction, with the exception of their number of final states being 2, 5, and 8 respectively. We would expect the average test performance of DFAs with $|F| = 2$ and $|F| = 8$ to be normally distributed around $|F| = 5$. A T-test was performed to check whether these test performances differed significantly as $|F|$ changed.

2. Data collection A, consisting of 100 sets. A set consists of three DFAs of with $|Q_| = 10$, with $|\Sigma| \in \{1, 2, 3\}$. DFA 2 is constructed from DFA 1 by adding new transitions to randomly selected states, until a new complete DFA is derived. This is repeated to create DFA 3. A T-test was performed to see whether the test

performances for DFAs of their respective alphabet size differed significantly. As DFAs with larger alphabets are expected to have longer traces, but require more information to correctly learn, it could be presumed that resultant models would not differ significantly in their test scores.

3. Data collection N, consisting of 8 sets. A set consists of 12 DFAs are generated, with $|Q| \in \{3, \cdots, 10\}$ for each set. $|F| = |Q|/2$. Test performance is expected to decrease as $|Q|$ increases.

Each set consists of 100 traces. The proportion of positive to negative traces sampled is equal to the proportion of final to non-final states in the DFA. Fixing this data set size enables the results to be more readily comparable. This size was chosen due its relative ease to learn models from, especially for optimal methods. As this research is limited in computational resources, the choice was made to design the experiments around easily computable sizes.

3.4 Model Identification And Testing

The experiment itself consisted of FlexFringe being run on every data set. A particular property of the SAT solver mode of FlexFringe is that it does not necessarily find a minimal DFA for a certain data set. Instead, it attempts to find a DFA of a certain size, specified with the "satoffset" parameter. The given SAT solver evaluates whether the translated problem is satisfiable, and if yes, returns a DFA of that size if it is. As such, finding a minimal DFA requires some amount of searching. The following procedure was used to find the minimal optimally trained DFA for some training set T.

1. Let N_H equal the number of states in the generating DFA D_G.
2. Let $\overline{Q}$ be the expected number of nodes for the model. For example, a data set generated using a DFA with $|Q| = 10$ will have an expected size of $\overline{Q} = 10$.
3. Let $N_O = min(\overline{Q}, N_H)$.
4. Run FlexFringe using the SAT solver mode, with parameter 'satoffset' set to N_H.
5. If the SAT solver indicates satisfiability, repeat step 3, decrementing $N_O = N_O - 1$ each time, until satisfiable solution is found.
6. Else, if the SAT solver indicates unsatisfiability, repeat step 3, incrementing $N_O = N_O + 1$ each time, until a satisfiable solution is found.
7. Output the most recent generated model that indicated satisfiability.

Alternative methods for finding the size of the minimal model include incrementing from some lower bound until a satisfiable solution is found, or performing a binary search within some range. Ultimately, this method was chosen, as at large training set sizes, $N_H \approx N_O$, under the presumption that large training set sizes with result in both learned models being close in size to the generating DFA. This minimized how many times FlexFringe was called on any given data set, although whether this method is significantly better than binary search is unknown.

The exact commands and parameters used for FlexFringe were as follows:

.FlexFringe <path to training file> –ini=ini/edsm.ini –mode=satsolver –satoffset=N_O

.FlexFringe <path to training file> –ini=ini/edsm.ini

Where 'mode' indicates whether the SAT solver is used. If not set, the heuristic mode defined by 'ini' is used. This is a required parameter, and indicates what heuristic will be used, when mode is not set. 'satoffset' indicates the DFA size FlexFringe will attempt to find a model for. By changing the size of this parameter, searching can be performed to find the a size for N_O until a satisfiable model is found. [12]

5-fold cross validation was used to test the performance of generated models. This should provide a small amount of bias between the training and testing set [10]. Thus for each data set sampled from a generated DFA D, 80% of words in $W \cap \overline{W}$ are assigned to the training set, with the remainder assigned to the test set. The procedure for calculating the test performance of the trained models is as follows:

1. Run FlexFringe with the relevant configuration on the training data $w \in S_{train}$ to learn model m.
2. Simulate each word $w \in S_{test}$, on m.
3. Calculate the ratio of words that are simulated correctly, from 0.0 to 1.0.

4 Results

The T-Test score between A_1 and A_2 was calculated to be 0.0. For A_2 and A_3, this was calculated to be 510^{-8}. The T-Test score between test performances of DFAs with $|F| = 2$ and $|F| = 5$ was ≈ 0.4. For $|F| = 5$ and $|F| = 8$, this was $\approx 1.410^{-12}$.

Table 1. Properties of models with increasing size for N

$\|Q\|$	3	4	5	6	7	8	9	10
Avg. Word Length	8.512	10.325	9.429	9.133	9.621	8.657	9.726	10.027

Table 2. Properties of models with increasing size for Σ

$\lvert\Sigma\rvert$	1	2	3
Average Test Performance	0.991	0.905	0.835
Average Word Length	2.333	4.567	6.698
Std. Deviation Word Length	0.388	0.771	1.069

Table 3. Properties of DFAs with increasing size for F

$\lvert F\rvert$	2	5	8
Average Test Performance	0.682	0.667	0.838
Average Word Length	20.659	9.626	5.578
Std. Deviation Word Length	14.914	5.042	1.883

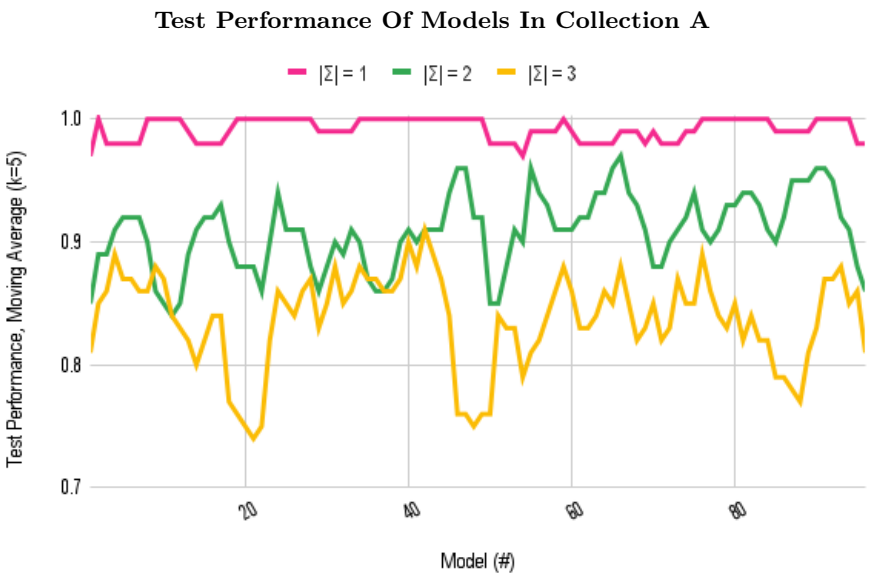

Fig. 2. Test performance of minimal models, grouped by alphabet size

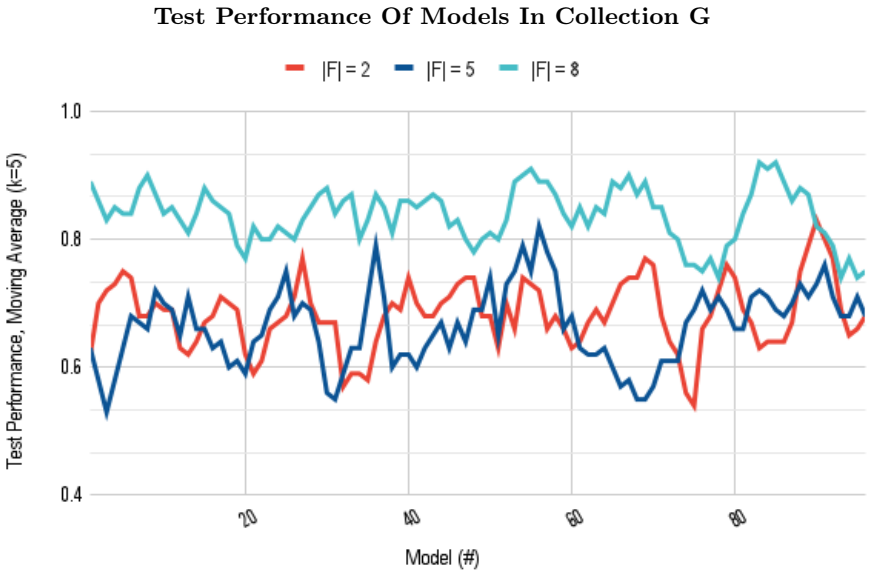

Fig. 3. Test performance of minimal models, grouped by final state count

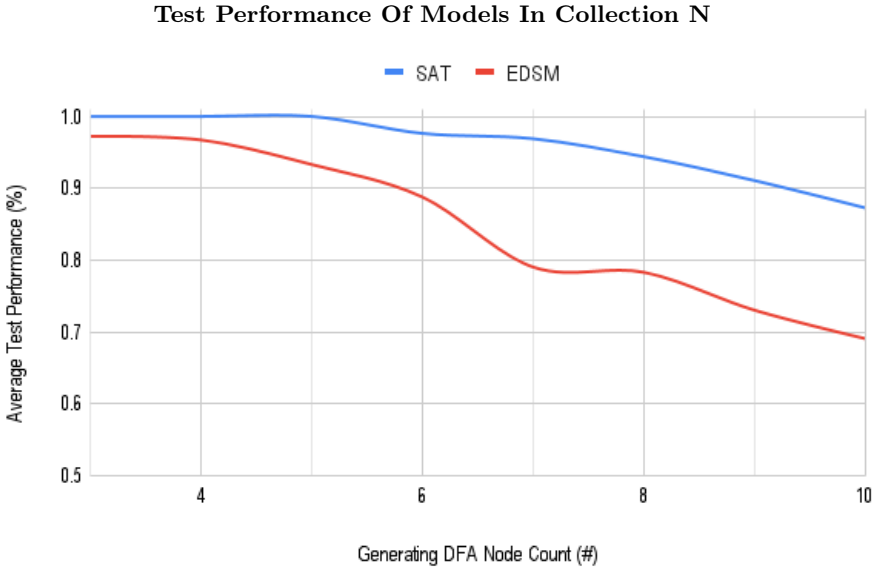

Fig. 4. Average test performs of models trained using the SATSOLVER and EDSM mode of FlexFringe as DFA node count increases.

5 Analysis

Models from A_1 has a noticeably higher test performance than equivalent models in A_2 and A_3. This difference was statically significant, due to their very low T-test scores. In addition to this, the average word length noticeably increased as the alphabet size of the generating DFA increased [Table 2], as expected. From this, we could conclude that the increased complexity of learning a model with a larger alphabet outweighs the benefit of longer, more informative traces, with this particular DFA construction and distribution of words. However, upon visual inspection, several models in A_1 did not accurately reflect their generating model. Take the pair of models in fig:2.

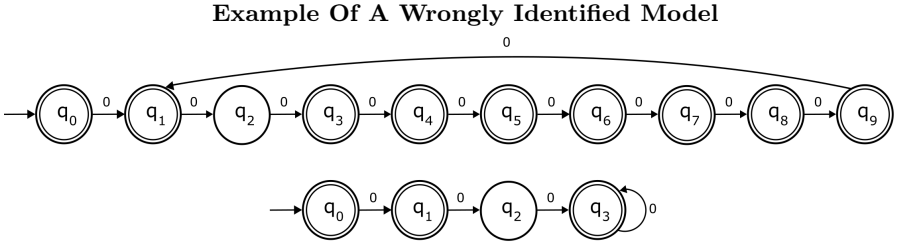

Fig. 5. Top: target model Bottom: trained model

Of the 84 models with a test performance of 1.0, 11 described a different language than their generating DFA. We can conclude that these models are over fitted, and could result in a lower test score when applied to additional data. Due to this flaw, the results of this experiment could be considered to be inconclusive

For data collection G, the results were rather unexpected. The test performances between models with $|F| = 2$ and $|F| = 5$ did not differ significantly, with a high t-test score. By contrast, between models with $|F| = 5$ and $|F| = 8$, there was a statistically significant difference. This difference can be seen in Fig. 3. This went contrary to the expected distribution of test scores, with $|F| = 5$ expected to perform best overall, and $|F| = 2$, $|F| = 8$ expected to perform similarly, and worse to some extend. This suggests that the proportion for final states does indeed have an impact on the test performance of the trained model, but that this follows some other distribution than a normal distribution centered around $|Q|/2$. Were we to speculate why this particular quirk in the procedures for STAMINA and Abbadingo was night accounted for, we might conclude that, in their DFA generation algorithms, all states have $p = 0.5$ chance of being a final state. This would result in the amount of final states being binomially distributed, with a mean of $|Q|/2$. As such, the discrepancy between the amount of traces need between DFAs with a large and small amount of final state could go unnoticed, or could be considered to be unimportant. For the purposes of

software model synthesis, this distribution of final states might not necessarily be reflective of reality. As such, other research might attempt to use a different DFA generation procedure that leads to a wider spread in the proportion of final states. It is the opinion of this paper that any further research that seeks to build on STAMINA should consider whether the sampling procedure, as described, is suitable for their own purposes, given the potential limitations found.

Furthermore, for data collection N, the average test performance of models decreased as the number of states in the generating model increased. Additionally, models trained heuristically consistently scored worse than their optimally trained counterparts. There were several cases of the heuristically trained models with equal test performances. Additionally, there were 2 cases where the heuristically trained model had a better test performance than it's optimally trained counterpart (Fig 4). This indicates that a while heuristically trained can perform than an optimal model learned on the same data, this is generally uncommon.

5.1 Recommendations For Further Research

The research in this paper looks at DFAs at particular sizes, and of particular constructions. Whether these findings, particularly the influence on final state count, are applicable for significantly larger DFAs, or for non-complete DFAs, is uncertain. This could further inform the design of DFA sampling techniques. Additionally, this research relies upon the sampling procedure described in STAMINA. As such, the results cannot necessarily be expected to be transferable to other sampling techniques. Examining whether other sampling procedures produce similar results could provide useful insight for creating more representative sampling techniques.

6 Ethical Considerations

In this section, several ethical points of contention are explored, as well as describing the steps taken to deal with them.

6.1 Ethics of DFA Identification

Many applications of machine learning, which DFA identification falls under, deals with certain ethical issues. These include, but are not limited to:

Misuse. While any particular model or model learning technique might not in itself be harmful, potential applications could be nefarious. As this paper does not look at any particular application for DFA identification, this is not an issue. Any person who does seek to apply any of the findings and techniques within this paper has the responsibility to do so ethically.

Power Consumption. Identifying DFAs can be very expensive computationally at larger training set sizes, particularly for optimal methods. As this experiment looks at a small range of DFA sizes at limited sizes, and does not use particularly large data set sizes, this issues is largely mitigated. Nevertheless, were this scope to significantly increase, power consumption could increase drastically. Future research should keep this in mind.

6.2 Reproducibility of Experiment

All data used in this paper, namely DFAs and their sample traces, was randomly generated. As this paper does not rely on any proprietary data set, the methods used for data generation should suffice. Furthermore, the exact implementation of DFA generation and sampling are publicly available at <>. Both FlexFringe and Glucose are open source under the GNU and MIT licenses respectively, and as such are freely available for the purpose of reproducing the work presented[12][14]. Performing the data generation described should be manageable on commonly available personal computers. DFA learning was, by far, the most resource intensive aspect of the experiment. Nonetheless, using the techniques described above should be manageable on common hardware.

6.3 Academic Transparency

For the sake of transparency, it should be noted that Sicco Verwer, who's work is cited several times throughout this paper, as well as being a contributor and maintainer for FlexFringe, also served as a supervising professor for this work. It was due to this that FlexFringe was suggest and ultimately used as the framework for this paper. Additionally, [9] was suggested as a starting point for further research. Besides the aforementioned instances, any further references to his work are entirely the choice of the author.

7 Conclusion

Several experiments were described to demonstrate whether certain generating DFA properties had an effect on the test performance of minimal models trained on their traces, using the sampling procedure used for the STAMINA competition. Traces were considerably longer when sampled from DFAs with larger alphabets. Nonetheless, target models with larger alphabets resulted in worse performing models, despite this increase in word length. Additionally, target models with more states were noticeably harder to learn. Target models with a larger proportion of final states had significantly higher test performance than equivalent models with fewer. Heuristic methods generally resulted in worse performing models when compared to optimal methods. These findings could prove to be informative for other researchers in designing or selecting DFA sampling techniques.

References

1. Mowtani, R. et al.: Equivalence and Minimization of Automata. In: Hirsch, M. (ed.) Introduction to Automata Theory, Languages, and Computation. pp. 155–167 Greg Tobin, Boston (1979).
2. Nadeem, A. et al.: Enabling Visual Analytics via Alert-driven Attack Graphs. Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. 21, 2420–2422 (2021). https://doi.org/https://doi.org/10.1145/3460120.3485361.
3. Gold, E.M.: Complexity of automaton identification from given data. Information and Control. 37, 3, 302–320 (1978). https://doi.org/https://doi.org/10.1016/S0019-9958(78)90562-4.
4. Walkinshaw, N. et al.: STAMINA: a competition to encourage the development and assessment of software model inference techniques. Empirical Software Engineering. 18, 4, 791–824 (2013). https://doi.org/10.1007/S10664-012-9210-3.
5. Lang, K.J. et al.: Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm. Springer.
6. Soubki, A., Heinz, J.: Benchmarking State-Merging Algorithms for Learning Regular Languages. Proceedings of Machine Learning Research. 217, 181–198 (2023).
7. Almeida, M. et al.: Enumeration and generation with a string automatarepresentation. Theoretical Computer Science. 387, 2, 93–102 (2007). https://doi.org/https://doi.org/10.1016/j.tcs.2007.07.029.
8. Verwer, S., Hammerschmidt, C. A.: flexfringe: A Passive Automaton Learning Package. In L. O'Conner (Ed.), 2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017 (pp. 638-642). IEEE. (2017). https://doi.org/10.1109/ICSME.2017.58
9. Heule, M.J.H., Verwer, S. Exact DFA Identification Using SAT Solvers. In: Sempere, J.M., García, P. (eds) Grammatical Inference: Theoretical Results and Applications. ICGI 2010. Lecture Notes in Computer Science(), vol 6339. Springer, Berlin, Heidelberg. (2010). https://doi.org/10.1007/978-3-642-15488-1_7
10. Kuhn, M., Johnson, K.: Applied Predictive Modeling. Springer (2013).
11. Costa Florêncio, C., Verwer, S. (2012). Regular Inference as Vertex Coloring. In: Bshouty, N.H., Stoltz, G., Vayatis, N., Zeugmann, T. (eds) Algorithmic Learning Theory. ALT 2012. Lecture Notes in Computer Science(), vol 7568. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-34106-9_10
12. Verwer, S., Baumgartner, R., Catshoek, T., Hammerschmidt, C., Tsoni, S.: FlexFringe, https://github.com/tudelft-cda-lab/FlexFringe.
13. Forbes, C.S. et al.: Statistical Distributions: Forbes/Statistical Distributions 4E, https://core.ac.uk/display/44256489.
14. Audemard, G., Simon, L.: On the Glucose SAT Solver. International Journal on Artificial Intelligence Tools. 21, 1, (2018). https://doi.org/10.1142/S0218213018400018.
15. Berend, D., Kontorovich, A.: The state complexity of random DFAs. Theoretical Computer Science. Volume 652, 102–108 (2016). https://doi.org/https://doi.org/10.1016/j.tcs.2016.09.012.