Chaining Heuristic and Exact Methods for DFA Identification

Vasil Chirov[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: Vasil Chirov
Final project course: CSE3000 Research Project
Thesis committee: Sicco Verwer, Simon Dieck, Soham Chakraborty

Abstract

This paper investigates a hybrid approach to deterministic finite automata (DFA) identification by combining heuristic (EDSM) and exact (reduction to SAT) methods. The hybrid strategy implies first partially identifying the DFA heuristically and then minimizing it with an exact method. Two implementations of the hybrid approach are tested - one using binary search on the number of states of the intermediate model, and one that adjusts the SAT offset to control its search space. The results obtained on datasets from the STAMINA competition show that while the hybrid approach reduces the size of the inferred models compared to EDSM, this does not necessarily translate to better test performance. Nevertheless, the methods used in this work demonstrate how a hybrid approach can be applied to infer more compact models in DFA identification.

1 Introduction

One of the most well-studied problems in grammatical inference is DFA identification. A deterministic finite automaton (DFA) is a model consisting of nodes, transitions, and an alphabet, that recognizes a certain regular language. DFA identification is the problem of identifying (learning) a minimal DFA that is consistent with a set of labeled examples. The inferred model needs to be minimal due to an important philosophical principal in problem-solving, known as Occam's razor, which recommends to search for the smallest possible set of elements. Generally, DFA identification asks the question: From which regular language is it most likely that the given set of labeled examples has been generated? DFA identification has many applications across various fields [9].

One practical and increasingly important application of DFA identification is in the field of software engineering, where the behavior of software systems can be specified using finite state machines. However, the process of creating and maintaining such models proves to be costly and inefficient and is therefore usually omitted during software development. One alternative approach is to derive these finite state machines by learning them from labeled software execution traces, generated actively (collecting input/output examples by interacting with the system) or passively (using already existing inputs/outputs from the system). This approach is called software model synthesis [3]. In software model synthesis, deterministic finite automata are usually used to represent the behavior of a system, and the process of inferring them is called DFA identification.

Different approaches to DFA identification exist. Normally, it is done heuristically, which helps to find a "good" (close to minimal) DFA that is consistent with the data effieciently. However, there are also exact methods for identifying minimal DFAs, which could be used to improve the performance of learning algorithms. The current state-of-the-art in DFA identification is the evidence-driven state merging (EDSM) algorithm - a heuristic-based algorithm aiming to find a local optimum efficiently, that won the Abbadingo competition in 1997, see [11].

Aside from heuristic-based algorithms, exact methods for DFA identification are also worth considering. Since the decision version of DFA identification has been shown to be an NP-complete problem [8], it can be translated into a more well-studied problem and its solver can be used to obtain a minimal DFA. Such a translation has been made from DFA identification into SAT, see [9]. This translation provides us with a way to find the exact identification of a DFA of minimal size, unlike the heuristic methods (e.g. EDSM) which find a close to minimal DFA. Of course, this comes at the cost of higher runtime due to the problem being NP-hard.

One way of combining the heuristic methods with optimal methods is to first partially learn the DFA from the input data heuristically and then feed the output as an input to an exact solver such as the SAT one discussed above. This approach could prove to be more effective in certain situations and improve the performance of DFA learning algorithms, which raises the question: *To what extent and in what ways does partially learning a model heuristically and then applying exact minimization - based on the decisions already made by the heuristic - affect model size and test-performance in DFA identification?*

This paper aims to answer the research question through an experiment, by implementing two variations of the hybrid approach to DFA identification and comparing their effectiveness against each other and against an entirely heuristic approach (i.e. EDSM). Our experiments show that while the hybrid approach manages to noticeably reduce the size of the inferred models compared to pure EDSM, this does not translate to consistently improving the test performance. Section 2 discusses the ethical considerations related to this work. In Section 3 the background on which this research is made is outlined. Section 4 describes in detail the experimental setup used to obtain the results of this work, while Section 5 presents the results and analyzes the important observations. Section 6 puts the experiment and the results in a broader context and compares them with previous results obtained with a similar hybrid approach. Finally, in Section 7, conclusions are made, and potential future work is suggested.

2 Ethical Analysis

This section discusses the ethical considerations related to this work.

Applications and impact

The techniques described in this work can be used for generating deterministic finite automata that act as surrogate models for software systems. Although this could have beneficial applications, such as improving software robustness, optimizing maintenance, or identifying bugs, it could also be misused for reverse engineering or discovering vulnerabilities in proprietary software. It is therefore important that these techniques are applied responsibly and within legal bounds.

Fairness and Dependability

The methods investigated in this work can be used for generating models that simulate the behavior of a given system. There is a significant possibility that incorrect models -

whether due to sparse data or algorithm limitations - get generated. This could lead to incorrect interpretations and potentially to safety risks if the models are used in safety-critical contexts such as finance and healthcare. Therefore, it is important to validate the correctness of these models, especially when they are used in safety-critical contexts.

Reproducibility of the experiments

To promote fairness and reproducibility, all datasets (STAMINA), algorithm configurations, and the software used are made available. The link to the repository of the used software (FlexFringe [14]) can be found in the references of the paper, while the STAMINA datasets can be found in the repository.

3 Background

In this section we discuss the relevant background in the research area for this project. First, in 3.1 a short explanation of deterministic finite automata and DFA Identification is provided. After that, in 3.2, two approaches to DFA identification are presented. Finally, a hybrid approach to DFA identification is introduced in 3.3.

3.1 DFA Identification

A deterministic finite automaton (DFA) is a finite state machine that recognizes a certain regular language. It consists of a set of states (including a starting state and accepting states), a set of transitions between states, as well as an alphabet. The set of all words that are accepted by the DFA forms its (regular) language. For a more detailed description, see [13].

DFA Identification is the problem of learning a minimal DFA from a given set of labeled examples. Usually a set of positive (accepted) and a set of negative (rejected) instances is provided as input. Several different approaches to DFA identification exist, including state-merging ones such as RPNI [2] and EDSM [11], as well as SAT-based methods [9]. In this work, we only consider EDSM and a SAT-based approach - specifically, a hybrid method that combines the two.

3.2 Approaches to DFA Identification

Evidence-driven state merging (EDSM)

Evidence-driven state merging (EDSM) is a heuristic-based state merging algorithm in the Red-Blue Framework, used for DFA identification. First appearing in the Abbadingo competition [11] in 1997, EDSM is the current state-of-the-art in DFA identification.

The algorithm starts by building an Augmented prefix tree acceptor (APTA) from the input. An APTA is a tree-shaped DFA, such that when parsing two different input strings, they go through the same state in the APTA only if they share the same prefix until they reach that state. Positive instances end in accepting states, while negative instances, respectively, end in rejecting states.

After building an APTA from the input, EDSM starts merging states heuristically. Due to EDSM being an algorithm in the Red-Blue Framework, it starts with the root of the APTA being colored red and all its neighboring states being colored blue. In the red-blue framework, the red states are states that are confirmed to be part of the final DFA, while blue states are candidate states (a state is colored blue once it becomes a neighbor of a red state). On each iteration two events might take place - either a blue state is merged with a red state (only if they are behaviorally equivalent), or a blue state is promoted to a red state. This process is repeated until no more blue states are left. EDSM is a specific algorithm that operates within the Red-Blue Framework. It is also worth mentioning that for a merge to be possible, it needs to be consistent. Inconsistent merges are those that merge an accepting with a rejecting state. A merge may also introduce nondeterminism (i.e. a state becomes the source of two transitions with the same label), which is then followed by a determinization process. The determinization process performs additional merges until there are no non-deterministic choices left. If the determinization process encounters an inconsistency, the initial merge is not considered possible.

In order to avoid bad merges, a statistical score is computed for each possible merge, and the highest scoring one is performed. A merge between two states is scored as the evidence supporting their equivalence. When a merge is considered, EDSM simulates the merge (including the determinization process) and determines which states would be merged as a consequence. For each group (also called an equivalence class) of states that would be combined into one, the algorithm checks the consistency of their labels. If any class contains both accepting and rejecting states, it is marked invalid and is given a score equal to $-\infty$. Otherwise, the class is given a score equal to the number of labeled states (i.e. accepting or rejecting) minus one - subtracting one because the first label sets the standard. The total merge score is the sum of the scores of all equivalence classes. This scoring system favors merges that are backed by more label agreement, improving the chance that early merge decisions are correct, and the resulting DFA generalizes well.

EDSM is a heuristic algorithm that converges to a local optimum efficiently. However, it does not guarantee a globally optimal (i.e. minimal) inferred DFA.

Reduction to SAT

A different approach to DFA Identification is reducing the problem to SAT and using a SAT solver to infer a DFA. Unlike EDSM, the reduction to SAT guarantees finding a minimal DFA that is consistent with the input data. An explanation of SAT, and the proof that it is NP-complete can be found in [4].

The process of inferring a minimal DFA with a SAT solver goes through several reductions and adaptations. It is important to note that DFA Identification as defined previously is not exactly the problem that is reduced to SAT. Instead, a modified version - finding a consistent DFA of fixed size (a problem shown to be NP-complete [8]) - is used. To ensure a minimal DFA, this problem is run iteratively with target sizes k, k+1, k+2, and so on until a solution is found.

The first reduction is from fixed size DFA Identification to Graph coloring, where the number of colors in the graph corresponds to the size of the inferred DFA. This reduction is introduced in [6], where each vertex in the graph corresponds to a state in the APTA. The vertices are connected with edges to enforce equality and inequality constraints, with the final

goal being that vertices with the same colors are merged to obtain the DFA with a given size (the number of colors).

In [9] a compact encoding for a reduction from Graph coloring to SAT is proposed. The use of color variables, parent relation variables and accepting color variables allow for encoding logical dependencies and symmetry braking constraints into clauses to form a formula in conjunctive normal form. This formula is then passed to a SAT solver to try and infer a DFA.

3.3 A hybrid approach to DFA Identification

As heuristic approaches to DFA identification such as EDSM opt to find a close to optimal solution efficiently, while approaches such as reduction to SAT sacrifice runtime to find the minimal DFA, there also exists the idea to run a heuristic algorithm for a few steps, and then minimize the partially identified DFA. This approach allows for a SAT solver to be used on instances that are otherwise too difficult (i.e, too big) for it to solve in reasonable time. Such an approach called DFASAT was used in 2012 to win the STAMINA competition [15]. Due to DFASAT and the STAMINA competition making the foundations of this work, the most important information about them is summarized in the remaining part of this section.

STAMINA competition

STAMINA was a competition aiming to encourage the development, improvement, and empirical evaluation of techniques for software model synthesis. Software model synthesis is the problem of inferring models (DFAs) from labeled software execution traces, that can simulate the behavior of software systems, see [3].

The setup of the STAMINA competition was as follows. Each competitor was provided with 100 training sets with different difficulties. The datasets were generated from randomly generated DFAs, that were aiming to resemble actual software behavior. There were 20 problem-sets of varying difficulty, with each set containing 5 hidden randomly generated DFAs. The varying difficulties were introduced through different alphabet sizes of the DFAs (50, 20, 10, 5, 2) and different sparsity levels of the datasets (100%, 50%, 25%, 12.5%). For each training set, each competitor had to train a model using their technique, and then run the model on the given test sets. Finally, the competitors had to submit a bit-string, with each bit indicating whether a trace was accepted by the model (1), or whether it was rejected by the model (0).

To rank the competing techniques, the organizers of the competition compared the submitted bit-strings with the correct bit-strings, and the result was presented using the BCR score. The BCR score combines the sensitivity *SE* and specificity *SP*. The sensitivity is the proportion of positive matches that are predicted to be positive, while the specificity is the proportion of true negatives that are predicted to be negative. In terms of the sets of true positives (TP) and true negatives (TN), the BCR score is computed as follows:

$$\text{SE} = \frac{|TP|}{|TP \cup FN|}, \text{SP} = \frac{|TN|}{|TN \cup FP|}, \text{BCR} = \frac{2 \cdot \text{SE} \cdot \text{SP}}{\text{SE} + \text{SP}}$$

In this work we will take inspiration from the STAMINA competition and adopt a similar approach to comparing DFA identification techniques.

DFASAT

The STAMINA competition was won by a technique that is quite similar to the one being used in this work. The winning algorithm was called DFASAT [10], and it combined 5 techniques in order to adjust for the challenging setting of software model synthesis.

The main idea (also the one being tested in this work) was to identify a partial DFA by applying EDSM for several iterations, and then minimize the remaining model with a SAT solver. The reason behind this was to reduce the size of the input for the SAT solver. For some instances of the STAMINA competition, the resulting encoding was too large for the state-of-the-art SAT solvers, leading to millions of clauses and an unfeasible problem. It is important to note that the SAT encoding was slightly adjusted to work with a partially identified DFA instead of the initial APTA. The set of red states identified by EDSM was used as a clique of mutually inconsistent states - states that should not be merged. In the SAT encoding, each red state was assigned a fixed color using symmetry braking predicates. This served two purposes: avoiding the exploration of symmetrical solutions (i.e. equivalent DFAs with permuted colors), and guaranteeing that the red states remain separate in the final DFA, preserving the structure learned by EDSM.

Another problem for DFASAT was that the EDSM heuristic is not well suited for software model synthesis. In software models, there usually exist states that have disjunctive sets of labels on outgoing transitions. The traditional heuristic would usually combine these states, which leads to the model having a very different language. To fix this issue, DFASAT implements a new heuristic for EDSM which favors the merging of states with a high degree of overlap in positive fanout. The positive fanout of a state is the set of symbols of outgoing transitions that can be activated by an accepting computation. In addition to this technique, two other techniques are implemented. One is to try different greedy paths by introducing randomness in the heuristic score, and then selecting the best one. The other is adding an additional merge restriction - a merge is not allowed to add a new label to the positive fanout of a red state. [10]

The final issue for DFASAT was that even with the above mentioned techniques, it was still unlikely that the inferred model will be the same as the target machine, due to the sparseness of the data. This was mostly solved by using an ensembe technique - generate many different good solutions to the problem and accept a word when at least 10% of them accept it. Such ensemble methods are reviewed in [7].

After combining all of these techniques, DFASAT managed to achieve a significant improvement in accuracy compared to purely EDSM, and win the STAMINA competition. In this work we take inspiration from DFASAT, but we only consider the first technique - minimizing a DFA that is partially identified heuristically.

4 Experimental Setup

This section describes the experimental setup used to obtain the results of this work. In 4.1 the software and the datasets that were used are presented. 4.2 outlines the way STAMINA datasets were used. Finally, in 4.3 and 4.4 a detailed explanation of the two techniques used to apply the hybrid approach are presented.

4.1 Software and datasets

To answer the research question - *"To what extent and in what ways does partially learning a model heuristically and then applying exact minimization - based on the decisions already made by the heuristic - affect model size and test-performance in DFA identification?"* - we need a software that is able to both perform heuristic and exact methods, and chain them to perform the hybrid approach. Furthermore, we need different datasets with different difficulties and properties such that we can compare the different methods and draw conclusions about their usefulness. To achieve this, a software tool called FlexFringe [14], together with the STAMINA competition training sets were used.

Use of FlexFringe

FlexFringe is "an open-source software tool to learn variants of finite state automata from traces using a state-of-the-art evidence-driven state-merging algorithm at its core" [14, p. 1]. It can be used to run different heuristic algorithms (including EDSM), exact methods, and also chain both. Despite being able to encode DFA identification into SAT, FlexFringe does not have a built-in SAT solver. To obtain results for exact minimization, an external SAT solver needs to be provided. For this work we use version 4.2.1 of the Glucose SAT solver [1].

For the hybrid approach (partially identifying the DFA heuristically and then minimizing) to work in FlexFringe, there are 3 key parameters that the user needs to set. Two of them which control at what point the greedy (e.g. EDSM) merging stops for the problem to be passed to the SAT solver, and one that controls the size of the minimized DFA.

- The first important parameter is the APTA bound. The algorithm begins with a large initial APTA, which starts to become smaller as the greedy algorithm merges nodes. To stop the algorithm early, the user can set an APTA bound which, once reached (i.e. the DFA size becomes less than the bound), stops the greedy merging.
- The second parameter is also used to stop the greedy merging algorithm early. As mentioned earlier, algorithms such as EDSM operate in the Red-Blue Framework, where red states are states that are guaranteed to be part of the final DFA. FlexFringe gives the user the option to specify a DFA bound, which stops the greedy merging early once the number of red states reaches the specified DFA bound.
- The third parameter is the SAT offset. If one runs the hybrid approach with no offset, the SAT solver will try to merge all remaining non-red states into the red states. If this is not possible, the result will be an unsatisfiable problem. FlexFringe gives the user the option to specify an offset x, which gives the SAT solver the freedom to search a solution of size up to $the\ number\ of\ red\ states + x$.

STAMINA datasets

As mentioned earlier, the STAMINA competition provided the competitors with training sets with 20 different characteristics. There were 5 sparsity levels - 100%, 50%, 25%, 12,5% - and 5 alphabet sizes - 50, 20, 10, 5, 2. For each combination of sparsity and alphabet size, 5 different training sets were generated from 5 different random DFAs with approximately 50 states, resulting in 100 datasets in total. It is important to note that the sparsity of the datasets was defined by how much they cover the target DFA. That results in the sparser datasets having less traces compared to the denser ones.

To try out a DFA identification approach on the STAMINA datasets, it is not necessary to run it on all 100 datasets. Due to each combination of sparsity level and alphabet size having 5 representations, one could try their approach on 20 out of the 100 datasets and still get informative results. Therefore, in this work, we use the datasets that are multiples of 5 (i.e., 5, 10, 15, etc.). For more information on the STAMINA competition and the datasets used, see [15].

4.2 Use of STAMINA datasets

The goal of this experiment is to showcase how a hybrid approach (greedy, then exact) performs on the problem of DFA identification. To be able to assess the performance of such an approach, the datasets need to be used properly, and a criterion that measures the performance needs to be set.

For this experiment, we are using the training sets from the STAMINA competition. Unfortunately, the test sets that were used are no longer available and a model that is trained on an entire training set cannot be tested. To solve this issue, each training set is divided into a new training and test set. To ensure that the test sets give informative results, each trace needs to be unique, and it should be unseen data to the model. Therefore, all duplicates in the test sets are removed, along with all traces that also appear in the training set. This preprocessing step works well on datasets consisting of a large number of traces. However, some of the STAMINA training sets are sparse, which leads to some test sets containing only a few traces (about 20) after the preprocessing step. This could lead to results that are highly influenced by the randomness in the selection of the traces that form the test sets.

To reduce the randomness in the results, we apply k-fold cross validation on the STAMINA training sets. K-fold cross validation is a popular technique used in machine learning, where the entire dataset is split into k subsets, called folds. Each fold is used once as a test set, while the other k-1 folds are used as training. At the end, the average result is taken. As we are taking inspiration from the STAMINA competition, for this experiment we use the BCR score to measure the test performance of a model (explained in 3.3). For a more detailed explanation of k-fold cross validation, see [12].

To balance the size of the training and test sets, an appropriate number of folds for k-fold cross validation needs to be chosen. The two main factors in making the decision were efficiency and completeness of the model. The more folds we

make, the more models we need to train, which reduces efficiency as the problem is NP-complete. On the other hand, the more folds we make, the bigger the training sets are, which improves the completeness of the inferred model. All things considered, 5-fold cross validation was chosen, as it uses a large enough proportion of the data as a training set, while also running in reasonable time (up to 30 minutes per fold).

Two slightly different techniques for applying the hybrid approach on the STAMINA datasets were tested. One focused on stopping the greedy merging at different times, and one focused on stopping the greedy merging at a fixed time. These two techniques are explained in detail in the following two subsections.

4.3 Optimal DFA bound search

Saying that we first partially identify a model heuristically and then minimize it with an exact method is not an entirely proper definition of the hybrid approach. We still need to decide when to stop the greedy merging and minimize the partially identified DFA. In the technique described in this subsection, the focus is on trying different DFA bounds at which to stop the greedy merging and find the minimal one that results in a successful minimization. To ensure efficiency, we run the SAT solver with an offset of 3.

The main difficulty in the hybrid approach is how to make sure that the instances passed to the SAT solver are not too large. To ensure this, we first run EDSM with a fixed APTA bound of 1500, which ends up with a sufficiently small model for the SAT solver to minimize. We capture the number of red states of the partially identified model and perform an entire run of EDSM, again capturing the number of red states. Since we know that once a red state is colored red - it stays red, we can use these two numbers as a lower and upper bound for the number of red states reached before we stop the greedy merging and start the minimization process. The reason why the size of the model inferred with EDSM can be used as an upper bound is that we know that an exact method will always find a model with at most the same number of states as a heuristic method.

Once we have established the lower and upper bounds, we aim to determine the minimal number of red states within this range that are reached by EDSM prior to invoking the SAT solver, such that the problem is satisfiable. It is important to understand that the SAT solver will not find a solution every time (i.e., the expression will not always be satisfiable). Since we are using an offset of 3, the SAT solver will look for a solution of up to $the\ \ number\ \ of\ \ red\ \ states + 3$. If such a solution does not exist, the SAT solver will return unsatisfiable. Intuitively, one would want to iterate from the lower bound to the upper bound and stop once a satisfiable solution is found. However, this means that in the worst case all values would need to be checked. Instead, to find the optimal number of red states before we switch, we do binary search between the above-defined lower and upper bound. That way the search runs in logarithmic time instead of linear, which is a dramatic improvement in efficiency, given that each run of the SAT solver could take up to 5 minutes. To make sure that a run does not take more than 5 minutes and potentially run forever, a timeout of 5 minutes is set for each SAT solver run. If a run exceeds the 5 minute mark, it is considered unsatisfiable. Additionally, if the difference between the DFA bound for a certain run and the upper bound of the binary search is less than 3, the SAT solver is run with an offset equal to 0. This is done to avoid inferring a DFA that has more states than the upper bound. For an explanation of binary search, see [5, pp. 39–41].

To investigate how a different lower bound would affect the results, the experiment was once executed with a lower bound found by stopping EDSM with an APTA bound equal to 1000 in addition to the run with an APTA bound equal to 1500.

Example execution

Here is an example execution of this technique. EDSM is run with an APTA bound of 1500 and the partial resulting DFA has 40 red states. Then, an entire run of EDSM is performed and the resulting DFA has 60 states. Therefore, the range of the binary search is $[40, 60]$, the first run is with DFA bound equal to 50, and a SAT offset equal to 3. The SAT solver returns satisfiable, the run is successful, and the inferred model has 53 states. The next run of the binary search is with range $[40, 49]$, a DFA bound equal to 44, and SAT offset equal to 3. Depending on whether the run is successful or not, the range is reduced to either $[40, 43]$ or $[45, 49]$. This continues until the range is reduced to a single integer. The DFA inferred with the last successful run is returned.

4.4 Optimal offset search

The first technique focuses on running the SAT solver with a small offset, with the aim of avoiding long runtime. However, in this way, in most cases the greedy merging makes up most of the hybrid approach. To limit the greedy merging and make more use of the SAT solver, in this subsection we propose a slightly different technique.

The second technique fixes the point at which EDSM is stopped, and tries different offsets with which the SAT solver is run. Similarly to the first technique, the greedy merging is stopped with an APTA bound of 1500, and the number of red states of the obtained partially identified model is used as a lower bound, while the number of red states from a full EDSM run is used as an upper bound. The difference is that this time the binary search is not on the DFA bound but on the SAT solver offset. For each run of the binary search, the SAT offset is set to a certain integer x. Hence, the SAT solver could find a DFA of up to $lower\ \ bound + x$ states. To avoid runs that take too long (potentially forever) due to excessively large offsets, the range of the binary search is limited to 12. In other words, if $upper\ \ bound - lower\ \ bound > 12$, a DFA bound equal to $upper\ \ bound - 12$ is used to stop the greedy merging, instead of an APTA bound equal to 1500. Twelve was chosen as an upper bound for the range, since offsets larger than 12 mostly resulted in timeouts.

Similarly to the DFA bound experiment, to assess how a different lower bound would affect the results, the SAT offset experiment was also run with APTA bounds equal to both 1000 and 1500.

Example execution

Here is an example execution of this technique. EDSM is run with an APTA bound of 1500 and the partial resulting DFA

has 50 red states. Then, an entire run of EDSM is performed and the resulting DFA has 60 states. Therefore, the range of the binary search is [50, 60], and the first run is with a SAT offset equal to 5. The run is successful and the SAT solver finds a model of size 55. For the next run, the binary search range is reduced to $[50, 54]$, and the SAT offset is equal to 2. Depending on whether the run is successful, the range is reduced to either $[50, 51]$ or $[53, 54]$. This continues until the range is reduced to a single integer. The DFA inferred with the last successful run is returned.

5 Results

In this section, the results of the experiments described above are presented and analyzed. Subsection 5.1 discusses the differences in the state counts of the inferred DFAs with both experiments, compared to full EDSM runs. Subsection 5.2 does the same but for the differences in the BCR scores of the inferred DFAs. The experiments are run on an Acer Aspire 7 laptop with 16 GB DDR4 main memory and a 6-core, 12-thread Intel Core i7-8750H 2.2GHz with Turbo Boost up to 4.1GHz processor. The full results are provided in Appendix A.

5.1 Differences in the state counts of the inferred DFAs

We start by looking at how the state counts of the inferred DFAs change with the hybrid approach compared to a fully heuristic (i.e. EDSM) approach. Naturally, the expectations were that the hybrid approach would find DFAs with smaller state counts, due to the usage of an exact solver (i.e. the SAT solver). This hypothesis is confirmed by the experiments. In Figure 1 we can see how the state counts of the inferred DFAs with the SAT offset experiment differ compared to full EDSM runs, while in Figure 2 we can observe the same but for the DFA bound experiment. The blue lines (circle marker) show the differences with a lower bound for the search found by stopping the greedy merging with an APTA bound equal to 1000, while the orange lines (x marker) show the differences with a lower bound found with an APTA bound equal to 1500. Furthermore, in Figure 3, the comparison of the 2 experiments on this metric can be observed (with lower bounds found with APTA bounds equal to 1500). It is important to note that the values are an average from 5-fold cross validation, and hence some are not integers.

In Figure 1 and Figure 2 we can clearly see that stopping the greedy merging earlier (i.e., obtaining the lower bound for the search with a larger APTA bound) results in smaller models. This is expected for both experiments. For the SAT offset experiment, when the APTA bound is larger, the exact solver plays a bigger role, which, respectively, leads to smaller models. For the DFA bound experiment, when the APTA bound is larger, the range over which to search is bigger, meaning a satisfiable solution can be found when the greedy merging is stopped earlier, again leading to smaller models. The comparison of the two experiments in Figure 3 shows that the SAT solver experiment infers smaller models on almost all STAMINA datasets. This is once again expected, as the SAT solver experiment would always stop the greedy merging at most as early as the DFA bound experiment. However, this does not necessarily mean that the SAT offset experiment would always find smaller DFAs compared to the DFA bound experiment, as can be seen for the 65th STAMINA dataset. For some instances, stopping the greedy merging later could lead to a satisfiable problem with a lower offset, compared to stopping the greedy merging earlier. On certain datasets, additional greedy merging might improve the structure of the residual problem, leading to a satisfiable problem with fewer states. Nevertheless, we can conclude that while stopping the greedy merging earlier generally yields better results (i.e. smaller models), this is not universally true.

Finally, we perform a statistical test to see if there is any correlation between the dataset characteristics and the effectiveness of the hybrid approach in terms of state counts. For this purpose we use the SAT offset experiment results (APTA bound 1500). In Table 1 we can observe how the state counts of the inferred models change for the different dataset sparsities when the SAT offset experiment is used instead of pure EDSM. From this data, we compute the Pearson correlation coefficient, which is equal to 0.82, as well as the p-value, which is equal to 0.17. This indicates that there is strong positive correlation between the sparsity of the dataset and the effectiveness of the hybrid approach compared to pure EDSM (i.e., the sparser the dataset, the more effective the hybrid approach gets compared to pure EDSM). However, since the p-value is higher than 0.05, this means that the correlation is not statistically significant, and further experiments are needed to verify these results. Additionally, the same is done for the alphabet sizes of the datasets: see the differences in Table 2. From this data we compute the Pearson correlation coefficient, which is equal to 0.78, as well as the p-value, which is equal to 0.12. Again, this indicates strong positive correlation between the alphabet size of the DFA used to generate the dataset and the difference in the number of states of the hybrid approach compared to pure EDSM, but this time in the other direction (i.e., as the alphabet gets bigger, the SAT offset experiment gets less effective in terms of how much it reduces the state counts found by pure EDSM). However, the p-value is again higher than 0.05, indicating that further experiments are needed to confirm this correlation.

Sparsity	100%	50%	25%	12.5%
State count diff.	-1.44	-1.45	-2.5	-3.44

Table 1: Table representing the average DFA state counts obtained with the SAT offset experiment compared to pure EDSM, relative to the sparsity of the datasets.

Alph. size	2	5	10	20	50
St. c. diff.	-3.83	-2.81	-1.75	-1.55	-1.1

Table 2: Table representing the average DFA state counts obtained with the SAT offset experiment compared to pure EDSM, relative to the alphabet size of the DFAs used to generate the datasets.

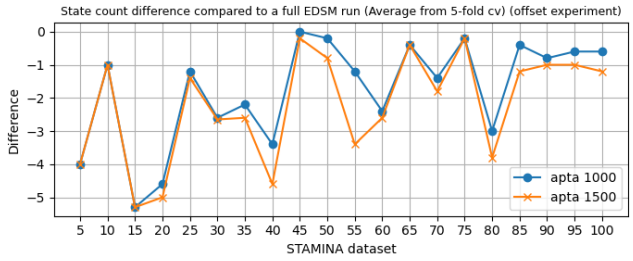

Figure 1: Difference in the (average) number of states of the identified DFAs with the offset experiment (APTA bounds 1000 and 1500) compared to full EDSM runs.

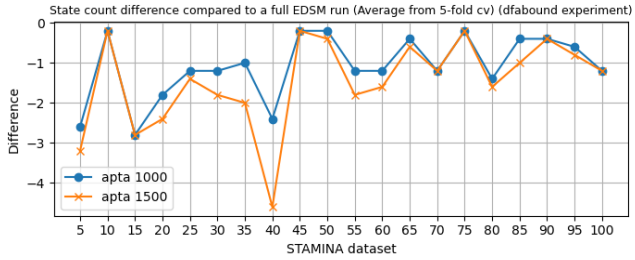

Figure 2: Difference in the (average) number of states of the identified DFAs with the DFA bound experiment (APTA bounds 1000 and 1500) compared to full EDSM runs.

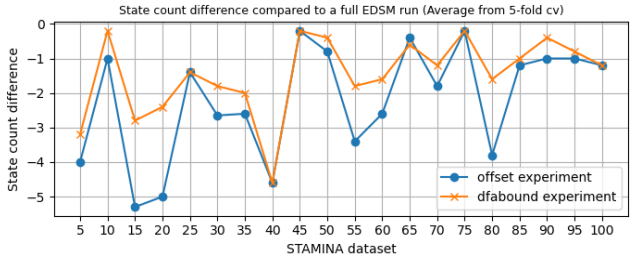

Figure 3: Difference in the (average) number of states of the identified DFAs with both experiments (with APTA bound 1500), compared to full EDSM runs.

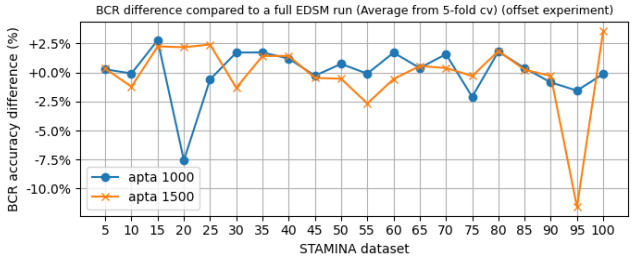

Figure 4: Difference in the (average) BCR scores of the identified DFAs with the offset experiment (APTA bounds 1000 and 1500) compared to full EDSM runs.

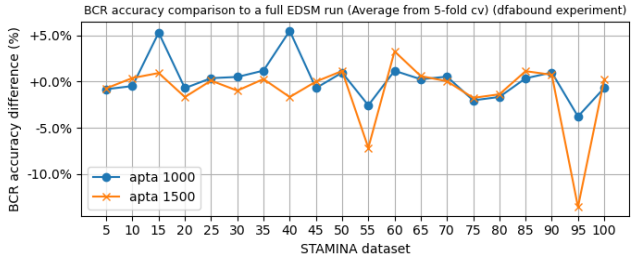

Figure 5: Difference in the (average) BCR scores of the identified DFAs with the DFA bound experiment (APTA bounds 1000 and 1500) compared to full EDSM runs.

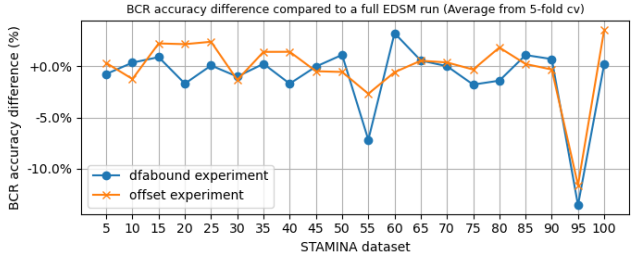

Figure 6: Difference in the (average) BCR scores of the identified DFAs with both experiments (with APTA bound 1500), compared to full EDSM runs.

5.2 Differences in the BCR scores of the inferred DFAs

In addition to the state count differences, the other important metric on which to assess the results of the two experiments is the difference in the BCR scores. Similarly to the plots for the state count differences, in Figures 4 and 5 we can observe the differences with the full EDSM runs, but this time for the BCR scores. Figure 4 presents the results from the SAT offset experiment, while Figure 5 presents the results from the DFA bound experiment. Once again, the blue lines (circle marker) show the differences with a lower bound found with an APTA bound equal to 1000, while the orange lines (x marker) show the differences with a lower bound found with an APTA bound equal to 1500. Furthermore, in Figure 6 we can observe the comparison of the two experiments on this metric (again with lower bounds found with APTA bounds equal to 1500).

Unlike the state counts of the inferred models, the BCR scores do not improve with the hybrid approach. In Figures 4, 5, and 6, we can see that not only does the BCR score not improve consistently compared to full EDSM runs, but it also does not improve with different APTA bounds or between the two experiments. However, there are still some interesting results. As we can see for the 95th STAMINA dataset in Figures 4 and 5, both experiments have noticeably worse BCR scores when run with APTA bounds equal to 1500, compared to when run with APTA bounds equal to 1000. There are two main circumstances that could lead to such a result. The first one is that when EDSM is run for longer (i.e. APTA bound equal to 1000), it could perform merges or promotions

to red states that lead to the inferred model being closer to the target DFA, despite not having much evidence. This would naturally lead to better performance on a test set. It is worth mentioning that it is a lot more likely for this to go the other way, and EDSM to perform bad merges or promotions due to having little evidence. The other reason for such a result is that the SAT solver that was used to run the experiments uses parallel processing. Parallel processing introduces nondeterminism in the search for a satisfiable solution, which could lead to different results on the same input, and hence occasionally to high BCR differences. To confirm the nondeterminism, the SAT offset experiment was run an additional 3 times on the 95th STAMINA dataset, resulting in BCR scores equal to 53%, 50%, and 54% (compared to the original 44%).

Sparsity	100%	50%	25%	12.5%
BCR diff. (%)	+0.62%	−0.6%	+2.18%	+1.7%

Table 3: Table representing the average BCR accuracy of the SAT offset experiment compared to pure EDSM, relative to the sparsity of the datasets.

Alph. size	2	5	10	20	50
BCR diff. (%)	+0.88%	+0.99%	−1.05%	+0.63%	−2.01%

Table 4: Table representing the average BCR accuracy of the SAT offset experiment compared to pure EDSM, relative to the alphabet sizes of the DFAs used to generate the datasets.

In order to see if there is any correlation between the BCR score results and the characteristics of the datasets, a statistical test is performed. For this purpose we use the SAT offset experiment (APTA bound 1500). In Table 3 we can observe how the BCR scores change for each dataset sparsity when we use the SAT offset experiment instead of pure EDSM. From this data, we compute the Pearson correlation coefficient, which is equal to 0.1, as well as the p-value, which is equal to 0.9. This indicates that there is practically no correlation. Furthermore, we do the same for the alphabet size: see the differences in Table 4. Once again we compute the Pearson correlation coefficient, which is equal to -0.78, and the p-value, which is equal to 0.12. This suggest strong negative correlation (i.e. as the alphabet size gets bigger, the SAT offset experiment gets worse than EDSM), but the p-value is above 0.05, which means the correlation is not statistically significant and further experiments would be needed to confirm it.

Finally, we perform a statistical test to see if there is any correlation between the differences in the BCR accuracies obtained with the SAT offset experiment and EDSM, and the BCR accuracies obtained with pure EDSM. In other words, does the SAT offset experiment perform better/worse compared to EDSM when EDSM performs well/poorly. We compute the Pearson correlation coefficient, which is equal to -0.06, as well as the p-value, which is equal to 0.79. These results indicate that there is practically no correlation.

6 Discussion

The main goal of this experiment was to test how a hybrid approach (greedy, then exact) performs on the problem of DFA identification and to assess its usefulness. Two different implementations of the exact approach were presented, both failing to outperform a purely heuristic approach (i.e. EDSM), despite managing to infer smaller models.

The two implementations of the hybrid approach were tested on the training sets from the STAMINA competition [15], which was, in fact, won by a hybrid approach called DFASAT [10]. DFASAT adopted the same approach as the one presented in this work: identify the model partially with EDSM, and minimize with a SAT solver. Nevertheless, it still achieved significantly better results than the two implementations of the hybrid approach tested here. The main reason behind this is that DFASAT used four additional techniques in addition to a hybrid approach. As explained in 3.3, these include a new heuristic for EDSM (more well suited for software model synthesis), trying different greedy paths, adding an additional merge restriction, and using an ensemble method. Adding these techniques to the two implementations of the hybrid approach presented in this work would potentially improve their test performances.

Despite not improving the performance on the test sets used in this experiment, the two implementations of the hybrid approach give valuable insight into how running the SAT solver as a larger part of the execution results in significantly smaller models. If a more powerful machine is available, the greedy merging could be stopped even earlier, which would result in even smaller models that generalize the input data better. To make sure the results that indicate that the hybrid approach finds smaller models than EDSM are statistically significant, a Wilcoxon signed-rank test [16] is performed. For this purpose the SAT offset experiment (with APTA 1500) is used. The test results in p-value equal to $8.8 * 10^{-5}$. Since this value is less than 0.05, the reslults are statistically significant.

7 Conclusions and Future Work

This section discusses the conclusions that could be drawn from this research, as well as the future work that could follow. In 7.1 the work is briefly summarized. Then, in 7.2 the conclusions that could be made are presented. Finally, in 7.3 we outline the potential future work.

7.1 Summary

The main objective of this research is to observe how a hybrid approach (learning a partial model heuristically, and then minimizing it with an exact method) to DFA identification affects model size and test performance. To achieve this an experiment was conducted. Evidence-driven state merging (EDSM) [11] was used as the heuristic method, while reduction to SAT [9] was used as the exact method. Two different variations of the hybrid approach were tested. The first one searched for a minimal point within a given range at which to stop EDSM before minimizing, while the second one stopped EDSM at a fixed point and minimized from there. The hybrid

approach used in this work was inspired by a similar approach called DFASAT [10]

The hybrid approach was tested on the training sets from the STAMINA competition [15]. K-fold cross validation was used to reduce the bias in the results, produced by the sparseness of some of the datasets.

7.2 Conclusions

The experiments showed a few interesting results. Despite the hybrid approach managing to reduce the sizes of the inferred models (compared to purely EDSM), it did not manage to consistently improve the test performances. This indicates that even though the hybrid approach puts us one step closer to achieving the main goal of DFA identification (i.e. finding the minimal DFA that is consistent with the training data), other factors play a significant role in the test performances of the models. These include how well-suited the merge heuristic used for EDSM is, as well as the randomness introduced by the parallel processing used by the SAT solver.

Despite not getting much information on how the test performances of the models change when the hybrid approach is applied with different settings, we can draw some conclusions about how the sizes of the inferred models change. We can conclude that when the exact method makes for a bigger part of DFA identification, that almost always results in smaller models. The earlier we stop the greedy merging (i.e. EDSM), the smaller the DFAs we obtain are.

7.3 Future work

The implementations of the hybrid approach discussed in this work are just a foundation of this idea. There are many optimizations that could be achieved, along with additional techniques that could further reduce the sizes of the inferred models, and improve the test performances.

The implementations of the hybrid approach were run with thought to the time and hardware available. It would be insightful to see what the model sizes and test performances would be if the greedy merging was stopped earlier, and a bigger part of the problem was solved by the SAT solver. This would of course require a larger computational effort, and therefore a very powerful machine.

The datasets used for the experiments are particularly designed for software model synthesis [3]. This leads to some of them being too sparse, which makes it more difficult for the inferred models to replicate the target DFAs. Future work could be to generate different datasets from DFAs with different characteristics (e.g. different alphabet sizes, different model sizes, etc.), and test the hybrid approach on them.

Finally, if we want to further improve the effectiveness of the hybrid approach on the STAMINA datasets, we could include the additional techniques in the DFASAT pipeline. These include a new heuristic for EDSM (more well suited for software model synthesis), trying different greedy paths, adding an additional merge restriction, and using an ensemble method.

References

[1] Gilles Audemard and Laurent Simon. Glucose: A solver that predicts learnt clauses quality. In *Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing (SAT 2011)*, volume 6695 of *Lecture Notes in Computer Science*, pages 7–10. Springer, 2011.

[2] A. Cano Gómez. Inferring regular trace languages from positive and negative samples. In J. M. Sempere and P. García, editors, *Grammatical Inference: Theoretical Results and Applications (ICGI 2010)*, volume 6339 of *Lecture Notes in Computer Science*, pages 24–37. Springer, Berlin, Heidelberg, 2010.

[3] Jonathan E. Cook and Alexander L. Wolf. Discovering models of software processes from event-based data. *ACM Trans. Softw. Eng. Methodol.*, 7(3):215–249, July 1998.

[4] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC)*, pages 151–158. ACM, 1971.

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 3 edition, 2009.

[6] François Coste and Jacques Nicolas. Regular inference as a graph coloring problem. 01 1998.

[7] Thomas G. Dietterich. Ensemble methods in machine learning. In Amanda Sharkey, Noel Sharkey, and Josef Kittler, editors, *Multiple Classifier Systems*, volume 1857 of *Lecture Notes in Computer Science*, pages 1–15. Springer, Berlin, Heidelberg, 2000.

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

[9] Marijn J. H. Heule and Sicco Verwer. Exact dfa identification using sat solvers. In José M. Sempere and Pedro García, editors, *Grammatical Inference: Theoretical Results and Applications. ICGI 2010*, volume 6339 of *Lecture Notes in Computer Science*, pages 66–79. Springer, Berlin, Heidelberg, 2010.

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

[11] Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm. In Vasant Honavar and Giora Slutzki, editors, *Grammatical Inference. ICGI 1998*, volume 1433 of *Lecture Notes in Computer Science*, pages 1–12. Springer, Berlin, Heidelberg, 1998.

[12] Rukshan Manorathna. k-fold cross-validation explained in plain english (for evaluating a model's performance and hyperparameter tuning). 12 2020.

[13] Thomas A. Sudkamp. *Languages and Machines: An Introduction to the Theory of Computer Science*. Addison-Wesley, 2nd edition, 1997.

[14] Sicco Verwer and C.A. Hammerschmidt. flexfringe: A passive automaton learning package. In L. O'Conner,

editor, *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 638–642. IEEE, 2017. Code available at https://github.com/tudelft-cda-lab/FlexFringe.

[15] Neil Walkinshaw, Bernard Lambeau, Cédric Damas, Pierre Dupont, Jan Van den Bergh, and Tom Mens. STAMINA: a competition to encourage the development and assessment of software model inference techniques. *Empirical Software Engineering*, 18(5):791–824, 2013.

[16] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

Appendix

A Full results

In this appendix, the results of the experiments described in Section 5 are presented. Table 5 contains the BCR scores for each algorithm on the used STAMINA datasets. In Table 6 the state counts of the inferred DFAs can be observed.

STAMINA dataset / Algorithm	5	10	15	20	25	30	35	40	45	50
EDSM	0.98	0.97	0.52	0.39	0.67	0.61	0.49	0.50	0.62	0.53
SAT offset experiment (APTA 1500)	0.98	0.95	0.54	0.42	0.69	0.60	0.50	0.51	0.61	0.53
SAT offset experiment (APTA 1000)	0.98	0.96	0.55	0.32	0.66	0.63	0.51	0.51	0.62	0.54
DFA bound experiment (APTA 1500)	0.97	0.97	0.53	0.38	0.67	0.60	0.49	0.48	0.62	0.54
DFA bound experiment (APTA 1000)	0.97	0.96	0.57	0.39	0.67	0.61	0.50	0.55	0.61	0.54
	55	60	65	70	75	80	85	90	95	100
EDSM	0.52	0.42	0.65	0.64	0.56	0.55	0.66	0.58	0.56	0.53
SAT offset experiment (APTA 1500)	0.49	0.41	0.65	0.64	0.55	0.57	0.67	0.57	0.44	0.57
SAT offset experiment (APTA 1000)	0.52	0.43	0.65	0.66	0.54	0.57	0.67	0.57	0.54	0.53
DFA bound experiment (APTA 1500)	0.45	0.45	0.65	0.64	0.54	0.54	0.67	0.58	0.42	0.53
DFA bound experiment (APTA 1000)	0.49	0.43	0.65	0.65	0.54	0.53	0.67	0.59	0.52	0.53

Table 5: Table with the BCR scores for each algorithm on the STAMINA datasets.

STAMINA dataset / Algorithm	5	10	15	20	25	30	35	40	45	50
EDSM	63.6	50.6	86.8	49.8	112.	79.4	68.8	38.8	108.	68.4
SAT offset experiment (APTA 1500)	59.6	49.6	81.5	44.8	111.0	76.7	66.2	34.2	108.	67.6
SAT offset experiment (APTA 1000)	59.6	49.6	81.5	45.2	111.0	76.8	66.6	35.4	108.	68.2
DFA bound experiment (APTA 1500)	60.4	50.4	84.0	47.4	111.0	77.6	66.8	34.2	108.	68.0
DFA bound experiment (APTA 1000)	61.0	50.4	84.0	48.0	111.0	78.2	67.8	36.4	108.	68.2
	55	60	65	70	75	80	85	90	95	100
EDSM	35.6	25.0	49.2	29.4	28.0	19.0	37.0	25.4	15.4	13.6
SAT offset experiment (APTA 1500)	32.2	22.4	48.8	27.6	27.8	15.2	35.8	24.4	14.4	12.4
SAT offset experiment (APTA 1000)	34.4	22.6	48.8	28.0	27.8	16.0	36.6	24.6	14.8	13.0
DFA bound experiment (APTA 1500)	33.8	23.4	48.6	28.2	27.8	17.4	36.0	25.0	14.6	12.4
DFA bound experiment (APTA 1000)	34.4	23.8	48.8	28.2	27.8	17.6	36.6	25.0	14.8	12.4

Table 6: Table with the state counts of the inferred models for each algorithm on the STAMINA datasets.