# Learning DFAs by Evolving Short Sequences of Merges

**Kristian Guillaumier** KRISTIAN.GUILLAUMIER@UM.EDU.MT
*Dept. of Artificial Intelligence, Faculty of ICT, University of Malta, Msida, Malta*
**John Abela** JOHN.ABELA@UM.EDU.MT
*Dept. of Computer Information Systems, Faculty of ICT, University of Malta, Msida, Malta*

**Editors:** Jane Chandlee, Rémi Eyraud, Jeffrey Heinz, Adam Jardine, and Menno van Zaanen

## Abstract

The grammatical inference community has been studying evolutionary methods for DFA learning for almost three decades. These methods typically operate by learning a representation of the target DFA either as a partitioning the states of a prefix tree acceptor or as an encoding of its transition matrix. In this paper, we present an alternative approach for learning random DFAs over binary alphabets from sparse training data. We first conducted several experiments on thousands of problem instances to study their behaviour and to better understand the conditions under which state merging algorithms succeed or fail. Motivated by these observations, we implemented an evolutionary algorithm in which the chromosomes encode short sequences of merges selected from a subset of high state-reduction merges. The fitness of a chromosome is then measured by extending it using the EDSM heuristic and the size of the final hypothesis is used to score the entire sequence. To improve runtime performance, we use a method that can reliably estimate the fitness of a sequence of merges without extending it completely. We use the state-of-the-art EDSM algorithm as a baseline to compare our results to and observe that we can find low-error hypotheses or the exact target DFAs with a considerably higher likelihood.

**Keywords:** Grammatical Inference, Deterministic Finite-State Automata, Formal Languages, Genetic Algorithms, Greedy Search, Pattern Analysis, Regular Languages.

## 1. Introduction

DFA learning is the task of inferring a minimum-state deterministic finite-state automaton from positive and negative training data. The hardness of this task has been extensively studied (Gold, 1978) and the academic community has proposed several challenges to promote research in the area and encourage the development of new and better algorithms. Examples include *Abbadingo One* for learning DFAs from sparse training data (Abb, 1997; Lang et al., 1998), *Gowachin* and *GECCO* which allowed for noise (Gow, 1997; GEC, 2004), *Zulu* for learning DFAs from membership queries (Combe et al., 2010), and *StaMInA* for learning DFAs having large alphabets (Walkinshaw et al., 2010).

Among the most successful approaches are state merging algorithms which iteratively select and merge pairs of states starting from a highly specific hypothesis until a more compact solution consistent with the training data is found (Wieczorek and Unold, 2014). Since each merge choice imposes constraints on subsequent ones, the quality of the results found using such methods is sensitive to the initial choices made by the algorithm (Lang et al., 1998). In this paper, we describe an evolutionary approach which attempts to find an initial sequence of high-quality merges which, when extended using a state merging algorithm, would considerably increase the likelihood of finding good solutions. To test the effectiveness of our method, we use EDSM as a baseline[1].

1. Datasets, results, and code are available at https://github.com/kguil2/ICGI2020-21.

This paper is organised as follows. In §2, we highlight several evolutionary methods used in DFA learning. In §3, we present the necessary preliminaries. In §4, we discuss the observations we made from the experiments that motivated our approach. In §5, we describe our genetic algorithm (GA). §6 and §7 describe our experimental setup, results, and observations. Conclusions, limitations, and research perspectives are presented in §8.

## 2. Related Work

An early evolutionary method, due to Dupont (1994), attempts to find an optimal (smallest cardinality) partitioning of the states in a highly specific automaton constructed from the training data. This method has been shown to be successful and comparable to the RPNI algorithm (Oncina and García, 1992) on problems whose target DFAs had up to 5 states. Lucas and Reynolds (2005) proposed an approach called smart state labelling where a candidate DFA is represented by a transition matrix. A multistart, random hill-climber is used to modify the candidate until it either fits the training data well or a maximum number of evaluations are performed. In their experiments, they found that their algorithm outperforms EDSM (Lang et al., 1998) on target DFAs having between 4 and 16 states created according to the protocol of the Abbadingo One learning competition. Shayani and Bentley (2007) proposed a novel representation scheme based on gene regulatory networks to successfully evolve automata having between 2 and 16 states. They used a similar statistical method to that of Lucas and Reynolds to learn the output function of the DFA. Rather than using randomly created DFAs to create a training set, the authors used DFAs which represent modulo-N up/down counters. Tsarev and Egorov (2011) describe a method where an initial population of random $n$-state finite-state machines is created and a model checking approach, inspired by Johnson (2007), is used to determine their fitness. Chromosomes are encoded as vectors of transitions, and crossover was implemented by either selecting transitions from the parents after they have been randomly shuffled or by choosing transitions from the parents based on a statistical test. Mutation involved randomly changing the initial state, or randomly changing a transition, or adding/deleting transitions according to a mutation rate. The method was applied to automata having 6 states and 7 transitions for a simple real-world problem. More recently, Oğuz (2020) has proposed a genetic algorithm which runs on a graphical processing unit (GPU) for evolving DFAs that serve as robot controllers in the Tartarus[2] environment. Chromosomes encode DFAs as vectors of transitions and uniform crossover is used. Rather than searching for a minimum state DFA, the number of states is fixed and the fitness function is a score of how well the robot performed after eighty moves. Target DFAs had between 4 and 12 states.

## 3. Preliminaries

A DFA is 5-tuple $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F \rangle$, where $\Sigma$ is a finite set of symbols, $Q$ is a finite set of states, the state $q_0 \in Q$ is distinguished as the starting state, $\delta : Q \times \Sigma \to Q$ is the set of transitions, and $F \subseteq Q$ is the set of accepting states. States that are not accepting are, by definition, *rejecting*. Informally, a string $s$ is *accepted* by a DFA if for every character in $s$ starting from $q_0$, the automaton moves from one state to the next using $\delta$ until a

2. A grid-based problem proposed by Teller (1994) used as a benchmark problem in evolutionary algorithms.

state in $F$ is reached for the last character in the string. The string is, otherwise, *rejected*. In the context of DFA learning, it is sometimes useful to consider states that are neither accepting nor rejecting. A DFA is then defined as $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F_A, F_R \rangle$, where $F_A \subseteq Q$ and $F_R \subseteq Q$ are the disjoint sets of accepting and rejecting states respectively. States in $Q$ which are neither in $F_A$ nor in $F_R$ are called *unlabelled* states.

Consider a DFA $\mathcal{A}$ and a partition $\pi$ of its states. Two states are said to be *equivalent* or *merged together* if they belong to the same block (subset) in $\pi$. The partition $\pi$ induces an equivalence relation on the states of $\mathcal{A}$ (de la Higuera, 2010). A *quotient automaton*, denoted by $\mathcal{A}/\pi$, is an automaton *derived* from $\mathcal{A}$ with respect to the partition $\pi$, where its states are the equivalence classes defined by $\pi$ (Dupont et al., 1994).

Let $S_+$ and $S_-$ be two finite disjoint sets of strings which are accepted and rejected by a DFA respectively. We call $S_+$ the set of *positive samples*, and $S_-$ the set of *negative samples*. The pair $\langle S_+, S_- \rangle$ is called a *training set*. A DFA is *consistent* with $\langle S_+, S_- \rangle$ if it accepts all the strings in $S_+$ and rejects all the strings in $S_-$. The *prefix tree acceptor* $\mathrm{PTA}(S_+)$ is the smallest tree-shaped DFA which accepts exactly the strings in $S_+$. Likewise, the prefix tree acceptor $\mathrm{PTA}(S_-)$ is the smallest tree-shaped DFA which rejects exactly the strings in $S_-$. The *augmented prefix tree acceptor*, or APTA, is the superposition of $\mathrm{PTA}(S_+)$ and $\mathrm{PTA}(S_-)$ (Coste and Nicolas, 1997). A set of positive samples $S_+$ is said to be *structurally complete* with respect to a DFA if every transition is exercised and every final state is reached when parsing the strings in $S_+$ (Dupont et al., 1994). This idea can be extended to *symmetrical structural completeness* where every transition must be exercised by the strings in $S_+ \cup S_-$. We further require that all final rejecting states are reached by the strings in $S_-$ (de la Higuera, 2010).

## 3.1. State Merging

A commonly used approach in DFA learning is to first construct an APTA from the training data and then iteratively selecting and merging pairs of states until a DFA is obtained where no further merges are possible. When a pair of states is merged, we must ensure that the merge is valid (the automaton remains consistent with the training data). Additionally, merging a pair of states may introduce non-determinism. This is resolved by recursively merging the successor states having the same symbol. A rigorous description of this constraint may be found in Coste and Nicolas (1997) and pseudo-code for the procedure may be found in López and García (2016). Throughout this document, whenever we refer to a merge, we are always merging for determinisation.

A DFA that contains one or more valid merges is called a *partial hypothesis*. Otherwise, the DFA is called a *final hypothesis* and is usually the result returned by a state merging algorithm. In a partial hypothesis that contains more than one possible valid merge, a state merging algorithm must *choose* which one of these merges to proceed with. Algorithms such as RPNI (Oncina and García, 1992) make this choice by imposing an order on the set of all pairs of the states in a DFA and then selecting the first valid merge in that order. Algorithms such as EDSM (Lang et al., 1998) use a heuristic to score merges and choose the one having the highest score. Such methods perform a greedy search in a space of DFAs starting from a highly specific hypothesis (the APTA) and select merges without backtracking until a final hypothesis is reached. The target DFA is guaranteed to be in the search space if the training data is structurally complete with respect to it (Dupont et al., 1994; de la Higuera, 2010).

### 3.2. Evidence Driven State Merging (EDSM)

Whenever a merge is performed, states are being grouped together. If a labelled state is grouped with one having the same label, this is considered to be evidence supporting that merge and the EDSM score is incremented by one for each such match. Since we are merging for determinisation, successor states may be also merged contributing to the label matching count. EDSM is commonly considered to be a state-of-the-art state merging algorithm in DFA learning (Heule and Verwer, 2013; López and García, 2016) and is used as a supporting heuristic by other algorithms such as Ed-Beam (Lang, 1999), SAGE (Juillé and Pollack, 1998), TBW-EDSM (Cicchello, 2002), S-EDSM (Abela et al., 2004), and dfasat (Heule and Verwer, 2013). As such, EDSM is typically used as a baseline for comparing other DFA learning techniques to.

### 3.3. Colour-Compatible Merges

Consider the target DFA and APTA shown in Figure 1 (i) and (ii) respectively.[3] For the moment we can ignore the state colours. In the APTA, merging the unlabelled state 4 with the labelled state 8 is valid and is therefore a candidate for selection by a state merging algorithm. However, the resulting DFA is no longer consistent with the target DFA since it now classifies the string *ab* as accepting whereas the target DFA rejects it. After this merge is performed, no further sequence of merges can possibly lead to the target DFA since the unlabelled state 4 is now labelled as accepting when it really should be rejecting. In such a case, we could say that, although valid, this is a 'wrong' merge.

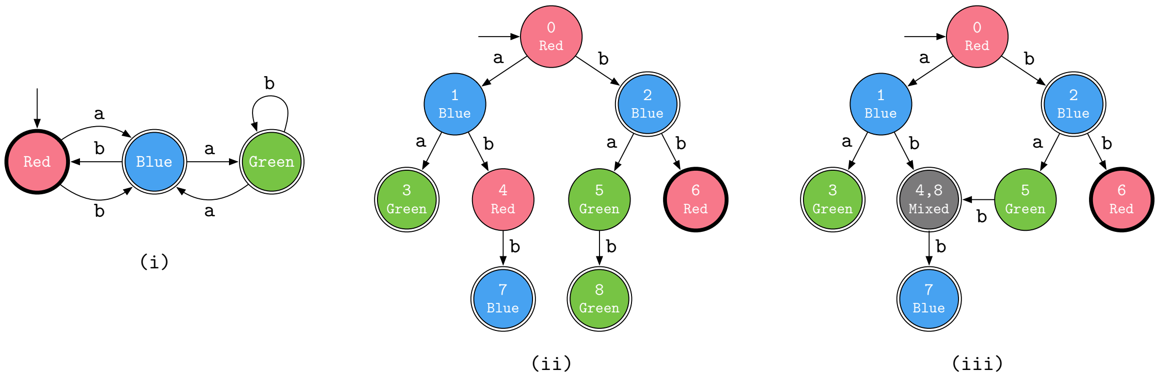

Figure 1: A coloured target DFA (i), the APTA (ii) for the training set $S_+ = \{aa, abb, b, bab\}$ and $S_- = \{bb\}$, and (iii) merging the state 4 with 8 results in mixed colours.

3. By convention, states having a double outline are accepting, states having a thick outline are rejecting, and states with a single thin outline are unlabelled states.

In our preliminary experiments we investigated when state merging algorithms such as EDSM choose such wrong merges. That is, a merge that is valid but that misclassifies strings not in the training set. We were inspired by the work of Coste and Nicolas (1997) who pose the DFA learning task as a graph colouring problem. Since, by construction, every state in an APTA maps to a state in the corresponding target DFA, if a distinct colour is assigned to every state in the target, then every state in the APTA will map to one of these colours. Any valid merge that merges states having different colours will then be a wrong merge. Back to the example in Figure 1, the state 4 in the APTA maps to the red state in the target DFA while state 8 maps to the green one. Although both states can be validly merged together, the merge is not *colour-compatible* as can be seen in (iii).

Of course, during real-world DFA learning, we are not privy to the colours of the states in the APTA. However, let us suppose that *for the purpose of experimentation only*, an Oracle does reveal this colouring information to us. This would allow us to always choose merges which are not only valid but are also colour-compatible. If a state merging algorithm is to converge to the target DFA it must, somehow, select only colour-compatible merges although many valid merges are not.

### 3.4. Oracle-Assisted Heuristics

We will refer to any heuristic which used extra information not in the training set as an *Oracle-assisted* heuristic. While it is not possible to use such a heuristic in real-world DFA learning, using such a heuristic allows us to analyze, and better understand, the behaviour of a state-merging algorithm by studying the conditions under which it succeeds or fails. We implemented two algorithms that use such a heuristic - Col-EDSM which is a modified version of EDSM where the highest-scoring merge is the one which has the highest EDSM score *and is also colour-compatible* and FullCol-EDSM which behaves exactly like Col-EDSM but also fully labels the states in the APTA using the Oracle.

## 4. Experimental Observations

In our experiments, all problem instances are constructed according to the Abbadingo One competition specification as described in Abb (1997) and Lang et al. (1998). The unknown target DFAs each have $n$ states and the training sets are at the sparsest density.

1. In the first experiment, we observed that the target DFA is never found in fewer than $n + 1$ merge steps. Furthermore, hypotheses which have $\leq 1\%$ error on a testing set were always found in close to $n+1$ merge steps. In general, as the length of a sequence of merges increases, so does the error of the final hypothesis in the testing set. We observe a similar correlation between the size of the hypothesis DFA found and the error of that hypothesis in the testing set. These relationships are shown in Figures 4 and 5 in Appendix A.

2. The importance of the initial merge choices made by a state merging algorithm has been stressed in the literature (Lang et al., 1998). Cicchello and Kremer (2002) use a stochastic search to attempt to quantify how important these first moves are. In this experiment, we build on this work by using Col-EDSM to efficiently select the first $k$

highest scoring and colour-compatible merges before proceeding with EDSM to obtain a final hypothesis. As expected, our experiments show that as $k$ increases, so does the likelihood of finding a low-error hypothesis. For example, over 1,024 instances of 64-state target DFA problems at the sparsest density of 1,521 training samples, when the first 8 merges are guaranteed to be colour-compatible, we find low-error hypotheses about 48% of the time, compared to 15% using EDSM.

3. Whenever a merge is performed, there is a reduction in the number of states. The *reduction count* is therefore the size of the resulting DFA subtracted from the size of the DFA before the merge. When studying merge sequences having exactly $n+1$ merge steps which lead to the exact target (constructed using FullCol-EDSM), we observe that merges having very low reduction counts are avoided in the initial merge steps. An example of this can be seen in Figure 6 which shows that during the first eight merge choices in a sequence leading to the target DFA, merges reducing the partial hypothesis by fewer than 64 states are never selected.

4. Following observations 2 and 3, let the *APTA reduction table* $\mathcal{T}$, be the set of all valid merges in an APTA whose reduction count is at least some value $\alpha$ and whose EDSM score is at least some value $\beta$. We observe that while the set of merges in $\mathcal{T}$ is substantially smaller than the set of all possible merges in the APTA, it nonetheless contains many colour-compatible merges. For example, over 256 problem instances of 64-state target DFAs at the sparsest density, APTAs contain around 26 million possible merges, while the corresponding APTA reduction tables for $\alpha \geq 60$ and $\beta \geq 1$ contain, on average, only 1,291 merges out of which 74 are colour-compatible.

5. The merges selected by an Oracle-assisted heuristic such as FullCol-EDSM are all colour-compatible. Our experiments show that there is a significant overlap between the merges in an APTA reduction table and the colour-compatible merges in a sequence leading to the target DFA from that APTA. For example, over 1,024 instances of 64-state target DFA problems, over 97% of APTA reduction tables for $\alpha \geq 60$ and $\beta \geq 1$ contain at least 8 colour-compatible merges in common with the corresponding FullCol-EDSM sequence leading to the target DFA. The significance of this is that, with high likelihood, a search in the considerably smaller space of merges in the APTA reduction table can allow us to identify sufficiently long sequences of colour-compatible merges which establish enough constraints that greatly improve our chances of finding low-error hypotheses (see point 2 above).

6. Consider a sequence of merges starting from an APTA which lead to a final hypothesis. There exists a strong correlation between the size of that hypothesis and the cumulative EDSM score of the partial hypothesis at merge step $n+1$ in the sequence. In other words, using simple linear regression, we can reliably guess the size of a final hypothesis after performing only the first $n+1$ merge steps of the entire sequence leading to that hypothesis. Figure 7 in Appendix A shows this correlation.

### 4.1. Adversarial Cases

EDSM always selects the highest scoring merge available and, when more than one merge shares a highest score, a random one is chosen from that rank.[4] When EDSM makes a wrong (non-colour-compatible) choice, it is either because a colour-compatible merge has a highest score but another equally high scoring (but not colour-compatible) merge was selected during tie-breaking, or because the colour-compatible merge was 'outside' of the highest scoring rank. These two situations are illustrated in Figure 2. During our experimentation, we identified three scenarios which increase the likelihood of EDSM making merge choices that are not colour-compatible. These adversarial cases are when the target DFA does not contain any loop transitions (a transition from a state to itself), when the training set is not structurally complete with respect to the target DFA, and when none of the highest EDSM-scoring merges in the APTA (the ties) are colour-compatible. In these scenarios we observe a considerable drop in EDSM's performance, and we also note that the likelihood of randomly creating such adversarial problem instances is significant.

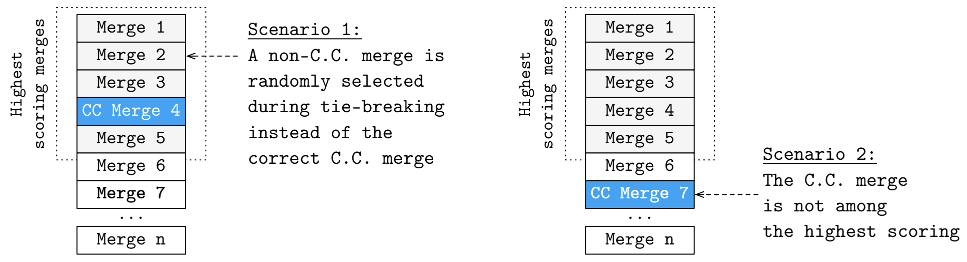

Figure 2: How EDSM can select a merge which is not colour-compatible.

## 5. Evolving Merge Sequences

An APTA reduction table is the collection of valid merges in an APTA having some minimum reduction count and a minimum EDSM score. While this table contains substantially fewer merges than the set of all possible merges, experiments 4 and 5 in §4 indicate that it will, typically, still contain many colour-compatible ones. Since initial sequences of colour-compatible merges considerably increase the likelihood of finding the target DFA or a low-error hypothesis (result 2 in §4), the GA attempts to find such a sequence while only searching in the APTA reduction table.[5] Given a problem instance consisting of a training set, the GA works as follows:

1. *Setup*: an APTA is created from the training set. An APTA reduction table for hyperparameters $\alpha$ and $\beta$ is created from the valid merges in the APTA.

2. *Chromosome representation*: a chromosome consists of a sequence of $k$ merges drawn from the APTA reduction table. When creating and manipulating chromosomes, the following two situations must be considered:

4. A rank is a set of merges having the same EDSM score.
5. The GA does not know which merges are colour-compatible.

- A merge in a chromosome may cause a subsequent one to be a null operation. This happens when the states in a subsequent merge have already been merged by a previous one. Such merges should be avoided in favour of 'useful' ones.
- If after performing a merge, a subsequent one becomes invalid, we say that the two merges *block* each other (this relation between merges is symmetric since merge order is not important[6]). This happens when one merge labels some previously unlabelled state, and the second merge requires that same unlabelled state to be labelled differently. Clearly, these two merges cannot exist in the same chromosome sequence.

3. *Initialisation*: the initial population consists of chromosomes each consisting of $k$ random merges selected from the APTA reduction table subject to the two conditions described in Point 2 above.

4. *Fitness evaluation*: the fitness of a chromosome is computed in two steps. Firstly, the merges in a chromosome are executed in the APTA to obtain a partial hypothesis. The partial hypothesis is then extended using windowed EDSM to reach a final hypothesis having size $m$. The score of a chromosome is given by $\mathrm{ABS}(m - n)$ where $n$ is the size of the target DFA. A chromosome having a lower score is fitter than one having a higher score since it is closer in size to the target DFA.

5. *Crossover*: since the order in which merges are performed is unimportant, we use a variation of uniform crossover (Goldberg, 1989) where the $k+k$ merges in two parents are pooled together and $k$ distinct merges (an offspring) are randomly selected from the pool subject to the previous two conditions.

6. *Mutation*: involves picking a random merge in a chromosome and substituting it with a random one from the APTA reduction table also subject to the previous conditions.

7. Fitness evaluation and genetic operators are illustrated in Figure 3.

8. *Selection*: to select parents for crossover, we use deterministic tournament selection where the fittest chromosome is selected from a random subset of $t$ chromosomes in the population (Blickle and Thiele, 1997). Hyperparameter $t$ is the tournament size.

9. *New generation:* the composition of a new generation is determined by the size $p$ of the initial population, and the *crossover* and *elitism rates* which are both ratios whose sum is $\leq 1\%$. Specifically, a new population consists of:

    - $p \times$ CrossoverRate offspring chromosomes obtained by selecting parents using deterministic tournament selection and mating them. Each offspring is mutated with a probability determined by a *mutation rate*.
    - $p \times$ ElitismRate fittest chromosomes in the previous generation are carried over.
    - In the interest of diversity, random chromosomes are created and added to the new population until its size reaches $p$.

6. Since set union is associative over an arbitrary finite number of sets (Halmos, 1960).

10. *Termination condition:* the GA terminates either when an extended chromosome having a fitness score of zero is found (the hypothesis is equal in size to the target DFA) or when a maximum number of generations have elapsed. In this second case, the hypothesis corresponding to the fittest extended chromosome is returned.

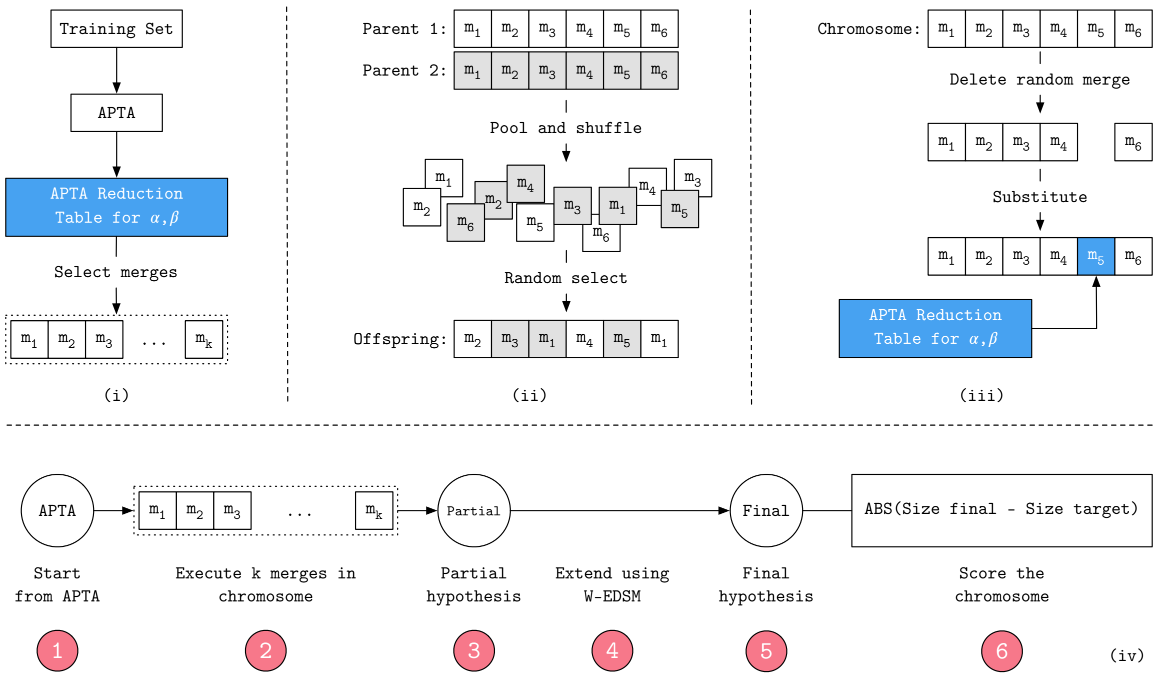
Training Set
APTA
APTA Reduction Table for $\alpha,\beta$
Select merges
$m_1$ $m_2$ $m_3$ ... $m_k$
(i)
Parent 1: $m_1$ $m_2$ $m_3$ $m_4$ $m_5$ $m_6$
Parent 2: $m_1$ $m_2$ $m_3$ $m_4$ $m_5$ $m_6$
Pool and shuffle
Random select
Offspring: $m_2$ $m_3$ $m_1$ $m_4$ $m_5$ $m_1$
(ii)
Chromosome: $m_1$ $m_2$ $m_3$ $m_4$ $m_5$ $m_6$
Delete random merge
Substitute
APTA Reduction Table for $\alpha,\beta$
(iii)
APTA
Partial
Final
ABS(Size final - Size target)
Start from APTA
Execute k merges in chromosome
Partial hypothesis
Extend using W-EDSM
Final hypothesis
Score the chromosome
1
2
3
4
5
6
(iv)

Figure 3: (i) A chromosome consists of merges drawn from the APTA reduction table, (ii) crossover, (iii) mutation, and (iv) the fitness computation of a chromosome.

The vast majority of the computational resources expended by the GA are used to evaluate the fitness of chromosomes in a population. This is because computing the fitness of a chromosome involves first performing its merges and then extending the partial hypothesis *completely* until a candidate DFA is identified. Since the candidate DFA is only needed to determine its size, we can exploit the correlation discussed in point 6 in §4 to estimate this size while performing fewer merge steps. The method works as follows:

- All the merges in the chromosome are applied to the APTA. The partial hypothesis is extended using windowed EDSM until a maximum of $n + 1$ merges are performed.
- The cumulative EDSM score of every $n + 1$ merge is computed.
- Linear regression is used to estimate the size of the candidate DFA from the cumulative EDSM score of the $n + 1$ merges.

- The regression coefficients are determined by creating 1,024 random problem instances and correlating the cumulative EDSM score at merge step $n + 1$ and the final DFA size. An example of this is shown in Figure 7.

## 6. Experimental Setup

Problem instances are randomly generated according to the Abbadingo One competition protocol (Lang et al., 1998). In this setup, target DFAs will have $n$ states and a depth of $2 \log_2 n - 2$. Training and testing sets are created by drawing samples uniformly and without replacement from the set of all binary strings having length between 0 and $2 \log_2 n + 3$. The proportion of positive and negative strings in a training set is not allowed to differ by more than 20% to avoid rare cases where the set may be disproportionately composed of strings of one class. Testing sets consist of 1,800 strings not in the training set.

We evaluate our methods on instances of target DFAs having exactly 32 and 64 states and having training sets at the sparsest density. Abbadingo specifies that the sparsest training set for 64-state target DFA problems is 1,521 strings but does not specify the number of training strings for 32-state problems. For 32-state problems we use training sets of 607 strings. This number has been found experimentally so that the likelihood of identifying low-error hypotheses using EDSM is similar to that of sparse 64-state problems. Since randomly created problem instances may vary in difficulty, each experiment is evaluated against exactly the same sets of problems. Additionally, in order to better study the behaviour of each experiment, the problems are grouped into sets according to their characteristics. Training sets may or may not be structurally complete, the target DFA may or may not have loop transitions, and the highest EDSM-scoring merges in the first step (at the APTA) may or may not contain a colour-compatible merge.

In Table 1, we describe the GA configurations which are used to evaluate our sets of problem instances. Each configuration specifies the chromosome length $k$, the population size $p$, the maximum number of generations allowed $g$, the tournament size $t$, the APTA reduction table parameters $\alpha$ and $\beta$, the crossover rate $c$, the mutation rate $m$, and the elitism rate $e$. Configurations marked with a dagger symbol (such as 32v1$^\dagger$) use the performance optimisation described in §5. The hyperparameters were chosen by inspecting the merges contained in APTA reduction tables as well as by trial-and-error.

In order to better evaluate the effectiveness with which our fitness function is directing the search, we also compare our results to a 'blind' variant of the GA. This works by selecting parents for crossover randomly without considering their fitness. Specifically, an initial population is created containing $p$ random chromosomes. Two parents are selected randomly and an offspring is created using crossover. The offspring is mutated according to the mutation rate and is added to the current population. This is repeated until the population size grows to $p \times g$ chromosomes, where $g$ is the maximum number of generations. This means that the algorithm will evaluate $p \times g$ chromosomes which is, at least, the number of chromosomes evaluated by the fitness-directed GA described earlier. The smallest DFA obtained after extending every chromosome is returned as the final hypothesis.

| For 32-state target DFAs, 607 training strings | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | $k$ | $p$ | $g$ | $t$ | $\alpha,\beta$ | $c$ | $m$ | $e$ |
| 32v1 | 8 | 100 | 50 | 5 | 25,1 | 0.8 | 0.01 | 0.1 |
| 32v2 | *best of two runs of 32v1* | | | | | | | |
| 32v3 | 8 | 200 | 100 | 5 | 25,1 | 0.8 | 0.01 | 0.1 |
| 32v3$^\dagger$ | 8 | 200 | 100 | 5 | 25,1 | 0.8 | 0.01 | 0.1 |
| For 64-state target DFAs, 1,521 training strings | | | | | | | | |
| | $k$ | $p$ | $g$ | $t$ | $\alpha,\beta$ | $c$ | $m$ | $e$ |
| 64v1 | 6 | 100 | 50 | 5 | 60,1 | 0.8 | 0.01 | 0.1 |
| 64v2 | 6 | 200 | 50 | 5 | 60,1 | 0.8 | 0.01 | 0.1 |
| 64v2$^\dagger$ | 6 | 200 | 50 | 5 | 60,1 | 0.8 | 0.01 | 0.1 |

Table 1: Genetic algorithm hyperparameters.

## 7. Results and Observations

In this section, we summarise and discuss the results that we obtained when running our GA in the configurations described in the previous section. A complete breakdown of these results as well as a description of the problem sets used can be found in Appendix A.

- On structurally complete training data, the GA is, compared to EDSM, much more successful in finding low-error ($\leq 1\%$) hypotheses compared to EDSM. For 32-state target DFA problems (see Table 2), the GA in configuration 32v3 identifies low-error DFAs in 88 out of 128 problem instances (69%) compared to EDSM which does so in 21 out of the same problems (16%).It must be pointed out that 20 out of the 21 low-error hypotheses found by EDSM were also found by the GA. When the target DFA increases to 64 states (see Table 3), 64v2 finds low-error DFAs in 36 out of 64 problem instances (56%) while EDSM does so in 8 out of the same problems (13%). Again, 7 out of the 8 hypotheses found by EDSM were also found by the GA. Similar observation as above can be made with respect to finding the exact target DFA where in its best parameter configuration the GA substantially outperforms EDSM.

- Using the faster estimation method to compute the fitness of a chromosome affects the quality of our results. When using the true fitness configuration 32v3 we find low-error DFAs in 88 out of 128 problem instances (69%). Using the exact same GA configuration 32v3$^\dagger$ which *estimates* fitness we find 75 low-error DFAs out of 128 problems (59%). In spite of this, the fitness-estimation method can still find low-error DFAs with considerably higher likelihood than EDSM while running significantly faster. A similar trade-off applies to 64-state target DFA problems.

- Tuning the parameters of the GA can yield noticeably better results. This can be seen in the parameter sets 32v1 and 32v3 where the population size and the maximum generations allowed parameters are doubled.

| | ≤ 1% Error | Exact Target | Mean Error (SD) | Mean Size (SD) |
|---|---|---|---|---|
| **Set A.32: Structurally complete problem set** | | | | |
| EDSM | 21 | 5 | 18% (19.5) | 72 (27) |
| 32v2 | 80 | 21 | 2.1% (6.2) | 35 (10) |
| 32v3 | 88 | 21 | 1.8% (6) | 34 (8) |
| 32v3[†] | 75 | 21 | 3.7% (9.5) | 37 (14) |

Table 2: Summary of results for 32-state target problems. Configurations with † use the optimisation described in §5.

| | ≤ 1% Error | Exact Target | Mean Error (SD) | Mean Size (SD) |
|---|---|---|---|---|
| **Set A.64: Structurally complete problem set** | | | | |
| EDSM | 8 | 1 | 32.3% (20.4) | 161 (60) |
| 64v2 | 36 | 3 | 5.9% (12) | 86 (40) |
| 64v2[†] | 28 | 4 | 8.6% (15.3) | 95 (50) |

Table 3: Summary results for 64-state target problems. Configurations with † use the optimisation described in §5.

- In each of the three adversarial cases we described earlier, the GA outperforms EDSM by a considerable margin. The mean error and hypothesis size over all problem instances is also greatly reduced. In all cases, the performance characteristics of the GA on 64-state target DFA problems are similar to those obtained on 32-state target DFA problems. This indicates that this method scales well to larger problems.
- Analysing the rate with which the GA converges on 32-state target DFA problems, shows that low-error hypotheses are found in an average of 23 generations. Interestingly, in 27% of the cases where the GA was not able to find a low-error hypothesis, the sizes of those hypotheses were still equal to the target DFA. If our criterion for success is finding a DFA equal in size to the target, this approach is even more promising.
- We also evaluate performance against a purely random search as well as the blind variant which does not use a fitness-directed search. On the same set of problems, the configuration 32v3 finds 88 out of 128 low-error DFAs whereas both a random search and the blind variant never manage to do so in spite of the fact that the blind variant is performing at least the same number of extensions as the GA. This confirms our intuition that the selection pressure applied by the fitness function is directing the search towards more productive regions of the search space. Interestingly, the mean error and DFA sizes of the hypotheses found by the blind variant are much better than those obtained by a random search. This result supports our belief that starting searches with high-reduction merges is a promising approach.

- On our hardware,[7] the optimised version of the GA using the best parameter configuration on 32-state problems takes an average of 19 seconds and 3,195 fitness evaluations to run. For 64-state problems, the algorithm takes an average of 295 seconds and 4,542 fitness evaluations. A direct runtime performance comparison with other evolutionary methods is not meaningful since the chromosome encoding schemes and fitness functions are completely different. For example, on noise-free training data, the method by Lucas and Reynolds (2005) performed a maximum of 1,000,000 evaluations $\times 50$ trials using a simpler and faster fitness function.

## 8. Conclusion

In this paper, we have described a GA that evolves short initial sequences of merges which, when extended using a greedy heuristic based on EDSM, results in a much higher likelihood of finding a low-error hypothesis. Our work is mostly comparable to Lucas and Reynolds (2005) since we also focused on Abbadingo-style problem instances and used a similar evaluation strategy. Their experiments show that their smart state labelling approach performs better than EDSM on target DFAs having between 4 and 16 states but is outperformed by EDSM on 32-state target DFA problems at a density of 3,275 training strings. In comparison, we evaluated our method on larger 32-state and 64-state target DFAs and find that it consistently outperforms EDSM. Moreover, we used just 607 training strings to learn 32-state target DFAs from. On the other hand, their method can deal with noisy data whereas we have not considered this challenge as yet.

We have also evaluated how our method behaves in several scenarios that are adversarial to EDSM and found that it outperforms EDSM in all these cases too. We have also proposed an optimisation strategy which can reliably estimate the size of a final hypothesis without needing to complete an entire merge sequence. This technique results in a significant speedup and may be applicable to other methods such as SAGE (Juillé and Pollack, 1998), Ed-Beam (Lang, 1999), and dfasat (Heule and Verwer, 2013) which all rely on constructing and evaluating large numbers of merge sequences to find good hypotheses.

Despite the promising results that we have obtained, the main weakness of our method lies in the high computational cost associated with computing the fitness of a candidate solution. Every chromosome (sequence of merges) must be extended using EDSM and, therefore, the number of times that EDSM is called equals the population size multipled by the number of generations. Although this cost is somewhat alleviated using our optimisation, we are now exploring new techniques to make our GA computationally feasible on target DFAs having 128 states or larger. These improvements will also allow us to perform more effective grid searches for better hyperparameters. Another enhancement we are currently investigating is modifying our algorithm to deal with noise in the training data. The techniques for dealing with noisy data used by Habrard et al. (2003) and Lucas and Reynolds (2005) are likely starting points for this investigation.

7. iMac 2019 with a six-core Intel i5 CPU running at 3.7 GHz and 24GB of DDR4 memory.

## References

Abbadingo One: Dfa learning competition. http://abbadingo.cs.nuim.ie, 1997. Accessed: 2017-05-19.

The Gowachin dfa learning benchmark. http://www.irisa.fr/Gowachin, 1997. Accessed: 2020-09-01.

Gecco: Learning dfas from noisy samples. http://cswww.essex.ac.uk/staff/sml/gecco/NoisyDFA.html, 2004. Accessed: 2017-05-19.

John Abela, François Coste, and Sandro Spina. Mutually compatible and incompatible merges for the search of the smallest consistent dfa. In *International Colloquium on Grammatical Inference*, pages 28–39. Springer, 2004.

Tobias Blickle and Lothar Thiele. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, 4:361–394, 1997.

Orlando Cicchello. A new limited search approach to learning abbadingo-style finite state automata. Master's thesis, The Faculty of Graduate Studies, University of Guelph, 2002.

Orlando Cicchello and Stefan C. Kremer. Beyond edsm. In Pieter W. Adriaans, Henning Fernau, and Menno van Zaanen, editors, *ICGI*, volume 2484 of *Lecture Notes in Computer Science*, pages 37–48. Springer, 2002. ISBN 3-540-44239-1.

David Combe, Colin de la Higuera, and Jean-Christophe Janodet. Zulu: An interactive learning competition. In Anssi Yli-Jyrä, András Kornai, Jacques Sakarovitch, and Bruce Watson, editors, *Finite-State Methods and Natural Language Processing*, pages 139–146, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-14684-8.

François Coste and Jacques Nicolas. Regular inference as a graph coloring problem. In *In Workshop on Grammar Inference, Automata Induction, and Language Acquisition (ICML' 97)*, pages 9–7, 1997.

Colin de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, New York, NY, USA, 2010. ISBN 9780521763165.

P. Dupont, L. Miclet, and E. Vidal. What is the search space of the regular inference? In Rafael C. Carrasco and Jose Oncina, editors, *Grammatical Inference and Applications*, volume 862 of *Lecture Notes in Computer Science*, pages 25–37. Springer Berlin Heidelberg, 1994. ISBN 978-3-540-58473-5.

Pierre Dupont. *Regular Grammatical Inference from Positive and Negative Samples by Genetic Search: the GIG Method*, pages 236–245. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994. ISBN 978-3-540-48985-6.

E Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978. ISSN 0019-9958.

David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989. ISBN 0201157675.

Amaury Habrard, Marc Bernard, and Marc Sebban. Improvement of the state merging rule on noisy data in probabilistic grammatical inference. In *Proceedings of the 14th European Conference on Machine Learning*, ECML'03, pages 169–180, Berlin, Heidelberg, 2003. Springer-Verlag. ISBN 978-3-540-20121-2.

Paul Halmos. *Naive Set Theory.* Van Nostrand, 1960. ISBN 0387900926. Reprinted by Springer-Verlag, Undergraduate Texts in Mathematics, 1974.

Marijn J. H. Heule and Sicco Verwer. Software model synthesis using satisfiability solvers. *Empirical Software Engineering*, 18(4):825–856, 2013.

Colin G. Johnson. Genetic programming with fitness based on model checking. In Marc Ebner, Michael O'Neill, Anikó Ekárt, Leonardo Vanneschi, and Anna Isabel Esparcia-Alcázar, editors, *Genetic Programming*, pages 114–124, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-71605-1.

Hugues Juillé and Jordan B. Pollack. A sampling-based heuristic for tree search applied to grammar induction. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, pages 776–783, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence. ISBN 0-262-51098-7.

Kevin J. Lang. Faster algorithms for finding minimal consistent dfas, 1999.

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*, pages 1–12, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. ISBN 978-3-540-68707-8.

Damián López and Pedro García. *On the Inference of Finite State Automata from Positive and Negative Data*, pages 73–112. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016. ISBN 978-3-662-48395-4.

S. M. Lucas and T. J. Reynolds. Learning deterministic finite automata with a smart state labeling evolutionary algorithm. 2005.

Jose Oncina and Pedro García. Identifying regular languages in polynomial time. In *Advances in Structural and Syntactic Pattern Recognition, Volume 5 of the Series in Machine Perception and Artificial Intelligence*, pages 99–108. World Scientific, 1992.

Kaya Oğuz. True scores for tartarus with adaptive gas that evolve fsms on gpu. *Information Sciences*, 525:1–15, 2020. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2020.03.072.

Hooman Shayani and Peter J. Bentley. A more bio-plausible approach to the evolutionary inference of finite state machines. In *Proceedings of the 9th Annual Conference Companion* *on Genetic and Evolutionary Computation*, GECCO '07, pages 2937–2944, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-698-1.

Astro Teller. The evolution of mental models. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 9, pages 199–219. MIT Press, 1994.

Fedor Tsarev and Kirill Egorov. Finite state machine induction using genetic algorithm based on testing and model checking. In *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '11, pages 759–762, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450306904.

Neil Walkinshaw, Kirill Bogdanov, Christophe Damas, Bernard Lambeau, and Pierre Dupont. A framework for the competitive evaluation of model inference techniques. In *Proceedings of the First International Workshop on Model Inference In Testing*, MIIT '10, pages 1–9, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450301473.

Wojciech Wieczorek and Olgierd Unold. Induction of directed acyclic word graph in a bioinformatics task. In Alexander Clark, Makoto Kanazawa, and Ryo Yoshinaka, editors, *The 12th International Conference on Grammatical Inference*, volume 34 of *Proceedings of Machine Learning Research*, pages 207–217, Kyoto, Japan, 2014. PMLR.

## Appendix A. Detailed Results and Additional Figures

Table 4 shows the composition of the sets of problem instances used to evaluate our GA. Sets A.32 through D.32 each contain 128 problem instances for 32-state target DFAs and training sets containing 607 strings. Sets A.64 through D.64 each contain 64 random problem instances for 64-state target DFAs and training sets containing 1,521 strings. Each of the sets A through D are characterised by whether the training sets are or are not structurally complete, whether the target DFA has loop transitions or not, and whether the rank of highest EDSM-scoring merges in the first step (at the APTA) contains at least one colour-compatible merge.

Tables 5 and 6 show the results we obtained using various configurations of our GA and compare them to monotonic EDSM. The tables show the number of problem instances where the final hypothesis has an error of $\leq 1\%$ on the testing set, the number of problem instances where we find the exact target DFA, the mean error of the final hypotheses over all problem instances, and the mean size of the final hypotheses over all problem instances.

| **32-state target DFAs** | | | |
|---|---|---|---|
| Set | Struct. compl. | Loops | CC merges in 1st step |
| A.32 | Yes | Mixed | Mixed |
| B.32 | Yes | No | Mixed |
| C.32 | Yes | Mixed | No |
| D.32 | No | Mixed | Mixed |
| **64-state target DFAs** | | | |
| Set | Struct. compl. | Loops | CC merges in 1st step |
| A.64 | Yes | Mixed | Mixed |
| B.64 | Yes | No | Mixed |
| C.64 | Yes | Mixed | No |
| D.64 | No | Mixed | Mixed |

Table 4: The composition of problem instance sets.

| | $\leq$ 1% Error | Exact Target | Mean Error (SD) | Mean Size (SD) |
|---|---|---|---|---|
| **Set A.32: Structurally complete problem set** | | | | |
| EDSM | 21 | 5 | 18% (19.5) | 72 (27) |
| Random | 0 | 0 | 48.7% (2.2) | 776 (21) |
| Blind | 0 | 0 | 29.6% (10.1) | 81 (10) |
| 32v1 | 60 | 12 | 4.4% (8.7) | 39 (14) |
| 32v2 | 80 | 21 | 2.1% (6.2) | 35 (10) |
| 32v3 | 88 | 21 | 1.8% (6) | 34 (8) |
| 32v3† | 75 | 21 | 3.7% (9.5) | 37 (14) |
| **Set B.32: No loops in target DFAs** | | | | |
| EDSM | 16 | 1 | 30.6% (20.3) | 74 (27) |
| 32v3† | 76 | 25 | 5.5% (13.1) | 39 (19) |
| **Set C.32: No colour-compatible merge in 1st merge** | | | | |
| EDSM | 0 | 0 | 43.3% (8.3) | 92 (9) |
| 32v3† | 66 | 15 | 6.1% (13.3) | 41 (20) |
| **Set D.32: Not structurally complete** | | | | |
| EDSM | 5 | 0 | 31.7% (17.8) | 77 (23) |
| 32v3† | 52 | 0 | 4.2% (10) | 36.8 (16) |

Table 5: Results for 32-state target problems. Configurations with † use the optimisation described in §5.

| | $\leq 1\%$ Error | Exact Target | Mean Error (SD) | Mean Size (SD) |
|---|---|---|---|---|
| **Set A.64: Structurally complete problem set** | | | | |
| EDSM | 8 | 1 | 32.3% (20.4) | 161 (60) |
| 64v1 | 17 | 1 | 12.6% (16) | 110 (50) |
| 64v2 | 36 | 3 | 5.9% (12) | 86 (40) |
| 64v2$^\dagger$ | 28 | 4 | 8.6% (15.3) | 95 (50) |
| **Set B.64: No loops in target DFAs** | | | | |
| EDSM | 6 | 0 | 38.4% (17.6) | 179 (50) |
| 64v2$^\dagger$ | 24 | 0 | 14.5% (19.4) | 112 (62) |
| **Set C.64: No colour-compatible merge in 1st merge** | | | | |
| EDSM | 1 | 0 | 45.9% (8.1) | 200 (23) |
| 64v2$^\dagger$ | 16 | 0 | 18.2% (20.6) | 126 (62) |
| **Set D.64: Not structurally complete** | | | | |
| EDSM | 4 | 0 | 35.6% (18.1) | 173 (52) |
| 64v2$^\dagger$ | 22 | 0 | 13.5% (18.7) | 109 (59) |

Table 6: Results for 64-state target problems. Configurations with $^\dagger$ use the optimisation described in §5.

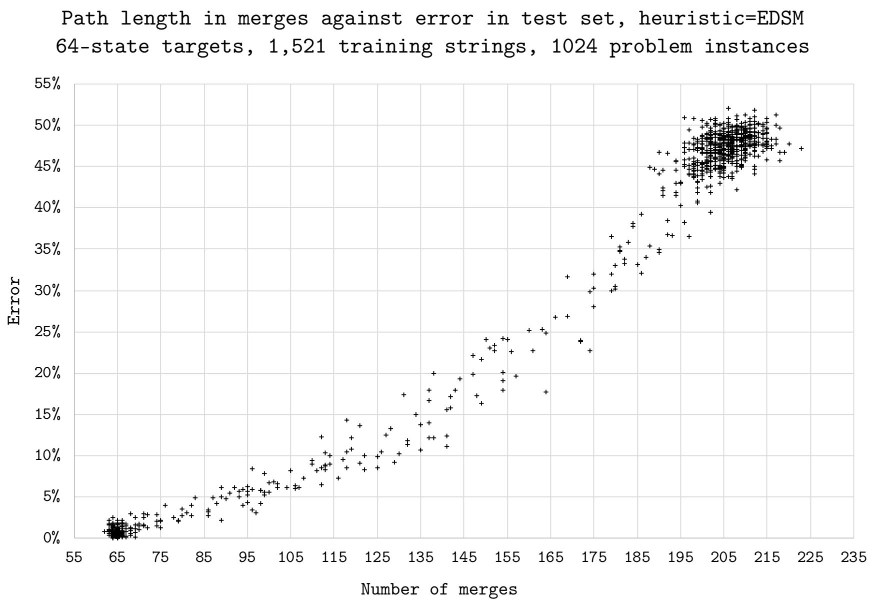

Figure 4: The length of a merge sequence against error in testing set.

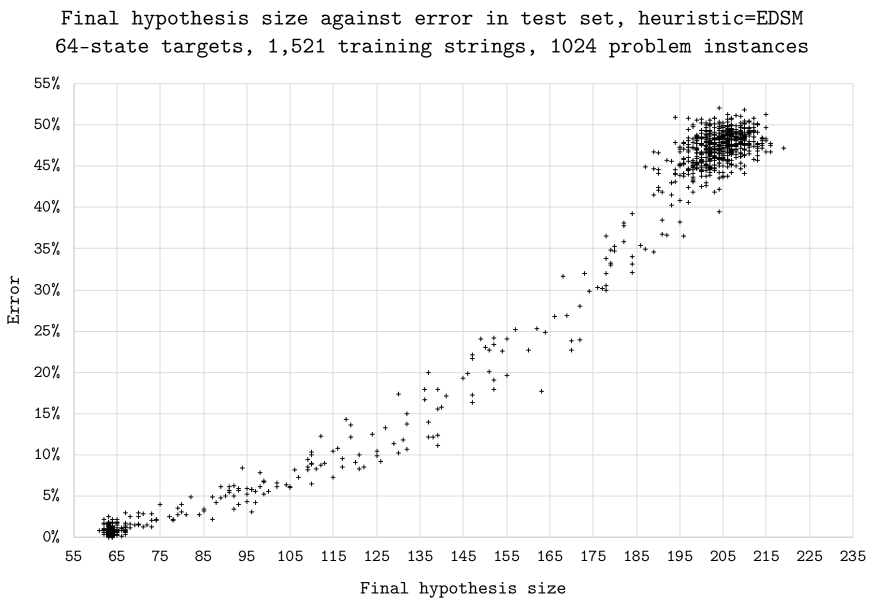

Figure 5: The size of a final hypothesis against error in testing set. Data is similar but not identical to Figure 4.

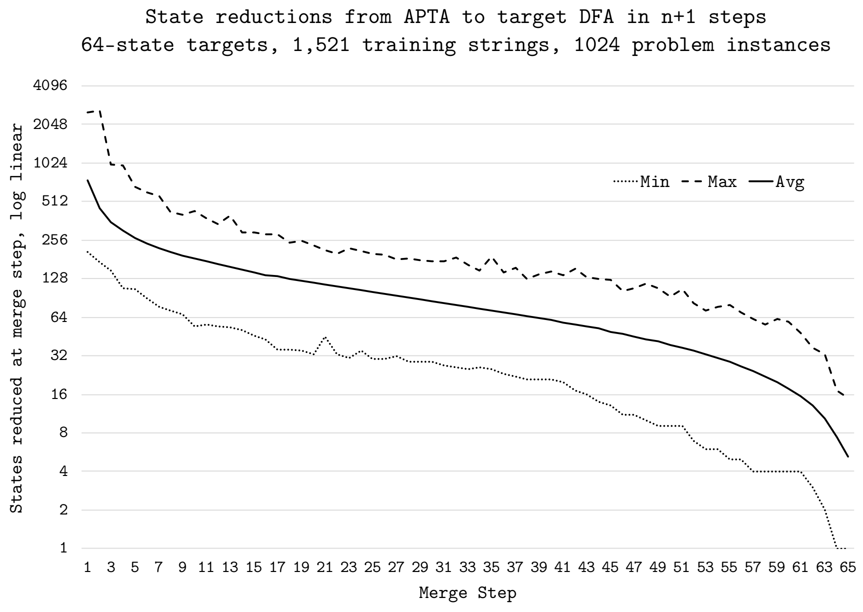

Figure 6: The number of states reduced in a partial hypothesis at each merge step.

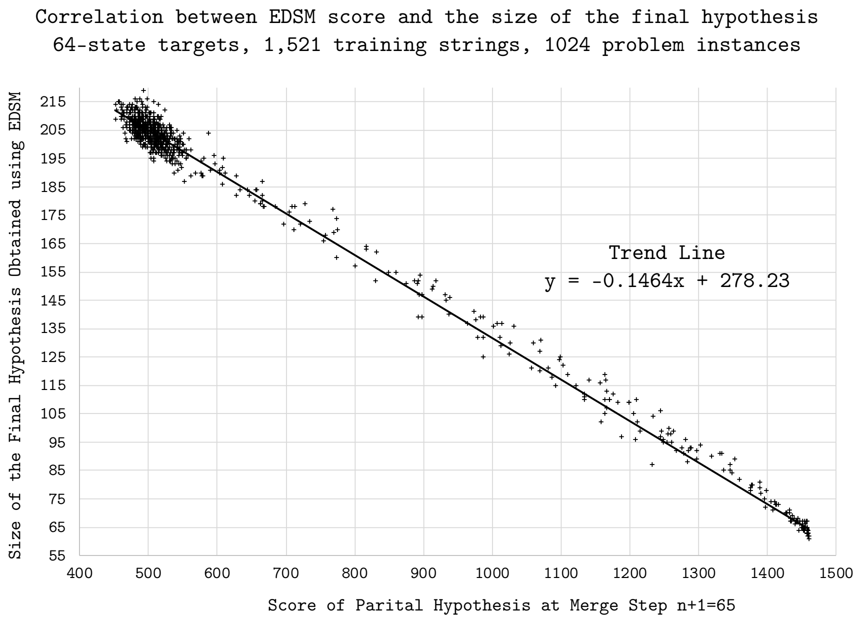

Figure 7: Cumulative EDSM score of a partial hypothesis at merge step $n + 1$ against the final hypothesis size.