An active learning algorithm for bidirectional deterministic finite automata

Simon Dieck[1] and Sicco Verwer[1]

Department of Software Technology, TU Delft, Netherlands
{S.Dieck,S.E.Verwer}@tudelft.nl

Abstract. In this paper, we present an L^*-style algorithm for actively learning a bidirectional deterministic finite automaton (biDFA) in polynomial time using three types of oracles. We show how the W-method for the equivalence oracle can be adapted to our algorithm and present a novel heuristic for choosing the orientation of states. With this algorithm, one can identify automata for a subset of the linear languages that includes but is not limited to the regular languages. Since the equivalence oracle is an important part of the algorithm, we also discuss complexity bounds for different versions of the language equivalence problem for biDFAs. These results, together with our algorithm, also prove complexity bounds for the biDFA minimisation problem. Finally, we provide an implementation of the algorithm and experimentally show its performance with different approximation heuristics.

1 Introduction

A deterministic finite automaton (DFA) is a versatile model that is used across a variety of different fields ranging from applications in biology for parsing DNA [18], in compilers as scanners [12], to cyber security for fingerprinting [9]. While there are several ways of obtaining DFAs, for many fields, it is useful to infer them by interacting with a system whose behaviour the DFA should replicate [7]. This process, normally referred to as active learning, ideally produces a DFA whose behaviour is equivalent to the system from which it was inferred. For example, one can use DFAs inferred from programs that were infected by malicious software to identify security vulnerabilities [5,6,2]. Similarly, one can analyse the behaviour of opaque models, like neural networks, by analysing a DFA surrogate model inferred from them instead [17].

One issue that all of these applications face is that DFAs are limited to recognising the regular languages. And while learning procedures for more expressive models exist, often based on a notion of substitutability, these algorithms are computationally significantly more expensive [4,23].

In this context bidirectional deterministic automata (biDFAs) are interesting. One can think of these models as DFAs with two reading heads. At the beginning, one reading head is placed at the start of the string, and the other one is placed at the end. In each step, the model decides on which reading head will read the next character. This way, a string is parsed from both ends by heads moving inwards. The process terminates if both heads meet. One can think of this meeting point as a "centre" of a string. For the deterministic version of these models, the decision of which reading head to move is also deterministic, and as such, every word has a fixed centre where the heads meet. Nondeterministic biDFAs can recognise all linear languages, while the deterministic version can recognise a subset of the linear languages that strictly includes the regular languages. Importantly, they can do this without requiring additional memory structures like, for example, pushdown automata [13,15].

While not significantly more expressive than DFAs, the additional languages biDFA can recognise are such that normally require counting, like $a^n b^n$. They can also recognise languages like palindromes, which notably even deterministic pushdown automata cannot. Further, Dieck and Verwer have shown that on palindrome-like regular languages biDFAs can be exponentially smaller than DFAs [8]. Especially in the field of inferring models from software, one can expect such patterns. If software produces some output at the beginning and end of a recursive function, the output will have a palindrome-like structure. Accordingly, an inference algorithm for biDFAs has potential uses in many fields but is of special interest for software inference.

In this paper, we show that the L^* inference algorithm for DFA can be adapted to infer a biDFA. We show that if an oracle for state orientation, that is which reading head is used in a state, is given, this algorithm has the same runtime guarantees as L^* for DFAs. We also prove that the algorithm produces a biDFA that is correct and minimal with regard to a centre function defined by the oracle. Since L^* requires an equivalence oracle, we analyse the complexity

of this problem, which is an open question for biDFAs, and prove results for a special case. The algorithm introduced in this work also proves that the biDFA minimisation problem is NP-complete if the biDFA recognises a regular language and in P if the centre function for the minimised model is equivalent to the centre function of the input model.

In practice, especially when inferring software, one has only access to membership queries. Accordingly, model-checking techniques such as the W-method [3] are often used to approximate an equivalence query heuristically [14]. In this work, we also prove that an adaptation of the W-method heuristic [3] works for biDFAs, giving similar guarantees. We also provide a heuristic that can be used for the state orientation oracle to decide which end of the string to read from in a given state.

Finally, we test the algorithm and heuristics experimentally. Here, we show that with the right setup for the heuristics, the algorithm consistently correctly identifies non-regular languages. This includes a palindrome language and even some that were chosen to be difficult for the heuristic. We also show that it often finds the globally minimal model on those languages.

1.1 Related work

Bidirectional automata, then called biautomata, were first introduced by Klíma and Polák [16]. While these models also read a string from both ends moving inwards and are almost equivalent to biDFAs, Klíma and Polák defined the transition function in such a way that these models could only recognise regular languages. This transition function was then changed to allow non-deterministic transitions by Holzer and Jakobi [13]. They analysed the language theoretical properties of these biautomata and showed that they could recognise all linear languages [13].

Jirásková and Klíma then adjusted the transition function of Holzer and Jakobi once again to be deterministic both in allowing only one transition per character and state and enforcing that all transitions from a single state need to read a character from the same end [15]. These models are equivalent to biDFAs since they implicitly introduce an orientation of the state and otherwise are fully deterministic. Jiráskova and Klíma prove most language theoretical properties of these models in their work and show that the class of languages they can recognise is a proper subset of the linear languages and a proper superset of the regular languages [15]. Dieck and Verwer provide a Myhill-Nerode style characterisation of the class of languages recognised by these models [8].

The contributions of these works regarding the language equivalence and minimisation problem, together with our novel contributions are summarised in Table 1.

Independently by Nagy very similar bidirectional models have been proposed in the context of DNA processing [18,19]. Whether these models are equivalent has not been proven yet.

Given B_1 and B_2, recognising L_1 and L_2, defining centres c_1 and c_2	L_1 or L_2 is regular	$c_1 = c_2$	No restrictions
Emptiness of intersection	NL-complete	**NL-complete**	Undecideable
Language equivalence problem	NL-complete	**In NL**	Unknown
Minimisation problem	**NP-complete**	**In P**	NP-hard

Table 1. Summary of some problems relating to biDFA. Results in bold are novel results from this paper. NP-hardness of the minimisation problem was proven by Dieck and Verwer [8] and the other known results by Jirásková and Klíma [15].

For the active learning algorithm, much of our work is based on the L^* algorithm introduced by Angluin [1]. However, for the counterexample processing we use the method first introduced by Rivest and Schapire [21,20].

2 Background

In this section, we will introduce some notation and results from other works that are used in this work. For notation, we will mostly refer to what was used by Dieck and Verwer [8] since we are focused on the deterministic version of these bidirectional models and will heavily use their characterisation. We assume familiarity with standard definitions and notations regarding formal languages and DFA.

For biDFAs we need to introduce the notion of a centre function c.

Definition 1 (Centre function). *A centre function $c : \Sigma^* \to \mathbb{N}$ maps a word $w \in \Sigma^*$ to the range $[0, |w|]$, where $|w|$ denotes the length of a word which is equal to the number of characters it contains.*

Given such a centre function, every word can be split into a left and a right half $w = w_{c,l} \cdot w_{c,r}$ s.t. if $w = \sigma_1\sigma_2 \dots \sigma_{|w|}$ then $w_{c,l} = \sigma_1 \dots \sigma_{c(w)}$ and $w_{c,r} = \sigma_{c(w)+1} \dots \sigma_{|w|}$.

For the definition of a biDFA we use the one given by Dieck and Verwer [8]:

Definition 2 (biDFA). *A bidirectional deterministic finite automaton (biDFA) is a six-tuple $B = (Q_l, Q_r, \Sigma, \delta, q_\lambda, F)$, where $Q = Q_l \cup Q_r$, with $Q_l \cap Q_r = \emptyset$, forms a finite set of states, Σ is an alphabet, $\delta : Q \times \Sigma \to Q$ a transition function, $q_\lambda \in Q$ the initial state and $F \subseteq Q$ the set of accepting states.*

For this definition to work on words and not just characters, δ is extended to $\delta^* : Q \times \Sigma^* \to Q$ with the following definition:

$$\delta^*(q, (\sigma_1 \dots \sigma_n)) = \begin{cases} q & \text{if } \sigma_1 \dots \sigma_n = \lambda \\ \delta(q, (\sigma_1 \dots \sigma_n)) & \text{if } n = 1 \\ \delta^*(\delta(q, \sigma_1), (\sigma_2 \dots \sigma_n)) & \text{if } n > 1 \text{ and } q \in Q_l \\ \delta^*(\delta(q, \sigma_n), (\sigma_1 \dots \sigma_{n-1})) & \text{if } n > 1 \text{ and } q \in Q_r \end{cases}$$

The language L accepted by B can then be defined as $\{w \in \Sigma^* | \delta^*(q_\lambda, w) \in F\}$. One can think of a biDFA as parsing a word by parsing a left string from left to right and a right string from right to left. Whenever B is in a state in Q_l, δ^* will process the next character from the left. When in a state of Q_r, it will process the next character from the right. With this, one can use a biDFA B to define a centre function c_B. To obtain $c_B(w)$ one can count the number of states in Q_l that were encountered when parsing w with B, except for the final state in which the process terminates. This is exactly the number of characters that were read from the left end of the string.

Dieck and Verwer have shown, given a centre function c and a language L one can define an equivalence relation on Σ^* as follows [8]:

$u \equiv_{c,L} v$ iff for all $m \in \Sigma^* : u_{c,l} \cdot m \cdot u_{c,r} \in L \iff v_{c,l} \cdot m \cdot v_{c,r} \in L$

In words, two words are considered inter-equivalent w.r.t. c and L if and only if all words that can be produced from them by inserting a word at their centre behave the same with regards to membership in L. As this interjection comes up often, we introduce a shorthand $u \cdot_c m = u_{c,l} \cdot m \cdot u_{c,r}$, where "$\cdot_c$" is a left-associative operator. We will denote the equivalence classes for this by $[w]_{c,L}$ where $[w]_{c,L} = \{w' \in \Sigma^* | w' \equiv_{c,L} w\}$.

Finally, there is the important restriction for a centre function of "strict stability" which was introduced by Dieck and Verwer [8]:

Definition 3 (Strict stability). *A centre function c is considered stable if for all words $w \in \Sigma^*$ and all characters $s \in \Sigma$ $c(w) \leq c(w \cdot_c s) \leq c(w) + 1$.*

A stable centre function c is considered "strictly stable" for a language L if for all $u, v \in \Sigma^*$ $u \equiv_{c,L} v$ implies that for all $s, s' \in \Sigma : c(u) = c(u \cdot_c s) \iff c(v) = c(v \cdot_c s')$

In words, a strictly stable centre function shifts the centre by either 0 or 1 if a single character is inserted and two inter-equivalent words shift by the same amount. Since every word is inter-equivalent to itself, that also means that for a single word, the centre shifts in the same direction independent of which character is inserted. Also note that every centre function defined by a biDFA is strictly stable.

Given this, Dieck and Verwer prove a Myhill-Nerode style theorem whose corollary is important for this work [8]:

Corollary 1. *For a given strictly stable centre function c and a language L the number $|Q|$ of states of the minimal biDFA defining the same centre function c and recognising L is equal to the number of equivalence classes of the inter-equivalence w.r.t. c and L. There exists an isomorphism between such a minimal biDFA and the "canonical" minimal biDFA w.r.t. c and L.*

This corollary means that we can identify a biDFA that is minimal w.r.t. a given centre function if we can identify the corresponding equivalence classes.

3 The algorithm

In this section, we will introduce the active learning algorithm for biDFA. The algorithm is in large parts an adaptation of the L^* algorithm introduced by Angluin [1] but uses a counterexample processing based on Rivest and Schapire [21,20]. For the description of the algorithm, we will assume the existence of a teacher that acts as a membership and equivalence oracle, like the original L^* algorithm, in addition to a centre function oracle. For a teacher of a language $L \subseteq \Sigma^*$ the oracles are expected to have the following capabilities:

The membership oracle can answer whether a given word $w \in \Sigma^*$ is in L or not. The equivalence oracle can answer for a given biDFA B whether the language recognised by B is equivalent to L. If it is not equivalent the oracle returns a counterexample. The centre function oracle for a fixed strictly stable centre function c will return for a given word $w \in \Sigma^*$ its centre $c(w)$.

We will discuss how each of the oracles can be realised in Section 3.2. Algorithm 1 shows the pseudocode for the main routine of the algorithm assuming such oracles are given.

input : A teacher T for a language L, alphabet Σ
output: A biDFA B which recognises L

1 $A \leftarrow \{\lambda\}$, $E \leftarrow \{\lambda\}$;
2 $d_{\lambda,\lambda} = T.\texttt{MembershipQuery}(\lambda)$;
3 $o(\lambda) =$ "l" if $T.\texttt{CentreQuery}(s) > T.\texttt{CentreQuery}(\lambda)$, "r" otherwise ; // For some $s \in \Sigma$
4 For all $s \in \Sigma$: $x_{\lambda,s,\lambda} = T.\texttt{MembershipQuery}(s)$;
5 **while** true **do**
6 **while** A *is not* ***complete*** **do**
  // $\exists a' \in A, s \in \Sigma : \forall a \in A : \exists e \in E : d_{a,e} \neq x_{a',s,e}$
7 $a^* \leftarrow a' \cdot_c s$ $A \leftarrow A \cup \{a^*\}$;
8 $o(a^*) =$ "l" if $T.\texttt{CentreQuery}(a^* \cdot_c s') > T.\texttt{CentreQuery}(a^*)$, "r" otherwise ; // for some $s' \in \Sigma$
9 For all $e \in E$: $d_{a^*,e} = x_{a',s,e}$;
10 For all $s'' \in \Sigma, e \in E$: $x_{a^*,s'',e} = T.\texttt{MembershipQuery}(a^* \cdot_c s'' \cdot_c e)$;
11 $B \leftarrow \texttt{BuildbiDFA}(A, d, x, o)$;
12 $x \leftarrow T.\texttt{EquivalenceQuery}(B)$;
13 **if** x *is empty* **then**
14 **return** B
15 **else**
16 $e^* \leftarrow \texttt{ProcessCounterexample}(x, B, T)$;
17 $E \leftarrow E \cup \{e^*\}$;
18 For all $a \in A$: $d_{a,e^*} = T.\texttt{MembershipQuery}(a \cdot_c e^*)$;
19 For all $a \in A, s \in \Sigma$: $x_{a,s,e^*} = T.\texttt{MembershipQuery}(a \cdot_c s \cdot_c e^*)$;

Algorithm 1: Active learning algorithm for biDFA

The algorithm at all times maintains two sets $A, E \subset \Sigma^*$, the set A of access words and the set E of extensions. Both are initialised with $A, E = \{\lambda\}$. In addition to these two sets it maintains two tables, D, the distinguishing table, with $|A|$ rows and $|E|$ columns and X, the extended table, with $|A||\Sigma|$ rows and $|E|$ columns. For $a \in A$ and $e \in E$, the distinguishing table has the entry $d_{a,e} = 1$ if $a \cdot_c e \in L$ and 0 otherwise. Similarly for $a \in A$, $s \in \Sigma$ and $e \in E$ the extended table has the entry $x_{a,s,e} = 1$ if $a \cdot_c s \cdot_c e \in L$ and 0 otherwise.

We consider A to be **complete** if for all $a' \in A$ and $s \in \Sigma$ there exists an $a \in A$ s.t. $x_{a',s,e} = d_{a,e}$ for all $e \in E$. If A is not **complete** we will add one $a' \cdot_c s$ to A for which for all $a \in A$ there exists an $e \in E$ s.t. $x_{a',s,e} \neq d_{a,e}$. One can think of A as the set of unique equivalence classes and, therefore, states that have already been identified. We know for all $a, a' \in A$ that $[a]_{c,L} \neq [a']_{c,L}$ since there exists an $e \in E \subset \Sigma^*$ s.t. $a \cdot_c e \in L$ and $a' \cdot_c e \notin L$ or vice versa, since $d_{a,e} \neq d_{a',e}$ for at least one $e \in E$. Since we want to maintain this property we will not add new access words outside of this routine to make A complete. This differs from the original L^* algorithm, which also adds new access words when obtaining a counterexample [1].

An important observation here is that we can make the queries to the membership oracle, that are necessary to fill the tables, even with only partial information on the centre function. If we know $c(a)$, we can form $a \cdot_c e$ for any $e \in E$. The same is true for $a \cdot_c s \cdot_c e$ if we know $c(a)$ and $c(a \cdot_c s)$. Further, since c is strictly stable, it suffices to know $c(a \cdot_c s)$ for one $s \in \Sigma$. This way, we can limit the number of times we need to call the centre function oracle. We can save all the information the algorithm needs to function by having a label $o(a)$ for all $a \in A$. This label will be "l" if we believe the state associated with a is in Q_l, and the centre function will shift forward by 1 upon inserting a single character and "r" otherwise. We can obtain this label by querying the centre function oracle for the centre of $a \cdot_c s$ for any $s \in \Sigma$ when a is added to A. Since $c(\lambda) = 0$ and all other access words are one-letter extensions of other access words, this is sufficient to build o starting from the initialisation.

Given a **complete** A, we can now construct a biDFA by creating a state $q_a \in Q_l$ for all $a \in A$ with $o(a) =$"l" and a state $q_{a'} \in Q_r$ for all $a' \in A$ with $o(a') =$"r". Since A is **complete** for every $a' \in A$ and $s \in \Sigma$ there exists a unique $a \in A$ s.t. $d_{a,e} = x_{a',s,e}$ for all $e \in E$. We will call this a $x(a', s)$. With this we can define $\delta(q_a, s) = q_{x(a,s)}$. Further, we will set q_λ to be the state created from $\lambda \in A$ and $F = \{q_a | d_{a,\lambda} = 1\}$. This fully defines a biDFA since we assume Σ is given.

Once a biDFA B is built from a **complete** set A we can use the equivalence oracle. If the oracle certifies that B recognises L, we terminate returning B. Otherwise, we process the counterexample w given by the oracle. Here, we use a method based on the work of Rivest and Schapire [21,20]. Since w is a counterexample, meaning either B accepts it, while it is not in L or vice versa, we can use it to identify a new extension that distinguishes a word $a' \cdot_c s$ from every row in the distinguishing table. There must exist a pre- and inter- and suffix of w, say w_p,w_i and w_s with $w_p \cdot w_i \cdot w_s = w$ and $c(w_p w_s) = |w_p|$ s.t. $\delta^*(q_\lambda, w_p w_s) = q_a$

but $[w_p w_s]_{c,L} \neq [a]_{c,L}$ and w.l.o.g. $w_p w_s \cdot_c w_i \notin L$ and $a \cdot_c w_i \in L$. If this was not the case, B would also reject w. Let $w_p w_s$ be the shortest of such pre- and suffixes. So if w.l.o.g. $w'_p \cdot \sigma \cdot w_s = w_p w_s$ for some $\sigma \in \Sigma$, and $\delta^*(q_\lambda, w'_p w_s) = q_{a'}$, then $x_{a',\sigma,w_i} \neq d_{a,w_i}$. But since the row associated with a is unique and otherwise equal with the row associated with $a', \sigma\ w_i$ distinguishes $a' \cdot \sigma$ from every row in the distinguishing table, which was our claim. Finally, observe that we can identify w_i since for some pre- and suffix $w_p w_s$ with $\delta(q_\lambda, w_p w_s) = q_a$ if $[w_p w_s]_{c,L} = [a]_{c,L}$ then $w_p w_s \cdot_c w_i \in L \iff a \cdot_c w_i \in L$. It, therefore, suffices to process w character by character and substituting the pre- and suffix already processed with the access trace of the state that was reached. A membership query is asked before and after the substitution. The first time the queries differ before and after substitution, the shortest pre- and suffixes are identified. This can also be used to improve the query complexity of this search by using binary search to find the shortest pre- and suffix from which predictions begin to differ, which is the result obtained by Rivest and Schapire [21,20]. Pseudocode for this counterexample processing is shown in Algorithm 2.

input : A teacher T for $L \subseteq \Sigma^*$, a biDFA $B = (Q_l, Q_r, \Sigma, \delta, q_\lambda, F)$, and a word $x \in \Sigma^*$ where T and B differ
output: A string $e^* \in \Sigma^*$

```
1  x_i ← x_1 ... x_|x|, p ← 0;
2  q ← q_λ, b_high ← |x|, b_low ← 0 ;
3  while b_high ≠ b_low do
4      b ← b_low + ⌊(b_high − b_low)/2⌋;
5      while p ≤ b do
6          if q ∈ Q_l then
7              s ← x_{i,1} ;          // Where x_{i,1} is the first character of x_i
8              x_i ← x_{i,2} ... x_{i,|x_i|} ;
9          else
10             s ← x_{i,|x_i|} ;      // Where x_{i,|x_i|} is the last character of x_i
11             x_i ← x_{i,1} ... x_{i,|x_i|−1} ;
12         p ← p + 1 ;
13         q ← δ(q, s) ;
14     Let a be the access word associated with q ;
15     if T.MembershipQuery(a ·_c x_i) ≠ T.MembershipQuery(x) then
16         b_high ← b ;
17     else
18         b_low ← b ;
19     p ← 0, x_i ← x;
20 return x_i
```

Algorithm 2: ProcessCounterexample

3.1 Correctness and Running time

While many of the key arguments have already been made in the previous section, we formalise the correctness of the presented algorithm with the following theorem:

Theorem 1. *Given a teacher T for a language L and a strictly stable centre function c, the presented algorithm will return a biDFA that is isomorphic to the canonical biDFA given c and L.*

Proof. Let c' denote the centre function defined by the returned biDFA $B = (Q_l, Q_r, \Sigma, \delta, q_\lambda, F)$. First, it is clear, that B recognises L since otherwise, a counterexample would exist.

We claim that B is minimal w.r.t. c' and L. Assume B was not minimal w.r.t. c' and L. This implies that there exist two distinct states $q, q' \in Q$ with $L_q = \{w \in \Sigma^* | \delta^*(q_\lambda, w) = q\}$, $L_{q'} = \{w \in \Sigma^* | \delta^*(q_\lambda, w) = q'\}$ s.t. there exists three inter-equivalence classes $[u]_{c',L}, [v]_{c',L}, [w]_{c',L}$ with $L_q \cap [u]_{c',L} \neq \emptyset$, $L_{q'} \cap [u]_{c',L} \neq \emptyset$, $L_q \cap [v]_{c',L} \neq \emptyset$ and $L_{q'} \cap [w]_{c',L} \neq \emptyset$ and $[v]_{c',L} \neq [w]_{c',L}$. The implication follows from the following observations: An inter-equivalence class $[u]_{c',L}$, that is intersected by both L_q and $L_{q'}$, must exist by pigeon hole principle since there are more states than inter-equivalence classes. But since q and q' are distinct there exists an extension $e \in E$ s.t. $d_{a_q,e} \neq d_{a_{q'},e}$ where a_q and $a_{q'}$ are the access words associated with q and q' respectively. But this implies L_q and $L_{q'}$ must also intersect inter-equivalence classes, that differ from each other, namely $[v]_{c',L}$ and $[w]_{c',L}$. At least one of them must be different from $[u]_{c',L}$ for which we choose w.l.o.g. $[v]_{c',L}$. This means there must exist a counterexample since there exists an x s.t. for all $v^* \in [v]_{c',L}$ and $u^* \in [u]_{c',L}$ $v^* \cdot_{c'} x \in L$ and $u^* \cdot_{c'} x \notin L$ or vice versa. But B will either accept or reject both of them depending on whether q is in F. However, this is a contradiction to B recognising L.

Since B is minimal w.r.t. c' and L it now suffices to show that $c = c'$ to prove the theorem. Here we can use the strict stability of c and c'. When an access word a is added to A we use the oracle to determine which direction the centre function shifts, which due to strict stability must be the same direction for all words in that inter-equivalence class. We therefore also know at this point that $[a]_{c,L} \neq [a']_{c,L}$ and $[a]_{c',L} \neq [a']_{c',L}$ for all $a' \in A$ with $a' \neq a$. But since this is an invariant strict stability guarantees $c = c'$. □

Theorem 1 shows that if a biDFA is returned by the algorithm, then it will be the minimal one w.r.t. c and L. However, it gives no guarantee for the algorithm terminating and how long it will take for the algorithm to terminate. For the L^*-style algorithms, they are often evaluated on how many calls to the membership and equivalence oracle they make [1,7]. We will do the same but also include the number of calls to the centre function oracle.

Theorem 2. *Given a teacher T for a language L and a centre function c the presented algorithm will identify the minimal biDFA $B = (Q_l, Q_r, \Sigma, \delta, q_\lambda, F)$*

after asking at most $O(|Q|)$ equivalence queries, $O(|Q|)$ centre function queries and $O(|Q|^2|\Sigma| + |Q|\log_2(|x|))$ membership queries, where $Q = Q_l \cup Q_r$ and x is the longest counterexample.

Proof. Note that the algorithm adds exactly $|Q|$ words to A. Since the centre function oracle is called only when a new word is added it can be called at most $|Q|$ times. Similarly, we have shown at the beginning of Section 3 that after asking the equivalence oracle and processing the given counterexample, exactly one element is added to E, and at least one element will be added to A. This also shows that the equivalence oracle can be called at most $|Q|$ times. The membership oracle is used in two places. First, when filling in the information in the tables. The size of the distinguishing table is $|A||E|$ while the size of the extended table is $|A||E||\Sigma|$ since $|A| \leq |Q|$ and the only time a new element is added to the extended table is when the equivalence oracle is called we can limit the number of times the membership oracle is called by $|A||E| + |A||E||\Sigma| \leq |Q|^2 + |Q|^2|\Sigma|$. The other time the membership oracle is used is when processing a counterexample in Algorithm 2. The subroutine is called at most $|Q|$ times since it is only called when the equivalence query is called. During the processing of a single counterexample x^*, the membership oracle is called twice each time a substitution is made. This happens $\log_2(|x^*|)$ times with the binary search. □

Essentially, we retain the same running time guarantees as modern L^*-style algorithms used for learning a DFA while learning a biDFA.

3.2 Heuristics

While the version of the algorithm relying on oracles is interesting from a theoretical perspective for many applications where DFA active learning is normally used, namely learning surrogate models for more complex models, these oracles are normally not available. In this section, we will discuss how each of the oracles can be implemented.

First, note that the membership oracle does not interact with a biDFA and, therefore, is independent of the model we are trying to learn. The implementation of the membership oracle just relies on what model is used as a teacher. When learning from a more complex model, this normally involves running the complex model with the queried input to obtain the output [14].

While in section 4 we will discuss how the language equivalence problem, and therefore oracle, can be solved if the teacher is a biDFA or DFA, this problem cannot be solved exactly for arbitrary models. There exist methods that heuristically predict equivalence while using only membership queries. The two most popular ones are the W-method [3] and random walks [14]. The W-method generates the set of all words of length less than a given parameter m, $W = \{u \in \Sigma^* || u| \leq m\}$ and then asks membership queries of both the teacher and the hypothesized model for all words $a \cdot u \cdot e$ for all $a \in A$, $u \in W$ and $e \in E$. It compares the output on both queries. When learning DFA, if all query pairs give the same

output, it guarantees that the models either recognise the same language or that the DFA would require at least m more states to recognise the same language [3]. We can easily adjust this method to work with biDFAs by querying for all words $a \cdot_c u \cdot_c e$ for all $a \in A, u \in W, e \in E$ instead. One can prove that such an adaptation retains the same guarantees by showing that the shortest distinguishing sequence being longer than m implies that at least m states need to be split into at least two states.

For the following Lemma and proof we call A a set of access words of B, since $\{\delta^*(q_\lambda, a) | a \in A\} = Q_l \cup Q_r$ and E a distinguishing set of B since for all $q, q' \in Q$ there exists an $e \in E$ s.t. $\delta^*(q, e) \in F$ and $\delta^*(q', e) \notin F$ or vice versa.

Lemma 1 (W-method for biDFA). *Given two biDFAs $B = (Q_l, Q_r, \Sigma, \delta, q_\lambda, F)$ and $B' = (Q'_l, Q'_r, \Sigma, \delta', q'_\lambda, F')$ which are minimal w.r.t. their shared centre function c, a set of access words A for B, a distinguishing set E for B and a set W of all words shorter or of equal size than a constant $w \in \mathbb{N}$.*

If for all $a \in A, u \in W, e \in E$ $\delta^*(q_\lambda, a \cdot_c u \cdot_c e) \in F \iff \delta'^*(q'_\lambda, a \cdot_c u \cdot_c e) \in F'$ then either $L_B = L_{B'}$ or $|Q'_l| + |Q'_r| > |Q_l| + |Q_r| + w$.

Proof. Since both B and B' are minimal w.r.t. c they are isomorphic to the set of all inter-equivalence classes $[u]_{c,L_B}$ and $[u]_{c,L_{B'}}$. This immediately gives us that for all $u \in \Sigma^*$ $[u]_{c,L_{B'}} \subseteq [u]_{c,L_B}$ since for all u' with $[u']_{c,L_B} \neq [u]_{c,L_B}$ there exists an $e \in E$ s.t. $u' \cdot_c e \in L_{B'}$ and $u \cdot_c e \notin L_{B'}$ or vice versa. Now assume there exists a $u, u' \in \Sigma^*$ s.t. $[u]_{c,L_{B'}} \neq [u']_{c,L_{B'}}$ but $[u]_{c,L_{B'}} \subset [u]_{c,L_B}$ and $[u']_{c,L_{B'}} \subset [u]_{c,L_B}$. This means $L_B \neq L_{B'}$ and there exists a word $s \in \Sigma^*$ s.t. w.l.o.g. $u \cdot_c s \in L_{B'}$ but $u' \cdot_c s \notin L_{B'}$.

First observe that $[u]_{c,L_{B'}} \neq [u']_{c,L_{B'}}$ implies $[u \cdot_c s_1]_{c,L_{B'}} \neq [u' \cdot_c s_1]_{c,L_{B'}}$ since $u \cdot_c s_1 \cdot_c s_2 \ldots s_{|s|} \in L$ and $u' \cdot_c s_1 \cdot_c s_2 \ldots s_{|s|} \notin L$. But $[u \cdot_c s_1]_{c,L_{B'}} \subseteq [u \cdot_c s_1]_{c,L_B}$ and $[u' \cdot_c s_1]_{c,L_{B'}} \subseteq [u \cdot_c s_1]_{c,L_B}$. This means one new state implies another new state until $[u \cdot_c s_1 \ldots s_k]_{c,L_B} = [u]_{c,L_B}$ and $u \cdot_c s_1 \ldots s_k \in [u']_{c,L_{B'}}$ for some $k \leq |s|$. Therefore, B' has an additional state compared to B for each unique state in B that is encountered when processing $\delta^*(q_u, s)$. In this case let us partition s into $s^* = s_1 \ldots s_k$ and $s' = s_{k+1} \ldots s_{|s|}$. Since $[u \cdot_c s^*]_{c,L_B} = [u]_{c,L_B}$ and $u \cdot_c s^* \in [u']_{c,L_{B'}}$, we can construct $\hat{u}' = u \cdot_c s^*$ s.t. $\hat{u}' \in [u]_{c,L_B}$ with w.l.o.g. $u \cdot_c s' \in L_{B'}$ and $\hat{u}' \cdot_c s' \notin L_{B'}$. Note that if s' contains some prefixes that again loop back to $[u]_{c,L_B}$ and $[u']_{c,L_{B'}}$ we can drop them until we obtain the shortest possible string s'' s.t. s'' distinguishes $[u]_{c,L_{B'}}$ and $[u']_{c,L_{B'}}$. Note that every state encountered when processing $\delta^*(q_u, s^* \cdot_c s'')$ leads to an additional state in B'.

Now assume $|Q'| \leq |Q| + w$ but $L_B \neq L_{B'}$. Since less than w new states can be encountered when processing $s^* \cdot_c s''$, and only s^* loops back to a previously encountered state $|s^*| + |s''| < w$. But then we can use that $s^* \cdot_c s''$ is in W and $\lambda \in E$, to observe that $\delta^*(q_\lambda, u \cdot_c s^* \cdot_c s'' \cdot_c \lambda) \iff \delta'^*(q'_\lambda, u \cdot_c s^* \cdot_c s'' \cdot_c \lambda)$. But this is a contradictions, since $u \cdot_c s^* \in [u']_{c,L_{B'}}$ and $\delta^*(q_\lambda, u \cdot_c s^*) = \delta^*(q_\lambda, u)$ but s'' distinguishes $[u]_{c,L_{B'}}$ and $[u']_{c,L_{B'}}$. □

While the W method offers nice guarantees it is computationally very expensive, as it requires $|A||E||\Sigma|^w$ membership queries to both models. Accordingly, the more commonly used method in practice is random walks. Here, long strings

are generated, sometimes with some problem-specific heuristics, and then the output of both models is compared on these strings. Since this only requires a membership oracle it is compatible with the algorithm presented in this work. The disadvantage of the random walk method is that it provides no theoretical guarantees and blows up the sizes of the counterexamples. Therefore, it is highly recommended to use the binary search implementation of the counterexample processing shown in algorithm 2.

Lastly, we will discuss how the centre function oracle can be implemented. Unless P=NP, a centre function oracle, that runs in polynomial time but results in the globally minimal biDFA will not be possible even if a biDFA is given as a teacher since that would result in a P-time algorithm for the minimisation problem with a regular language, which we show in Section 4 to be an NP-complete problem. In the more realistic setting where the teacher is some opaque model, which we can only interact with through membership queries and the centre function is no longer given, it is, therefore, unrealistic to expect a heuristic that leads to a globally good solution. Accordingly, we propose a greedy heuristic that attempts to minimise the immediate number of new states that would appear as a result of fixing the centre function. This heuristic is shown in Algorithm 3.

A simple summary of the heuristic is that it chooses the direction in such a way that the new rows that will be added to the extended table have the maximum overlap with already existing rows in the distinguishing table. If a newly added row in the extended table overlaps with an existing one the distinguishing table one can think of this as an indication that likely no new state will be introduced by this row. The heuristic therefore choses the direction of a state in such a way that locally the number of states is greedily minimised. If no direction is better, it flips a coin. One can think of this as looking one step into the future since only single-character extensions are considered.

One could also extend the heuristic by running the algorithm for n steps and choosing the direction that minimised the number of states after those steps. But since each step requires choosing a direction again, this would require making $2^n|A||E||\Sigma|$ membership queries. This quickly becomes infeasible but might be worth considering for very small values of n.

As given, there are instances where the heuristic of Algorithm 3 will result in Algorithm 1 taking exponential time in expectation if we assume E always contains a distinguishing sequence if it exists. For example $L = \{a, b\}^n(c\{a, b\}^{10})^n$ over the alphabet $\Sigma = \{a, b, c\}$ is problematic for the heuristic. Since every time the algorithm chooses a state to read from the left without choosing right-oriented states for the next 11 successors, the number of minimal states of a model which respect this centre function will increase by 12. But in most situations, this language contains no information locally. If a c is read from either left or right, it leads to a sink state unless it is the final c in the right sequence. Otherwise, reading any other character from the left or right will introduce a new state. Therefore, the overlap with existing rows for both is 1 most of the time. A previously observed state is only encountered again in a one-step future if the random choice at the end of Algorithm 3 works out favourably often enough.

input : A word $a^* = a \cdot_c s'$, a teacher T, a set of access words A, a set of distinguishing words E, a partial centre function o and a distinguishing table d
output: An orientation $o(a^*)$

```
1  l ← 0 ;                    // An overlap counter for assuming o(a*) =''l''
2  r ← 0 ;                    // An overlap counter for assuming o(a*) =''r''
3  for all s ∈ Σ do
4  |  Assume o(a*) =“l”;
5  |  x^l_{a*,s,e} =MembershipQuery(a* ·_c s ·_c e);
6  |  if ∃a ∈ A : ∀e ∈ E : d_{a,e} = x^l_{a*,s,e} then
7  |  |  l ← l + 1 ;
8  |  Assume o(a*) =“r”;
9  |  x^r_{a*,s,e} =MembershipQuery(a* ·_c s ·_c e);
10 |  if ∃a ∈ A : ∀e ∈ E : d_{a,e} = x^r_{a*,s,e} then
11 |  |  r ← r + 1 ;
12 if l > r then
13 |  return o(a*) = “l”
14 else if r > l then
15 |  return o(a*) = “r”
16 else
17 |  return o(a*) = “l” or o(a*) = “r” with probability 0.5 each
```

Algorithm 3: Heuristic for choosing state orientation

However, while this example gives a good idea that the heuristic performs worse the longer and more generic the pumpable sequences are, it is not very useful for the actual application of the algorithm. This is due to the assumption that E contains a distinguishing sequence if it exists. Generally, the heuristic approach relies on an accurate prediction of inter-equivalence. This is also the main motivation for using the counterexample processing of Algorithm 2. If we did not have the invariant that entries in A represent unique states then in a heuristic setting we would need to guarantee that entries in A that cannot be distinguished by E get assigned the same orientation. However, after adding new entries to E one might discover that those states are not equivalent but it would be no longer possible to assign a new orientation to one of the states. By setting up the algorithm as we did we delay orientation assignments as long as possible. This should lead to more accurate assessments of overlap by the heuristic since E will contain more entries. For our experiments, we wanted to further improve this overlap assessment by initialising E to contain several words. We achieved this in two different ways. One was to simply add all words up to a certain length to E. The other method was to run the algorithm for some iterations and then restart it while retaining E between restarts. We call this second method **Reset and Remember**.

3.3 Experiments

Setup For experiments, we implemented the algorithm in C++ and ran it on several simple languages as a proof of concept. The implementation can be found at `https://anonymous.4open.science/r/biDFA_L_star-5F08/`. For each language, the teacher was given in the form of a biDFA. However, the algorithm only had access to it in the form of membership queries to simulate the more common use case. For the equivalence query, we implemented the W-method, and for the centre function oracle, we implemented algorithm 3. Additionally, we used three methods to initialise the algorithm, one **standard** method where $A = E = \{\lambda\}$ and one **μ-warm start** method, where E is initialised as $\{s \in \Sigma^* || s| \leq \mu\}$. Finally, we implemented the **Reset and Remember** method described at the end of section 3.2. For this, we reset the algorithm for the kth time if a model with at least 2^{1+k} states was discovered. The algorithm terminated if it predicted the smallest so far known model twice in a row. For all methods, we also set a limit to forcibly terminate once the distinguishing table A contained more than 100 rows. The algorithm was used 10 times on each language for each experiment, and we recorded the average runtime, the number of times the forced termination occurred, as well as the maximum, minimum, and average number of states of the final model, excluding the ones that were forcibly terminated.

The languages we tested on where $L_1 = a^n b^n$, $L_2 = a^n ccb^n$, $L_3 = a^n (bb)^n$ as some simple linear languages that give an idea of how the algorithm performs on these simple variations of a two-sided pump pattern. We also ran experiments on a regular language $L_4 = (abbb)^n$, which illustrates some of the pitfalls of the algorithm. To showcase the strength of the algorithm, we used the palindrome language over $\Sigma = \{a, b, c\}$, which is $L_5 = ww^{-1}$ a string followed by its reversed string. Finally, we chose two languages designed to challenge our proposed heuristic of Algorithm 3. The chosen languages try to avoid revealing local information immediately. These languages are $L_6 = ((\{a, b\})^2)^n (b\{a, b\}^5)^n$ and $L_7 = (\{a, b\}^2 a\{a, b\}^2)^n (\{a, b\}^2 b\{a, b\}^2)^n$.

Experimental results A table with the full results can be found in Appendix A. , and a less detailed version is in Table 2.

Experiments with the standard initialisation show that L_6 requires an m value of 7 while L_7 requires one of 8 to not return a trivially incorrect model that accepts the empty word and rejects everything else. For all other languages, an m value of 3 sufficed. One can reduce the value of m by the value of μ since the W method compares outputs on all strings $a \cdot_c u \cdot_c e$ for all $a \in A, u \in W$ and $e \in E$. Normally E initially contains only λ. If strings of length μ are added, the W-method can find μ longer counterexamples during the first equivalence query.

In the experiments, both the **μ-warm start** method and **Reset and Remember** method were successful in making the heuristic notably more performant. Without it, the algorithm essentially always failed on languages L_5 and L_6 and struggled with L_2. Combining the two methods showed some improvement in the average number of states and positively impacted running time when

		$m = 3, 7, 8$ $\mu = 0$ "RR"=No	$m = 1, 5, 6$ $\mu = 2$ "R and R"=No	$m = 3, 7, 8$ $\mu = 0$ "R and R"=Yes	$m = 1, 5, 6$ $\mu = 2$ "R and R"=Yes
L_1	#MinDisc	6	10	10	10
	#ForcedTerm	0	0	0	0
	Average #states	5.8	3	3	3
L_2	#MinDisc	5	10	10	10
	#ForcedTerm	0	0	0	0
	Average #states	7.1	5	5	5
L_3	#MinDisc	6	5	10	8
	#ForcedTerm	0	0	0	0
	Average #states	7.3	5.1	4	4.5
L_4	#MinDisc	1	1	1	5
	#ForcedTerm	0	0	0	0
	Average #states	8.8	7.4	9.0	6.3
L_5	#MinDisc	2	10	10	10
	#ForcedTerm	8	0	0	0
	Average #states	5	5	5	5
L_6	#MinDisc	0	1	0	2
	#ForcedTerm	10	3	8	5
	Average #states	-*	14.4	18	10.8
L_7	#MinDisc	2	4	6	7
	#ForcedTerm	0	0	0	0
	Average #states	15.8	12.9	14.2	12.6

Table 2. Experimental results for different initialisation methods. Each setup was run 10 times. #MinDisc is how often the globally minimal model was discovered. #ForcedTerm is how often the algorithm terminated because a hypotheses with more than 100 states was formed. Average # states is the average size of the inferred models, excluding the ones that were forcibly terminated.

compared to running **Reset and Remember** alone. Generally, the **Reset and Remember** method had a worse impact on running time than the **μ-warm start** method.

When run with one of the initialisation methods, the algorithm almost always finds a minimal size model for our simple languages L_1 and L_2. At the same time, only **Reset and Remember** was very consistent on L_3. It also often finds these minimal size models for these simple languages when run without adding extra words to E. On the palindrome language L_5 however, it seems the initialisation methods are necessary. Without them, the model struggled to find the minimal size model and even often encountered the forced termination. Nevertheless, if one of the initialisation methods was used, the algorithm consistently found a minimal-size model for the palindrome language. For L_4, the regular language, one can however observe that the algorithm rarely finds the minimal size model with only the combined initialisation performing well. The minimal size model orients all states in the same direction, and as soon as one non-sink state is

oriented differently the number of required states is almost doubled. However, the heuristic cannot obtain this information by looking only one step into the future and thus struggles to find a minimal-size model. Surprisingly though the algorithm performed well on L_6 and L_7 which were designed to exploit the same problems of the centre function heuristic. While the algorithm, even with the initialisation methods, still sometimes hit forced termination on those languages it often found a small model that recognised those languages and sometimes even a minimal-size model.

4 Language theoretical properties

Language equivalence The algorithm that is presented in this work needs access to an equivalence oracle that produces a counterexample if languages are not equivalent. However, the complexity of the problem of deciding whether two arbitrary biDFA recognise the same language is currently not known. Jirásková and Klíma have shown that deciding whether the intersection of two languages given as two biDFAs is empty, is undecidable [15]. Therefore, it is not even clear if the language equality problem will be decidable. We however conjecture that the problem is at least in co-NP. Jirásková and Klíma have shown that the problem is in NL if one of the languages is regular [15]. For the non-regular case we believe that for a biDFA to correctly identify a two-sided pump-pattern like uv^nwx^ny, where both v and x are non-empty it will have to place the centre in between the copies of v and x. If this placement can be limited to at most some constant it would force two biDFAs whose languages have a significant overlap to define similar centre functions. If one can prove and combine these results it should be possible to limit the size of the smallest counterexample.

A simpler version of the language equality problem we can consider, even if the defined languages are non-regular, is if two biDFA define the same centre function.

Lemma 2. *Given two biDFAs* $B_1 = (Q_l, Q_r, \Sigma, \delta, q_\lambda, F)$ *and* $B_2 = (Q'_l, Q'_r, \Sigma, \delta', q'_\lambda, F')$ *which recognise* L_1 *and* L_2 *and define the centre functions* c_1 *and* c_2 *respectively. If* $c_1 = c_2$ *determining whether* $L_1 \cap L_2 = \emptyset$ *is NL-complete.*

Proof. We can construct an intersection biDFA B^I that recognises $L^I = L_1 \cap L_2$ as follows: $Q^I_l = \{q_{ij} | i \in Q_l, j \in Q'_l\}$, $Q^I_r = \{q_{ij} | i \in Q_r, j \in Q'_r\}$, Σ stays the same, the starting state is $q_{q_\lambda, q'_\lambda}$ and $F^I = \{q_{ij} | i \in F, j \in F'\}$. Finally δ^I is defined as $\delta^I(q_{ij}, s) = q_{\delta(q_i,s),\delta(q_j,s)}$. This construction is possible since both biDFAs respect the same centre function and thus $\delta^*(q_\lambda, w) \in Q_l \iff \delta'^*(q'_\lambda, w) \in Q_l$. From this observation, it also follows that B^I defines the same centre function as B_1 and B_2. Accordingly deciding the emptiness of the intersection comes down to deciding whether L^I is empty. However, to decide this we do not need to construct B^I in its entirety. For an NL algorithm it suffices to always simply remember the current state it is in, starting with q^I_λ, guessing a transition and constructing the next state based on the guess. Since the size of $Q^I \leq |Q||Q'|$

the algorithm can terminate after making that many guesses and not finding a state in F^I. This gives us an NL algorithm for deciding the emptiness of the intersection since NL is closed under complementation. Since Jirásková and Klíma have shown that the emptiness problem is NL-hard [15] this concludes the proof. □

The core idea of the proof is that once the centre function is fixed one can construct an intersection biDFA which respects the same centre function.

Corollary 2. *The language equality problem for two biDFAs B_1 and B_2 defining L_1 and L_2 respectively, that both define the same centre function c is in NL.*

To prove the corollary one can use the standard construction of $L_1 \cap \overline{L_2} = \emptyset$ and $\overline{L_1} \cap L_2 = \emptyset$ implies $L_1 = L_2$.

Minimisation The algorithm shown in the previous section also answers some questions about the complexity of the biDFA minimisation problem. One can define this problem as given a biDFA B which recognises L and a constant k to decide whether there exists a biDFA B^* that also recognises L with $|Q^*| \leq k$.

Theorem 3 (Minimisation complexity). *Given a membership and equivalence oracle for L there exists a nondeterministic polynomial time algorithm that solves the biDFA minimisation problem which requires a polynomial number of calls to the membership oracle and a linear number of calls to the equivalence oracle.*

Proof. Note that there are only two possible outcomes whenever algorithm 1 calls the centre function oracle. A nondeterministic algorithm can guess the outcome and otherwise run algorithm 1 as described. The algorithm will return "yes" if the final biDFA has at most k states and "no" otherwise. Theorems 1 and 2 guarantee that the returned algorithm recognises L and that this procedure had a nondeterministic polynomial running time.

Since the input biDFA to the minimisation problem can be given as the teacher, and therefore the membership query can be solved in linear time, the complexity of the minimisation problem depends on the complexity of the language equivalence problem. This leads us to the following corollary:

Corollary 3. *Given a biDFA B recognising the language L and defining the centre function c and a constant k the minimisation problem is:*

1. *In P when restricted to finding a biDFA that also defines c.*
2. *NP-complete if L is regular.*

Proof. 1. follows from theorem 3 and corollary 2 as well as the observation that a biDFA can be used as an oracle for its own centre function.

2. follows from Theorem 3 and Dieck and Verwer's proof that the problem is NP-hard [8] as well as Jirásková and Klíma's proof that the language equivalence problem is NL-complete if L is regular [15].

5 Discussion

In this work, we have introduced an active learning algorithm for biDFAs and some proofs about the complexity of the language equivalence problem for biDFAs. Together this also answered some open questions about the complexity of the biDFA minimisation problem. When combined with the heuristics we provided for the oracle required by the algorithm, we believe this work to be an important step to move bidirectional automata from a theoretical novelty towards something that can be used in practice. Indeed for DFA applications where the memory footprint is of significant importance, one could use the NP algorithm for minimisation introduced in this work. For the more common reverse engineering applications in security where DFA are currently learned one could experiment with the algorithm introduced in this work. For this, we expect that there is also significant room for improving the heuristics we provided. Especially the centre function heuristic we proposed, while working well on toy examples, provides no theoretical guarantees. Since beyond their superior expressiveness biDFA are also interesting because they can be potentially exponentially smaller than DFA when recognising regular languages [8], it would be desirable to find a centre function heuristic that works well on regular languages.

Another issue that keeps these results from being applicable to many real-life scenarios is that often not DFA are learned but Mealy machines [7,14]. Nevertheless, with the theoretical foundations provided in this work, we expect the same algorithm can used to learn bidirectional Mealy machines, similarly to how L^* is often used to learn them [14].

If one could show that the language equivalence problem is indeed in co-NP it would likely also imply that biDFAs are learnable in the limit [10]. If that is the case it would also be interesting if passive learning algorithms [22] or exact algorithms [11], that learn from a fixed sample could be adapted to learn biDFAs instead.

From a theoretical perspective, there are also some open questions left. Most importantly, whether the language equality problem is indeed as we conjectured in co-NP for the general case. Together with Theorem 3 this would imply that minimisation is also in co-NP.

For two biDFA that define the same centre function we have proven membership in NL for the equivalence problem and membership in P for the minimisation problem but completeness remains an open question.

In conclusion, we believe that bidirectional automata can be useful in real-life applications and to that end, we have in this work provided a foundation upon which such methods can be built and with which initial experiments can be conducted.

References

1. Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

2. Chia Yuan Cho, Domagoj Babić, Pongsin Poosankam, Kevin Zhijie Chen, Edward XueJun Wu, and Dawn Song. Mace: Model-inference-assisted concolic exploration for protocol and vulnerability discovery. In *20th USENIX Security Symposium (USENIX Security 11)*, 2011.
3. Tsun S. Chow. Testing software design modeled by finite-state machines. *IEEE transactions on software engineering*, 4(3):178–187, 1978.
4. Alexander Clark and Rémi Eyraud. Polynomial identification in the limit of substitutable context-free languages. *Journal of Machine Learning Research*, 8(8), 2007.
5. Jonathan E Cook and Alexander L Wolf. Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 7(3):215–249, 1998.
6. Weidong Cui, Jayanthkumar Kannan, and Helen J Wang. Discoverer: Automatic protocol reverse engineering from network traces. In *USENIX Security Symposium*, pages 1–14, 2007.
7. Colin De la Higuera. *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.
8. Simon Dieck and Sicco Verwer. On bidirectional deterministic finite automata. In *International Conference on Implementation and Application of Automata*, pages 109–123. Springer, 2024.
9. Domenico Ficara, Stefano Giordano, Gregorio Procissi, Fabio Vitucci, Gianni Antichi, and Andrea Di Pietro. An improved dfa for fast regular expression matching. *ACM SIGCOMM Computer Communication Review*, 38(5):29–40, 2008.
10. E Mark Gold. Language identification in the limit. *Information and control*, 10(5):447–474, 1967.
11. Marijn JH Heule and Sicco Verwer. Exact dfa identification using sat solvers. In *Grammatical Inference: Theoretical Results and Applications: 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings 10*, pages 66–79. Springer, 2010.
12. Alfred V Hoe, Ravi Sethi, and Jeffrey D Ullman. *Compilers—principles, techniques, and tools*. Pearson Addison Wesley Longman, 1986.
13. Markus Holzer and Sebastian Jakobi. Minimization and characterizations for biautomata. *Fundamenta Informaticae*, 136(1-2):113–137, 2015.
14. Malte Isberner, Falk Howar, and Bernhard Steffen. The open-source learnlib: a framework for active automata learning. In *Computer Aided Verification: 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I 27*, pages 487–495. Springer, 2015.
15. Galina Jirásková and Ondřej Klíma. On linear languages recognized by deterministic biautomata. *Information and Computation*, 286:104778, 2022.
16. Ondřej Klíma and Libor Polák. On biautomata. *RAIRO-Theoretical Informatics and Applications*, 46(4):573–592, 2012.
17. Edi Muškardin, Bernhard K Aichernig, Ingo Pill, and Martin Tappler. Learning finite state models from recurrent neural networks. In *International Conference on Integrated Formal Methods*, pages 229–248. Springer, 2022.
18. Benedek Nagy. On a hierarchy of $5' \to 3'$ sensing watson–crick finite automata languages. *Journal of Logic and Computation*, 23(4):855–872, 2013.
19. Benedek Nagy and Shaghayegh Parchami. On deterministic sensing $5' \to 3'$ watson–crick finite automata: a full hierarchy in 2detlin. *Acta Informatica*, 58:153–175, 2021.

20. R.L. Rivest and R.E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, 1993. URL: `https://www.sciencedirect.com/science/article/pii/S0890540183710217`, `doi:10.1006/inco.1993.1021`.
21. Ronald L Rivest and Robert E Schapire. Inference of finite automata using homing sequences. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 411–420, 1989.
22. Sicco Verwer and Christian A Hammerschmidt. Flexfringe: a passive automaton learning package. In *2017 IEEE international conference on software maintenance and evolution (ICSME)*, pages 638–642. IEEE, 2017.
23. Ryo Yoshinaka. Efficient learning of multiple context-free languages with multidimensional substitutability from positive data. *Theoretical Computer Science*, 412(19):1821–1831, 2011.

A Full experimental results

A reminder what languages were learned:

- $L_1 = a^n b^n$
- $L_2 = a^n ccb^n$
- $L_3 = a^n (bb)^n$
- $L_4 = (abbb)^n$
- $L_5 = ww^{-1}$ ("Palindrome") over $\Sigma = \{a, b, c\}$
- $L_6 = (\{a, b\}^2)^n (b\{a, b\}^5)^n$
- $L_7 = (\{a, b\}^2 a\{a, b\}^2)^n (\{a, b\}^2 b\{a, b\}^2)^n$

w is the input of the W-method, μ the input of the warm start initialisation and "RR" stands for Reset and Remember.

- Runtime: The average execution time over 10 runs in seconds
- #MinDisc: How many times the minimal size model was discovered in 10 runs.
- #ForcedTerm: How many times in 10 runs the forced termination was hit after having a hypotheses with more than a 100 states.
- Average #states: The average number of states of the predicted models over 10 runs.
- Min size — Max size: The size of the smallest and largest models discovered during 10 runs.

Also note that for the average number of states and the minimal and maximal models the runs that were forcibly terminated were excluded. For the average runtime the runs that were forcibly terminated were included.

		$w=3$ $\mu=0$ "RR"=No	$w=3,7,8$ $\mu=0$ "RR"=No	$w=3,6,7$ $\mu=1$ "RR"=No	$w=3,6,7$ $\mu=2$ "RR"=No
L_1	Runtime	<1s	-	<1s	<1s
	#MinDisc	6	-	5	10
	#ForcedTerm	0	-	0	0
	Average #states	5.8	-	4.6	3
	Min size — Max size	3|17	-	3|9	3|3
L_2	Runtime	<1s	-	<1s	<1s
	#MinDisc	5	-	6	10
	#ForcedTerm	0	-	0	0
	Average #states	7.1	-	6.4	5
	Min size — Max size	5|13	-	5|11	5|5
L_3	Runtime	<1s	-	<1s	<1s
	#MinDisc	6	-	4	4
	#ForcedTerm	0	-	0	0
	Average #states	7.3	-	5.6	10.4
	Min size — Max size	4|15	-	4|7	4|25
L_4	Runtime	<1s	-	<1s	<1s
	#MinDisc	1	-	2	3
	#ForcedTerm	0	-	0	0
	Average #states	8.8	-	8.6	7.2
	Min size — Max size	5|12	-	5|15	5|10
L_5	Runtime	1.9s	-	1.8s	0s
	#MinDisc	2	-	2	10
	#ForcedTerm	8	-	7	0
	Average #states	5	-	6	5
	Min size — Max size	$5\|5^*$	-	5|8	5|5
L_6	Runtime	<1s	11.5s	5.1s	4.5s
	#MinDisc	0	0	0	0
	#ForcedTerm	0	10	6	4
	Average #states	2	$-^*$	15	22.3
	Min size — Max size	$2^*\|2$	$-\|-^*$	10|20	10|50
L_7	Runtime	<1s	<1s	<1s	<1s
	#MinDisc	0	2	1	3
	#ForcedTerm	0	0	0	0
	Average #states	2	15.8	17.6	13.4
	Min size — Max size	$2^*\|2$	11|27	11|49	11|19

		$w = 1,5,6$ $\mu = 2$ "RR"=No	$w = 1,5,6$ $\mu = 3$ "RR" = No	$w = 3,7,8$ $\mu = 0$ "RR"=Yes	$w = 1,5,6$ $\mu = 2$ "RR"=Yes
L_1	Runtime	< 1s	< 1s	< 1s	< 1s
	#MinDisc	10	10	10	10
	#ForcedTerm	0	0	0	0
	Average #states	3	3	3	3
	Min size — Max size	3|3	3|3	3|3	3|3
L_2	Runtime	< 1s	< 1s	< 1s	< 1s
	#MinDisc	10	10	10	10
	#ForcedTerm	0	0	0	0
	Average #states	5	5	5	5
	Min size — Max size	5|5	5|5	5|5	5|5
L_3	Runtime	< 1s	< 1s	< 1s	< 1s
	#MinDisc	5	8	10	8
	#ForcedTerm	0	0	0	0
	Average #states	5.1	4.7	4	4.5
	Min size — Max size	4|9	4|8	4|4	4|8
L_4	Runtime	< 1s	< 1s	< 1s	< 1s
	#MinDisc	1	1	1	5
	#ForcedTerm	0	0	0	0
	Average #states	7.4	7.4	9.0	6.3
	Min size — Max size	5|10	5|9	5|11	5|9
L_5	Runtime	< 1s	< 1s	< 1s	< 1s
	#MinDisc	10	10	10	10
	#ForcedTerm	0	0	0	0
	Average #states	5	5	5	5
	Min size — Max size	5|5	5|5	5|5	5|5
L_6	Runtime	1.2s	3.9s	16.2s	2.5
	#MinDisc	1	2	0	2
	#ForcedTerm	3	3	8	5
	Average #states	14.4	15.7	18	10.8
	Min size — Max size	9|31	9|41	13|23	9|14
L_7	Runtime	< 1s	< 1s	4.9s	< 1s
	#MinDisc	4	4	6	7
	#ForcedTerm	0	0	0	0
	Average #states	12.9	13.2	14.2	12.6
	Min size — Max size	11|18	11|20	11|36	11|19