JMLR: Workshop and Conference Proceedings 57:15–29, 2016

Query Learning Automata with Helpful Labels

Adrian-Horia Dediu ADRIANHORIADEDIU@YAHOO.COM
Superdata, Bucharest, Romania

Joana M. Matos JMF.MATOS@FCT.UNL.PT
FCT – New University of Lisbon, Portugal

Claudio Moraga CLAUDIO.MORAGA@TU-DORTMUND.DE
TU Dortmund University, Germany

Abstract

In the active learning framework, a modified query learning algorithm benefiting by a nontrivial helpful labeling is able to learn automata with a reduced number of queries. In extremis, there exists a helpful labeling allowing the algorithm to learn automata even without counterexamples.

We also review the correction queries defining them as particular types of labeling. We introduce minimal corrections, maximal corrections, and random corrections. An experimental approach compares the performance and limitations of various types of queries and corrections. The results show that algorithms using corrections require fewer queries in most of the cases.

Keywords: Active Query Learning of Automata, Helpful Labeling of States, Correction Queries

1. Introduction

In 1987, Angluin (1987) introduced query learning, a well-known algorithm, namely L^* which is considered the basis of a theory called *exact query learning*. In query learning, we assume the existence of a *Teacher* who knows a target regular language and answers (correctly) specific kinds of queries asked by the *Learner*. There are various types of queries a Teacher can answer during the learning process. We give several examples: *membership* (MQ for short), *equivalence* (EQ), *subset*, *superset*, *disjointness*, and *exhaustiveness*, as described by Angluin (1988).

The algorithm L^* uses the notion of a *minimally adequate Teacher* (MAT), which is a fairly wide class of Teachers. A minimally adequate Teacher is able to answer correctly two types of queries, for the initial algorithm proposed by Angluin, a MAT answers membership and equivalence queries.

The motivation for studying such learning algorithms gave place to many discussions. There are some radical voices arguing that if someone knows a target automaton, then there is no point to learn it, we can simply transfer the automaton to some machine to use it directly. There is also some criticism about the nature of counterexamples, saying that for some practical applications, there is no way for the Teacher to identify them. Nevertheless, there are known applications to *learning maps*, which represent the interactions of a robot with his environment as a DFA (Rivest and Schapire (1993)). The states of the automaton

represent locations on the map, the input alphabet specifies different actions of the robot, and the output alphabet indicates observable parameters of the places. There is also a new direction using learning algorithms for model checking and program verification, as for example Neider (2014). The Teacher knows the initial conditions of a program as a regular language I, the program itself is modeled by a transducer T that gives the output for some input sequence and the Teacher also knows some invalid configurations of the data, again specified as a regular language B. We need to learn if the closure of language I through the transducer T, gives null intersection with B. For this purpose, Angluin's algorithm works perfectly.

The improvements brought to L^* over the time were marked by slight changes in the terminology. If the initial article Angluin (1987) talks about membership queries, as the algorithm learns regular languages, the article by Vilar (1996) presents query learning of subsequential transducers with *translation queries*. The article of Angluin et al. (2009) uses *output queries* (OQ) since the algorithm learns also the output of the states. The same article studies an interesting topic, *the influence of the cardinality of the output alphabet on the learning speed* for various learning scenarios. The Teacher gives a new dimension for the output symbols, adding labels either carefully chosen or simply randomly. In the case of the target automaton is labeled by a Teacher, the new type of query is known as a *label query* (LQ) which are used for several active and passive learning scenarios; we give the formal definition in the next section. In this paper, we present label queries only for L^* and study how much help a Teacher can give without transforming the learning process into a trivial one.

Correction queries (CQ) were introduced by Becerra-Bonache et al. (2005, 2006) and have been studied intensively since then. We mention only several papers, such as Tîrnăucă and Knuutila (2007); Kinber (2008); Becerra-Bonache et al. (2008); Mitrana and Tirnăucă (2011). Although the label queries appear in the literature after correction queries, the presentation of corrections queries as a particular type of label queries gives a new perspective, allowing us to experiment several new types of corrections: *minimal*, *maximal*, and *random corrections*.

2. Preliminaries

We assume that the reader is familiar with the basic notions of formal languages and complexity. We briefly present an overview of the basic concepts we use in this article.

Let Σ be a finite set of symbols, called the *alphabet*. A finite sequence of elements of Σ is called a *string* over Σ. For a given string w, $|w|$ represents the *length* of the string. We denote by ϵ the *empty string*, with $|\epsilon| = 0$. We define a binary operation between strings in the following way. For two strings $w = a_1 \dots a_n$ and $x = b_1 \dots b_m$ over Σ, the *concatenation* of the two strings is the string $a_1 \dots a_n b_1 \dots b_m$. The concatenation operation is denoted by $w \cdot x$ (or simply wx when there is no confusion).

Let Σ^* be the set of strings over Σ. A *language* L over Σ is a subset of Σ^*. The elements of L are also called *words*. For any given nonnegative integer k, $\Sigma^{\leq k}$ denotes the language of words w with $|w| \leq k$.

Definition 1 (Deterministic finite automata, Freund et al. (1997)) *A* deterministic finite automaton *(DFA[1]) is a tuple* $M = (Q, \Sigma, \Gamma, \tau, \lambda, q_0)$*, where* Q *is a finite* set of states*,* Σ *is a finite and nonempty alphabet called the* input alphabet*,* Γ *is a finite and nonempty alphabet called the* output alphabet*,* τ *is a partial function from* $Q \times \Sigma$ *to* Q *called the* transition function*,* λ *is a mapping from* Q *to* Γ *called the* output function*, and* q_0 *is a fixed state of* Q *called the* initial state*.*

We extend τ to a map from $Q \times \Sigma^*$ to Q in the usual way. We take $\tau(q, \epsilon) = q$ and $\tau(q, \alpha \cdot a) = \tau(\tau(q, \alpha), a)$, for all states q in Q, for all strings α in Σ^*, and for all characters a in Σ, provided that $\tau(q, \alpha)$ and $\tau(\tau(q, \alpha), a)$ are defined. For a state q and a string x over the input alphabet, we denote by qx the state $\tau(q, x)$, and by $q\langle x \rangle$, the sequence of length $|x|+1$ of output labels observed upon executing the transitions from state q dictated by x, that is, the string $\lambda(q)\lambda(qx_1), \ldots, \lambda(qx_1 \ldots x_n)$, where n is the length of the string x and $x_1, \ldots, x_n$ are its characters.

A deterministic *finite acceptor* is a DFA with the output alphabet $\Gamma = \{0, 1\}$; if $\lambda(q) = 1$, then q is an *accepting state*, otherwise, q is a *rejecting state*. A string x is *accepted* or *recognized* by a finite acceptor with the initial state q_0 if $q_0 x$ is an accepting state. The definition of automata with final states (see, for example, Hopcroft and Ullman (1979)) is equivalent with our definition of finite acceptors, with the convention that *final states* are the accepting states. For a finite acceptor $A = (Q, \Sigma, \Gamma, \tau, \lambda, q_0)$, we define the *language* $L(A)$ as the set of strings accepted by the acceptor A.

A finite acceptor is a *password automaton* if it accepts a single word. In order to be complete, a password automaton has a *dead state*, that is a non-final state from which we cannot reach a final state. Note that for a given word w with length $|w| = n$, there exists a password automaton with $n + 2$ states accepting w.

A deterministic finite acceptor $A = (Q, \Sigma, \{0, 1\}, \tau, \lambda, q_0)$ is a *zero-reversible acceptor* if and only if A has at most one final state and for no two distinct states q_1 and q_2 do there exist an input symbol $b \in \Sigma$ and a state $q_3 \in Q$ such that $\tau(q1, b) = q3 = \tau(q2, b)$ (see Angluin (1982)).

For a DFA $M = (Q, \Sigma, \Gamma, \tau, \lambda, q_0)$, two states p, q in Q are called *k-distinguishable* if there exists a string w of length not greater than k such that $p\langle w \rangle \neq q\langle w \rangle$. If for any string of length k, where k is a nonnegative integer, we have $p\langle w \rangle = q\langle w \rangle$, the states p, q are called *k-indistinguishable*. Note that k-indistinguishability defines an equivalence relation between states. Two states are called *equivalent* (also *indistinguishable*) if they are k-indistinguishable for every k.

For d a nonnegative integer, we define the *d-signature tree* of a state q as the finite function mapping each input string x of length at most d to the output symbol of the state qx. We denote the d-signature tree of state q by $\sigma_d(q)$ and by $\sigma_d(q)(x)$ the output symbol of the state qx, where $|x| \leq d$.

We will assume that the target automata are minimal, accessible, and complete. For automata that are non minimal, there exist states that the Learner is not able to distinguish. For automata having inaccessible states, there is no way to get information about these

1. Angluin et al. (2009) use the term "automaton with output". In formal language books, like Hopcroft and Ullman (1979), this definition corresponds to a Moore automaton, and the notion of acceptors that we define next, corresponds to a DFA.

states. If the automata are not complete, there is a simple construction to make them complete. We add a new virtual state q_v not existing in Q, using as output a special symbol # not existing in the output alphabet. All the undefined transitions are reassigned as going to q_v. The transitions from q_v go to q_v as well. The Teacher and the Learner are both aware of this convention. Note that in the case of acceptors, there is no need to add a special symbol # for the virtual state q_v: it suffices to label q_v with zero and the language of the automaton remains the same.

In L^*, the Learner uses a table called the *observation table* to store the Teacher's answers and to construct a hypothesis automaton.

The set of indices for rows are discovered incrementally by the Learner. Starting with the empty string, the Learner forms a nonempty, finite, prefix-closed set S over Σ. The rows of the observation table are indexed by the set $S \cup S \cdot \Sigma$. The indices for columns form a nonempty finite suffix-closed set of strings E over Σ. The Teacher's answers, collected in the observation table, represent a finite function C, mapping $(S \cup S \cdot \Sigma) \cdot E$ to Γ. We denote the observation table by (S, E, C). If s is an element of $(S \cup S \cdot \Sigma)$, then $row(s)$ denotes the finite function from E to Γ defined by $row(s)(e) = C(s \cdot e)$.

If $M = (Q, \Sigma, \Gamma, \tau, \lambda, q_0)$ is a finite target DFA, then a *labeling* of M is a function ℓ mapping Q to a set Λ of labels, the *label alphabet*. We use M to construct a new automaton $M^\ell = (Q, \Sigma, \Gamma \times \Lambda, \tau, \lambda_\ell, q_0)$, where the output function $\lambda_\ell(q) = (\lambda(q), \ell(q))$. That is, the new output for a state is a pair of symbols, the output symbol together with the label attached by the Teacher to that state. We call this pair of symbols the *labeled output*. If it is clear from the context that we deal with a labeled automaton, then we call the labeled output only *output*. We assume that ℓ is surjective. The Learner has access to output queries for a labeled target automaton and this kind of query will be referred to as a *label query*.

There exists a variant of the L^* algorithm, also discussed by Tîrnăucă and Knuutila (2007), that initializes the content of the set E of an observation table in a different way. Instead of using only the empty word ϵ, the algorithm initializes E with $\Sigma^{\leq j}$, for j a nonnegative integer. We denote this variant of the algorithm by L^*_j. Clearly, L^*_0 corresponds to the original L^*.

3. Helpful Label Queries

A Teacher performs a *helpful labeling* of the states of the target automaton creating the possibility of a reduce number of queries for the learning algorithm. In this section we show that there exists an algorithm that adds helpful labels to the states of an automaton, such that the algorithm L^*_1 can completely learn the automaton only with label queries and without needing any equivalence query. We present first the theoretical background followed by an experimental approach.

3.1. Theoretical Aspects

We present the theoretical complexity of the original L^* and we compare with the trivial type of labeling that assigns a different label to each state, to give an idea about the complexity limits of L^*. We have omitted several simple proofs. For full proofs and examples, please consult the PhD thesis Dediu (2015).

Theorem 2 (Angluin (1987); Dediu (2015)) *Given any minimally adequate Teacher presenting an unknown minimal DFA $M = (Q, \Sigma, \Gamma, \tau, \lambda, q_0)$, the Learner L^* terminates and correctly outputs a DFA isomorphic to M. If $|Q|$ is the number of states of M and m is an upper bound on the length of any counterexample provided by the Teacher, then the total running time of L^* is bounded by a polynomial in m and $|Q|$. The number of OQ is $O(|Q|^2|\Sigma|m)$ and the number of EQ is at most $|Q|$.*

For labeled automata, there are two trivial situations. One of the cases, if the Teacher labels each of the states with the same symbol, then there is no influence on the learning process. In the other case, if the Teacher labels the states in such a way that for each state there results a different pair of symbols, then we easily obtain the following result.

Proposition 3 (Angluin et al. (2009)) *Let $M^\ell = (Q, \Sigma, \Gamma \times \Lambda, \tau, \lambda_\ell, q_0)$ be a finite target labeled automaton having $\ell : Q \to \Lambda$, a labeling function such that the resulting output function λ_ℓ is injective. Then L^* can learn the target automaton using $|Q||\Sigma| + 1$ label queries and no counterexamples are needed.*

Proposition 4 (Dediu (2015)) *Let $M = (Q, \Sigma, \Gamma, \tau, \lambda, q_0)$ be a minimal automaton. There exists a labeling ℓ of M, with $|\lambda_\ell(Q)| < |Q|$, such that for $j \geq 1$, L_j^* is able to learn the target automaton with a number of label queries that is $O(|Q||\Sigma|^j)$ and without needing counterexamples.*

Proof Let n be the number of states, m the number of output symbols and k the number of input symbols, such that $m < n$.

For a nonnegative integer j, let us denote by R_j the equivalence relation of being j-indistinguishable on Q. We also denote by Π_j the partition of states that R_j defines ($\Pi_j = Q/R_j$).

The partition Π_{j+1} is a refinement of Π_j since two states that are indistinguishable by any string of length $j + 1$, are also indistinguishable by any string of length j. Moreover, the following Lemma was proved by Moore (1956).

Lemma 5 (Moore (1956)) *If $|\Pi_j| < n$, then Π_{j+1} is a strict refinement of Π_j.*

We also have the following Lemma.

Lemma 6 *All elements of each equivalence class π of Π_j share the same j-signature tree. For a string s such that $q_0 s = q' \in \pi$, then for any string $e \in \Sigma^{\leq j}$ and for any state $q \in \pi$ we have $\sigma_j(q)(e) = row(s)(e)$.*

This Lemma shows that there exists a one-to-one correspondence between j-signature trees and rows in the observation table of L_j^*. The Teacher does not need to have access to the Learner's observation table, or to construct one, the Teacher uses j-signature trees to find a helpful labeling; however, the labeling has a direct effect on the Learner's observation table.

We associate to the partition of states Π_j a labeling function ℓ_j in the following way. For each element p from an equivalence class π of Π_j, we assign a unique label, such that the set $\{\ell_j(p) \mid p \in \pi\}$ becomes a permutation of the set $\{1 \dots |\pi|\}$. This labeling assures

unique values for the j-signature tree of each state. We used the assumption that the range of the labeling function consists of a subset of the natural numbers for simplicity and this serves to our goal to prove the existence of a non-injective labeling. Of course, following the spirit of this proof, one can use other labeling alphabets as well.

We count the number of labels needed to distinguish all the states for different values of j. For $j = 0$, the labels in each equivalence class are integers from one to the number of elements in each class, thus we have $|\lambda_{\ell_0}(Q)| = \sum_{p\in\Pi_0} |p| = n$. For $j > 0$, the partition Π_j is a strict refinement of Π_0 (see Lemma 5), there exist two states p and q in Q such that pR_0q holds while pR_jq does not hold. The states p and q have distinguishable j-signatures and can have both assigned the same label, 1 ; thus the total number of elements in $|\lambda_{\ell_j}(Q)|$ must be less than n.

We analyze now the size of the observation table for L_j^*. Note that L_j^* needs in the observation table as many rows as L^*, since the size of $S \cup S \cdot \Sigma$ is $O((k+1)n)$. The set E contains $(k^{j+1} - 1)/(k - 1)$ strings. The queries corresponding to $S \cdot E$ with the exception of column ϵ exist also in $S \cdot \Sigma \cdot E$. Thus, the number of queries is $O(nk^j)$. To conclude the proof, we give also the labeling algorithm.

Algorithm 1: Helpful labeling
foreach *signature S in signatureTrees* **do**
  newLabel← 1 **foreach** *state q with signature S* **do**
    addLabel(q,newLabel) newLabel← newLabel +1

The set *signatureTrees* keeps all the j-signature trees for all the states and the variable *newLabel* keeps the label to assign to some state. ■

Remark 7 *Note that for one state automata, clearly the output function is already injective, there is no need to add supplementary labels for L^* to learn them without counterexamples.*

We did not search for a minimal number of labels, we were interested to find some examples of automata with the largest number of helpful labels needed for a non-injective labeling function, thus, we concentrated mostly on the case $j = 1$.

3.2. Comparative Results

Our test set contains randomly generated automata using a binary alphabet and a number of states between 2 and 20. We have 11 different automata for each number of states. In fact, the tested automata are all available in Bonache (2005), see the annex for a complete list. In Table 1, for each number of states, we collect the minimum, the average, and the maximum number of membership queries (asked by L^*), as well as the number of label queries (asked by L_1^* with a helpful labeling). These results are presented graphically in Figure 1, the axis y is logarithmic for better visualization.

3.3. Remarks

The number of label queries is concordant with Proposition 4, not depending on the automaton structure.

Table 1: Comparative results between L^* and L_1^* with a helpful labeling showing number of queries depending on the number of states

States	Min MQ	Avg MQ	Max MQ	LQ
2	5	5.00	5	11
3	11	14.00	17	15
4	14	20.45	27	19
5	23	33.18	54	23
6	31	57.09	83	27
7	39	68.18	90	31
8	44	76.00	111	35
9	49	80.00	125	39
10	65	137.73	269	43
11	77	139.09	229	47
12	90	159.45	215	51
13	101	151.73	263	55
14	143	234.36	615	59
15	95	205.27	389	63
16	167	277.91	509	67
17	175	295.45	506	71
18	231	356.91	629	75
19	183	278.64	405	79
20	215	330.73	854	83

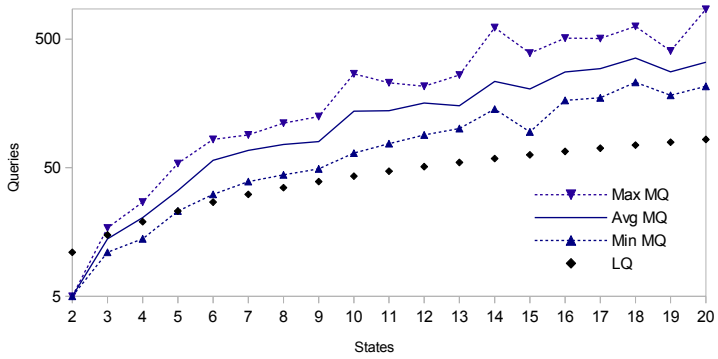

Figure 1: Membership and label queries depending on the number of states

From the proof of Proposition 4, it clearly results that the complexity of a helpful Teacher labeling a DFA for allowing L_j^* to learn the target automaton without needing counterexamples is polynomial, for $j \geq 1$.

During our experiments we observed that for the automaton P167 having 17 states, we need 16 different output symbols to allow L_1^* to work without counterexamples. We can see this automaton represented in Figure 2.

From the experimental results, we see that only if the number of states is less than five, L_1^* (with a helpful labelling) does not perform better than L^*.

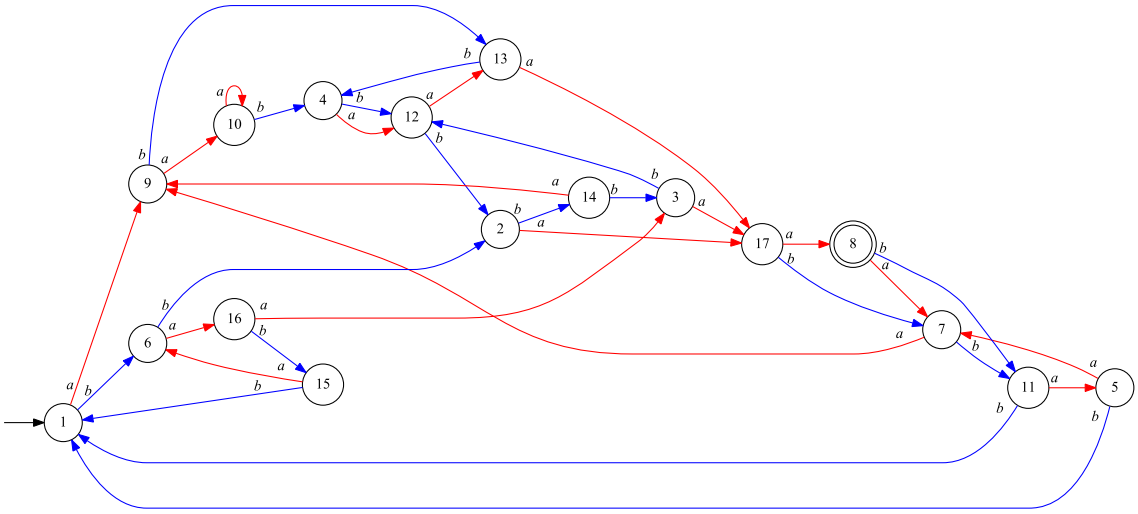

Figure 2: For the automaton P167 having 17 states, we need 16 different output symbols to allow L_1^* to work without counterexamples

4. Reviewed Correction Queries

In this section, we study a labeling method combined with some partial guidance along the learning path, which we call correction. The target automata are only acceptors, as the answer to a *correction query* u gives a word w such that uw is in the target language. That is, when the Learner asks u, the Teacher answers, "you mean uw." There are two particular cases: if $w = \epsilon$, then this answer could be interpreted as "yes, u is in the target language"; if there is no such word w, then the answer is a special symbol not belonging to the input alphabet.

4.1. Theoretical Approach

Let $M = (Q, \Sigma, \{0,1\}, \tau, \lambda, q_0)$ be a finite target acceptor, and $\#$ a special symbol not existing in the input alphabet Σ. If θ is a labeling of M that maps Q to $\Sigma^{\leq |Q|-1} \cup \{\#\}$, then θ is called a *correction function* if the following conditions hold.

$$\theta(q) = \begin{cases} \epsilon & \text{if } \lambda(q) = 1, \quad (1) \\ b\theta(q') & \text{if } \exists\, q' \in Q, b \in \Sigma, qb = q' \text{ and } \theta(q') \text{ is defined}, \quad (2) \\ \# & \text{otherwise.} \quad (3) \end{cases}$$

We assume that once a state is assigned with a label, that is a correction string, that state is never reassigned with another correction string.

For an acceptor labeled with a correction function, there is no need for the Teacher to communicate the output of the states together with the labeling. The output can be deduced by the Learner, employing condition (1): if the label of a state q is ϵ, then the state has the output $\lambda(q) = 1$; otherwise, the output is $\lambda(q) = 0$.

After labeling the final states, there might be several states for which the condition (2) holds, which means that for a given automaton there might exist several correction functions. The condition (2) creates the possibility of reducing the number of queries: once the Teacher answers with the label of a state, the Learner can deduce the output for all the states along the path until the final state.

Given an acceptor $M = (Q, \Sigma, \{0, 1\}, \tau, \lambda, q_0)$, we present a simple algorithm to construct a correction function for it (Algorithm 2). Let F be the set of final states, that is, $F = \{q \in Q \mid \lambda(q) = 1\}$. Let n be $|Q|$, the number of states of the acceptor. The algorithm works with a set of candidate assignments Θ containing pairs formed by a state and a string (q, w) such that $qw \in F$. The correction function $\theta(q)$ is initially assigned with either the empty string ϵ for the final states or with the special symbol #. The symbol # plays a double role. First, for some state labeled with #, it shows that a correction was not yet assigned. The second role of the symbol # is to indicate the dead state at the end of the algorithm. We note that once a label other than # was assigned, it is never reassigned. When there are no more candidates to assign, eventually remains only the dead state already assigned with # as the correction.

Algorithm 2: Constructing a correction function
Initialize $\theta(q) \leftarrow \epsilon$ for all $q \in F$ and $\theta(q) \leftarrow \#$ for all $q \in Q \setminus F$ Initialize $\Theta \leftarrow (q, c)$ for $q \in Q, c \in \Sigma$, $\theta(q) = \#$ and $qc \in F$ **while** Θ *is nonempty* **do**
  select (r, w), a pair from Θ remove all pairs containing r from Θ assign $\theta(r) \leftarrow w$
  Add to Θ all pairs (p, bw), $p \in Q$, $b \in \Sigma$, $\theta(p) = \#$ and $pb = r$

Theorem 8 *Algorithm 2 terminates in $O(|Q|^2|\Sigma|)$ for any acceptor M with the states Q and input alphabet Σ, and gives as output a correction function. For any correction function C of M, there exists an execution of Algorithm 2 that outputs C.*

Proof We note that once an element containing a state r is removed from Θ, there are no other states adding a tuple containing r to Θ. The algorithm ends, the cycle "while" assigns each time one state r, after that it removes all the other candidates r. The number of non-final states is clearly bounded by n and in Θ there cannot exist more than $n|\Sigma|$ candidates.

Algorithm 2 follows exactly the definition of a correction function. For each final state the label is ϵ as assigned in Line 2. The dead states do not have outgoing transitions other than to themselves, hence, the correction function remains as initially assigned with #. It can be easily shown by induction on the length of the correction, that all the states p other than the dead states are added together with a word w to Θ, where pw is a final state. For each state existing in Θ, one correction is assigned in Line 2.

We show now that Algorithm 2 can output any correction function function C of M. For final states, both C and Algorithm 2 assign the same value, ϵ. For the other states (except the dead state), the correction function C is defined recursively. We note that the candidate set Θ keeps all the possibilities of assignments initially for states with outgoing transitions to final states, and then for all assigned states. Thus, for the states with outgoing transitions to the final states, Algorithm 2 should select and assign to θ one of the possible values, namely the one indicated by C. We can show by induction on the length

of the correction word that for each state q, Algorithm 2 can select for the assignment to θ (Line 2) the pair (q, w) such that $C(q) = w$. Finally, for the dead states, C is defined to be #; Algorithm 2 assigns # in Line 2 and never reassigns it, as for dead states there are no outgoing transitions to other states with corrections assigned. ■

If in Line 2 there is more than one element available in Θ, then there exists more than one correction function for the given automaton. When constructing a correction function, Algorithm 2 can select constantly one candidate using, e.g., any of the following strategies.

- Select one state from the candidate assignments with a minimum length of the word reaching a final state. If the algorithm works in this way, then we call the resulting correction function a *minimal correction*.
- If the algorithm selects always one of the candidates with the maximum length of the word reaching a final state, then we call the resulting correction function a *maximal correction*.
- Selecting a random candidate, we get a *random correction*.

For the selection of a minimal or a maximal correction there can be also several choices depending on the order within the input alphabet; however we believe that the order within the input alphabet should not influence the results of the learning algorithm. Actually, we tested a version of lexicographic correction and the results were similar to those of a minimal correction.

We present a minimal, a maximal, and a random correction for the automaton P167 (Table 2).

Table 2: Correction functions examples for the automaton P167

Automaton				Correction		
State	$\tau(a)$	$\tau(b)$	λ	minimal	maximal	random
1	9	6	0	b^2a^2	$b^3a^2b^2a^3$	aba^2
2	17	14	0	a^2	$ba^2b^2a^3$	a^2
3	17	12	0	a^2	ba^3	a^2
4	12	12	0	aba^2	ba^3	ba^3
5	7	1	0	b^3a^2	$ab^5a^2b^2a^3$	a^2ba^2
6	16	2	0	ba^2	$b^2a^2b^2a^3$	ba^2
7	9	11	0	aba^2	$b^5a^2b^2a^3$	aba^2
8	7	11	1	ϵ	ϵ	ϵ
9	10	13	0	ba^2	ab^2a^3	ba^2
10	10	4	0	$baba^2$	b^2a^3	b^2a^3
11	5	1	0	b^3a^2	$b^4a^2b^2a^3$	$baba^2$
12	13	2	0	ba^2	a^3	a^3
13	17	4	0	a^2	a^2	a^2
14	9	3	0	ba^2	$a^2b^2a^3$	ba^2
15	6	1	0	aba^2	$ab^2a^2b^2a^3$	aba^2
16	3	15	0	a^3	$bab^2a^2b^2a^3$	a^3
17	8	7	0	a	a	a
Labels				9	16	10

The Learner has access to the labels assigned to states of a target automaton by a correction function and this kind of query will be referred to as a *correction query* (CQ for short).

We enumerate now several properties of correction functions.

Lemma 9 (Dediu (2015)) *If for a state q of an acceptor there exists a word w such that qw is a final state, then a correction function maps q to a string other than #.*

Proposition 10 (Dediu (2015)) *Let q be a state of an acceptor and $w = \theta(q)$ the value assigned to q by a correction function. Then on the path from q obtained by following w, there are no loops.*

We conclude this section showing several learnability properties via correction queries for some particular classes of automata.

Theorem 11 (Dediu (2015)) *Let $M = (Q, \Sigma, \Gamma, \tau, \lambda, q_0)$ be a password automaton with $|\Sigma| \geq 2$. Then for M there exists a unique correction function and L^* can learn the target automaton using $(|Q|-1)(|\Sigma|-1)+2$ correction queries and no counterexamples are needed.*

Theorem 12 *Let $M = (Q, \Sigma, \Gamma, \tau, \lambda, q_0)$ be a complete zero-reversible automaton with input alphabet $|\Sigma| \geq 2$. Then for M there might exist several correction functions. Each correction function is injective and L^* can learn the target automaton using $|Q||\Sigma| + 1 - (|Q| - 1)$ correction queries and no counterexamples are needed.*

Proof The existence of multiple correction functions is simple, let us take for example the automaton $(Q = \{1, 2\}, \Sigma = \{a, b\}, \Gamma = \{0, 1\}, \tau, \lambda, q_0 = 1)$, where $\tau(1, a) = \tau(1, b) = 2$, $\lambda(1) = 0$ and $\lambda(2) = 1$. Then we have two possible correction functions, for the initial state they could return either a or b. The injectivity results from the definition of zero-reversible automata; there exists at most one final state, following the correction path from the final state, we can reach a unique state. With an injective labeling, according to Proposition 3, there are $|Q||\Sigma| + 1$ queries needed. However, from a correction query we can deduce the answers of the Teacher for all the states on the path until the final state, thus there are needed less correction queries. For each state $q \in Q \setminus \{q_0\}$, from a correction $b\alpha$, with $b \in \Sigma$ and $\alpha \in \Sigma^*$, we can deduce exactly one correction for the state qb. ■

For zero-reversible automata that are not complete, a correction function is not injective anymore. We recall that for the undefined transitions the correction function returns # and the same for the dead state. However, the learning algorithm needs the same number of queries as for a complete automaton; for each undefined transition, in the complete automaton the correction uniquely identify the state of the transition. For the dead state there is no need for supplementary correction queries, according to the definition, all its transitions are labeled by #.

4.2. Experimental Results

We use the same test set as for the label queries. There are 11 different automata for each number of states between 2 and 20. We experimented with three different correction functions: one minimal, another random, and the last one a maximal correction.

We also represent graphically the comparative results. We take the average MQ/CQ per number of states, according to our tests (Table 3), and we represent graphically their values. Figure 3 shows on the same graph the average number of MQ and CQ depending on the number of states, while Figure 4 shows the number of queries for three different corrections.

Table 3: Average number of queries (MQ/CQ) per number of states

States	MQ (L*)	CQ (minimal)	CQ (random)	CQ (maximal)
2	5.00	3.82	3.82	3.82
3	14.00	8.09	7.27	7.36
4	20.45	10.36	10.45	10.27
5	33.18	19.36	19.18	18.55
6	57.09	25.18	22.55	22.18
7	68.18	34.64	34.82	29.82
8	76.00	43.00	40.45	40.27
9	80.00	49.00	50.55	47.64
10	137.73	46.55	39.82	38.27
11	139.09	65.91	65.09	60.09
12	159.45	65.36	63.82	45.91
13	151.73	90.73	83.09	80.64
14	234.36	151.73	150.64	146.45
15	205.27	125.82	114.09	116.09
16	277.91	107.18	82.27	87.36
17	295.45	141.18	150.36	128.18
18	356.91	185.18	185.45	182.64
19	278.64	147.45	142.82	155.27
20	330.73	147.18	142.55	125.82

4.3. Remarks

Analyzing the results, we observe that, for almost all our test problems, the number of correction queries is lower than the number of MQ needed to learn the automata. For several exceptions, the negative influence comes from different choices of counterexamples. In addition, the maximal correction performs better in almost all cases, because for a maximal correction there are more chances to have more labeling symbols attached as corrections.

5. Concluding Remarks

We believe that the minimal number of labels needed to learn automata without counterexamples can be considerably reduced for various classes of automata, as in general the same state appears in different d-signature trees on different levels, from the root to the leaves. The algorithm that we present to assign helpful labels is not optimal, in the sense that we

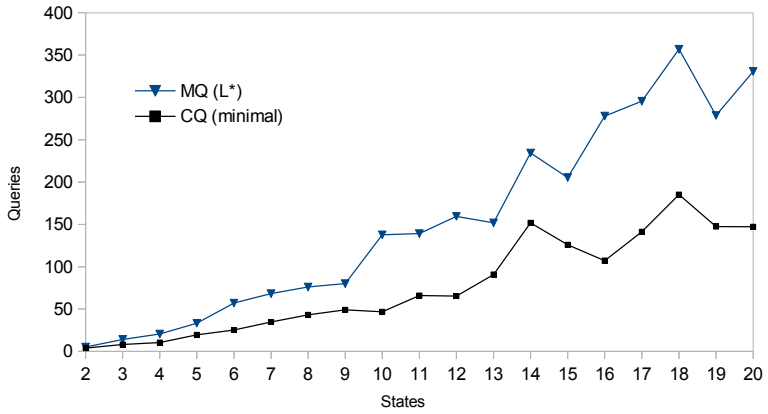

Figure 3: Comparative results MQ/CQ

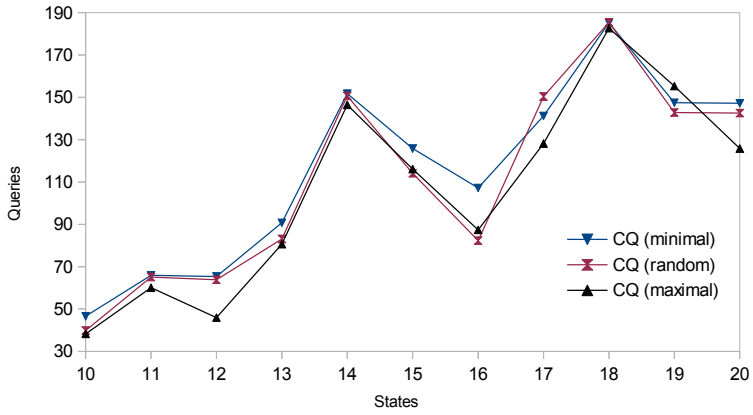

Figure 4: Comparative results, CQ minimal, random and maximal

did not try to minimize the number of labels needed. We give only several ideas to reduce the number of labels. For example, after assigning a label to one state, we can reevaluate the signature trees, and only after that to assign a new label. The order of assignment is also important, if we assign first a label to a state appearing more often in signatures, that can also contribute to obtain a lower number of labels needed.

We can also try to generalize corrections for DFA (that is for more than two symbols in the output alphabet). In fact, Becerra-Bonache and Dediu (2009), propose a method to learn DFA using a query similar with a correction query; however, the method is not a pure generalization of correction queries.

Acknowledgments

We thank Colin de la Higuera, University of Nantes, France for his recommendations and suggestions.

Joana M. Matos acknowledges the FCT Project UID/MAT/00297/2013 of CMA and of Departamento de Matemática da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa.

References

Dana Angluin. Inference of reversible languages. *J. ACM*, 29(3):741–765, July 1982. ISSN 0004-5411. doi: 10.1145/322326.322334. URL http://doi.acm.org/10.1145/322326.322334.

Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987. doi: http://dx.doi.org/10.1016/0890-5401(87)90052-6.

Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988. doi: http://dx.doi.org/10.1023/A:1022821128753.

Dana Angluin, Leonor Becerra-Bonache, Adrian-Horia Dediu, and Lev Reyzin. Learning finite automata using label queries. In Ricard Gavaldà, Gábor Lugosi, Thomas Zeugmann, and Sandra Zilles, editors, *Proceedings of the 20th International Conference on Algorithmic Learning Theory, Porto, Portugal, October 3–5, 2009*, volume 5809 of *Lecture Notes in Computer Science*, pages 171–185. Springer-Verlag, 2009.

Leonor Becerra-Bonache and Adrian-Horia Dediu. Learning from a smarter teacher. In Emilio Corchado and Hujun Yin, editors, *Proceedings of the 10th International Conference on Intelligent Data Engineering and Automated Learning, Burgos, Spain, September, 2009*, volume 5788 of *Lecture Notes in Computer Science*, pages 200–207, Berlin, 2009. Springer-Verlag.

Leonor Becerra-Bonache, Cristina Bibire, and Adrian-Horia Dediu. Learning DFA from corrections. In Henning Fernau, editor, *TAGI*, WSI-2005-14, pages 1–11. Technical Report, University of Tubingen, 2005.

Leonor Becerra-Bonache, Adrian-Horia Dediu, and Cristina Tîrnăucă. Learning DFA from correction and equivalence queries. In *Proceedings of the 8th International Conference on Grammatical Inference: Algorithms and Applications*, volume 4201 of *Lecture Notes in Computer Science*, pages 281–292, Berlin, 2006. Springer-Verlag.

Leonor Becerra-Bonache, Colin de la Higuera, Jean-Christophe Janodet, and Frédéric Tantini. Learning balls of strings from edit corrections. *Journal of Machine Learning Research*, 9:1841–1870, 2008.

Leonor Becerra Bonache. On the learnibility of mildly context-sensitive languages using positive data and correction queries, 2005. URL http://www.tdx.cat/bitstream/handle/10803/8780/DissertationLeonorBecerra.pdf?sequence=1&isAllowed=y. Ph.D. thesis, Rovira i Virgili University, Tarragona, Spain.

Adrian-Horia Dediu. Learning automata with help, 2015. URL http://www.tdx.cat/bitstream/handle/10803/314182/Tesi-Horia%20Dediu.pdf?sequence=1. Ph.D. thesis, Rovira i Virgili University, Tarragona, Spain.

Yoav Freund, Michael J. Kearns, Dana Ron, Ronitt Rubinfeld, Robert E. Schapire, and Linda Sellie. Efficient learning of typical finite automata from random walks. *Inf. Comput.*, 138(1):23–48, 1997.

John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

Efim Kinber. On learning regular expressions and patterns via membership and correction queries. In Alexander Clark, François Coste, and Laurent Miclet, editors, *Proceedings of the 9th International Colloquium on Grammatical Inference: Algorithms and Applications, Saint-Malo, France, September 22–24, 2008*, volume 5278 of *Lecture Notes in Computer Science*, pages 125–138, Berlin, 2008. Springer-Verlag.

Victor Mitrana and Cristina Tirnăucă. New bounds for the query complexity of an algorithm that learns DFAs with correction and equivalence queries. *Acta Inf.*, 48(1):43–50, 2011.

Edward F. Moore. Gedanken experiments on sequential machines. In *Automata Studies*, pages 129–153. Princeton U., 1956.

Hermann Daniel Neider. Applications of automata learning in verification and synthesis, 2014. URL http://d-nb.info/1059276062/34. Ph.D. thesis, RWTH Aachen University.

Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, April 1993. doi: 10.1006/inco.1993.1021. URL http://dx.doi.org/10.1006/inco.1993.1021.

Cristina Tîrnăucă and Timo Knuutila. Efficient language learning with correction queries. Technical Report 822, Turku Center for Computer Science, 2007.

Juan Miguel Vilar. Query learning of subsequential transducers. In Laurent Miclet and Colin de la Higuera, editors, *Proceedings of the Third International Colloquium on Grammatical Interference (ICGI-96): Learning Syntax from Sentences, Montpellier, France, September*, volume 1147 of *Lecture Notes in Computer Science*, pages 72–83. Springer-Verlag, 1996.