UNIVERSITAT ROVIRA I VIRGILI

LEARNING AUTOMATA WITH HELP.

Adrian-Horia Dediu

Dipòsit Legal: T 1472-2015

Adrian-Horia Dediu

LEARNING AUTOMATA WITH HELP

DOCTORAL THESIS

Supervisor: Victor Mitrana
Co-supervisor: Claudio Moraga
Tutor: María Dolores Jiménez López

Departament de Filologies Romàniques

UNIVERSITAT ROVIRA I VIRGILI
Tarragona
2015

UNIVERSITAT ROVIRA I VIRGILI
Departament de Filologies Romàniques
Av. Catalunya, 35
43002 Tarragona, Spain

I STATE that the present study, entitled "Learning automata with help", presented by Adrian-Horia Dediu for the award of the degree of Doctor, has been carried out under my supervision, and that it fulfils all the requirements to be eligible for the European Doctorate Award.

Date: 24.04.2014

Doctoral Thesis Supervisor

Prof. Dr. Victor Mitrana

ABSTRACT

In this thesis, we analyze and propose several enhanced versions of existing classic learning algorithms for deterministic finite automata. Our improvements belong to the category of helpful learning, aiming to speed up, or to influence the quality of the learning process. A considerable part of our work is based on a practical approach, for each algorithm we discuss, there are comparative results, obtained after the implementation and testing the learning processes on sets of randomly generated automata. After extended experiments, we present graphs and numerical data with the comparative results that we obtained.

We study algorithms belonging to two different learning models: active and passive. Our algorithms have features belonging to one or more categories of help for learning automata. Thus, an increased number of output symbols allows a reduced number of queries; some partial guidance along the learning path gives the results of several queries as a single one; enhancing the learning structure permits a better exploration of the learning environment.

In the active learning framework, a modified query learning algorithm benefiting by a nontrivial helpful labeling is able to learn automata without counterexamples.

We 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.

A classic algorithm learns typical automata from random walks in the online passive learning framework. For the original algorithm, we cannot estimate the number of trials needed to learn completely a target automaton for some cases. Adding inverse transitions to the underlying graph of the target automaton, the random walk acts as a random walk on an undirected graph. The advantage is that for such graphs, there exists a polynomial upper bound for the cover time. The new algorithm is still an efficient algorithm with polynomial upper bounds for the number of default mistakes and the number of trials.

ACKNOWLEDGEMENTS

I thank my supervisors, Professors Victor Mitrana and Claudio Moraga for their advice, experience, and the information I received, contributing with their knowledge and wisdom to the final form of this thesis. I thank Professor Carlos Martín-Vide for his patience, support, and confidence during all these years. I also thank my tutor, María Dolores Jiménez López for her careful reading, help and guidance.

I wish to address special thanks to Dana Angluin, Leonor Becerra-Bonache, and Lev Reyzin: their kindness and broad vision enlarged my horizons.

I thank all my professors and teachers: during the PhD studies, during the University and high school, during various research projects; they all contributed to my knowledge and education. I am grateful to my professor of mathematics from high school, Chiţa Popovici, and my professors from the university, Moisa Trandafir and Paul Cristea.

I thank all my colleagues from different projects and workplaces. Their confidence gave me the persistence and inspiration needed to finish this doctoral dissertation.

I thank my family for providing the support and understanding I needed, and especially Joana, for her patience in reading and checking my results were very helpful to me for my work.

I am grateful for the grants that I received from Universitat Rovira i Virgili (Rovira i Virgili University), 2002CAJAL-BURV4 and the postdoctoral research project 2010PFR-URV-B2-02: they allowed me to continue my work and to improve my knowledge.

Finally, I am grateful for the free tools and software programs that helped me to work on this dissertation, MiKTEX[1], OpenOffice[2], GAP[3] and Visual Studio 2010 Express (C#)[4].

1 http://miktex.org/
2 http://www.openoffice.org/
3 GAP – Groups, Algorithms, Programming – http://www.gap-system.org/
4 http://www.microsoft.com/visualstudio/eng/downloads#d-2010-express

CONTENTS

LIST OF NOTATIONS

Notation	Definition	Page
i.e.	The abbreviation of *id est* (Latin) meaning "that is"	15
$\emptyset$	The empty set	19
$\mathbb{N}$	The set of natural numbers	19
$[n]$	The finite set $\{1,\ldots,n\}$	19
$[0]$	The empty set	19
$\mathbb{R}$	The set of real numbers	19
$\|A\|$	The number of elements in A	19
$\mathscr{P}(A)$	The powerset of A	19
$\langle x \rangle_R$	The equivalence class of x, for $R \subseteq A \times A$ an equivalence relation on a set A	19
ϵ	The empty string	19
$\|w\|$	The length of the string w	19
Σ^*	The set of strings over Σ	19
$\Sigma^{\leq k}$	The set of words w with $\|w\| \leq k$, for some nonnegative integer k	19
DFA	Deterministic Finite Automaton	20
$M = (Q,\Sigma,\Gamma,\tau,\lambda,q_0)$	A deterministic finite automaton • 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.	20

Continued on next page

List of Notations

Notation	Definition	Page
qx	The state reached from q by consecutive transitions following the characters of x	20
$q\langle x\rangle$	The sequence of length $\|x\|+1$ of output labels observed executing the transitions from state q dictated by x	20
$A=(Q,\Sigma,\{0,1\},\tau,\lambda,q_0)$	A deterministic finite acceptor	20
$A=(Q,\Sigma,\tau,F,q_0)$	The same, a deterministic finite acceptor, F is the set of final states, $F=\{q \mid \lambda(q)=1, q\in Q\}$	20
$L(A)$	The language of the acceptor A	20
$\rho(M)$	The degree of distinguishability of the DFA M	24
$G=(V,E)$	A graph that consists of a set V of vertices and a set $E\subseteq V\times V$ of edges	19
$O(f)$	The set of functions $\{g:\mathbb{N}\to\mathbb{N} \mid \exists c>0$ and n_0 such that $c\cdot g(n)\geq f(n), \forall n>n_0\}$	25
$\Omega(f)$	The set of functions $\{g:\mathbb{N}\to\mathbb{N} \mid \exists c>0$ and n_0 such that $c\cdot g(n)\leq f(n), \forall n>n_0\}$	25
MQ	Membership Queries	29
EQ	Equivalence Queries	29
L^*	The algorithm proposed by Angluin [6] for learning DFA	29
OQ	Output Queries	30
(S,E,C)	An observation table, S a prefix closed set of state accessing strings, EA suffix closed set of distinguishing strings, C a function mapping $(S\cup S\cdot\Sigma)\cdot E$ to Γ giving the outputs answered by the Teacher	31
$\mathscr{M}(S,E,C)$	The Learner's conjecture	31
$M^\ell=(Q,\Sigma,\Gamma\times\Lambda,\tau,\lambda_\ell,q_0)$	A labeled automaton, the output function $\lambda_\ell(q)=(\lambda(q),\ell(q))$, where the labeling function ℓ maps Q to Λ	45
LQ	Label Queries	45
L^*_j	A variant of L^* initializing E with $\Sigma^{\leq j}$, for some nonnegative integer j	45
CQ	Correction Queries	58

Continued on next page

List of Notations

Notation	Definition	Page
$E[X]$	The expected value of the random variable X	68
δ	A confidence parameter, $0 < \delta \leq 1$	68
CCP	Coupon Collector's Problem	72
H_N	$\sum_{i=1}^{N} \frac{1}{i}$, the Nth harmonic number	72
$\Delta'(q')$	The state signature tree of state q'	71
$\Delta'(q',p')$	The node of an (incomplete) signature tree of state q' accessed by path p'	71
$v = (v_i : i \in [n])$	A vector v and its components	82
$\mathbb{R}^n$	The vector space of ordered n-tuples of real numbers	82
$\mathbf{0}_n$ or $\mathbf{0}$	A vector with all n components equal to 0; without the subscript n if this is clear from the context	82
$A = (a_{ij} : i \in [m], j \in [n])$	A matrix A and its entries	82
$M_{m \times n}$	The set of all the matrices with m rows and n columns	82
$\mathbf{I}_n$ or $\mathbf{I}$	The identity matrix with n by n entries, with ones on the main diagonal and zeros elsewhere; we drop the subscript n if this is clear from the context	82
A^{-1}	The inverse of matrix A	82
Probability: S, P, A	S is a sample space, P probability, $A \subseteq S$ is an event	83
Markov chain: I, u, $\mathbf{P}, f_{ij}(n), T_j, \mu_i$	I is a countable set of states; u the initial probability distribution; $\mathbf{P}$ is a transition stochastic matrix; $f_{ij}(n)$ is the probability that the first visit to state j, starting from i, takes place at the nth step; T_j, is the time of the first visit to state j; μ_i is the mean recurrence time of state i.	83
G an undirected graph: d_i, Δ_{ij}	d_i is the number of edges incident with vertex i; Δ_{ij} is the distance between vertices i and j	84

Continued on next page

List of Notations

Notation	Definition	Page
G a graph: h_{ij}, c_{ij}, C_v, C_G	h_{ij} is the hitting time, the expected number of transitions until a random walk starting from i reaches j; c_{ij} is the commute time, $c_{ij} = h_{ij} + h_{ji}$; C_v is the expected time to visit all the vertices of G starting from the vertex v; C_G is the cover time, $C_G = max_v C_v$	84
Σ^{-1}	A one to one disjoint set of complementary symbols of an alphabet Σ used for inverse transitions, $\Sigma^{-1} = \{a^{-1} \mid a \in \Sigma\}$	87
T_q^-	The set of inverse transitions from q	87
T_q	The set of direct transitions from q	87
$N_{q'}^b$	The set of states accessed by an inverse transition starting in q' with the character b	87
trial type: cA, pM, dM, ndM	Correct answer, prediction mistake, default mistake, nondeterministic mistake	88

1 INTRODUCTION

Conceptual metaphors[1] associate common notions with specific aspects proposed by new theories or models. The processes of learning often use conceptual metaphors to transmit and develop existing knowledge. There are several disciplines investigating learning capabilities, among them *machine learning* represents a relatively new domain that proposes a series of (self) improving algorithms, trying to model the processes of knowledge acquisition. Machine learning studies the possibilities of certain algorithms to act on new data and to generalize the results after training on existing data. Machine learning is a vast domain, grouping together a series of methods coming from statistics, neural modeling, adaptive and artificial intelligence, formal language theory, evolutionary algorithms, etc. The term "machine learning" is a conceptual metaphor in itself, at least for those times when most of the machine actions only follow concrete specifications.

There are several classes of machine learning algorithms, such as *supervised*, *unsupervised* and *reinforcement* learning. We mention only several of the newly emerged paradigms in this context, such as connectionism, classification and self-organization, statistical algorithms and interpretations, and various inference mechanisms.

Within *computational learning theory*, there are several directions that serve as a reference. In 1967, Gold [25] proposed *learning in the limit.* According to Gold, a *language learnability model* consists in:

1. A definition of learnability,
2. A method of information presentation,
3. A naming relation, which assigns names to languages.

A Learner[2] is a function assigning names to strings of units of information. A language L is said to be *identified in the limit* if after some time, the guesses of a Learner are all the same and are a name of L.

In 1984, Valiant [47] proposed *probably approximately correct learning* (PAC learning). In this framework, the Learner receives samples and must select a hypothesis function from a certain class of possible functions. The goal is that, with a high probability, the hypothesis function has a low generalization error. The Learner has access to two special routines, called EXAMPLES

1 I.e., metaphors that are more than a matter of language, with associations at the level of thoughts and actions (Lakoff and Johnson [34]).

2 In Angluin [6], "the Teacher" and "the Learner" are written with uppercase, however, over the years this was changed; we prefer to use uppercase characters to show that these are special concepts and not the usual teachers and learners as we know from the real life.

and ORACLE. A call to EXAMPLES provides a positive example, while ORACLE tells whether the concept (the predicate) to be learned is true or false for some presented data.

In 1987, Angluin [6] introduced query learning, which is considered the basis of a theory called *exact query learning*. A Teacher knowing a target automaton, answers correctly specific kinds of queries asked by the Learner. We present in detail the query learning algorithm proposed by Angluin in Chapter 3.

Although the internal representation of knowledge uses various terms, such as learning hypotheses, concepts, subsequential functions, and so on, deterministic finite automata (DFA) are widely used for their simplicity, generality and representation power. Traditionally, a finite automaton consists of a set of states connected by different transitions. Automata might be with or without output; in the case of automata with output, this can be associated either to states or to transitions. We give more details about this topic in the next chapter.

We can classify the results about the learning of automata as *active* or *passive learning*. In the passive model, the Learner has no control over the data received, while in the active model, the Learner can experiment with the target machine. Learning in the limit is an example of passive learning, while query learning (Angluin [6]) represents an example of active learning.

Various studies have tried to improve the performance of learning algorithms by introducing parallelism, enhancements in the data structure, or the concept of a helpful Teacher. Studying the effects and costs of different techniques of help is important not only to speed up the machine learning process, but also to tell us about possible implications for real life learning environments. Parekh and Honavar [42] presented a survey of several helpful environments. A helpful Teacher for a learning algorithm is a controversial topic. For example, we consider providing information about the number of states as acceptable help, while data about the internal names of the states is an example of collusion. However, there is no clear border on how much help a Teacher can give without oversimplifying the learning process. In this dissertation, we study several helpful methods showing the advantages and the limitations of such techniques.

We divided the contents into two parts: one dedicated to active learning and the other to passive learning. Before the two parts, we have a chapter containing common notations and definitions.

The first part starts with the well known query learning algorithm L^*, while the next two chapters describe modifications applied to this algorithm. In Chapter 4, starting from the definition of a label query introduced by Angluin et al. [9], we present a new version of L^* that is able to learn DFA with a helpful labeling without needing counterexamples. Finding a helpful labeling means automatically more resources for the Teacher's algorithm. However, this is balanced by far for two reasons: first, the Teacher does not need to find counterexamples; second, and this is more important, the total number of queries needed to learn an automaton is considerably reduced as also shown by the results of our tests.

In Chapter 5, we present a particular type of labeling that represents packed information about a complete learning path. This method was already investigated by Becerra-Bonache, Bibire and Dediu [14] and Becerra-Bonache, Dediu and Tîrnăucă [17] starting from the premise that a correction gives the shortest path from a state to a final state. We show here that other corrections are possible as well. The total number of queries needed to learn automata with correction queries is in general lower than the number of MQ needed. In case of password automata we show that we need a linear number of correction queries, comparing with a quadratic number of MQ, depending on the number of states of the target automaton. We conclude both chapters about helpful conditions for L^* with experimental comparative results.

The second part of this dissertation presents helpful conditions for a passive on-line learning algorithm, **Reset** (Freund et al. [23]). The first chapter of the second part describes the original algorithm that learns uniformly almost all automata. We also discuss several inherent limitations of the algorithm, and extend the study of **Reset** for almost all automata. In Chapter 7, we extend the transitions of the learned automaton allowing the random walks to go forward and backward on transitions. We obtain thus an upper bound for the number of trials needed to learn almost all automata. We also implement the new algorithm, and present a comparative study showing the improvements and the results of the experiments.

CHAPTER REFERENCES

[6] Dana Angluin. "Learning regular sets from queries and counterexamples". In: *Information and Computation* 75.2 (1987), pp. 87–106. DOI: http://dx.doi.org/10.1016/0890-5401(87)90052-6 (cit. on pp. 12, 15, 16, 29, 30, 32, 33, 35).

[9] Dana Angluin, Leonor Becerra-Bonache, Adrian-Horia Dediu, and Lev Reyzin. "Learning finite automata using label queries". In: *Proceedings of the 20th International Conference on Algorithmic Learning Theory, Porto, Portugal, October 3–5, 2009*. Ed. by Ricard Gavaldà, Gábor Lugosi, Thomas Zeugmann, and Sandra Zilles. Vol. 5809. Lecture Notes in Computer Science. Springer-Verlag, 2009, pp. 171–185 (cit. on pp. 16, 20, 30, 45, 46, 95).

[14] Leonor Becerra-Bonache, Cristina Bibire, and Adrian-Horia Dediu. "Learning DFA from Corrections". In: *Proc. Workshop on Theoretical Aspects of Grammar Induction*. Ed. by Henning Fernau. WSI-2005-14. Technical Report, University of Tubingen, 2005, pp. 1–11 (cit. on pp. 17, 45, 55, 95).

[17] 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. Vol. 4201. Lecture Notes in Computer Science. Tokyo, Japan: Springer-Verlag, 2006, pp. 281–292 (cit. on pp. 17, 30, 45, 55, 95).

[23] Yoav Freund, Michael J. Kearns, Dana Ron, Ronitt Rubinfeld, Robert E. Schapire, and Linda Sellie. "Efficient Learning of Typical Finite Automata from Random Walks". In: *Inf. Comput.* 138.1 (1997), pp. 23–48 (cit. on pp. 17, 20, 67–69, 71, 77).

[25] E. Mark Gold. "Language identification in the limit". In: *Information and Control* 10.5 (1967), pp. 447–474 (cit. on p. 15).

[34] George Lakoff and Mark Johnson. "Conceptual Metaphor in Everyday Language". In: *The Journal of Philosophy* 77.8 (1980), pp. 453–486. DOI: 10.2307/2025464. URL: http://dx.doi.org/10.2307/2025464 (cit. on p. 15).

[42] Rajesh Parekh and Vasant Honavar. "On the Relationship between Models for Learning in Helpful Environments". In: *Proceedings of the 5th International Colloquium on Grammatical Inference: Algorithms and Applications, Lisbon, Portugal, September 11–13, 2000*. Ed. by Arlindo L. Oliveira. Vol. 1891. Lecture Notes in Computer Science. Springer-Verlag, 2000, pp. 207–220 (cit. on p. 16).

[47] Leslie Gabriel Valiant. "A theory of the learnable". In: *Communications of the ACM* 27.11 (1984), pp. 1134–1142. DOI: http://doi.acm.org/10.1145/1968.1972 (cit. on pp. 15, 29).

2 PRELIMINARIES

In this chapter, we present the common notations and definitions used in the next chapters. The subsequent chapters contain sections with specific definitions for the learning algorithms that we discuss. We assume that the reader is familiar with the basic notions of combinatorics, graph theory, probability, and complexity. We briefly present an overview of the basic concepts we use in the thesis.

Notation $[n]$

We denote by $\mathbb{N}$ the set of natural numbers. For $n \in \mathbb{N}$, the notation $[n]$ represents the finite set $\{1,\dots,n\}$, while $[0] = \emptyset$ is the *empty set*. We denote by $\mathbb{R}$ the set of real numbers.

For a given finite set A, we denote by $|A|$ the number of elements in A. We use the notation $\mathscr{P}(A)$ for the powerset of A, i.e. the set of all subsets of A.

Let $R \subseteq A \times A$ be an equivalence relation. For $x \in A$, the set

$$\langle x \rangle_R = \{z \in A \mid xRz\}$$

of all elements related to x by R is called the *equivalence class* of x. The set of all equivalence classes $A/R = \{\langle x \rangle_R \mid x \in A\}$ is the *quotient set* of A by R. For any equivalence relation R on a set A, the set A/R is a partition of A. We recall that a *partition* of a set A is a set Π of nonempty subsets of A such that the union of the elements of Π is equal to A and the intersection of any two elements of Π is the empty set. Conversely, from any partition Π of A, we can define an equivalence relation on A by setting xRy precisely when x and y are in the same equivalence class of Π. We say that a partition Π' of a set A is a *refinement* of a partition Π of A (or Π' is *finer* than Π), if every element of Π' is a subset of some element of Π. A refinement Π' of Π is *strict* if $\Pi' \neq \Pi$.

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 $|w| \leq k$. The *prefixes* of a string s is the set of all prefixes, $\mathtt{Pref}(s) = \{a \mid ab = s, \text{ for } a, b \in \Sigma^*\}$. A language L is *prefix-closed* if $\mathtt{Pref}(s) \subseteq L$ for all $s \in L$. Similarly, the *suffixes* of a string s is the set of all suffixes, $\mathtt{Suff}(s) = \{b \mid ab = s, \text{ for } a, b \in \Sigma^*\}$. A language L is *suffix-closed* if $\mathtt{Suff}(s) \subseteq L$ for all $s \in L$.

A *graph* $G = (V, E)$ consists of a set V of *vertices (or nodes)*, and a set $E \subseteq V \times V$ of *edges*. A *path* in a graph is a sequence of vertices, $v_1 v_2, \dots, v_k$

for $k \geq 2$ such that every (v_i, v_{i+1}) is an edge in the graph, for $1 \leq i \leq k-1$. A path is a *cycle* if the first and the last vertex are the same. A graph $G = (V, E)$ is *connected* if for all $v, w \in V$ there exists a path between v and w. A graph G is *undirected* if for all $v, w \in V$ we have $(v, w) \in E \Leftrightarrow (w, v) \in E$, otherwise G is *directed*. A graph G is called *acyclic* if there are no cycles in G.

A graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E \cap (V' \times V')$. A *tree* is a connected, acyclic graph. The tree with no nodes is called the null or empty tree. A tree that is not empty consists of a *root* node r and zero or more (sub)-trees, each of whose roots connected to r. The root of each subtree is called a *child* of r, and r is called the *parent* of each child. In a tree, nodes with no children are called *leaf* nodes.

A *spanning tree* for a connected graph G is a tree subgraph of G including all the vertices of G.

Next, we give the definition of a formalism that operates on strings.

Definition 2.1 (Deterministic finite automata,[1] Freund et al. [23])**.** *A deterministic finite automaton (DFA) is a tuple*

$$M = (Q, \Sigma, \Gamma, \tau, \lambda, q_0), \textit{ where}$$

- 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*.*

Notations qx *and* $q\langle x\rangle$

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 *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 having the initial state q_0 if q_0x is an accepting state. The definition of automata with final states (see, for example, Hopcroft and Ullman [31]) is equivalent with our definition of finite acceptors, with the convention that *final states* are the accepting states.

1 Angluin [9] uses the term "automaton with output". In formal language books, like Hopcroft and Ullman [31], this definition corresponds to a Moore automaton, and the notion of acceptors that we define below corresponds to a DFA.

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.

The next example illustrates the way we graphically represent deterministic finite automata.

Example 2.2. *Take the automaton* $M_1 = ([4], \{a, b\}, \{0, 1\}, \tau, \lambda, 1)$*, where the transition function* τ *and the output function* λ *are given in Table 2. In Figure 1, we see the corresponding graphical representation for the automaton* M_1*, where the states are labeled with the state number together with the corresponding output symbol. We mark the initial state with an arrow.*

Table 2: The transition function τ, and the output function λ of the automaton M_1

State	$\tau(a)$	$\tau(b)$	λ
1	2	4	1
2	1	3	0
3	4	2	0
4	3	1	0

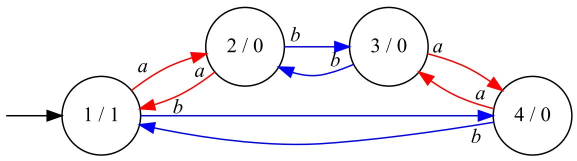

Figure 1: A graphical representation of the automaton M_1

The DFA M_1 *is a finite acceptor whose language is the set of all strings over* $\{a, b\}$ *with an even number of* a*'s and an even number of* b*'s (zero is considered here as an even number). In the case of finite acceptors, for a simplified graphical representation, we may also use double circles for accepting states, single circles for rejecting states, and only the state numbers for labels. In Figure 2, we see the simplified graphical representation of the acceptor* M_1.

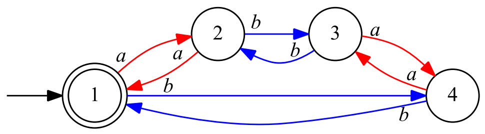

Figure 2: An alternative graphical representation of the acceptor M_1

Let two DFA be $M = (Q, \Sigma, \Gamma, \tau, \lambda, q_0)$ and $N = (Q', \Sigma, \Gamma, \tau', \lambda', q'_0)$. We say that M is *homomorphic* to N if there exists a mapping $\varphi : Q \to Q'$ such that the following conditions hold for any state $q \in Q$ and for any letter $a \in \Sigma$.

(i) $\varphi(q_0) = q'_0$,

(ii) $\varphi(\tau(q, a)) = \tau'(\varphi(q), a)$ and

(iii) $\lambda(q) = \lambda'(\varphi(q))$.

If φ is bijective, we call the automata *isomorphic*. Two isomorphic automata are also called *equivalent*. Let us note that in some works on automata theory, there appears a more general notion of homomorphism, where the sets of input symbols and output symbols are transformed by mappings as well, while our previous notion of homomorphism is referred to as a "state-homomorphism".

Example 2.3. *Let us consider two automata,* $M = (Q, \Sigma, \Gamma, \tau, \lambda, q_0)$ *and* $N = (Q', \Sigma, \Gamma, \tau', \lambda', q'_0)$*, where* $Q = Q' = [10]$*, the input alphabet* $\Sigma = \{a, b\}$*, the output alphabet* $\Gamma = [5]$*, and the initial states are* $q_0 = 1$ *and* $q'_0 = 1$*. Table 3 shows the transition functions and the output functions of these automata.*

Table 3: The transition functions and the output functions of automata M and N

Q	$\tau(a)$	$\tau(b)$	λ
1	2	3	3
2	2	4	5
3	5	2	1
4	6	7	1
5	5	6	4
6	5	8	2
7	3	9	5
8	10	5	2
9	1	6	5
10	2	4	3

Q'	$\tau'(a)$	$\tau'(b)$	λ'
1	3	5	3
2	4	10	2
3	3	8	5
4	4	2	4
5	4	3	1
6	1	2	5
7	5	6	5
8	2	7	1
9	3	8	3
10	9	4	2

The two automata are isomorphic and Table 4 shows an isomorphism between them.

Table 4: An isomorphism between the automata M and N

Q	1	2	3	4	5	6	7	8	9	10
Q'	1	3	5	8	4	2	7	10	6	9

We say that a DFA $M = (Q, \Sigma, \Gamma, \tau, \lambda, q_0)$ is *complete* if for all states $q \in Q$ and letters $a \in \Sigma$, $\tau(q, a)$ is defined (that is, τ is a total function). At the end of this chapter, we will give a simple construction to complete an undefined transition. A state q is called *reachable* or *accessible* if there exists a string w such that $\tau(q_0, w) = q$. An automaton all of whose states are accessible is called *accessible*. If q_1 and q_2 are states of a DFA, then q_1 and q_2 are *distinguishable* if there exists a *distinguishing string* for them, i.e., a string w such that $q_1\langle w\rangle \neq q_2\langle w\rangle$.

An automaton all of whose states are accessible and distinguishable is called *minimal*. The algorithms for minimizing finite acceptors (see, for example, Hopcroft and Ullman [31]) can be easily generalized for automata with more than two symbols in the output alphabet.

Example 2.4. *Take the finite acceptor* $M_2 = ([8], \{0,1\}, \{0,1\}, \tau, \lambda, 1)$*, where the transition function* τ *and the output function* λ *are given in Table 5. In Figure 3, we see a corresponding graphical representation for the acceptor* M_2*.*

Table 5: The transition function τ and the output function λ of the acceptor M_2

State	$\tau(0)$	$\tau(1)$	λ
1	8	2	0
2	8	1	0
3	5	6	0
4	5	6	0
5	6	7	0
6	6	6	1
7	7	6	1
8	3	4	0

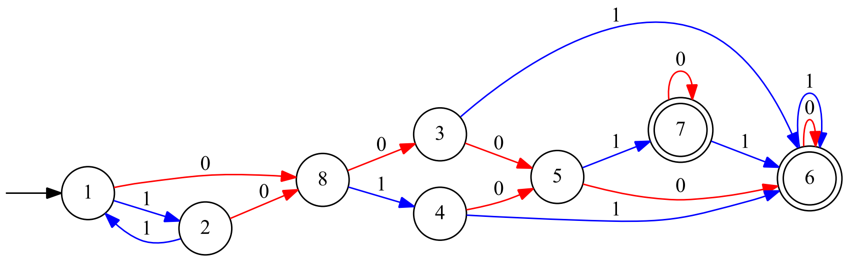

Figure 3: A graphical representation of the acceptor M_2

After minimizing the automaton M_2*, we get the acceptor*

$$M_3 = (\{\{1,2\},\{8\},\{3,4\},\{5\},\{6,7\}\}, \{0,1\}, \{0,1\}, \tau, \lambda, 1),$$

with the transition function and the output function described in Table 6. In Figure 4, we see a graphical representation of M_3*.*

Table 6: The transition function τ and the output function λ of the acceptor M_3

State	$\tau(0)$	$\tau(1)$	λ
$\{1,2\}$	$\{8\}$	$\{1,2\}$	0
$\{8\}$	$\{3,4\}$	$\{3,4\}$	0
$\{3,4\}$	$\{8\}$	$\{6,7\}$	0
$\{5\}$	$\{6,7\}$	$\{6,7\}$	0
$\{6,7\}$	$\{6,7\}$	$\{6,7\}$	1

We note that after minimization, the new automaton has each state as a set of states from the initial automaton in order to indicate how the new states were obtained. We can also rename the states of the minimal automaton using a bijective mapping form $\{\{1,2\},\{8\},\{3,4\},\{5\},\{6,7\}\}$ to $[5]$.

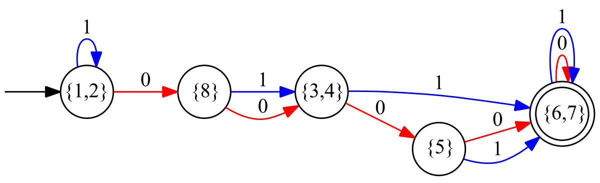

Figure 4: A graphical representation of the acceptor M_3

Two states p, q in Q are called k-*distinguishable* if there exists a string w of length $\leq k$ 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.

The *degree of distinguishability* of a minimal automaton M, denoted by $\rho(M)$, or simply ρ when M is clear from the context, is the minimal number r such that for any pair of states from M, there exists a distinguishing string not longer than r.

Definition 2.5. *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.

Of course, alternatively we can see and represent graphically a d-signature tree as a tree, with the edges labeled by the elements of the domain of definition and the nodes with output symbols.

Example 2.6. *Table 7 presents the* 2*-signature tree of the initial state of the automaton* M_1 .

Table 7: The 2-signature tree of the initial state of automaton M_1

ϵ	a	b	aa	ab	ba	bb
1	0	0	1	0	0	1

In Trakhtenbrot and Barzdin' [46], also in Andras [1], we can find more information about automata. Automata with output on states are closely related to automata with output on transitions, also called transducers. For more information about transducers, we recommend the book of Berstel [18] and an article of Mohri [38].

In this thesis, we present several considerations about the time complexity of the algorithms we discuss. Let us clarify the notation we will use. Let f, g be functions defined from $\mathbb{N}$ into $\mathbb{N}$. The following definitions also make sense for real valued functions (Balcázar, Díaz, and Gabarró [10]). Although the notations O and Ω are commonly used in computer science, there are still several things to comment on about the operators and the definitions associated with these notions. Some authors define $O(f) = \{g \mid \exists c > 0$ and n_0 such that $c \cdot g(n) \geq f(n), \forall n > n_0\}$ as a set of functions. In a similar

Notations O and Ω

way, $\Omega(f) = \{g \mid \exists c > 0 \text{ and } n_0 \text{ such that } c \cdot g(n) \leq f(n), \forall n > n_0\}$. If we accept this definition, for a function g satisfying the previous conditions, we say that $g \in O(f)$ or $g \in \Omega(f)$ Brassard [19]. Some authors, even accepting that O and Ω are sets, use $g = O(f)$ or "g is $O(f)$" for simplicity in writing, Gurevich [30]. Of course the relation "is $O(f)$" is not symmetric. Moreover, to simplify the notation, we identify functions with their expressions, as in the following example.

Example 2.7. *We define the following functions*

$$\begin{cases} f : \mathbb{N} \to \mathbb{N}, f(n) = n \\ g : \mathbb{N} \to \mathbb{N}, g(n) = 2n \\ h : \mathbb{N} \to \mathbb{N}, h(n) = n^2 \end{cases}$$

We have: $g = O(f)$ *and we can also write* $2n = O(n)$ *or even* $2n$ *is* $O(n)$ *or* n *is* $O(n^2)$.

In the following chapters, we will assume that the learned automata are minimal, accessible, and complete. For automata that are non minimal, there exist states that the Learner is not be able to distinguish. For automata having inaccessible states, there is no way to get information about these 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 it with zero with the language of the automaton remaining the same.

CHAPTER REFERENCES

[1] András Ádám. *The behaviour and simplicity of finite Moore automata*. Budapest,Hungary: Akademiai Kiado, 1996 (cit. on p. 24).

[9] Dana Angluin, Leonor Becerra-Bonache, Adrian-Horia Dediu, and Lev Reyzin. "Learning finite automata using label queries". In: *Proceedings of the 20th International Conference on Algorithmic Learning Theory, Porto, Portugal, October 3–5, 2009*. Ed. by Ricard Gavaldà, Gábor Lugosi, Thomas Zeugmann, and Sandra Zilles. Vol. 5809. Lecture Notes in Computer Science. Springer-Verlag, 2009, pp. 171–185 (cit. on pp. 16, 20, 30, 45, 46, 95).

[10] José L. Balcázar, Josep Díaz, and Joaquim Gabarró. *Structural Complexity*. Berlin: Springer-Verlag, 1990 (cit. on p. 24).

[18] Jean Berstel. *Transductions and Context-free Languages*. Vol. 38. Leitfäden der angewandten Mathematik und Mechanik. Teubner, 1979 (cit. on p. 24).

[19] Gillea Brassard. "Crusade for a better notation". In: *SIGACT News* 17.1 (June 1985), pp. 60–64. DOI: 10.1145/382250.382808. URL: http://doi.acm.org/10.1145/382250.382808 (cit. on p. 25).

[23] Yoav Freund, Michael J. Kearns, Dana Ron, Ronitt Rubinfeld, Robert E. Schapire, and Linda Sellie. "Efficient Learning of Typical Finite Automata from Random Walks". In: *Inf. Comput.* 138.1 (1997), pp. 23–48 (cit. on pp. 17, 20, 67–69, 71, 77).

[30] Yuri Gurevich. "What does $O(n)$ mean". In: *SIGACT News* 17.4 (Mar. 1986), pp. 61–63. DOI: 10.1145/8307.8311. URL: http://doi.acm.org/10.1145/8307.8311 (cit. on p. 25).

[31] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979 (cit. on pp. 20, 22).

[38] Mehryar Mohri. "Finite-state transducers in language and speech processing". In: *Comput. Linguist.* 23.2 (June 1997), pp. 269–311 (cit. on p. 24).

[46] Boris A. Trakhtenbrot and Ya. M. Barzdin'. *Finite automata*. Vol. 1. Fundamental Studies in Computer Science. Behavior and Synthesis, Translated from Russian by D. Louvish, English translation edited by E. Shamir and L. H. Landweber. Amsterdam: North-Holland, 1973 (cit. on pp. 24, 68, 69, 75–77).

Part I.

Helpful Query Learning of Automata

3 LEARNING DFA FROM QUERIES

In this part, we present several helpful conditions for query learning. First, we present the theoretical basis of a well-known algorithm, namely L^*. Although the original algorithm worked for finite acceptors, we present a version for DFA (the generalization is straightforward). After presenting the basic algorithm, we discuss several possibilities created by a helpful Teacher's labeling the states with supplementary output symbols. We also study a particular type of labeling that we call correction, which gives supplementary information about the learning path.

After the initial presentation of Angluin [6], there have been several studies of the use of L^* for learning DFA. We mention Grinchtein and Leucker [28] and a paper where similar problems have been solved in a very different manner, Gasarch et al. [24].

In query learning, we assume the existence of a Teacher who knows a target DFA and answers (correctly) specific kinds of queries asked by the Learner. This type of learning has been intensively studied and categorized as a method of exact learning, in contrast with the model proposed by Valiant[47] in 1984 about probably approximately correct learning.

There are various types of queries a Teacher can answer during the learning process. We give several examples: membership, equivalence, subset, superset, disjointness, and exhaustiveness, as described by Angluin [7].

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 about a known target finite acceptor.

- *Membership queries* (MQ). The Learner asks whether a string w is recognized by the target acceptor and the Teacher answers "yes" or "no".

- *Equivalence queries* (EQ). The Learner produces a finite acceptor M and asks whether M is isomorphic to the target automaton; the Teacher answers "yes" in the affirmative case or "no" otherwise. If the answer is "no", the Teacher also returns a string w that is not interpreted (recognized or not recognized) correctly by M[1]. The returned string w is called a *counterexample*.

It is not possible using only membership queries to learn finite acceptors in polynomial time, as shown by Angluin [5]. Later, Angluin [8] proved that

1 We try to unify the terminology for DFA and acceptors, thus a DFA interprets an input string while an acceptor recognizes or not an input string.

also using only equivalence queries it is not possible to learn finite acceptors in polynomial time.

We briefly recall several improvements made to L^* since the original presentation. In 1993, Rivest and Schapire [43] described a more efficient version of L^* using homing sequences[2]. In 1996, Vilar [48] proposed a version of L^* for learning transducers, using translation queries instead of membership queries. A parallel version of the algorithm L^* was presented by Balcázar et al. [11]. In 2006, Becerra-Bonache, Dediu and Tîrnăucă used correction queries instead of membership queries for learning finite acceptors [17]. The initial approach defines a correction query as a shortest string from one state to a final state; however, there exist other corrections as well. In 2009, Angluin et al. [9] generalized membership queries to label queries.

3.1 LOCAL DEFINITIONS

This chapter follows the line of the article presented by Angluin [6] about learning regular sets from queries and counterexamples. Our data structure and proofs are adapted to work for DFA. Vilar [48] presents query learning of subsequential transducers, however, due to the fact that our automata have output only on states and not on transitions, our version is simpler.

We define in a natural way a minimally adequate Teacher for query learning DFA, that is, a Teacher which is able to answer correctly two types of queries about a given target DFA $M = (Q, \Sigma, \Gamma, \tau, \lambda, q_0)$[3].

- *Output queries* (OQ). The Learner asks about the output of the state reached following the transitions dictated by a string w, starting from the initial state. The Teacher answers with $\lambda(q_0 w)$.

- *Equivalence queries* (EQ). The Learner produces a finite DFA $M' = (Q', \Sigma, \Gamma, \tau', \lambda', q_0')$ and asks whether M' is isomorphic to M. The answer of the Teacher is "yes" in the affirmative case or "no" otherwise. If the answer is "no", the Teacher also returns a string w that is not interpreted correctly by M', that is, $\lambda(q_0 w) \neq \lambda'(q_0' w)$. The returned string w is called a *counterexample*.

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-

2 Informally, a homing sequence guides the Learner such that, the outputs produced by the homing sequence, completely determine the state reached by the automaton, regardless the state the sequence is applied

3 Recall that at the end of Chapter 2, without loss of generality, we assumed that the target automaton is complete, minimal, and with all states accessible. Otherwise, the Teacher can add a virtual state, labeled with a special symbol '#' and all undefined transitions are defined as going to the virtual state. After learning the target automaton, the Learner can remove the virtual state and the transitions to it. As a small optimization, the Learner does not need to ask queries about transitions from the virtual state, as they go to the virtual state itself.

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).

An observation table can be visualized as a two-dimensional array with rows labeled by elements of $S \cup S \cdot \Sigma$ and columns labeled by elements of E, with the entry for row s and column e being equal to $C(s \cdot e)$. If s is an element of $(S \cup S \cdot \Sigma)$, then $row(s)$ denotes the finite function from E to Γ defined by $row(s)(e) = C(s \cdot e)$. By $rows(S)$, we understand the set $\{row(s) \mid s \in S\}$.

An observation table is called *closed* if for every s in $(S \cdot \Sigma \setminus S)$, there exists an s' in S such that $row(s) = row(s')$. An observation table is called *consistent* if for any s_1, s_2 in S such that $row(s_1) = row(s_2)$, we have $row(s_1 \cdot a) = row(s_2 \cdot a), \forall a \in \Sigma$.

If (S, E, C) is a closed and consistent observation table, we define a corresponding DFA, also called the *Learner's conjecture*, denoted by $\mathscr{M}(S, E, C) = (Q', \Sigma, \Gamma', \tau', \lambda', q'_0)$, where the set of states Q', the output alphabet Γ', the transition function τ', the output function λ' and the initial state q'_0, are defined as follows:

$$
\begin{aligned}
&Q' = \{row(s) \mid s \in S\} = rows(S),\\
&\Gamma' = \{C(w) \mid w \in (S \cup S \cdot \Sigma) \cdot E\},\\
&\tau'(row(s), a) = row(s \cdot a), \text{ for all } s \in S \text{ and for all } a \in \Sigma,\\
&\lambda'(row(s)) = C(s \cdot \epsilon), \text{ for all } s \in S,\\
&q'_0 = row(\epsilon).
\end{aligned}
$$

We use primes for the Learner automaton to avoid confusion with the target automaton. To see that this is a well defined automaton, note that since S is a nonempty prefix-closed set, it must contain ϵ, so q'_0 is defined. Also, since E is a nonempty suffix-closed set, it must contain ϵ. Thus, if s_1 and s_2 are elements of S such that $row(s_1)$=$row(s_2)$, then $C(s_1) = C(s_1 \cdot \epsilon) = row(s_1)(\epsilon)$ and $C(s_2) = C(s_2 \cdot \epsilon) = row(s_2)(\epsilon)$ are defined and equal to each other, hence λ' is well defined. To see that τ' is well defined, suppose s_1 and s_2 are elements of S such that $row(s_1) = row(s_2)$. Then since the observation table (S, E, C) is consistent, for each a in Σ, $row(s_1 \cdot a) = row(s_2 \cdot a)$, and since it is closed, this common value is equal to $row(s)$ for some s in S.

Let us take a finite DFA $M = (Q, \Sigma, \Gamma, \tau, \lambda, q_0)$ and a function C defined on a set of strings L over Σ^*. We say that the automaton M *is consistent* with the function C, if for any element $w \in L$, we have $C(w) = \lambda(q_0 w)$.

3.2 DATA STRUCTURE—PROPERTIES

The next series of theoretical results show several properties of closed and consistent observation tables, as well as the relation between a target automaton M and a Learner constructed conjecture $\mathscr{M}(S, E, C)$ such that M is consistent with C.

Lemma 3.1 (Angluin [6]). *Assume that* (S, E, C) *is a closed and consistent observation table. For the automaton* $\mathcal{M}(S, E, C)$ *and for every* s *in* $S \cup S \cdot \Sigma$, $\tau'(q'_0, s) = row(s)$.

Proof. This lemma can be proved by induction on the length of s.

Induction base: For the length of s being equal to 0, that is, for $s = \epsilon$, we have $\tau'(q'_0, \epsilon) = q'_0 = row(\epsilon)$ by definition of $\mathcal{M}(S, E, C)$.

Induction hypothesis: Assume that for any string s in $S \cup S \cdot \Sigma$ having length less than or equal to some i, $\tau'(q'_0, s) = row(s)$.

Induction step: For t in $S \cup S \cdot \Sigma$ having length equal to $i+1$, we can write $t = s \cdot a$, where $a \in \Sigma$ and the length of s is i. As $S \cup S \cdot \Sigma$ is prefix closed and $t \in S \cup S \cdot \Sigma$, then s is also in $S \cup S \cdot \Sigma$. We have

$$\begin{aligned}
\tau'(q'_0, t) &= \tau'(\tau'(q'_0, s), a) & \\
&= \tau'(row(s), a), & \text{by the induction hypothesis,} \\
&= row(s \cdot a), & \text{by the definition of } \tau' \text{ in } \mathcal{M}(S, E, C) \\
&= row(t) & \text{as } t = s \cdot a.
\end{aligned}$$

This completes the proof of Lemma 3.1. □

Lemma 3.2 (Angluin [6]). *Assume that* (S, E, C) *is a closed and consistent observation table. Then the automaton* $\mathcal{M}(S, E, C)$ *is consistent with the finite function* C. *That is, for every* $s \in S \cup S \cdot \Sigma$ *and* $e \in E$, $\lambda'(\tau'(q'_0, s \cdot e)) = C(s \cdot e)$.

Proof. The proof is by induction on the length of e.

Induction base: If $e = \epsilon$, and $s \in S \cup S \cdot \Sigma$, then $\tau'(q'_0, s \cdot e)$ is just $\tau'(q'_0, s) = row(s)$. If s is in S, then $\lambda'(row(s)) = C(s)$ by definition of λ' in $\mathcal{M}(S, E, C)$. If s is in $S \cdot \Sigma$, then as the observation table is closed, there exists $s' \in S$ such that $row(s') = row(s)$. We have $C(s' \cdot \epsilon) = C(s \cdot \epsilon) = \lambda'(row(s)) = \lambda'(row(s'))$.

Induction hypothesis: Suppose that for all words e' in E of length at most i, $\lambda'(\tau'(q'_0, s \cdot e')) = C(s \cdot e')$. Let e be an element of E of length $i+1$.

Induction step: Since E is suffix-closed, $e = a \cdot e'$ for some a in Σ and e' in E. Let s be an element of $S \cup S \cdot \Sigma$. As the observation table is closed, there exists $s' \in S$ such that $row(s) = row(s')$. We have

$$\begin{aligned}
\tau'(q'_0, s \cdot e) &= \tau'(q'_0, s \cdot a \cdot e'), & \\
&= \tau'(\tau'(q'_0, s), a \cdot e'), & \\
&= \tau'(row(s), a \cdot e'), & \text{by Lemma 3.1,} \\
&= \tau'(row(s'), a \cdot e'), & \text{as } row(s) = row(s'), s' \in S, \\
&= \tau'(\tau'(row(s'), a), e'), & \\
&= \tau'(row(s' \cdot a), e'), & \text{by the definition of } \tau', \\
&= \tau'(\tau'(q'_0, s' \cdot a), e'), & \text{by Lemma 3.1,} \\
&= \tau'(q'_0, s' \cdot a \cdot e'). &
\end{aligned}$$

By the induction hypothesis, $\lambda'(\tau'(q'_0, s' \cdot a \cdot e')) = C(s' \cdot a \cdot e')$; as $row(s') = row(s)$, then $C(s' \cdot a \cdot e') = C(s \cdot a \cdot e')$. Finally, we have $\lambda'(\tau'(q'_0, s \cdot e)) = \lambda'(\tau'(q'_0, s' \cdot a \cdot e')) = C(s \cdot a \cdot e') = C(s \cdot e)$. □

Lemma 3.3 (Angluin [6]). *Assume that* (S,E,C) *is a closed and consistent observation table and the conjecture* $\mathscr{M}(S,E,C)$ *has* n *states. Any automaton consistent with the finite function* C *either has more than* n *states or is isomorphic to* $\mathscr{M}(S,E,C)$.

Proof. Let us consider a new automaton $M'' = (Q'', \Sigma, \Gamma', \tau'', \lambda'', q_0'')$ having n or fewer states, consistent with C. For each state q'' in Q'' we define a finite function $f_{q''}$ from E to Γ' such that $f_{q''}(e) = \lambda''(q''e)$.

As M'' is consistent with C, we have $\lambda''(q_0''s \cdot e) = C(s \cdot e)$ for each s in $(S \cup S \cdot \Sigma)$ and for each e in E. This implies that $f_{q_0''s} = row(s)$ in $\mathscr{M}(S,E,C)$. As s ranges over S, we get at least n different values for $f_{q_0''s}$, that is, $|Q''| \geq n$. According to the initial assumption that $|Q''| \leq n$, it follows that Q'' must have exactly n states. We define a function φ from Q' to Q'' by $\varphi(row(s)) = q_0''s$. For each distinct $row(s)$ corresponding to a unique state in Q' there is a unique q'', namely $q_0''s$. This mapping is injective and surjective. Let us check that $\varphi(q_0') = q_0''$. We have $\varphi(q_0') = \varphi(row(\epsilon)) = q_0''\epsilon = q_0''$. For each s in S and a in Σ, there exists s_1, an element of S, such that $row(s \cdot a) = row(s_1)$ because the table is closed. Then

$$\varphi(\tau'(row(s), a)) = \varphi(row(s \cdot a)) = \varphi(row(s_1)) = q_0''s_1,$$

and

$$\tau''(\varphi(row(s)), a) = \tau''(q_0''s, a) = q_0''s \cdot a.$$

As the states $q_0''s_1$ and $q_0''s \cdot a$ correspond to the same $row(s_1)$ value, they must be the same state in M''. Thus we can conclude that φ preserves the transition function.

We now show that φ preserves the output function. The output of an arbitrary state $row(s)$ from M' is $C(s)$. At the same time, $\varphi(row(s)) = q_0''s$, and as M'' is consistent with C, we also have that $\lambda''(q_0''s) = C(s)$, which implies that $\lambda'(row(s)) = \lambda''(\varphi(row(s)))$, as claimed. ☐

Considering together the results of Lemma 3.2 and Lemma 3.3, we can state the following theorem.

Theorem 3.4 (Angluin [6]). *If* (S,E,C) *is a closed and consistent observation table, then the conjecture* $\mathscr{M}(S,E,C)$ *is consistent with the finite function* C. *Any other DFA consistent with* C *but not equivalent to* $\mathscr{M}(S,E,C)$ *must have more states.*

3.3 PRESENTATION OF THE ALGORITHM

Next we present the way in which the L^* algorithm develops a series of closed and consistent observation tables to discover the target automaton[4]. Actually,

4 Although in the definition of automata there should be an initial state, it would be possible to learn a target "automaton" with no states: the answer at the first $OQ(\epsilon)$ is the special symbol # and the learning algorithm stops.

L^* has two parts: a Learner L^* together with a Teacher component; we discuss separately each one of them.

The Learner algorithm uses as its main data structure the observation table. We present L^* for DFA as Algorithm 1.

Algorithm 1: L^* for DFA

```
1  Initialize $S \leftarrow \epsilon$ and $E \leftarrow \epsilon$
2  Ask $OQ(\epsilon)$
3  Ask $OQ(a)$ for each $a \in \Sigma$
4  Construct the initial observation table $(S, E, C)$
5  repeat
6      repeat
7          if (($S, E, C$) is not closed) then
8              find $s$ in $S$ and $a$ in $\Sigma$ such that $row(s \cdot a) \notin rows(S)$
9              add $s \cdot a$ to $S$
10             extend $C$ to $(S \cup S \cdot \Sigma) \cdot E$ asking $OQ(s \cdot a \cdot c \cdot e)$, $c$ in $\Sigma$ and $e$
               in $E$
11         if (($S, E, C$) is not consistent) then
12             find $s_1, s_2 \in S$, $a \in \Sigma$ and $e \in E$ such that $row(s_1) = row(s_2)$
               and $C(s_1 \cdot a \cdot e) \neq C(s_2 \cdot a \cdot e)$
13             add $a \cdot e$ to $E$
14             extend $C$ to $(S \cup S \cdot \Sigma) \cdot E$ asking for not asked before
               $OQ(\alpha \cdot a \cdot e)$ for all $\alpha$ in $S \cup S \cdot \Sigma$
15     until (($S, E, C$) is closed and consistent)
16     Construct the conjecture $\mathscr{M}(S, E, C)$
17     if (the Teacher replies with a counter-example $s$) then
18         add $s$ and all its prefixes to $S$
19         extend $C$ to $(S \cup S \cdot \Sigma)E$ asking (not asked before) $OQ(\alpha \cdot c \cdot e)$,
           for all $\alpha$ in $Pref(s)$, $c$ in $\{\epsilon\} \cup \Sigma$ and $e$ in $E$
20 until (the Teacher replies yes to the conjecture)
21 Output $\mathscr{M}(S, E, C)$
```

We describe now how L^* works. Initially $S = E = \{\epsilon\}$. To determine C, the Learner asks output queries for ϵ and each a in Σ.

The inner **"repeat"** loop starting at line 6 checks whether the current observation table (S, E, C) is closed and consistent. If (S, E, C) is not closed (consistent), the algorithm adds a new string to S (to E) and updates the table asking output queries for the missing elements.

When the Learner's automaton is closed and consistent, the Learner asks an equivalence query. The Teacher's answer can be 'yes' (in this case the algorithm ends with the output $\mathscr{M}(S, E, C)$) or 'no'. In this case, the Teacher provides a counterexample, while the Learner adds all its prefixes to S and updates the observation table using the output queries.

3.4 CORRECTNESS AND COMPLEXITY

Theorem 3.5 (Angluin [6]). *Given any minimally adequate Teacher presenting an unknown minimal DFA M, the Learner L^* terminates and correctly outputs a DFA isomorphic to M. If n is the number of states of M and m 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 n.*

Proof. The correctness of the algorithm follows from the following fact. If the algorithm eventually terminates, the answer to the last conjecture is "yes", that is the output of the algorithm is an automaton isomorphic to the target DFA (by the definition of a minimally adequate Teacher).

Let us note that in line number 16, the algorithm uses a closed and consistent observation table. The number of different rows in the observation table monotonically increases as L^* runs. If a string is added to E (the "not consistent" branch), the number of different rows increases, two previous equal values $row(s_1)$ and $row(s_2)$ becoming different after adding the new string to E. Similarly, on the "not closed" branch, a new value s is added to S, and $row(s)$ is different from all existing rows in S. The total number of times the algorithm performs these operations (closing, or solving the consistency) is bounded by $n-1$, as initially the observation table contains $row(\epsilon)$. For an incorrect conjecture $\mathscr{M}'(S', E', C')$, the target automaton M which is consistent with C' but inequivalent with $\mathscr{M}'(S', E', C')$ should have at least one more state. Thus, there can be at most $n-1$ incorrect conjectures and L^* correctly terminates after making at most n conjectures.

Let k be the number of input symbols, that is, $k = |\Sigma|$. Let us denote by m the maximum length of a counterexample returned by the Teacher. Initially the sets S and E each contain one element, namely, ϵ. There can be at most n "not closed" or "not consistent" operations. Thus, the total number of elements in E cannot exceed n and the maximum length of a string in E can be at most $n-1$. The total number of strings in S cannot exceed $n+m(n-1)$, for the first n strings a similar reasoning as for E holds, and there can be at most $n-1$ counterexamples, each of which can cause at most m strings to be added to S. The maximum length of a string from S can be $n-1+m$: for every closing operation, the maximum length a string in S is increased by at most one, and for a counterexample, by no more than the length of the counterexample. To conclude, we get the maximum number of elements in $(S \cup S \cdot \Sigma) \cdot E$ being at most

$$(k+1)(n+m(n-1))n = O(mn^2),$$

and the maximum length of any string in $(S \cup S \cdot \Sigma) \cdot E$ being at most

$$m+2n-1 = O(m+n).$$

The number of output queries could be reduced, when compared with the size of $(S \cup S \cdot \Sigma) \cdot E$: some of the queries being asked several times. The queries from the first column of the observation table, that is, $(S \cup S \cdot \Sigma) \cdot \epsilon$, should be asked. The queries from the area $S \cdot (E \setminus \{\epsilon\})$ of the observation table are initially asked in the area $(S \cdot \Sigma) \cdot E$. Counting the number of OQ, we get $n + m(n-1) + nk(n + m(n-1))$ and $O(OQ) = O(mn^2)$.

We now show that L^* performs in polynomial time. Checking whether the observation table is closed or consistent can be done in a time polynomial in the size of the observation table, and these operations are performed at most $n-1$ times. The number of queries asked by the Learner is polynomial. Constructing a conjecture can be done in a time polynomial in the size of the observation table, and is performed at most n times.

Thus, the total running time of L^* is bounded by a polynomial in m and n.

□

The Teacher of L^* should answer OQ and EQ. To answer an OQ, the Teacher receives a string and should move in the target automaton starting from the initial state, returning back the output of the state reached, which is an algorithm linear in the length of the interrogating word. As the maximum length of any string in $(S \cup S \cdot \Sigma) \cdot E$ is at most $m + 2n - 1$, then answering an OQ is clearly a polynomial algorithm.

We describe now the procedure for answering an EQ. If the target automaton $M = (Q, \Sigma, \Gamma, \tau, \lambda, q_0)$ has more states than the conjecture $M' = (Q', \Sigma, \Gamma', \tau', \lambda', q'_0)$, then the answer for the EQ is negative and a counterexample should be provided. The Teacher constructs the finite acceptor $D = (Q \times Q', \Sigma, \{0,1\}, \tau'', \lambda'', (q_0, q'_0))$, where

$$\tau''((q,q'),a) = (\tau(q,a), \tau'(q',a))$$

for all states $q \in Q$, $q' \in Q'$ and input symbols $a \in \Sigma$. The output function λ'' gives an accepting state for counterexamples as follows:

$$\lambda''((q,q')) = \begin{cases} 1, & \text{if } \lambda(q) \neq \lambda'(q'), \\ 0, & \text{otherwise.} \end{cases}$$

Thus, the strings recognized by D are the counterexamples.

We note that constructing an isomorphism between the target automaton and the queried automaton is also a polynomial time algorithm (we map the initial states, after that for each transition we map the destinations) and we get the total running time of the Teacher and the Learner together to be polynomial.

3.5 ILLUSTRATIVE EXAMPLE

Now, we present the steps performed by L^* to learn a given DFA. Take the automaton $M_4 = ([10], \{a,b\}, [5], \tau, \lambda, 1)$ where the transition function τ, the output function λ, and a corresponding graphical representation are given in Figure 5.

State	$\tau(a)$	$\tau(b)$	λ
1	3	5	3
2	4	10	2
3	3	8	5
4	4	2	4
5	4	3	1
6	1	2	5
7	5	6	5
8	2	7	1
9	3	8	3
10	9	4	2

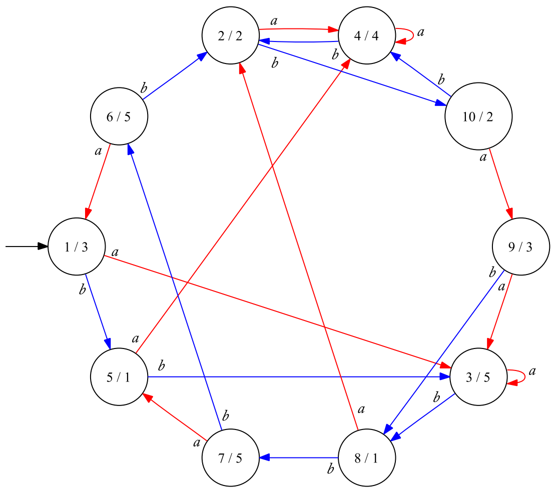

Figure 5: The transition function τ, the output function λ, and a graphical representation of the automaton M_4

In the initial observation table, Table 8, $row(a)$ is not closed, a is added to S. The next observation tables, Tables 9–11, are not closed, and the corresponding values are added to S.

Table 8: The initial observation table, $S = E = \{\epsilon\}$; $row(a) \notin rows(S)$

$(S,E,C)_1$		E
Row	$S \cup S\Sigma$	ϵ
1	ϵ	3
2	a	5
3	b	1

Table 9: $S = \{\epsilon, a\}$, $E = \{\epsilon\}$; $row(b) \notin rows(S)$

$(S,E,C)_2$		E
Row	$S \cup S\Sigma$	ϵ
1	ϵ	3
2	a	5
3	b	1
4	aa	5
5	ab	1

Table 10: $S = \{\epsilon, a, b\}$, $E = \{\epsilon\}$; $row(ba) \notin rows(S)$

$(S,E,C)_3$		E
Row	$S \cup S\Sigma$	ϵ
1	ϵ	3
2	a	5
3	b	1
4	aa	5
5	ab	1
6	ba	4
7	bb	5

Table 11: $S = \{\epsilon, a, b, ba\}$, $E = \{\epsilon\}$; $row(bab) \notin rows(S)$

$(S,E,C)_4$		E
Row	$S \cup S\Sigma$	ϵ
1	ϵ	3
2	a	5
3	b	1
4	ba	4
5	aa	5
6	ab	1
7	bb	5
8	baa	4
9	bab	2

The observation table described by Table 12 is closed and consistent and L^* constructs the first conjecture M_5, presented in Figure 6.

Table 12: $S = \{\epsilon, a, b, ba, bab\}$, $E = \{\epsilon\}$; table closed and consistent

$(S,E,C)_5$		E	Q'
Row	$S \cup S\Sigma$	ϵ	
1	ϵ	3	q'_1
2	a	5	q'_2
3	b	1	q'_3
4	ba	4	q'_4
5	bab	2	q'_5
6	aa	5	q'_2
7	ab	1	q'_3
8	bb	5	q'_2
9	baa	4	q'_4
10	baba	4	q'_4
11	babb	2	q'_5

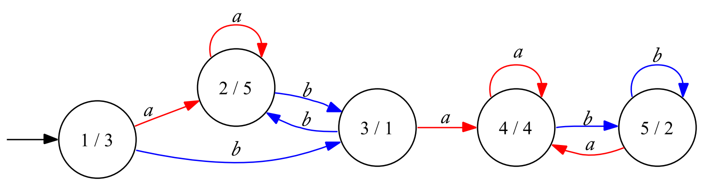

Figure 6: First conjecture M_5

As the conjecture M_5 is not the correct automaton, the Teacher constructs the acceptor of counterexamples. A minimized version is presented in Figure 7.

The problem of choosing the "best" counterexample still has some unknown aspects. Choosing a larger counterexample, the Teacher communicates about the existence of more states to the Learner, something which could reduce the number of future equivalence queries, however at the price of an increased number of output queries. In our case, a minimal counterexample would be aba. But, following this way would require a supplementary counterexample. We could choose $ababa$ and this requires no additional

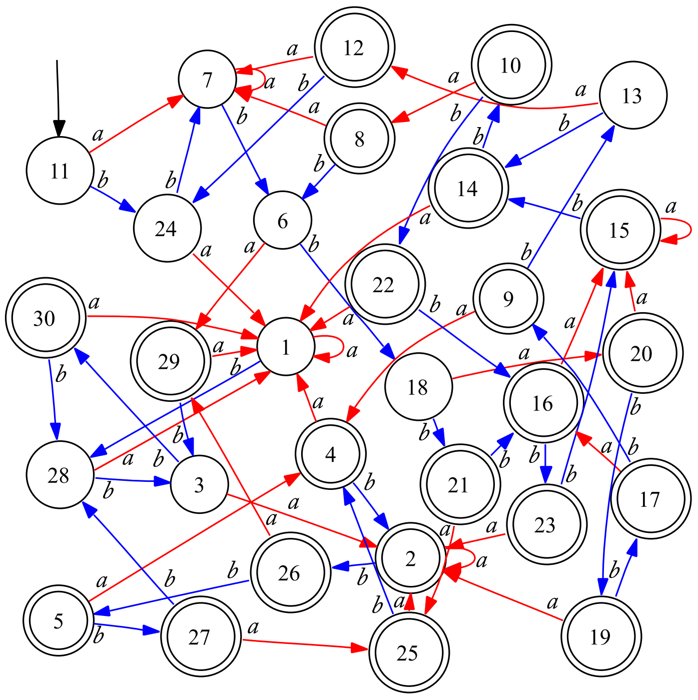

Figure 7: A minimal acceptor for counterexamples for conjecture M_5

counterexamples until the target automaton is discovered. We follow this way.

In Table 13 we see that $row(b)$ and $row(ab)$ have the same value, while a transition with a sends to different states, that is $row(ba)$ is different from $row(aba)$. Thus, L^* adds the string a to E and asks output queries for the new entries of the observation table.

Table 14 is not closed: $row(abb)$ is in $S \cdot \Sigma$ and not in S. The L^* algorithm adds abb to S and consequently the strings $abba$ and $abbb$ to $S \cdot \Sigma$, asking output queries for the new entries.

In Table 15 we see that $row(\epsilon)$ and $row(ababa)$ have the same output. A transition with b sends to different states, that is $row(b)$ is different from $row(ababab)$; more precisely, the inconsistency is in column a. Thus, L^* adds the string ba to E and asks output queries for the new entries of the observation table.

Table 16 is not closed: $row(abbb)$ is in $S \cdot \Sigma$ and not in S. The L^* algorithm adds $abbb$ to S and consequently the strings $abbba$ and $abbbb$ to $S \cdot \Sigma$, asking output queries for the new entries.

Table 13: $S = \{\epsilon, a, b, ba, aba, bab, abab, ababa\}$, $E = \{\epsilon\}$; $row(b)$, $row(ab)$ are not consistent, transition with a

$(S,E,C)_6$		E
Row	$S \cup S\Sigma$	ϵ
1	ϵ	3
2	a	5
3	b	1
4	ab	1
5	ba	4
6	aba	2
7	bab	2
8	abab	2
9	ababa	3
10	aa	5
11	bb	5
12	abb	5
13	baa	4
14	abaa	4
15	baba	4
16	babb	2
17	ababb	4
18	ababaa	5
19	ababab	1

Table 14: $row(abb) \notin rows(S)$

$(S,E,C)_7$		E	
Row	$S \cup S\Sigma$	ϵ	a
1	ϵ	3	5
2	a	5	5
3	b	1	4
4	ab	1	2
5	ba	4	4
6	aba	2	4
7	bab	2	4
8	abab	2	3
9	ababa	3	5
10	aa	5	5
11	bb	5	5
12	abb	5	1
13	baa	4	4
14	abaa	4	4
15	baba	4	4
16	babb	2	3
17	ababb	4	4
18	ababaa	5	5
19	ababab	1	2

Table 15: row(ϵ), row(ababa) are not consistent, transition with b

$(S,E,C)_8$		E	
Row	$S \cup S\Sigma$	ϵ	a
1	ϵ	3	5
2	a	5	5
3	b	1	4
4	ab	1	2
5	ba	4	4
6	aba	2	4
7	abb	5	1
8	bab	2	4
9	abab	2	3
10	ababa	3	5
11	aa	5	5
12	bb	5	5
13	baa	4	4
14	abaa	4	4
15	abba	1	4
16	abbb	5	3
17	baba	4	4
18	babb	2	3
19	ababb	4	4
20	ababaa	5	5
21	ababab	1	2

The observation table described by Table 17 is closed and consistent and L^* constructs a second conjecture M_6. This time the conjecture M_6 is isomorphic with the target automaton and we present this isomorphism in Table 18.

Table 16: row(abbb) $\notin$ rows(S)

$(S,E,C)_9$		E		
Row	$S \cup S\Sigma$	ϵ	a	ba
1	ϵ	3	5	4
2	a	5	5	2
3	b	1	4	5
4	ab	1	2	1
5	ba	4	4	4
6	aba	2	4	3
7	abb	5	1	3
8	bab	2	4	3
9	abab	2	3	4
10	ababa	3	5	2
11	aa	5	5	2
12	bb	5	5	2
13	baa	4	4	4
14	abaa	4	4	4
15	abba	1	4	5
16	abbb	5	3	4
17	baba	4	4	4
18	babb	2	3	4
19	ababb	4	4	4
20	ababaa	5	5	2
21	ababab	1	2	1

Table 17: table closed and consistent

$(S,E,C)_{10}$		E			Q'
Row	$S \cup S\Sigma$	ϵ	a	ba	
1	ϵ	3	5	4	q'_1
2	a	5	5	2	q'_2
3	b	1	4	5	q'_3
4	ab	1	2	1	q'_4
5	ba	4	4	4	q'_5
6	aba	2	4	3	q'_6
7	abb	5	1	3	q'_7
8	bab	2	4	3	q'_6
9	abab	2	3	4	q'_8
10	abbb	5	3	4	q'_9
11	ababa	3	5	2	q'_{10}
12	aa	5	5	2	q'_2
13	bb	5	5	2	q'_2
14	baa	4	4	4	q'_5
15	abaa	4	4	4	q'_5
16	abba	1	4	5	q'_3
17	baba	4	4	4	q'_5
18	babb	2	3	4	q'_8
19	ababb	4	4	4	q'_5
20	abbba	3	5	4	q'_1
21	abbbb	2	4	3	q'_6
22	ababaa	5	5	2	q'_2
23	ababab	1	2	1	q'_4

Table 18: The isomorphism between the conjecture and the target automaton

Q'	q'_1	q'_2	q'_3	q'_4	q'_5	q'_6	q'_7	q'_8	q'_9	q'_{10}
Q	1	3	5	8	4	2	7	10	6	9

We note that during all these steps needed to discover the target automaton, the Learner asks

- only two equivalent queries, the last one successful,
- 47 output queries that are enough to fill in all the entries in the observation tables.

3.6 REMARKS

The L^* algorithm presented in this chapter learns DFA, generalizing the algorithm presented by Angluin, about learning regular sets (finite acceptors).

This chapter constitutes the theoretical base for the next two chapters, learning with label queries and learning with correction queries, both focusing on particular aspects of learning DFA. Thus, learning with label queries emphasizes the influence of a helpful Teacher, which is able to assign output symbols, labeling the states for a faster learning, while learning from correction queries represents a particular type of labeling.

The next chapter will present more experimental results, comparing the number of queries needed by L^* and the number of label queries needed to learn a large set of test automata.

CHAPTER REFERENCES

[5] Dana Angluin. "A Note on the Number of Queries Needed to Identify Regular Languages". In: *Information and Control* 51.1 (1981), pp. 76–87 (cit. on p. 29).

[6] Dana Angluin. "Learning regular sets from queries and counterexamples". In: *Information and Computation* 75.2 (1987), pp. 87–106. DOI: http://dx.doi.org/10.1016/0890-5401(87)90052-6 (cit. on pp. 12, 15, 16, 29, 30, 32, 33, 35).

[7] Dana Angluin. "Queries and Concept Learning". In: *Machine Learning* 2.4 (1988), pp. 319–342. DOI: http://dx.doi.org/10.1023/A:1022821128753 (cit. on p. 29).

[8] Dana Angluin. "Negative Results for Equivalence Queries". In: *Machine Learning* 5.2 (1990), pp. 121–150. DOI: http://dx.doi.org/10.1023/A:1022692615781 (cit. on p. 29).

[9] Dana Angluin, Leonor Becerra-Bonache, Adrian-Horia Dediu, and Lev Reyzin. "Learning finite automata using label queries". In: *Proceedings of the 20th International Conference on Algorithmic Learning Theory, Porto, Portugal, October 3–5, 2009*. Ed. by Ricard Gavaldà, Gábor Lugosi, Thomas Zeugmann, and Sandra Zilles. Vol. 5809. Lecture Notes in Computer Science. Springer-Verlag, 2009, pp. 171–185 (cit. on pp. 16, 20, 30, 45, 46, 95).

[11] José L. Balcázar, Josep Díaz, and Ricard Gavaldà. "Algorithms for Learning Finite Automata from Queries: A Unified View". In: *Advances in Algorithms, Languages, and Complexity*. 1997, pp. 53–72 (cit. on p. 30).

[17] 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*. Vol. 4201. Lecture Notes in Computer Science. Tokyo, Japan: Springer-Verlag, 2006, pp. 281–292 (cit. on pp. 17, 30, 45, 55, 95).

[24] William I. Gasarch, Efim B. Kinber, Mark G. Pleszkoch, Carl H. Smith, and Thomas Zeugmann. "Learning via Queries with Teams and Anomalies". In: *Fundam. Inf.* 23.1 (Jan. 1995), pp. 67–89. URL: http://dl.acm.org/citation.cfm?id=2383376.2383378 (cit. on p. 29).

[28] Olga Grinchtein and Martin Leucker. "Learning finite-state machines from inexperienced teachers". In: *Proceedings of the 8th International Conference on Grammatical Inference: Algorithms and Applications*. Vol. 4201. Lecture Notes in Computer Science. Tokyo, Japan: Springer-Verlag, 2006, pp. 344–345. DOI: 10.1007/11872436_30. URL: http://dx.doi.org/10.1007/11872436_30 (cit. on p. 29).

[43] Ronald L. Rivest and Robert E. Schapire. "Inference of Finite Automata Using Homing Sequences". In: *Information and Computation* 103.2 (Apr. 1993), pp. 299–347. DOI: 10.1006/inco.1993.1021. URL: http://dx.doi.org/10.1006/inco.1993.1021 (cit. on p. 30).

[47] Leslie Gabriel Valiant. "A theory of the learnable". In: *Communications of the ACM* 27.11 (1984), pp. 1134–1142. DOI: http://doi.acm.org/10.1145/1968.1972 (cit. on pp. 15, 29).

[48] Juan Miguel Vilar. "Query learning of subsequential transducers". In: *Proceedings of the Third International Colloquium on Grammatical Interference (ICGI-96): Learning Syntax from Sentences, Montpellier, France, September*. Ed. by Laurent Miclet and Colin de la Higuera. Vol. 1147. Lecture Notes in Computer Science. Springer-Verlag, 1996, pp. 72–83 (cit. on p. 30).

4 LABEL QUERIES

In this chapter, we study the influence of an increased number of output symbols on the learning process. We allow a Teacher to add labels to a target automaton and to answer Learner's queries with both the output of the target automaton and the label attached.

In 2009, Angluin et al. [9] presented label queries in a general context, with labels carefully chosen or random labels for different learning scenarios. Here, we present label queries only for L^*. Although the label queries were introduced after correction queries (Becerra et al. [14, 17]), we find the presentation of corrections queries as a particular type of label queries to be more appropriate.

4.1 LOCAL DEFINITIONS

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 (defined in the previous chapter) for a labeled target automaton and this kind of query will be referred to as a *label query* (LQ for short).

There exists a variant of the L^* algorithm, also discussed by Tîrnăucă and Knuutila [45], that initializes the content of the set E of an observation table (defined in the previous chapter) 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^*.

4.2 THEORETICAL ASPECTS

For labeled automata, there are two particular situations. Suppose that the Teacher labels each of the states with the same symbol and then there is no influence on the learning process. The other case would be when the Teacher labels the states in such a way that for each state there results a different pair of symbols. We easily obtain the following result.

Proposition 4.1 (Angluin et al. [9]). *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 were needed.*

Proof. Let n be the number of states of the target automaton. The observation table of L^* is not closed until discovering all the states. We prove by induction on the number of states that an accessible, minimal automaton labeled with an injective λ_ℓ function can be learned using only $|Q||\Sigma| + 1$ label queries.

Induction base: Clearly for an automaton with one state, we need one query for the output of the initial state and $|\Sigma|$ queries for transitions.

Induction hypothesis: Assume that any accessible, minimal automata with n states and with an injective λ_ℓ function can be learned using only $n|\Sigma| + 1$ label queries.

Induction step: Suppose given

$$M_{n+1} = (Q_{n+1}, \Sigma, \Gamma \times \Lambda, \tau_{n+1}, \lambda_{\ell,n+1}, q_{0,n+1})$$

an accessible, minimal automaton with $n+1$ states with an injective labeling function.

We mark all the states with levels, starting with the initial state, we assign to it level 0. We assign level 1 to all states not yet assigned, reached by one step transitions from the initial state. In general we assign level $i+1$ to all states not yet assigned, reached by one step transitions from all states with level i. The assignment should stop after all states get a level, as all the states are reachable from the initial state, and we have a finite number of states. Let μ be the maximum level. The states reached with transitions from one state on level μ should also be one step transitions from some states with the level strictly less than μ, otherwise μ would not be the maximum level.

We take one arbitrary state p, from the level μ. We replace all input transitions to p with self-loops to the states they are coming from, that is, we construct a new transition function $\tau' : (Q_{n+1} \setminus \{p\}) \times \Sigma \to Q_{n+1} \setminus \{p\}$, defined as follows.

$$\tau'(q,a) = \begin{cases} \tau_{n+1}(q,a) & \text{if } \tau_{n+1}(q,a) \neq p, \\ q & \text{if } \tau_{n+1}(q,a) = p. \end{cases}$$

We get an automaton

$$M_n = (Q_{n+1} \setminus \{p\}, \Sigma, \Gamma \times \Lambda, \tau', \lambda', q_{0,n+1}),$$

where λ' is the restriction of $\lambda_{\ell,n+1}$ to $Q_{n+1} \setminus \{p\}$. The automaton M_n satisfies the conditions of the induction hypothesis, that is, can be learned using only $n|\Sigma| + 1$ label queries. We modify the automaton M_n in the following way. For a particular symbol $b \in \Sigma$, we construct a new transition function $\tau'_b : Q_{n+1} \times \Sigma \to Q_{n+1}$, defined as follows.

$$\tau'_b(q,a) = \begin{cases} \tau_{n+1}(q,a) & \text{if } \tau_{n+1}(q,a) \neq p, \\ q & \text{if } \tau_{n+1}(q,a) = p \text{ and } a \neq b, \\ p & \text{otherwise.} \end{cases}$$

Let $M_{n+1,b}$ be the automaton $(Q_{n+1}, \Sigma, \Gamma \times \Lambda, \tau'_b, \lambda_{\ell,n+1}, q_{0,n+1})$. To learn the automaton $M_{n+1,b}$, the algorithm L^* follows exactly the same steps as for M_n except when querying for the transition connecting p, the observation table becomes non-closed. We follow the learning steps as for M_n, not including the row accessing p in S until all the other n states are included in S (it was possible to learn the complete M_n, and all its states are one step transitions from some states with the level strictly less than μ). According to the induction hypothesis, up to this moment there have been $n|\Sigma|+1$ label queries and the observation table is not closed. We add to S the string leading from the initial state to p; there are $|\Sigma|$ queries needed to discover the transitions from p. Thus, to learn the automaton $M_{n+1,b}$, we need $(n+1)|\Sigma|+1$ queries. However, if there are more than one input transitions to p, this is not yet the automaton M_{n+1}. For all the other transitions from M_{n+1} connecting to p, we take them one by one and we follow exactly the same learning steps as for $M_{n+1,b}$. This time no additional query are needed, after adding to S one of of the strings leading from the initial state to p, the observation table becomes closed. We get thus that we need $(n+1)|\Sigma|+1$ queries to learn the automaton M_{n+1}. □

From the complexity of L^* we also note that the number of output symbols does not influence the size of the observation table. However, the degree of distinguishability depends on the number of symbols of the output alphabet, and this could also change the number of strings from E.

We show now that for any automaton M, and $j \geq 1$, there exists a labeling with a non-injective output function λ_ℓ, such that L^*_j is able to learn M without counterexamples.

Proposition 4.2. *Let $M = (Q, \Sigma, \Gamma, \tau, \lambda, q_0)$ be a minimal automaton with n states, m output symbols and k input symbols, such that $m < n$. There exists a labeling ℓ of M, with $|\lambda_\ell(Q)| < n$, such that for $1 \leq j$, L^*_j is able to learn the target automaton with a number of label queries that is $O(nk^{j+1})$ and without needing counterexamples.*

Proof. 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, if $|\Pi_j| < n$, then Π_{j+1} is a strict refinement of $|\Pi_j|$. To show this, let us take two states, p and q in Q that are j-indistinguishable. As we established that M is minimal, that is, that all states are distinguishable, then there exists $r > j$ and a string $w = w_1 \dots w_r$, with $w_1, \dots, w_r$ in Σ, such that w is a shortest (in terms of the length order) distinguishing string for (p, q). Let $p' = pw_1 \dots w_{r-j-1}$ and $q' = qw_1 \dots w_{r-j-1}$ be the states reached from p and q following the transitions with the first $r-j-1$ characters of w. The string $w_{r-j} \dots w_r$ of length $j+1$ is a distinguishing string for the pair (p', q') which is j-indistinguishable, otherwise it would contradict the minimality of the length of the string w. This part of the proof followed the lines of the proof made by Moore [39].

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. To show this, let us take two states p and q in Q. If they are in different equivalence classes, then there must exists a string no longer than j to distinguish between p and q, thus their j-signature trees are different. If p and q are in the same equivalence class, they have different labels; therefore, their j-signature trees are different.

The Teacher can label the states such that each state has a unique j-signature tree. It is easy to show that there is a one-to-one correspondence between j-signature trees and the rows in the observation table of L_j^*. In these conditions the proof that L_j^* can learn an automaton without counterexamples is similar with the proof of Proposition 4.1. The greater is j, the less is the number of labels needed. 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, because there are m equivalence classes in Π_0 and by hypothesis we have $m < n$. As Π_j is a strict refinement of Π_0, 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 the same output, that is, $\lambda(p) = \lambda(q)$. The labels associated with $\langle p\rangle_{R_j}$ and $\langle q\rangle_{R_j}$ must share the common labels corresponding to the minimum from $|\langle p\rangle_{R_j}|$ and $|\langle q\rangle_{R_j}|$, that is at least one label, 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 of L_j^*. Note that L_j^* needs in the observation table as many rows as L^*, that is, the size of $S \cup S \cdot \Sigma$ is $O((k+1)n)$. The set E contains $k^{j+1} - 1$ strings. The queries corresponding to the area $S \cdot E$ with the exception of column ϵ exist also in the area $S \cdot \Sigma \cdot E$. Thus, the number of queries is $O(nk^{j+1})$ and this concludes the proof. $\square$

In particular, it is easy to prove by induction on the number of states the following result.

Corollary 4.3. *For a minimal automaton with n states, m output symbols and k input symbols, with $m < n$, there exists a labeling with less than n labeled output symbols such that L_1^* needs $nk^2 + k + 1$ label queries and no counterexample to learn the automaton.*

Proof. For a new state s added to S in an observation table of L_1^*, we note the following facts.

- as the row corresponding to the new state already existed in $S \cdot \Sigma$, there is no new label query needed to be asked for s itself.

- We add k rows in $S \cdot \Sigma$, each row labeled by $s \cdot b$, for all b in Σ. However, for the entries $s \cdot b \cdot \epsilon$, we do not need label queries, since these entries are already known from $s \cdot b$ (already added to S).
- For each of the k new lines added to the observation table, we need to ask k label queries for each symbol of the input alphabet, therefore k^2 new label queries.

For an automaton with one state, there are $k+1$ label queries for the line in S and k^2 queries for the lines corresponding to $S \cdot \Sigma$. For an automaton with n states, as for each state other than the initial state we need k^2 queries, we get $k+1+k^2+(n-1)k^2 = nk^2+k+1$ label queries needed to learn the automaton. □

4.3 ILLUSTRATIVE EXAMPLE

Example 4.4. *Let us consider the following target finite acceptor*

$$M_7 = ([6], \{a, b\}, \{0, 1\}, \tau, \lambda, 1),$$

with the transition function τ, and output function λ presented in Table 19.

For the automaton M_7, we have $\Pi_0 = \{\{1,3,5,6\},\{2,4\}\}$ and we can compute Π_1 using the 1-signature trees of states (Table 20).

We get $\Pi_1 = \{\{1\},\{3,5,6\},\{2\},\{4\}\}$. When L_1^* learns M_7, the first closed and consistent observation table has the same value for the rows corresponding to states 3, 5, and 6. A helpful teacher, assigning different labels to these states, makes also the first closed and consistent table to have different rows for these states. We note that there is no need for helpful labels if the states have different outputs or are distinguishable by a or b. That gives us a possible (helpful) labeling function ℓ_1 presented in Table 21 together with the labeled automaton $M_7^{\ell_1}$ (Figure 8). We note that the labeled automaton has only four different outputs, $(0,1), (1,1), (0,2), (0,3)$.

Table 19: The finite acceptor M_7

State	$\tau(a)$	$\tau(b)$	λ
1	2	2	0
2	6	4	1
3	1	6	0
4	3	5	1
5	1	3	0
6	1	1	0

Table 20: and the 1-signature trees

State	ϵ	a	b
1	0	1	1
2	1	0	1
3	0	0	0
4	1	0	0
5	0	0	0
6	0	0	0

We present now the first observation table of L_1^* learning $M_7^{\ell_1}$ (Table 22). After a series of non-closed observations tables, we get the final observation table (Table 23). We need 27 label queries (no counterexamples) to find a Learner's conjecture that is isomorphic to the target automaton.

Table 21: The finite DFA $M_7^{\ell_1}$

State	$\tau(a)$	$\tau(b)$	λ	ℓ_1
1	2	2	0	1
2	6	4	1	1
3	1	6	0	1
4	3	5	1	1
5	1	3	0	2
6	1	1	0	3

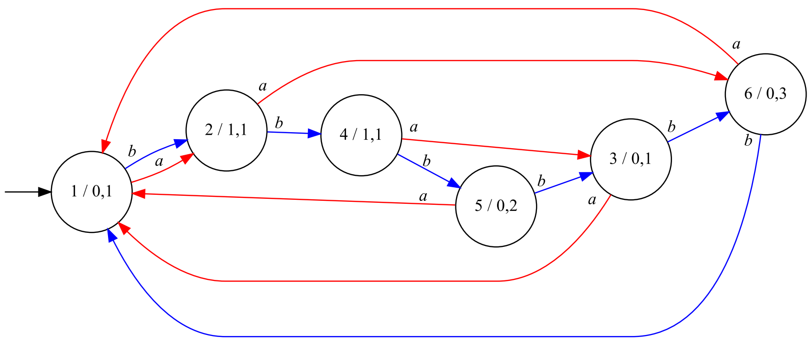

Figure 8: A graphical representation of $M_7^{\ell_1}$

Table 22: The initial observation table of L_1^* learning $M_7^{\ell_1}$, $S=\{\epsilon\}$, $E=\{\epsilon,a,b\}$; $row(a)\notin rows(S)$

$(S,E,C)_1$		E		
Row	$S\cup S\Sigma$	ϵ	a	b
1	ϵ	0,1	1,1	1,1
2	a	1,1	0,3	1,1
3	b	1,1	0,3	1,1

4.4 COMPARATIVE RESULTS

We present some of the results of testing more than 200 minimal, accessible and complete finite acceptors. Our test set contains randomly generated automata using a binary alphabet and a number of states between 1 and 20. Table 24 presents a sample from the results of our tests. In the Appendix, Table 35 gives the complete set of results for our experiments.

We present in parallel the comparative results after testing the original L^* algorithm and the version of L_1^* with helpful labels. We show the number of MQ and EQ for L^* as well as the number of LQ, the number of labels, and the number of output symbols after labeling for L_1^*.

We observe 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 9.

Table 23: The final observation table of L_1^* learning $M_7^{\ell_1}$, $S=\{\epsilon, a, aa, ab, aba, abb\}$, $E=\{\epsilon, a, b\}$

$(S,E,C)_6$		E		
Row	$S \cup S\Sigma$	ϵ	a	b
1	ϵ	0,1	1,1	1,1
2	a	1,1	0,3	1,1
3	aa	0,3	0,1	0,1
4	ab	1,1	0,1	0,2
5	aba	0,1	0,1	0,3
6	abb	0,2	0,1	0,1
7	b	1,1	0,3	1,1
8	aaa	0,1	1,1	1,1
9	aab	0,1	1,1	1,1
10	$abaa$	0,1	1,1	1,1
11	$abab$	0,3	0,1	0,1
12	$abba$	0,1	1,1	1,1
13	$abbb$	0,1	0,1	0,3

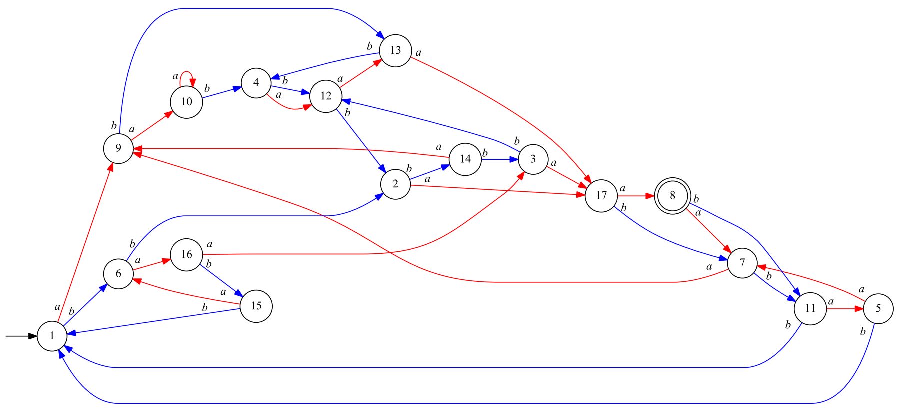

Figure 9: For the automaton P167 having 17 states, we need 16 different output symbols to allow L_1^* to work without counterexamples

The number of label queries is concordant with Corollary 4.3, not depending on the automaton structure. We wish to represent graphically the dependency on the number of states of the number of queries needed by L^* and L_1^* to learn different automata. We have 11 different automata for each number of states. In Table 25, 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 10, the axis y is logarithmic for better visualization.

Table 24: A sample from a test set for learning DFA with L^* and L_1^* with helpful labels

Test		L^*		L_1^* with helpful labels			
Id	States	MQ	EQ	LQ	Labels	$\lvert\lambda_{\ell_1}\rvert$	Observations
P1	2	5	1	11	1	2	$m = n$
P2	2	5	1	11	1	2	$m = n$
...	...	...	...	...	...	...	...
P160	16	242	5	67	4	7	
P161	16	509	9	67	11	13	
P162	16	415	9	67	10	11	
P163	16	167	6	67	3	6	
P164	16	206	6	67	4	7	
P165	16	215	4	67	6	8	
P166	17	247	6	71	9	12	
P167	17	506	7	71	15	16	$\lvert\lambda_{\ell_1}\rvert = n-1$
P168	17	383	8	71	8	10	
P169	17	351	8	71	10	12	
P170	17	285	7	71	8	11	
P171	17	351	7	71	8	11	
P172	17	239	5	71	7	9	
P173	17	274	5	71	11	14	
P174	17	206	6	71	6	9	
P175	17	175	5	71	3	6	
P176	17	233	6	71	4	7	
...	...	...	...	...	...	...	...
P199	20	242	5	83	4	8	
P200	20	467	7	83	16	17	
P201	20	224	5	83	5	8	
P202	20	279	5	83	5	8	
P203	20	269	6	83	5	8	
P204	20	854	9	83	16	17	
P205	20	215	6	83	5	9	
P206	20	314	7	83	8	10	
P207	20	231	6	83	4	7	
P208	20	274	6	83	8	11	
P209	20	269	5	83	7	10	

4.5 REMARKS

From the proof of Proposition 4.2, it clearly results that the complexity of a helpful Teacher labeling a DFA for allowing L_1^* to learn the target automaton without needing counterexamples is polynomial.

The d-signature trees represent a common aspect between the algorithms discussed in this chapter and passive learning.

Table 25: Comparative results, 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

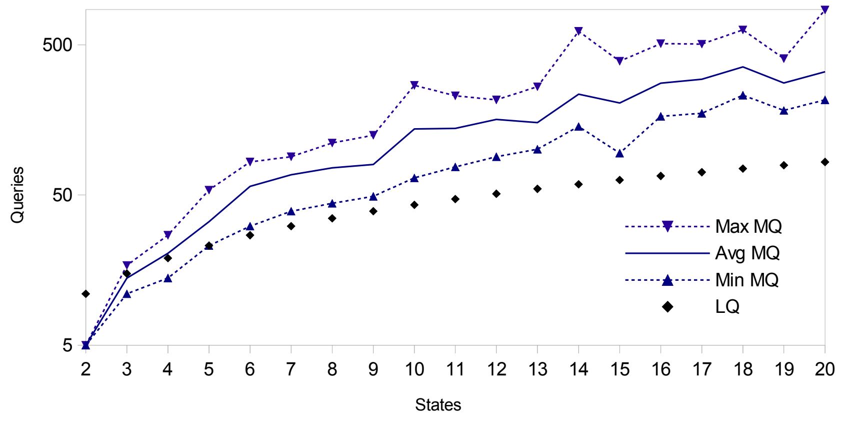

Figure 10: Membership and label queries depending on the number of states

We believe that the minimal number of labels needed to learn automata without counterexamples could 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.

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

CHAPTER REFERENCES

[9] Dana Angluin, Leonor Becerra-Bonache, Adrian-Horia Dediu, and Lev Reyzin. "Learning finite automata using label queries". In: *Proceedings of the 20th International Conference on Algorithmic Learning Theory, Porto, Portugal, October 3–5, 2009*. Ed. by Ricard Gavaldà, Gábor Lugosi, Thomas Zeugmann, and Sandra Zilles. Vol. 5809. Lecture Notes in Computer Science. Springer-Verlag, 2009, pp. 171–185 (cit. on pp. 16, 20, 30, 45, 46, 95).

[14] Leonor Becerra-Bonache, Cristina Bibire, and Adrian-Horia Dediu. "Learning DFA from Corrections". In: *Proc. Workshop on Theoretical Aspects of Grammar Induction*. Ed. by Henning Fernau. WSI-2005-14. Technical Report, University of Tubingen, 2005, pp. 1–11 (cit. on pp. 17, 45, 55, 95).

[17] 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*. Vol. 4201. Lecture Notes in Computer Science. Tokyo, Japan: Springer-Verlag, 2006, pp. 281–292 (cit. on pp. 17, 30, 45, 55, 95).

[39] Edward F. Moore. "Gedanken Experiments on Sequential Machines". In: *Automata Studies*. Princeton U., 1956, pp. 129–153 (cit. on p. 48).

[45] Cristina Tîrnăucă and Timo Knuutila. *Efficient Language Learning with Correction Queries*. Technical Report 822. Turku Center for Computer Science, May 2007 (cit. on pp. 45, 55).

5 CORRECTION QUERIES

In this chapter, 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.

We discuss a new type of query, introduced by Becerra-Bonache, Bibire and Dediu [14] and Becerra-Bonache, Dediu and Tîrnăucă [17]. Since then, correction queries have been studied intensively. We mention only several papers, such as Tîrnăucă and Knuutila [45], Kinber [32], Becerra-Bonache et al. [15], and Mitrana and Tîrnăucă [37]. Although, in the introductory articles a correction gives the shortest path from a state to a final state, we show that there are possible other corrections as well.

First, we give an example of a problem where corrections perform best. After that, we discuss the theoretical aspects and limitations of corrections, concluding with several comparative results.

5.1 A PARTICULAR CASE

Example 5.1. *Take the finite acceptor* $M_8 = ([5], \{0,1\}, \{0,1\}, \tau, \lambda, 1)$ *where the transition function* τ*, and the output function* λ *are given in Table 26. In Figure 11, we see a corresponding graphical representation for acceptor* M_8*. Note that the language of this acceptor consists in a single word, which is* 111.

Table 26: The transition function τ, and the output function λ of the acceptor M_8

State	$\tau(0)$	$\tau(1)$	λ
1	2	3	0
2	2	2	0
3	2	4	0
4	2	5	0
5	2	2	1

In our experiments with L^* *and the shortest counterexample we need 3EQ and 44MQ to learn this automaton. In accordance with Proposition 4.1, if a Teacher labels this automaton with an injective labeling output function, then*

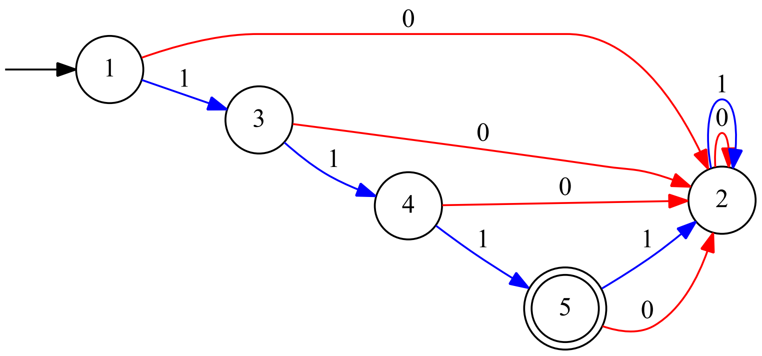

Figure 11: A graphical representation of acceptor M_8

we need 11 LQ and no counterexamples for L^* *to learn automaton* M_8*. Let us analyze what happens if the Teacher answers a query* u *with a* correction w *such that* uw *is a word in the target language. For all the states from* q_0u *until* q_0uw *there is no need for new queries, the answers can be deduced from* w*. In our example, we also see that for state number two there is no path reaching a final state. In this case, the Teacher should answer with a special symbol, # not existing in* Σ*, showing that there is no correction available. In Table 27 we give the final observation table of* L^* *learning the automaton* M_8*, assuming the Teacher answers with corrections.*

Table 27: *The final observation table of* L^* *learning* M_8*,*
$S=\{\epsilon,0,1,11,111\}$, $E=\{\epsilon\}$

$(S,E,C)_1$		E
Row	$S \cup S\Sigma$	ϵ
1	ϵ	111
2	1	11
3	11	1
4	111	ϵ
5	0	#
6	00	#
7	01	#
8	10	#
9	110	#
10	1110	#
11	1111	#

From the answer to query ϵ*, the Learner can infer the results of the queries* 1, 11, *and* 111*, adding them to the observation table together with the answer for* ϵ*. The same for* 00 *and* 01*, as these represent transitions from a state without a path to a final state. Thus, we can conclude that the Learner needs to ask only six queries, lines 1–4 count together as a single query, and the same for lines 5–7.*

5.2 LOCAL DEFINITIONS

A finite acceptor is a *password automaton* if it accepts a single word. The automaton M_8 is a password automaton accepting the word 111. 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.

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.

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

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 (1a): 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 (1b) holds, which means that for a given automaton there might exist several correction functions. The condition (1b) 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 $qw \in F$ and $\theta(q)$ is not defined yet.

Algorithm 2: Constructing a correction function

```
1 Initialize $\theta(q) \leftarrow \epsilon$ for all $q \in F$
2 Initialize $\Theta \leftarrow (q,c)$, $q \in Q, c \in \Sigma$, $\theta(q)$ not defined, $qc \in F$
3 while $\Theta$ is nonempty do
4     select $(r,w)$, a pair from $\Theta$
5     remove all pairs containing $r$ from $\Theta$
6     assign $\theta(r) \leftarrow w$
7     Add to $\Theta$ all pairs $(p,bw)$, $p \in Q$, $b \in \Sigma$, $\theta(p)$ not defined, $pb = r$
8 assign $\theta(q) \leftarrow \#$ to the states $q$ where $\theta(q)$ is not defined
```

We note that once an element r is removed from Θ there are no other states adding r to Θ, the elements in Θ are added only in increasing lexicographic order (of the access string of states).

The algorithm ends, the cycle "while" assigns each time one state (removing all the other appearances of the same state), the number of non-final states is clearly bounded by n, and in Θ, (even with repeated appearances) there cannot exist more than n different states.

Algorithm 2 follows exactly the definition of a correction function, the initialization of θ corresponds to condition (1a), the cycle "while" implements condition (1b), and the last line corresponds to condition (1c).

If in Line 4 there is more than one element available in Θ, then there exists more than one correction function for the given automaton. For a password automaton, there exists only one correction function, while for other automata there could exist several correction functions.

When constructing a correction function, Algorithm 2 could select constantly one candidate employing, 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*.

Selecting a minimal or a maximal correction there could be also several choices, depending on the input alphabet, however we believe that the order in 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 28). In Figure 12, we see a graphical representation of the automaton P167 labeled with the random correction from the example.

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).

5.3 THEORETICAL APPROACH

Lemma 5.2. *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 #.*

Proof. The proof follows easily after observing that once a state has been added to Θ, it is removed only if the state is labeled by Algorithm 2 either by a direct selection of the candidate itself, or from another path to a final state.

The proof is by induction on the length of the words leading to a final state.

Table 28: Correction functions examples

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

Induction base: For any state q, and for any $b \in \Sigma$, if qb is a final state, then q is added to Θ, according to the initialization of Θ by Algorithm 2.

Induction hypothesis: For any state q for which there exists a word w with length less than or equal to i such that qw is a final state, then q is labeled with a word other than #.

Induction step: Let us consider a state r for which there exists a character $b \in \Sigma$ such that $rb = q$. The state r is either already assigned, or is added to Θ when q is assigned. Thus, there exists a word bw of length $i+1$, such that rbw is a final state. □

Proposition 5.3. *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.*

Proof. Let us assume the contrary, that there exists a state q' appearing twice following the path from q dictated by w. As the values for the correction function are unique for each state, then necessarily the state q' that appears twice will appear the next time, following the same loop a second time, and so on, so there is no exit from the loop, which is false, as a correction value is a finite string. □

This is the reason we can bound by $|Q|-1$ the length of correction values, and hence we have a finite labeling alphabet in the definition of a correction function.

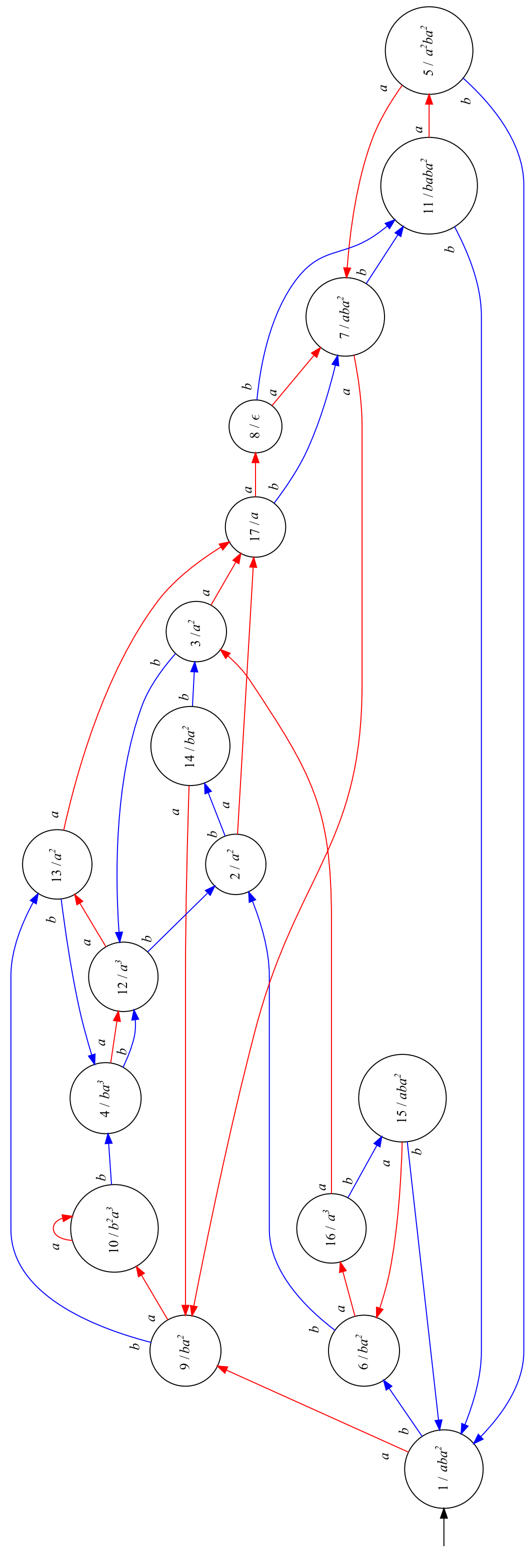

Figure 12: A graphical representation of the automaton P167 labeled with a random correction function

Proposition 5.4. *Let* $M = (Q, \Sigma, \Gamma, \tau, \lambda, q_0)$ *be a password automaton with* $|\Sigma| \geq 2$*. Then* L^* *can learn the target automaton using* $(|Q|-1)(|\Sigma|-1)+2$ *correction queries and no counterexamples are needed.*

Proof. Let us assume that the Teacher labels the states of the automaton M in such a way that the output function of the labeled automaton becomes injective. According to Proposition 4.1, the Learner needs $|Q||\Sigma|+1$ label queries and no counterexamples. Learning the automaton with correction queries, we can reduce the number of queries as follows. The answer for querying with ϵ contains the path to the only word in the language, w, where the length of $w = |Q|-2$. The Learner gets information about $|Q|-2$ other states different from the initial one. Also for the dead state, there is no need to ask $|\Sigma|$ queries, corresponding to each character of the input alphabet, the answer for all these queries being the special symbol #. The number of queries needed becomes $|Q||\Sigma|+1-|Q|+2-|\Sigma| = (|Q|-1)(|\Sigma|-1)+2$. □

Note that for $|\Sigma| = 2$ we get $|Q|+1$ correction queries needed to learn a password automaton.

5.4 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. Table 29 shows a sample from the comparative results, that is, the number of EQ and MQ/CQ for L^* with membership queries and corrections. In the Appendix, Table 37 presents the complete set of results from our experiments.

We also represent graphically the comparative results. We take the average MQ/CQ per number of states, according to our tests (Table 30), and we represent graphically their values. Figure 13 shows on the same graph the average number of MQ and CQ depending on the number of states, while Figure 14 shows the number of queries for three different corrections.

Table 29: A sample from a test set for learning DFA with L^* and correction queries

				Correction					
		L^*		Minimal		Random		Maximal	
Test	States	EQ	MQ	EQ	CQ	EQ	CQ	EQ	CQ
P1	2	1	5	1	4	1	4	1	4
P2	2	1	5	1	4	1	4	1	4
P3	2	1	5	1	4	1	4	1	4
…	…	…	…	…	…	…	…	…	…
P111	12	5	167	5	94	5	150	3	56
P112	12	5	186	3	60	3	53	1	14
P113	12	6	215	3	55	3	38	2	31

Continued on next page

				Correction					
		L*		Minimal		Random		Maximal	
Test	States	EQ	MQ	EQ	CQ	EQ	CQ	EQ	CQ
…	…	…	…	…	…	…	…	…	…
P206	20	7	314	5	182	6	257	5	172
P207	20	6	231	6	167	5	130	4	73
P208	20	6	274	4	89	4	90	3	61
P209	20	5	269	4	188	4	143	4	187

Table 30: 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

5.5 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. We note several exceptions for automata such as P132, P172 and P199. We believe that 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.

Recall that according to the proof of Theorem 3.5, the number of OQ (MQ) needed by L* is $O(\mathrm{OQ}) = O(mn^2)$. As in the particular case of Proposition

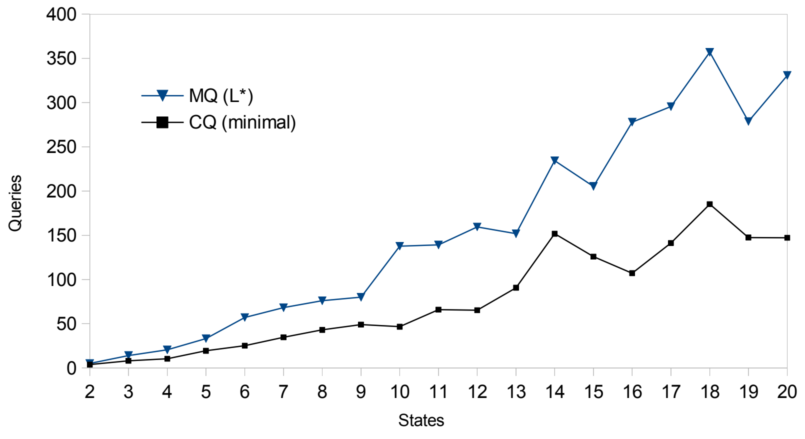

Figure 13: Comparative results MQ/CQ

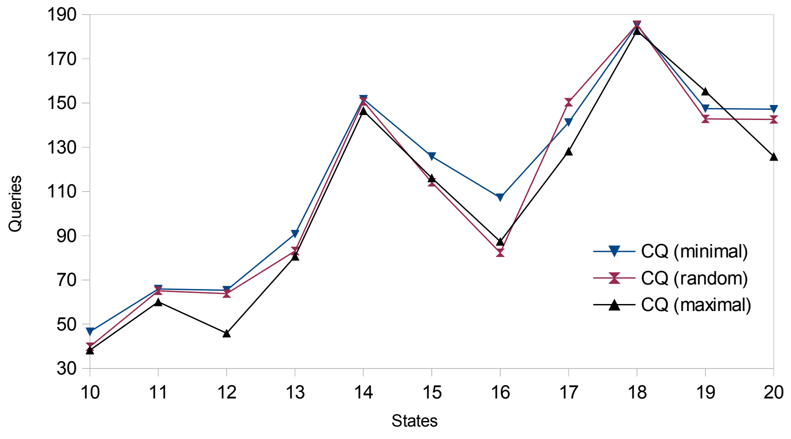

Figure 14: Comparative results, CQ minimal, random and maximal

5.4 we have $O(CQ) = O(n)$, this shows that a correction function has the potential to reduce considerably the number of queries. However, this is not the general case, for example for automata with a single non-final state, correction queries behave exactly as membership queries, that is, the worst case complexity is the same as shown for L^* and the CQ behave in the same way as the OQ (MQ).

We could also try to generalize corrections for DFA (that is for more than two symbols in the output alphabet), in fact Becerra-Bonache and Dediu [16], 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, additional information was used for marking the visited states.

CHAPTER REFERENCES

[14] Leonor Becerra-Bonache, Cristina Bibire, and Adrian-Horia Dediu. "Learning DFA from Corrections". In: *Proc. Workshop on Theoretical Aspects of Grammar Induction*. Ed. by Henning Fernau. WSI-2005-14. Technical Report, University of Tubingen, 2005, pp. 1–11 (cit. on pp. 17, 45, 55, 95).

[15] Leonor Becerra-Bonache, Colin de la Higuera, Jean-Christophe Janodet, and Frédéric Tantini. "Learning Balls of Strings from Edit Corrections". In: *Journal of Machine Learning Research* 9 (2008), pp. 1841–1870 (cit. on p. 55).

[16] Leonor Becerra-Bonache and Adrian-Horia Dediu. "Learning from a Smarter Teacher". In: *Proceedings of the 10th International Conference on Intelligent Data Engineering and Automated Learning, Burgos, Spain, September, 2009*. Ed. by Emilio Corchado and Hujun Yin. Vol. 5788. Lecture Notes in Computer Science. Springer-Verlag, 2009, pp. 200–207 (cit. on pp. 63, 95).

[17] 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*. Vol. 4201. Lecture Notes in Computer Science. Tokyo, Japan: Springer-Verlag, 2006, pp. 281–292 (cit. on pp. 17, 30, 45, 55, 95).

[32] Efim Kinber. "On Learning Regular Expressions and Patternsvia Membership and Correction Queries". In: *Proceedings of the 9th International Colloquium on Grammatical Inference: Algorithms and Applications, Saint-Malo, France, September 22–24, 2008*. Ed. by Alexander Clark, François Coste, and Laurent Miclet. Vol. 5278. Lecture Notes in Computer Science. Berlin: Springer-Verlag, 2008, pp. 125–138 (cit. on p. 55).

[37] Victor Mitrana and Cristina Tirnăucă. "New bounds for the query complexity of an algorithm that learns DFAs with correction and equivalence queries". In: *Acta Inf.* 48.1 (2011), pp. 43–50 (cit. on p. 55).

[45] Cristina Tîrnăucă and Timo Knuutila. *Efficient Language Learning with Correction Queries*. Technical Report 822. Turku Center for Computer Science, May 2007 (cit. on pp. 45, 55).

Part II.

Helpful Passive Learning of Automata

6 LEARNING TYPICAL AUTOMATA FROM RANDOM WALKS – RESET

In this chapter, we present the theoretical base for a passive learning algorithm called **Reset** that was introduced by Freund et al. [23]. We describe the original algorithm, together with several considerations about the expected number of default mistakes. This chapter ends by mentioning several inherent problems of the algorithm and several possible improvements.

Learning subsequential transducers—a broader class of transduction than the class of DFA presented here—was first presented by Oncina, Garcia and Vidal [41]. The class of subsequential functions (transducers) can be identified in the limit with positive presentation.

Well-known intractability results about passive learning show that finding a minimal automaton consistent with a set of positive and negative examples is NP-complete, Gold [26] and Angluin [4]. To deal with a possible infinite learning protocol, Freund et al. [23] use the notion of efficient learning. They presented two efficient algorithms, one based on a mechanism they called Reset and the other in a more general framework based on homing sequences. Although it does not look so passive, we consider the existence of a reset mechanism that brings the target machine into the initial state as a convention.

Practical implementations of the algorithms of Freund et al. experience real difficulties. The recommended signature depth leads to data structures that are difficult to handle, allowing experiments only for automata with a reduced number of states.

6.1 PRELIMINARIES

We follow the definitions and notations used by Feller [22] for the theory of probability. We also use several notions about graphs in this chapter.

We consider the concept of a *sample space* S and its points as given, being a set of all possible outcomes of an *experiment*. A sample space is called *discrete* if it contains only finitely many points or a countable number of points. *In what follows, we will use only discrete sample spaces*. For a discrete sample space S, an *event* A is a subset $A \subseteq S$ of possible outcomes. Given a discrete sample space S with sample points $E_1, E_2, \ldots$, we assume that with each point E_j there is associated a number called the *probability* of E_j and denoted by $P(E_j)$. It is to be non-negative and such that

$$P(E_1)+P(E_2)+\cdots=1.$$

Note that we do not exclude the possibility that a point has probability zero.

The probability $P(A)$ of any event A is the sum of probabilities of all sample points in it.

For a discrete sample space S, a *random variable* is a function $X : S \to \mathbb{R}$, that assigns a real value to each outcome in the sample space. Let X be a random variable and $x_1, x_2, \ldots$ be the values of X. The set of all sample points on which X takes the value x_j forms the event $X = x_j$ and its probability is denoted by $P\{X = x_j\}$. The function $P\{X = x_j\} = f(x_j)$, $j = 1, 2, \ldots$ is called the *probability distribution* of the random variable X. Clearly $f(x_j) \geq 0$ and $\sum\limits_j f(x_j) = 1$. The *mean* or *expected value* of X is defined by $E[X] = \sum\limits_j x_j f(x_j)$ if the series converges absolutely. If $\sum\limits_j |x_j| f(x_j)$ diverges, then we say that X has no finite expectation.

A *random walk* on a graph is a sequence of vertices $v_1 v_2, \ldots, v_t \ldots$, for $t \geq 0$, where $(v_t, v_{t+1}) \in E$ and the v_t are chosen randomly.

Given an automaton $M = (Q, \Sigma, \Gamma, \tau, \lambda, q_0)$, we can define the *underlying graph of the automaton* $G_M = (Q, E = \{(q, \tau(q, a)) \mid q \in Q, a \in \Sigma\})$. A *random walk* on an automaton refers to a random walk on the underlying graph of the automaton.

6.2 UNIFORM PROPERTIES OF AUTOMATA

Let δ be a confidence parameter, $0 < \delta \leq 1$, in the sense of Freund et al. [23], that is $1 - \delta$ represents the confidence level.

Definition 6.1 (Uniform properties of automata, Freund et al. [23])**.** *We say that* uniformly almost all automata *have property* $P_{n,\delta}$ *if for all* δ *and for any* n*-state underlying automaton graph* G_M*, if we randomly choose the output labeling of states associated to the vertices of* G_M*, then with probability at least* $1 - \delta$*, the property* $P_{n,\delta}$ *holds for the resulting automaton* M*.*

The term *uniform property*, also used by Trakhtenbrot and Barzdin' [46], refers to a property holding with high probability for any fixed underlying graph (even for the worst case) when assigning random output labels to the states associated with the vertices. From now on, δ indicates the confidence only over the random choice of labeling for the target automaton.

The following theorem exemplifies a uniform property.

Theorem 6.2 (Distinguishability degree for uniformly almost all automata, Freund et al. [23])**.** *For uniformly almost all automata with* n *states and* m *symbols in the output alphabet, every pair of inequivalent states has a shortest distinguishing string of length at most* $2\log_m(n^2/\delta)$.

The proof results from using the following theorem.

Theorem 6.3 (Trakhtenbrot and Barzdin' Theorem 5.1 [46]). *For any natural number* d *and for any finite automaton with* n *states and randomly chosen outputs from an alphabet with* m *symbols, the probability that the distinguishability degree* ρ *is greater than* d *satisfies the relation* $P(\rho > d) < n^2 m^{-d/2}$.

We give only a sketch of the proof, there are complete proofs available in both Freund et al. [23] and Barzdin' and Trakhtenbrot [46].

Let us consider a pair of different states (q_1, q_2) from an n-state automaton M. First, let us observe that for any shortest distinguishing string x for the states q_1 and q_2, any prefix $x^{(i)}$ of length i passes through at least $i+1$ states when taking x-walks from q_1 and q_2. We denote by (r_1^i, r_2^i) the pair of states $(q_1 x^{(i)}, q_2 x^{(i)})$. The set $\bigcup_{j=1,2} \{r_j^i \mid 1 \le i \le |x|\}$ contains at least $i+1$ elements as x is a shortest distinguishing string, at least one of the paths starting from q_1 and q_2 should have no loops, if both have loops, then x is not a shortest distinguishing string. The two paths might overlap, however, when assigning labels to the states of each path, there are at least $(i+1)/2$ independent events.

Let us fix d. If for any labeling of the underlying graph, the states q_1 and q_2 are distinguished by a string of length at most d, then the probability that q_1 and q_2 are distinguishable but not distinguished by any string of length d is zero. Otherwise for any labeling, there would exist a prefix $x^{(d)}$ of length d, $1 \le d < |x|$, such that x is a shortest distinguishing string. When assigning output labels to the states of the x-paths starting from q_1 and q_2, there are at least $(d+1)/2$ independent events, each of which has probability $1/m$ of not making $x^{(d)}$ a distinguishing string for q_1 and q_2. Thus, the probability that $x^{(d)}$ fails to become a distinguishing string for q_1 and q_2 is less than $m^{-(d+1)/2}$. For all pairs of states in M, the probability that a string of length d is not a distinguishing string is at most $n^2 \cdot m^{-(d+1)/2}$. If $d \ge 2\log_m(n^2/\delta)$, this probability becomes smaller than δ.

6.3 PRESENTATION OF THE ALGORITHM **RESET**

The main idea of the learning algorithm is to use distinct d-signature trees to identify the states, when d is the value indicated by Theorem 6.2. The model works for uniformly almost all automata, that is the worst-case underlying automaton graph, but with respect to a random labeling. The Teacher knows the *target automaton* and the Learner gradually constructs a *hypothesis automaton* homomorphic to the target automaton, the number of states being less than or equal to the number of states of the target automaton. We recall that in active learning algorithms, the Learner queries the Teacher, while in passive learning, the strategy is the opposite: the Teacher indicates the moves and queries the Learner about the output of the reached state. The Learner can be in one of the following situations:

a) does not know the answer—a *default mistake*,

b) gives the correct answer, or

c) gives an answer, but the answer is not the correct one—a *prediction mistake*.

While the Learner indicates the correct answers, the Teacher follows a random walk within the target automaton, communicating to the Learner the input symbol to follow and querying again about the output of the new state reached. We call a *trial* the sequence of actions and events that happen when the Teacher is in one state until moving to a new state. In case of a default mistake, both the Teacher and the Learner move back to the initial state of their automata. This action is called a *Reset*, after which the algorithm was named. The sequence of queries continues indefinitely, until eventually the Learner makes no mistakes.

Since the trial sequence is infinite, we measure the efficiency of such a protocol by the amount of computation per trial. We say that such an algorithm is *efficient* if the amount of computation in each trial is bounded by a fixed polynomial in the number n of states of the target machine and $1/\delta$.

The Teacher knows a target automaton $M = (Q, \Sigma, \Gamma, \tau, \lambda, q_0)$. We assume that the Teacher informs the Learner about the number of states of Q, and the input alphabet Σ, while the output alphabet Γ can be discovered during the learning process. At trial t, the Learner is asked to predict the output label of M at the current state $q(t) \in Q$. The current state is the initial state q_0 at trial 0 and is updated after each trial in the following manner. If no prediction mistake is made, a random input symbol b^{t+1} is presented by the Teacher, and the current state is updated to $q(t+1) = q(t)b^{t+1}$ and then the protocol proceeds with trial $t+1$. For a default mistake, the current state becomes $q(t+1) = q_0$.

From the Teacher's answers, the Learner constructs distinct d-signature trees, where d is the value indicated by Theorem 6.2. The Learner keeps a set Q'_{inc} of *incomplete states* and fills in the nodes of the signature trees associated with the states. When a signature tree is complete, the Learner moves the state associated with the complete signature tree to a set Q' of *complete states*. The Learner constructs a hypothesis automaton $M' = (Q', \Sigma, \Gamma, \tau', \lambda', q'_0)$. Based on the number of states $n = |Q|$ and the confidence parameter δ, the Learner computes d, the depth of the signature trees. The Learner keeps the current state $q'(t) \in Q' \cup Q'_{inc}$. When $q'(t) \in Q'$, the Learner predicts the output $\lambda(q'(t))$ and updates the next current state $q'(t+1)$ in the same way as the Teacher. When $q'(t)$ is incomplete, then the Learner needs to keep the current position $p'(t)$ within the d-signature tree. On entering into an incomplete state $q'(t)$, the path $p'(t)$ is the empty string ϵ. The state for the next trial remains the same, and $p'(t+1) = p'(t)b^{t+1}$. If the Learner does not know the output of the incomplete signature in position $p'(t)$, then the Learner predicts '?' (a symbol not existing in the output alphabet). The current state $q'(t+1)$ becomes the initial state q'_0 and the protocol proceeds with trial $t+1$. The Learner also predicts '?' when the path $p'(t)$ goes out of the signature tree.

We present this learning protocol (**Reset**) as Algorithm 3. In the description of the algorithm, we denote by $\Delta'(q')$ the signature tree of the state q' and by $\Delta'(q', p')$ the node of an incomplete signature tree of state q' accessed by the path p'. The sequence $\mathsf{ResetLearner}$ assigns the initial state to q', the empty string to p', and predicts '?', allowing the Teacher to reset the internal state of the target automaton as well.

Algorithm 3: Reset

```
Initialize $Q' \leftarrow \emptyset$ and $Q'_{inc} \leftarrow \{q'_0\}$
$q' \leftarrow q'_0$ and $p' \leftarrow \epsilon$
do            {Description of the protocol for each trial}
    if $q' \notin Q'_{inc}$ then
        predict $\lambda'(q')$ and on input symbol $b$, set $q' \leftarrow \tau'(q', b)$
    if $q' \in Q'_{inc}$ then
        predict $c$, the output symbol of $\Delta'(q', p')$
        if $c$ is '?' then
            label $\Delta'(q', p')$ with the output symbol
            if $\Delta'(q')$ is complete then
                if it exists $r'$ a state in $Q'$ such that $\Delta'(r') = \Delta'(q')$ then
                    Replace all transitions to $q'$ with transitions to $r'$
                else
                    $Q' \leftarrow Q' \cup \{q'\}$
                    foreach $b \in \Sigma$ do
                        Create a new state $r'_b$
                        $Q'_{inc} \leftarrow Q'_{inc} \cup \{r'_b\}$
                        Assign $\tau'(q', b) \leftarrow r'_b$
                        Partially fill in $\Delta'(r'_b)$ using $\Delta'(q')$
                $Q'_{inc} \leftarrow Q'_{inc} \setminus \{q'\}$
            ResetLearner
        else
            on input symbol $b$, set $p' \leftarrow p'b$
            if $|p'| > d$ then ResetLearner
forever
```

The Learner and the Teacher perform an unending protocol. Focusing on the Learner's answers, the main result of this learning protocol is that in appropriate conditions, there are no prediction mistakes and there exists a finite upper bound for the number of default mistakes.

All states except q_0 have almost complete signature trees when they are created (Line 19 of Algorithm 3), needing only the leaves to be completed by the learning process.

Theorem 6.4 (Complexity of **Reset**, Freund et al. [23]). *Algorithm **Reset** takes n and the confidence parameter δ as input and efficiently learns uniformly almost all n-state automata, making no prediction mistakes and having an expected number of default mistakes that is at most*

$$O((n^5/\delta^2)\log(n/\delta)),$$

where this expectation is taken over the space of all random walks.

Proof. The statement about no prediction mistakes is easy to prove if we select the depth of signatures according to Theorem 6.2.

For state q'_0, we can imagine a restrictive scenario in which the algorithm fills in, layer by layer, the nodes of the signature. Of course, we get an upper bound for the expected number of default mistakes if we count them in this way. For $0 \le i \le d$ and $q' \in Q'_{inc}$, let us denote by $X^i_{q'}$ the random variable counting the number of default mistakes encountered when completing the nodes of layer i of state q'. The expected number of default mistakes is the sum of the expected values for the variables previously defined. For q'_0, all layers should be filled in, while for all the other states, only the last layer is incomplete. Let X denote the random variable giving the total number of default mistakes.

$$E[X] = 1 + \sum_{i=1}^{d} E[X^i_{q'_0}] + \sum_{q' \in Q', q' \neq q'_0} E[X^d_{q'}]. \tag{2}$$

Computing the expected number of default mistakes made until an incomplete signature is filled in is similar to the so called "Coupon Collector's Problem" (CCP), see for example Mosteller [40]. There are N types of coupons. At each step, we are given a uniformly chosen coupon. The expected number of steps before we obtain at least one coupon of each type is given by the formula $\sum_{i=1}^{N} \frac{N}{i}$. The sum $\sum_{i=1}^{N} \frac{1}{i}$ that usually is denoted by H_N, is also known as the Nth *harmonic number*. The connection between CCP and the number of default mistakes while filling in an incomplete state is simple if we imagine the leaves of the incomplete signature as coupons and a random walk reaching one leaf as a coupon extraction. A good upper bound for $N \cdot H_N$ is $N(\ln N + 1)$. Using this approximation for the expected number of default mistakes, we get

$$E[X] \le 1 + \sum_{i=1}^{d} k^i(\ln k^i + 1) + kn\,k^d(\ln k^d + 1), \tag{3}$$

because we have at most kn states in Q'_{inc}, and for each state in Q' there are k possible transitions. Recall that $d = 2\log_m(n^2/\delta)$ (see Theorem 6.2). Thus, after some simple computations we get

$$E[X] = O((n^4/\delta^2)\log(n/\delta)) + O((n^5/\delta^2)\log(n/\delta)) = O((n^5/\delta^2)\log(n/\delta)).$$

The term $O((n^4/\delta^2)\log(n/\delta)$ corresponds to the expected number of defaults mistakes while filling in the initial state, and as we see, we can ignore it when we compare with the expected number of defaults mistakes for the rest of the states, that is $O((n^5/\delta^2)\log(n/\delta))$. This completes the proof of Theorem 6.4. □

6.4 AN EXAMPLE RUN OF **RESET**

Take the automaton $M_9 = ([4], \{0,1\}, \{+,-\}, \tau, \lambda, 1)$, where the transition function τ and the output function λ are given in Table 31. In Figure 15, we see a graphical representation for the automaton M_9.

Table 31: The transition function τ and the output function λ of the automaton M_9

State	$\tau(0)$	$\tau(1)$	λ
1	2	4	+
2	1	3	-
3	4	2	-
4	3	1	-

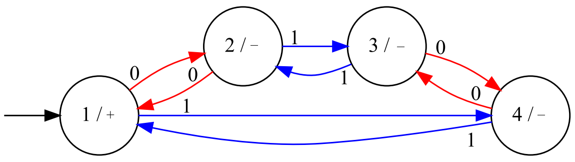

Figure 15: A graphical representation of the automaton M_9

We present a possible learning scenario for **Reset**, imagining that we observe the main variables of the algorithm after a default mistake, that is, the Learner does not know the output of a node of an incomplete state and predicts '?'. For a reasonable $\delta = 0.1$, from Theorem 6.2 there results a recommended depth for the signature trees $d \leq 2\log_2(4^2/0.1)$, that is, $d \leq 2 \cdot 7.32 = 14.64$. However, for this problem, it suffices that $d = 1$ to get all pairwise distinguishable 1-signature trees. We recall that the domain of the 1-signature trees for this problem is the set $\{\epsilon, 0, 1\}$. To simplify the notation, we represent a 1-signature tree f as a string $f(\epsilon)f(0)f(1)$: there is no danger of ambiguity, as for this problem we have only one character output symbol. For example, "+--" represents the 1-signature tree of state 1, and to not overload the notation we also drop the quotation marks. In the header of Table 32, dM, cA, and dE represent the value of the variables counting the default mistakes, correct answers, and depth exceeded, respectively. We recall that a depth exceed appears when a random walk goes out from an incomplete signature tree, and all the answers along the random walk were known. After a depth exceeded, the algorithm restarts the learning protocol from the initial state. The description of the learning process by Table 32 is a compromise between the details we give and the space used: we presented only the actions following a default mistake, as it is easy to imagine what happens for a correct answer or for a depth exceeded. If no action is given in the Note field, then one of the incomplete signatures received a useful answer from the Teacher.

Table 32: **Reset** learning the automaton M_9; the table shows the results after default mistake, before incrementing dM and before the Teacher's answer; the Note describes the action after the Teacher's answer

			M'					Random walk
dM	cA	dE	Q'	$\tau(0)$	$\tau(1)$	λ	Q'_{inc}	Note
0	0	0	1	?	?	?	1:???	ϵ
1	1	0	1	?	?	+	1:+??	0

Continued on next page

			M'					Random walk
dM	cA	dE	Q'	$\tau(0)$	$\tau(1)$	λ	Q'_{inc}	Note
2	4	1	1	?	?	+	1:+-?	1 move 1 to Q'
3	6	1	1:+--	2	3	+	2:-?? 3:-??	11
4	8	1	1:+--	2	3	+	2:-?? 3:-?+	10 move 3 to Q'
5	10	1	1:+-- 3:--+	2 4	3 5	+ -	2:-?? 4:-?? 5:+??	01
6	13	1	1:+-- 3:--+	2 4	3 5	+ -	2:-?- 4:-?? 5:+??	111
7	16	1	1:+-- 3:--+	2 4	3 5	+ -	2:-?- 4:-?? 5:+?-	110 merge 5 to 1
8	24	2	1:+-- 3:--+	2 4	3 1	+ -	2:-?- 4:-??	11101
9	26	2	1:+-- 3:--+	2 4	3 1	+ -	2:-?- 4:-?-	00 move 2 to Q'
10	29	2	1:+-- 2:-+- 3:--+	2 6 4	3 7 1	+ - -	4:-?- 6:+?? 7:-??	010
11	36	2	1:+-- 2:-+- 3:--+	2 6 4	3 7 1	+ - -	4:-?- 6:+?? 7:--?	1111011 move 7 to Q'
12	41	2	1:+-- 2:-+- 3:--+ 7:---	2 6 4 8	3 7 1 9	+ - - -	4:-?- 6:+?? 8:-?? 9:-??	11001
13	52	4	1:+-- 2:-+- 3:--+ 7:---	2 6 4 8	3 7 1 9	+ - - -	4:-?- 6:+?- 8:-?? 9:-??	100 merge 4 to 7
14	58	4	1:+-- 2:-+- 3:--+ 7:---	2 6 7 8	3 7 1 9	+ - - -	6:+?- 8:-?? 9:-??	111010
15	62	4	1:+-- 2:-+- 3:--+ 7:---	2 6 7 8	3 7 1 9	+ - - -	6:+?- 8:-?? 9:-+?	1001

Continued on next page

			M'					Random walk
dM	cA	dE	Q'	$\tau(0)$	$\tau(1)$	λ	Q'_{inc}	Note
16	72	5	1:+--	2	3	+	6:+?-	111011
			2:-+-	6	7	-	8:-?+	merge 9 to 2
			3:--+	7	1	-	9:-+?	
			7:---	8	9	-		
17	82	6	1:+--	2	3	+	6:+?-	01100
			2:-+-	6	7	-	8:-?+	merge 6 to 1
			3:--+	7	1	-		
			7:---	8	2	-		
18	135	9	1:+--	2	3	+	8:-?+	10100110001000
			2:-+-	1	7	-		merge 8 to 3
			3:--+	7	1	-		
			7:---	8	2	-		
19	135	9	1:+--	2	3	+		
			2:-+-	1	7	-		Q'_{inc} is empty
			3:--+	7	1	-		Stop
			7:---	3	2	-		

For this problem, we may stop the learning protocol when there are no more states in Q'_{inc}. The learned automaton is isomorphic to the target automaton.

6.5 RESET FOR ALMOST ALL AUTOMATA

In the framework of "uniform properties of automata," we assumed the underlying graph of the automaton to be fixed while we assign random symbols as the outputs of the states. In contrast, in this Section, we assume that also the graphs of the automata are randomly generated. The properties of automata with randomly generated underlying graphs were studied by Soviet mathematicians during the 60s, Korshunov [33] presents the results of more than a decade of work. We underline the importance of the distinguishability degree ρ of automata for learning algorithms, and in particular, for **Reset**, before introducing the framework of "almost all automata". First, we give a result about the upper and lower bound of ρ for DFA.

Theorem 6.5 (Trakhtenbrot and Barzdin' [46]). *Let M be an automaton with n states, k input symbols, and m output symbols. Then the distinguishability degree ρ of the automaton M satisfies*

$$(\log_k \log_m n) - 1 < \rho \leq n - m.$$

Proof. We follow the proof as cited in Trakhtenbrot and Barzdin' [46]. Although the theorem is for automata with output on transitions, the change for automata with output on states is straightforward.

As in the proof of Proposition 4.2, let us denote by $\mathscr{D}_M(j)$ the number of equivalence classes given by the relation "j-indistinguishable over the states of automaton M". The function $\mathscr{D}_M(j)$ (defined for j a nonnegative integer) is a nondecreasing function having the property

$$\mathscr{D}_M(0) = m < \mathscr{D}_M(1) < \ldots < \mathscr{D}_M(\rho) = \mathscr{D}_M(\rho+1) = \ldots = n.$$

Even for the slowest possible growth of $\mathscr{D}_M(j)$, that is
$\mathscr{D}_M(1) = m+1, \quad \mathscr{D}_M(2) = m+2, \ldots, \mathscr{D}_M(n-m) = m+n-m = n,$
the value of ρ can never exceed $n-m$.

For the second part of the inequality, we start counting how many different ρ-signature trees exist. Each signature tree has

$$1 + k + k^2 + \ldots + k^\rho < 2k^\rho$$

states, labeled in no more than m^{2k^ρ} ways. Thus, the number of states satisfies the inequality

$$n < m^{2k^\rho},$$

and by taking logarithms we get

$$\log_m n < 2k^\rho \text{ and } \log_k \log_m n < \log_k 2 + \rho,$$

that is, for $k \geq 2$ we have $(\log_k \log_m n) - 1 < \rho$. □

Definition 6.6 (Korshunov [33]). *Let $\mathscr{F}$ be an arbitrary class of automata with n states, m output symbols, and k input symbols. We say that almost all automata in $\mathscr{F}$ have the property B if the fraction of the automata in $\mathscr{F}$ having the property B approaches 1 as $n \to \infty$.*

The following results illustrate this definition.

Theorem 6.7 (Trakhtenbrot and Barzdin' Theorem 5.4 [46]). *There exist positive constants C_1 and C_2 such that almost all automata with k input symbols and n states have the following property: for any state q_i and natural number $d \leq C_1 \log_k n$ the set $A(q_i, d)$ of states accessible from q_i with strings no longer than d satisfies*

$$|A(q_i, d)| \geq k^{C_2 d}.$$

In other words, for almost all automata, the number of states accessible from any state with strings no longer than d is an exponential function on d, when d satisfies the condition from the theorem. The proof is based on a stochastic procedure that generates an underlying graph of an automaton.

We start from the initial state (level 0) and we generate transitions (for each letter of the input alphabet) to arbitrary states, the new states constitute the level 1. We repeat this procedure until there are no transitions going to a new level, but only to the current level, or to one of the preceding levels. The vertices of level $0, 1, \ldots, i$, for i a nonnegative integer, together with the edges issuing from them form an i-base.

The proof starts with the following Lemma.

Lemma 6.8 (Trakhtenbrot and Barzdin' [46]). *For $d \leq 1/6 \log_k n$, the probability $p(n)$ that the stochastic procedure generates an automaton graph whose d-base differs from a tree at most at one point, satisfies, for sufficiently large n,*

$$p(n) \leq n^{-8/7}.$$

We denote by $p(n, C_1, C_2)$ the probability that the stochastic procedure generates an automaton graph not satisfying the statement of the theorem, and $p_0(n, C_1, C_2)$ the probability that the stochastic procedure generates an automaton graph not satisfying the statement of the theorem for the initial state. We have $p(n, C_1, C_2) \leq n p_0(n, C_1, C_2)$. Using the result of Lemma 6.8, there exist constants C_1 and C_2 such that for sufficiently large n,

$$p_0(n, C_1, C_2) \leq \frac{1}{n^{8/7}}.$$

If the d-base of a generated graph differs from a tree at most at one point, then for $i \leq d$ we have $|A(q_0, i)| \geq 1 + (k-1) + (k-1)k + \ldots + (k-1)k^{i-1}$, and thus $|A(q_0, i)| \geq k^i$.

Corollary 6.9 (Korshunov [33]). *If $k \geq 2$, $m \geq 1$ and $n \to \infty$, then for almost all automata with n states, k input symbols and m output symbols, the degree of distinguishability ρ satisfies*

$$\log_k \log_m n - 1 < \rho \leq \log_k \log_m n + 4.$$

From the proof of Theorem 6.4, using the distinguishability degree given by Corollary 6.9, after some simple computations we can also complete the characterization of the performance of **Reset** for almost all automata.

Corollary 6.10 (Dediu and Moraga [21]). *Algorithm **Reset** can efficiently learn almost all n-state automata making no prediction mistakes and an expected number of default mistakes that is at most $O(n(\log n)\log\log n)$, where this expectation is taken over all choices of random walks.*

Proof. We rewrite equation 3 from the proof of Theorem 6.4 before making the logarithmic approximation for the harmonic number, and we get

$$\begin{aligned} E[X] &\leq 1 + \textstyle\sum_{i=1}^{d} k^i H_{k^i} + kn\, k^d H_{k^d+} \leq \quad 1 + (d + kn)\, k^d H_{k^d} \\ &\approx 1 + (d + kn) k^d (\ln k^d + 1). \end{aligned}$$

Recall that $d \leq \log_k \log_m n + 4$ (see Corollary 6.9). Thus, after some simple computations we get $E[X] = O(n(\log n)\log\log n)$. This completes the proof of Corollary 6.10.

□

6.6 REMARKS

In this chapter, we described the theoretical aspects of a passive learning algorithm called **Reset**. The original approach presented by Freund et al. [23]

within the framework of "uniformly almost all automata" was a starting point for our treatment. For practical implementations of **Reset**, we find the framework of "almost all automata" to be more appropriate since it yields more tractable sizes for the signature trees. During our experiments with the algorithm we found several other inherent difficulties in experimenting with **Reset**. The algorithm **Reset** does not say anything about prediction mistakes. The theoretical background of the algorithm gives the depth for the signature trees for (uniformly) almost all automata; however, in practical approaches, unexpected distinguishability degrees might appear. That the value of the distinguishability degree ρ is close to $\log_k \log_m n$ for almost all automata suggests the following simple solution for the case when a prediction mistake appears. We start with a depth d of the signature trees equal to $\log_k \log_m n$ and if a prediction mistake appears, we simply increment d and restart the learning algorithm.

During our experiments, we also remarked on several automata for which **Reset** was not able to learn all the states: it was cycling, and giving only correct answers, however, in the incomplete states there were always several states not reached anymore. We will try to formalize the conditions when **Reset** is not able to learn all the states.

Let M be an automaton with the set of states Q, the input alphabet Σ, and the transition function τ. If R is a subset of Q and if $\tau(q, x) \in R$ for any $q \in R$ and $x \in \Sigma$, then:

- R,
- the restriction of τ to R,
- the restriction of the output function λ to R and
- the initial state being one of the states reached with a transition from $Q \setminus R$,

form an automaton called a *subautomaton*. That is, a subautomaton absorbs all transitions: there are transitions to the states of a subautomaton from the outside, but all the transitions from inside a subautomaton remain inside the subautomaton (Liskovets [36]). For a $j \geq 1$, we call a subautomaton with j states a j-subautomaton.

Note that from the assumption that all states are accessible, it follows that there is no j-subautomaton containing the initial state such that $1 \leq j < n$.

Suppose that the following conditions hold:

i) In the target automaton there exists a subautomaton;

ii) All states from the subautomaton are learned while there exist other incomplete states;

iii) The random walk reaches the learned subautomaton.

Then the learning process "stagnates": **Reset** makes no mistakes, but does no longer learn any more states. As the goal of an on-line learning scenario is to give correct answers, not learning all the states is not considered a problem: however, avoiding stagnation during the learning process gives the Learner more possibilities in the case of unexpected situations. Let us observe that a 1-subautomaton can be obtained quite easily: for one state with all transitions as self-loops ($qb = q$ for all b in the input alphabet).

Wishing to improve the performance of the original **Reset**, we tried several helpful Teachers showing similar results as the algorithm without help. Using *extended answers*, we tried to reduce the number of trials in the sense that instead of one input character, the Teacher could communicate a whole string (possibly even with the length d given by Theorem 6.2). The Learner could also answer with a sequence, meaning the string of outputs obtained following Teacher's path. A partial answer could stand for a default mistake. As we can easily see from the proof of Theorem 6.4, this type of extended answer does not change the complexity class of the expected number of default mistakes, since the unknown outputs are located only on the leaves of states, for all states except for the initial state.

The same argument holds if we try a modified algorithm performing *reset only on depth exceeded*, that is after a default mistake there is no need to go to the initial state, but the learning could continue from the next node. Certainly, for the initial state, there are fewer trials, while for all the other states, the default mistakes and hence the number of trials would be the same.

In the next chapter, we present a helpful teacher that improves the performances of **Reset**.

CHAPTER REFERENCES

[4] Dana Angluin. "On the Complexity of Minimum Inference of Regular Sets". In: *Information and Control* 39.3 (1978), pp. 337–350 (cit. on p. 67).

[21] Adrian-Horia Dediu and Claudio Moraga. "Efficient Learning of Finite Automata from Undirected Random Walks". Submitted to Theoretical Computer Science-A, Elsevier, August, 2014 (cit. on pp. 77, 81, 86, 88, 95).

[22] William Feller. *An Introduction to Probability Theory and its Applications. Vol. I*. 3rd ed. New York: Wiley, 1968 (cit. on p. 67).

[23] Yoav Freund, Michael J. Kearns, Dana Ron, Ronitt Rubinfeld, Robert E. Schapire, and Linda Sellie. "Efficient Learning of Typical Finite Automata from Random Walks". In: *Inf. Comput.* 138.1 (1997), pp. 23–48 (cit. on pp. 17, 20, 67–69, 71, 77).

[26] E. Mark Gold. "Complexity of Automaton Identification from Given Data". In: *Information and Control* 37.3 (1978), pp. 302–320 (cit. on p. 67).

[33] Aleksej D. Korshunov. “The number of automata, boundedly determined functions and hereditary properties of automata”. In: *Kybernetika* 12.1 (1976), pp. 31–37 (cit. on pp. 75–77).

[36] Valery A. Liskovets. “Exact enumeration of acyclic deterministic automata”. In: *Discrete Appl. Math.* 154.3 (Mar. 2006), pp. 537–551. DOI: `10.1016/j.dam.2005.06.009`. URL: `http://dx.doi.org/10.1016/j.dam.2005.06.009` (cit. on p. 78).

[40] Frederick Mosteller. *Fifty Challenging Problems in Probability with Solutions*. Dover, 1965 (cit. on p. 72).

[41] José Oncina, Pedro García, and Enrique Vidal. “Learning Subsequential Transducers for Pattern Recognition Interpretation Tasks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15.5 (1993), pp. 448–458 (cit. on p. 67).

[46] Boris A. Trakhtenbrot and Ya. M. Barzdin’. *Finite automata*. Vol. 1. Fundamental Studies in Computer Science. Behavior and Synthesis, Translated from Russian by D. Louvish, English translation edited by E. Shamir and L. H. Landweber. Amsterdam: North-Holland, 1973 (cit. on pp. 24, 68, 69, 75–77).

7 LEARNING FROM UNDIRECTED RANDOM WALKS

The motivation for using a helpful Teacher for a passive on-line learning algorithm is complex, justified by the possibility of speeding up the learning process, in the context of the additional resources needed. In this chapter, we extend the study of the efficient passive learning algorithm **Reset** by allowing the algorithm to learn from undirected random walks. Thus, we can estimate an upper bound for the number of trials needed to completely learn almost all automata. A large number of experiments support the theoretical results.

Banderier and Dobrow [12] present an interesting metaphor, which, due to several similarities with the learning algorithm **Reset**, partially inspired our work. Imagine some guests around a table. One of them has a water carafe. Let p be the probability that a guest pours water into his glass; after that, he passes the carafe to his neighbor on the left or to the neighbor on the right. It is interesting to study the number of carafe moves needed before everyone receives some water. This problem is related to the cover time for random walks on graphs. In a passive learning algorithm like **Reset**, an incomplete state is the empty glass, while filling in the nodes of a signature tree is like pouring water into the glass.

This chapter represents the base of a journal article by Dediu and Moraga [21].

7.1 PRELIMINARIES

There are general definitions about probability and graphs in the preliminaries of the previous chapter. In this section, we present for self-consistency several well-known notions from linear algebra, Markov chains and an important theorem about cover times in undirected graphs.

For linear algebra, we follow the notations and the results from Schay [44]. A *scalar* is a real number.

Definition 7.1 (Vector Space, Schay [44]). *A set V is called a (real)* vector space *and its elements are called* vectors *if V is not empty and to each p, $q \in V$ and each real number c a unique sum $p+q \in V$ and a unique product $cp \in V$ are associated, satisfying the following eight axioms:*

1. $p+q=q+p$ *(commutativity of addition),*
2. $(p+q)+r=p+(q+r)$ *(associativity of addition),*
3. *There is a vector $\mathbf{0} \in V$ such that $p+\mathbf{0}=p$ for every p (existence of zero vector),*

4. *For every vector* p *there is an associated vector* $-p \in V$ *such that* $p + (-p) = \mathbf{0}$ *(existence of additive inverse),*
5. $1p = p$ *(rule of multiplication by* 1*),*
6. $a(bp) = (ab)p$ *(associativity of multiplication by scalars),*
7. $(a+b)p = ap + bp$ *(first distributive law),*
8. $a(p+q) = ap + aq$ *(second distributive law),*

where a, b *are scalars.*

Example 7.2. *For any positive integer* n*, the* vector space *of ordered real number* n*-tuples* $v = (v_i : i \in [n])$ *is denoted by* $\mathbb{R}^n$.

The basic operations defined for vectors in $\mathbb{R}^n$ are

$$(v_i : i \in [n]) + (u_i : i \in [n]) = (v_i + u_i : i \in [n])$$

and

$$c(v_i : i \in [n]) = (cv_i : i \in [n])$$

for all *vectors* $v = (v_i : i \in [n])$ and $u = (u_i : i \in [n])$ and every scalar c. The scalars $v_1, v_2, \ldots, v_n$ are called the *components* of the vector $v = (v_i : i \in [n])$, and two vectors are said to be equal if and only if their corresponding components are equal. We denote by $\mathbf{0}_n$ a vector with all n components equal to zero, or only $\mathbf{0}$ when n is clear from the context.

For all vectors $p = (p_1, p_2, \ldots, p_n)$ and $q = (q_1, q_2, \ldots, q_n)$ in $\mathbb{R}^n$, their *scalar* or *dot product* is $p \cdot q = p_1q_1 + p_2q_2 + \ldots + p_nq_n$. In $\mathbb{R}^n$, we define two vectors p and q to be *orthogonal* to each other if $p \cdot q = 0$.

The number of components of a vector is also called the *dimension* or the *size* of the vector.

A *matrix* with m rows and n columns (*size* m by n) is a rectangular array of numbers[1]. We can see a matrix as being formed by *row vectors* or *column vectors*. The individual items in a matrix are called its *elements* (or sometimes *entries*). We denote a matrix A and its entries by $A = (a_{ij} : i \in [m], j \in [n])$. We can add two matrices with the same size by adding their corresponding entries. The set $M_{m \times n}$ of all the matrices with m rows and n columns together with the operation of addition and the usual multiplication of matrices by scalars has the structure of a vector space. We define the multiplication of two matrices $A = (a_{ij} : i \in [m], j \in [n])$ and $B = (b_{ij} : i \in [n], j \in [p])$ according to the formula $AB = (c_{ij} : i \in [m], j \in [p])$, where $c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}$. We denote by $\mathbf{I}_n$ the *identity matrix*, $\mathbf{I}_n = (c_{ij} : i, j \in [n])$, where

$$c_{ij} = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{otherwise.} \end{cases}$$

When the size of the identity matrix is clear from the context we denote it only by $\mathbf{I}$. For any m by n matrix A, we have $A\mathbf{I}_n = \mathbf{I}_m A = A$. Let A and

1 In what follows, we need only real-valued vectors and matrices, although other types can be defined as well.

B be two n by n matrices. The matrix B is called *inverse* of A if and only if $AB = BA = \mathbf{I}$ and in this case, we also write $B = A^{-1}$.

We use the basic notions about vectors and matrices to introduce Markov chains. To fix notation, we work with a discrete sample space S and a probability P. A subset $A \subseteq S$ is an event and $P(A)$ is the probability of the event A.

Let $I = \{i_1, i_2, \ldots\}$ be a countable set of states[2]. We start with an initial probability distribution over I specified as a row vector

$$u = (u_i : i \in I), 0 \leq u_i \leq 1, \sum_{i \in I} u_i = 1.$$

We also have a transition matrix

$$\mathbf{P} = (p_{ij} : i, j \in I),$$

where $\mathbf{P}$ is a *stochastic matrix*, meaning that $p_{ij} \geq 0$, for all $i, j \in I$ and $\sum_{j \in I} p_{ij} = 1$. A sequence of random variables $(X_n)_{n \geq 0}$, $X_n : S \to I$ is a *Markov chain* with the initial distribution u and the transition matrix $\mathbf{P}$ if for all $n \geq 0$ and $i_0, \ldots i_{n+1} \in I$,

1. $P(X_0 = i_0) = u_{i_0}$ and
2. $P(X_{n+1} = i_{n+1} \mid X_0 = i_0, \ldots, X_n = i_n) = P(X_{n+1} = i_{n+1} \mid X_n = i_n) = p_{i_n i_{n+1}}$.

Theorem 7.3 (Grinstead and Snell [29]). *Let* $\mathbf{P}$ *be the transition matrix of a Markov chain, and let* u *be the initial distribution of the Markov chain. Then the probability that the chain is in state* s_i *after* n *steps is the* i*th entry in the vector* $u\mathbf{P}^n$.

Definition 7.4. *A Markov chain with the set of states* I *is* irreducible*, if for all states* $i, j \in I$ *there is an* n *such that* $P(X_n = j \mid X_0 = i) > 0$.

The states of a Markov chain can be classified as follows. A state $i \in I$ is *recurrent* if $P(X_n = i \text{ for some } n \geq 1 \mid X_0 = i) = 1$, which means that the probability of eventual return to i, having started from i is one. Otherwise i is called *transient*. Let $f_{ij}(n) = P(X_1 \neq j, X_2 \neq j, \ldots, X_{n-1} \neq j, X_n = j \mid X_0 = i)$ be the probability that the first visit to state j, starting from i, takes place at the nth step. Let $T_j = \min\{n \geq 1 \mid X_n = j\}$ be the *time of the first visit* to state j, with the convention that $T_j = \infty$ if this visit never occurs. The *mean recurrence time* μ_i of a state i is defined as

$$\mu_i = E(T_i \mid X_0 = i) = \begin{cases} \sum_n n f_{ii}(n) & \text{if } i \text{ is recurrent,} \\ \infty & \text{if } i \text{ is transient.} \end{cases}$$

Note that

- the mean recurrence time may be infinite even for recurrent states.
- an irreducible Markov chain has all the states either recurrent or transient.

2 We distinguish between I and the identity matrix $\mathbf{I}$.

A recurrent state i is called positive if μ_i is finite.

For a Markov chain with the transition matrix $\mathbf{P}$, a probability distribution π is called a *stationary probability distribution* if $\pi\mathbf{P} = \pi$.

Theorem 7.5 (Grimmett and Stirzaker [27]). *An irreducible Markov chain has a stationary distribution π if and only if all states are positive recurrent; in this case π is unique and is given by $\pi_i = 1/\mu_i$.*

Let us note that the existence of a stationary probability distribution π does not also mean that, starting from an arbitrary probability distribution other than π, the Markov chain will ever reach the stationary probability distribution[3].

A Markov chain can be represented by a directed graph with a vertex associated to each state and an edge labeled p_{ij} from vertex i to vertex j if $p_{ij} > 0$.

Let G be an undirected connected graph with n vertices. Let d_i denote the number of edges incident with vertex i and let Δ_{ij} denote the distance between vertices i and j, defined as the number of edges in the minimal path between i and j. To any random walk on G, we can associate a Markov chain in which the states are the vertices of G and the transition probability p_{ij} is given by

$$p_{ij} = \begin{cases} 0, & \text{if } (i,j) \text{ is not an edge,} \\ \frac{1}{d_i}, & \text{otherwise.} \end{cases}$$

We define the *hitting time* h_{ij} as the expected number of transitions until a random walk starting from i reaches j. In particular, h_{ii} is the mean recurrence time. The *commute time* between i and j is $c_{ij} = h_{ij} + h_{ji}$.

Lemma 7.6 (Aleliunas et al. [2]). *For each vertex i of a connected undirected graph $G(V,E)$, the hitting time $h_{ii} = \frac{2|E|}{d_i}$.*

Proof. To a random walk in G, we associate a Markov chain that is strongly connected and has the transition matrix $\mathbf{P}$. According to Theorem 7.5, there exists a unique stationary probability distribution π such that $\pi\mathbf{P} = \pi$. By direct substitution, it is easy to confirm that $\pi = (\frac{d_i}{2|E|} : i \in [|V|])$. The mean recurrence time of a state is the reciprocal of its stationary probability, thus $h_{ii} = \frac{2|E|}{d_i}$. □

Lemma 7.7 (Aleliunas et al. [2]). *For any edge (i,j) of a connected undirected graph $G(V,E)$, the commute time between i and j is $c_{ij} \leq 2|E|$.*

Proof. We interpret the result of Lemma 7.6 in the following way. The long-run frequency between successive traversals of an edge (i,j) from i to j is $1/2|E|$. We observe that all the transitions have the same long-run frequency. The expected number of transitions during a round trip from i to j and back is exactly $2|E|t_r$, where t_r is the expected number of transitions from i to j and back during such a round trip. We conclude the proof by observing that $t_r \leq 1$ during such a round trip. □

3 This would require an additional condition, called aperiodicity.

Lemma 7.8 (Aleliunas et al. [2]). *For a connected undirected graph* $G(V,E)$*, for all* $i,j \in V$*, the commute time between* i *and* j *is* $c_{ij} \leq 2|E|\Delta_{ij}$.

Proof. The proof results from Lemma 7.7 and induction on Δ_{ij}. □

For a graph $G=(V,E)$ and a random walk starting at $v \in V$, let C_v be the expected time to visit all the vertices of G. The *cover time* C_G is defined as $max_{v \in V} C_v$.

Theorem 7.9 (Aleliunas et al. [2]). *For a connected undirected graph* $G = (V,E)$*, if* $|V| = n$ *and* $|E| = m$*, then* $C_G \leq 2m(n-1)$.

Proof. Let $G' = (V',E')$ be a spanning tree of G. For any vertex $v \in V$ it is possible to traverse the entire tree G' starting at and returning to v, covering each edge exactly once in each direction. Let the successive vertices visited in this walk be $v = i_0, i_1, i_2, \ldots, i_{2n-2} = v$. Clearly, $C_v \leq h_{i_0 i_1} + h_{i_1 i_2} + \ldots + h_{i_{2n-3} i_{2n-2}} + h_{i_{2n-3} i_{2n-2}} = \sum_{(i,j)\in E'} c_{ij} \leq 2m(n-1)$ according to Lemma 7.7. If for any vertex $v \in V$ we have $C_v \leq 2m(n-1)$, then also $C_G \leq 2m(n-1)$. □

7.2 EXTENDING **RESET** WITH UNDIRECTED RANDOM WALKS

For the rest of this section, we fix the following notation. We denote by $M = (Q,\Sigma,\Gamma,\tau,\lambda,q_0)$ the target automaton and by $M' = (Q',\Sigma,\Gamma,\tau',\lambda',q_0')$ the Learner constructed hypothesis automaton. There are k symbols in Σ, m symbols in the output alphabet Γ, and n states in the target automaton.

We recall the learning protocol **Reset**, and we present it is as Algorithm 4. In fact, we kept the same description for our implementation and also for the next algorithms that we discuss. We note the protocol structure with a "**do–forever**" loop. However, "**return**" should be understood as jumping out of the loop.

We wish to present a simplified version of **Reset** which we can see as a Teacher's help, partially due to the extended answers we will introduce, partially by adjusting a probability coefficient p to get answers from the Teacher, can lead to a faster or slower learning process.

In the original algorithm, we allow the Teacher's answers to use a (helpful) labeling of states such that for each state q the labeling gives the output symbol $\lambda(q)$ together with the rest of the d-signature tree of state q. That is, for each state q, the Teacher constructs the set $S_q = \{(x,\lambda(qx) \mid x \in \Sigma^*, 1 \leq |x| \leq d\}$ and takes the labeling alphabet $\Lambda = \{S_q \mid q \in Q\}$ and $\ell(q) = S_q$.

Using only this labeling with **Reset** seems somehow oversimplifying the learning protocol, a collusion. To avoid this, we also introduce a probability p for the Teacher to answer with the labeled output in case of a default mistake. We denote this version of the learning algorithm by **Reset**$_p$. The new algorithm presents the same problems in case of an inappropriate selection of

Algorithm 4: Reset Type Learning Protocol

```
1  $Q' \leftarrow \emptyset$                                  // Learner's set of states
2  $Q'_{inc} \leftarrow \{q'_0\}$                        // Learner's incomplete set of states
3  $q' \leftarrow q'_0$                                   // Learner's current state
4  $q \leftarrow q_0$                                     // Teacher's current state
5  do
6      $tR \leftarrow$ Trial()                             // tR keeps trial result
7      if $Q'_{inc} = \emptyset$ then
           output : $M'$
8          return learnComplete
9      if $tR = pM$ then return pM                 // prediction mistake
10     if $tR = dM$ then                           // default mistake:  Reset
11         $q' \leftarrow q'_0$
12         $q \leftarrow q_0$
13 forever
```

d for the d-signature trees. Moreover, we can choose p in such a way as to simulate **Reset**.

Theorem 7.10 (Dediu and Moraga [21]). *Let X be the random variable giving the total number of default mistakes for* ***Reset*** *and Y_p be the random variable giving the total number of default mistakes for* ***Reset***$_p$ *when both algorithms successfully learn the same automaton M. Then there exists a probability p such that $O(E[X]) = O(E[Y_p])$.*

Proof. Assume that **Reset** uses an appropriate d for the d-signature trees, that is $d \geq \rho_M$. Recall that there are k input symbols and n states. From the proof of Corollary 6.10, we see that $O(E[X]) = O(knk^d H_{k^d})$.

For **Reset**$_p$, for every state, the expected number of default mistakes is $\frac{1}{p}$ (a shifted geometric distribution with parameter p). The expected number of default mistakes for **Reset**$_p$ is $E[Y_p] = (kn+1)/p$. Thus for $p = 1/(k^d H_{k^d})$, we get $O(E[X]) = O(E[Y_p])$. □

When implementing the algorithm **Reset**$_p$ there is only a slight simplification, the incomplete states are either empty or filled in when the Teacher answers. The rest of the algorithm works exactly in the same way as **Reset**.

Now, we present the changes to generalize the learning algorithm to work on undirected graphs. Thus, we can estimate an upper bound for the number of trials needed to learn almost all automata.

For directed graphs, the cover time has been less studied, and there are graphs with exponential cover time in the number of vertices (Cooper and Frieze [20]). For example, the graphs $G_n = (V, E)$ where $V = \{1, \ldots, n\}$ and $E = \{(i, i+1) \mid i = 1, \ldots, n-1\} \cup \{(i, 1) \mid i = 2, \ldots, n\}$ have $C_1 = \Omega(2^n)$. On the other hand, for undirected graphs there is the theorem of Aleliunas et al. [2] giving a cover time polynomial in the number of vertices (see also Theorem 7.9). Thus, modifying the original **Reset** to allow undirected random walks is strongly recommended from the theoretical point of view.

The modifications to be made in the Teacher in order to allow undirected random walks are quite simple. For the target automaton M, the Teacher constructs the extended labeled underlying graph $G'_M = (Q, E')$. For each state $q \in Q$, the Teacher also constructs a set of inverse transitions $T_q^- = \{(r,c) \mid (q,r,c) \in E', c \in \Sigma^{-1}\}$. The direct transitions are $T_q = \{(\tau(q,c),c) \mid c \in \Sigma\}$. Assuming that the current state in trial t is $q(t)$, the Teacher chooses randomly (uniformly) one element (r,c) from the set $T_q \cup T_q^-$, making the current state for the next trial $q(t+1) = r$ and communicating c to the Learner as the input symbol. Changing the current state with an input symbol c is a *positive move* if $c \in \Sigma$, and a *negative move* otherwise.

For the Learner, there are several problems related with the inverse transitions, which can be nondeterministic. For each state q', the Learner also constructs the set of possible transitions $T_{q'} \cup T_{q'}^-$ and for one input character b, the set $N_{q'}^b = \{q'' \mid (q'', b) \in T_{q'}^-\}$. In order to communicate to the Teacher that the Learner selected one particular state q'' from $N_{q'}^b$, instead of the simple output symbol $\lambda(q'')$, the Learner answers with the whole signature $\Delta'(q'')$.

In the case of an inverse transition, the Teacher can distinguish between three situations:

(i) The Learner is in the right state—a correct answer;

(ii) The Learner gives a wrong answer due to the nondeterminism of the move—the Teacher returns a new type of answer that we call a *nondeterministic mistake*;

(iii) There is simply a prediction mistake.

In Algorithm 5, we present all the changes made to **Reset**$_p$ for learning automata from undirected random walks, and we call the new algorithm **u-Reset**$_p$.

We use several variables with self-explanatory names: such as the Boolean variable $negativeMove$, which indicates a negative move. For several variables we explain as a comment their meaning, for example tA stores the Teacher's answer. We use $doDefaultMistke$ to denote the sequence of actions for a default mistake in case of a positive move: this sequence is similar to the one from **Reset** but also adding the inverse transitions. We consider the possible results of a trial as an enumeration type $\{cA, pM, dM\}$ that stands for {correct answer, prediction mistake, default mistake}, respectively. The Teacher can also return an answer ndM in case of a nondeterministic mistake.

When the Teacher is in one state, the number of all possible nondeterministic mistakes that can appear in one trial is bounded by n, thus **u-Reset**$_p$ is still an efficient algorithm.

From now on, we fix the probability $p = 1/(k^d H_{k^d})$ as we wish to compare **u-Reset**$_p$ with **Reset**.

Algorithm 5: u-Reset$_p$ Trial

```
trialType Trial()
    if negativeMove then
        $tA \leftarrow ndM$                                   // needed only if $N^b_{q'} = \emptyset$
        foreach $q'' \in N^b_{q'}$ do                          // nondeterministic candidates
            $tA \leftarrow$ndCheck($\Delta'(q'')$)
            if $tA = pM$ then return pM                       // prediction mistake
            $q' \leftarrow q''$
            if $tA = cA$ then Break;                          // correct answer
        if $tA = ndM$ then                                    // exhausted all possible moves
            $tA \leftarrow$predict('?')
            if $tA = \epsilon$ then return dM;                // default mistake
            if $\exists q'_a \in Q'$ such that $\Delta'(q'_a) = tA$ then add $\{(q'_a, b)\}$ to $T^-_{q'}$
            else
                create a new state $q''$ and set $\Delta'(q'') \leftarrow tA$
                $Q' \leftarrow Q' \cup \{q''\}$, assign $\tau'(q'', b^{-1}) \leftarrow q'$
                Add $\{(q'', b)\}$ to $T^-_{q'}$
                foreach $c \in \Sigma \setminus \{b^{-1}\}$ do
                    Create a new state $r'_c$ and set $Q'_{inc} \leftarrow Q'_{inc} \cup \{r'_c\}$
                    Assign $\tau'(q'', c) \leftarrow r'_c$ and add $\{(q'', c^{-1})\}$ to $T^-_{r'_c}$
            return dM                                         // default mistake
    if $q' \in Q'$ then
        $tA \leftarrow$ predict $\Delta'(q')$
        if $tA = pM$ then return pM                           // prediction mistake
        get an input string $b$
        if $b \in \Sigma$ then set $q' \leftarrow \tau'(q', b)$
        else $N^b_{q'} \leftarrow \{q'' \mid (q'', b) \in T^- q'\}$
        $negativeMove \leftarrow (b \notin \Sigma)$           // boolean value
        return cA                                             // correct answer
    if $q' \in Q'_{inc}$ then doDefaultMistake and return dM
end Trial
```

Theorem 7.11 (Dediu and Moraga [21]). *The algorithm **u-Reset**$_p$ efficiently learns almost all n-state automata with k input symbols, making no prediction mistakes, and:*

1. *The expected number of default mistakes is at most $n(2k-1)/p$;*
2. *The expected number of trials is in $O(n^3/p)$.*

Here, the expectations are taken over all choices of all random walks and the choice of the Teacher of answer or not to default mistakes.

Proof. For every state, the expected number of default mistakes is $\frac{1}{p}$ (a shifted geometric distribution with parameter p). Let us denote by $t_i = |T^-_{q_i} \cup T_{q_i}|$ the number of transitions for state q_i, for $1 \leq i \leq n$. We have $\sum_{i=1}^{n} t_i = 2kn$. From the initial state q_1 there are at most t_1 direct and inverse transitions. For a state q_i other than the initial state, for $2 \leq i \leq n$, there are at most $t_i - 1$ possible transitions to states producing default mistakes; one transition is to the state that allowed the creation of state q_i. We sum all the possible transitions that actually produce default mistakes, and we get $t_1 + \sum_{i=2}^{n}(t_i - 1) = 2kn - n - 1$. If we add the initial state (producing default mistakes without any transition) and use the fact that for each state there are $1/p$ expected default mistakes, then we get the result stated by the theorem.

To find an upper bound for the expected number of trials we use the result stated by Theorem 7.9. A random walk on the Learner's automaton starts in the initial state and ends with a default mistake. Thus, there is an upper bound for the expected number of trials: the product of the expected number of default mistakes by the cover time for the largest graph that the Learner can develop. There are at most $kn + 1$ states and kn transitions in this graph. Making the computations, we get an upper bound for the expected number of trials: $2(kn)^2 n(2k - 1)/p$. □

Comparing with **Reset**, there are more expected default mistakes, due to the undirected random walks; however, for **u-Reset**$_p$, we can give an upper bound for the expected number of trials. Next, we present the results of our experiments and for all the cases we tested, the number of trials was considerably lower for **u-Reset**$_p$ than for **Reset**.

7.3 COMPARATIVE RESULTS

There are several well-known methods to generate random automata, we mention the ones proposed by Bassino and Nicaud [13], or Almeida et al. [3]. For simplicity, we prefer the algorithm described by Lang [35], which we briefly present in the following.

We construct the underlying graph of a random automaton as shown in Algorithm 6. We generate random edges for each node of the underlying graph and for each character in the input alphabet. After assigning random output symbols to the states and after selecting the accessible automaton, we check whether the resulting automaton is minimal: if not, we generate a new automaton. We denote by N the initial number of nodes for the underlying graph the algorithm generates, Σ represents the input alphabet, and Γ is the output alphabet.

In practice, mostly of the generated accessible automata are already minimal.

Using the described procedure, we generate a large number of automata and we check the number of different d-signature trees for several values of d. We generate two sets of automata with the output alphabet containing two symbols: one by starting the generation routine from an initial number of nodes $N = 1000$ and the other from $N = 64000$. We also check the number of different signatures obtained for each automaton from using 4-signature trees

Algorithm 6: Generate an accessible, minimal random automaton

```
input  : $N, \Sigma, \Gamma$
1 repeat
2     foreach node $i$, $1 \leq i \leq N$ do
3         foreach character $b$ in $\Sigma$ do
4             add an edge $(i, j)$ labeled by $b$, $j$ a random number, $1 \leq j \leq N$
5         assign to $i$ a random output label from $\Gamma$
6     select $M$, the accessible automaton
7 until $M$ is minimal
output : $M$
```

and also from using 5-signature trees. In the case of automata starting from 1000 nodes, all automata except one have a number of signatures equal to the number of states when using 4-signature trees.

Table 33: Random automata obtained starting from 1000 nodes

Test Number	States	4-signature trees
1	809	809
2	779	779
3	792	792
...	...	...
33	814	814
34	808	807
35	796	796
...	...	...
99	813	813
100	806	806
Average	795.63	795.62

We note that for all automata using 5-signature trees, we get as many signatures as states. In Tables 33 and 34, we see several samples from our results.

For our tests, for each number between 140 and 300, we randomly generated ten different automata with that many states. For each automaton we tested how our algorithms worked for 10 different executions. For each number of states we recorded the average values for the number of trials and the number of default mistakes. We compared the results obtained by **Reset** and **u-Reset**$_p$. The expected number of default mistakes is concordant with the results from the experiments. The experimental average number of trials for **u-Reset**$_p$ is much lower than the upper bound from Theorem 7.11. Figure 16 compares the number of trials for **Reset** and **u-Reset**$_p$.

The very high values for the number of trials used by **Reset** at some points come from experiments that "stagnated." Due to the undirected random walk, the algorithm **u-Reset**$_p$ simply avoids this problem.

Table 34: Random automata obtained starting from 64000 nodes

Test Number	States	4-signature trees	5-signature trees
1	50923	50923	50923
2	50938	50938	50938
3	50931	50930	50931
4	51011	51008	51011
5	51213	51213	51213
...	...	...	...
97	51065	51065	51065
98	51095	51095	51095
99	50937	50936	50937
100	51231	51230	51231
Average	50997.48	50996.22	50997.48

7.4 REMARKS AND DISCUSSION

Using undirected random walks, we were able to estimate a polynomial upper bound for the number of trials needed to learn completely almost all automata in the on-line passive learning framework. A natural question arises: do we really need a more verbose Teacher for this result? Undirected random walks should use negative moves, thus Σ^{-1} should be included in the learning protocol. Could we imagine a Teacher without nondeterministic Mistakes? After all, the Teacher could answer simply with a prediction mistake. This could be a solution; however, recall that in theory, **Reset** does not use prediction mistakes at all, thus automatically a Teacher using prediction mistakes would be a more verbose one. In practice, we use prediction mistakes for **Reset** when the depth of the signature trees was not enough to discriminate between all the states. Hence, using the same mistake for a nondeterministic error that could appear quite often and a prediction error that appears rarely is in fact rather undesirable.

An interesting aspect would be to study what changes are needed to come up with an algorithm that could be called **u-Reset**, that is, undirected random walks while filling in the signature trees step by step as in **Reset**, the Teacher answering a default mistake only with the output of the state. The problem is how to deal with the nondeterministic mistakes? How could the Teacher know in case of a negative move only from a (partial) path in a d-signature tree that the Learner is in the correct state or not? What answer could the Teacher give to a default mistake, if the Learner is in an incorrect state, but the answers until the default mistake were correct?

We also considered a simplified version of negative moves, from time to time the Teacher could say “go back one step”. In this case the Teacher should keep a trace of a random walk, and for our tests this was requesting extensive memory, slowing down considerably the performances of our algorithms. Another difficulty for this scenario is also the fact that the Teacher should take

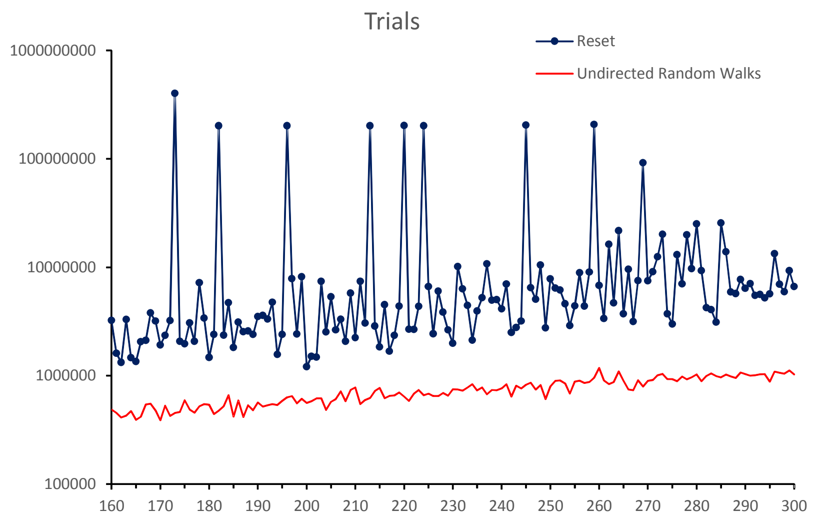

Figure 16: **Reset** and **u-Reset**$_p$—The average number of trials depending on the number of states

care to not go back more than the starting point of the random walk, with implications for the model of the new type of “random walk”. Our approach of **u-Reset**$_p$ avoided these problems.

We use a value for the probability p (in **Reset**$_p$ and **u-Reset**$_p$) such that **Reset** and **Reset**$_p$ give comparable results. The probability p depends on d, and hence on the number of states n. For experiments, we can use other values for p, creating the possibility of accelerating the learning process.

CHAPTER REFERENCES

[2] Romas Aleliunas, Richard M. Karp, Richard J. Lipton, Laszlo Lovasz, and Charles Rackoff. “Random walks, universal traversal sequences, and the complexity of maze problems”. In: *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 1979, pp. 218–223. DOI: 10.1109/SFCS.1979.34. URL: http://dx.doi.org/10.1109/SFCS.1979.34 (cit. on pp. 84–86).

[3] Marco Almeida, Nelma Moreira, and Rogério Reis. *Enumeration and Generation of Initially Connected Deterministic Finite Automata*. Tech. rep. Series: DCC-2006-07, Universidade do Porto, 2010 (cit. on p. 89).

[12] Cyril Banderier and Robert P. Dobrow. “A Generalized Cover Time for Random Walks on Graphs”. In: *FPSAC’00*. Springer-Verlag, June 2000, 113–124 (cit. on p. 81).

[13] Frédérique Bassino and Cyril Nicaud. "Enumeration and random generation of accessible automata". In: *Theoretical Computer Science* 381 (2007), pp. 86–104 (cit. on p. 89).

[20] Colin Cooper and Alan M. Frieze. "Stationary distribution and cover time of random walks on random digraphs". In: *Journal of Combinatorial Theory, Series B* 102.2 (2012), pp. 329–362 (cit. on p. 86).

[21] Adrian-Horia Dediu and Claudio Moraga. "Efficient Learning of Finite Automata from Undirected Random Walks". Submitted to Theoretical Computer Science-A, Elsevier, August, 2014 (cit. on pp. 77, 81, 86, 88, 95).

[27] Geoffrey R. Grimmett and David R. Stirzaker. *Probability and random processes*. Oxford science publications. Clarendon Press, 1985. ISBN: 9780198531852 (cit. on p. 84).

[29] Charles M. Grinstead and J. Laurie Snell. *Grinstead and Snell's Introduction to Probability*. Orange Grove Texts Plus, 2009 (cit. on p. 83).

[35] Kevin J. Lang. "Random DFA's Can Be Approximately Learned from Sparse Uniform Examples". In: *COLT*. Ed. by David Haussler. ACM, 1992, pp. 45–52 (cit. on p. 89).

[44] Géza Schay. *A Concise Introduction to Linear Algebra*. SpringerLink : Bücher. Springer, 2012. ISBN: 9780817683252 (cit. on p. 81).

8 CONCLUDING REMARKS

We investigated three major helpful conditions for the two different research directions of query learning and passive learning:

- an increased number of output symbols allows a reduced number of queries;
- some partial guidance along the learning path gives the results of several queries as a single one;
- enhancing the learning structure permits a better exploration of the learning environment.

During our research, we published the following papers presenting results related to the domain of learning automata: Becerra-Bonache, Bibire and Dediu [14], Becerra-Bonache, Dediu and Tîrnăucă [17], Angluin et al. [9], and Becerra-Bonache and Dediu [16]. These papers contains some of the ideas appearing in this thesis as well. We also submitted a journal article, Dediu and Moraga [21].

In Chapter 4, we presented a version of L^* based on a helpful labeling, enabling one to learn automata using only label queries without needing counterexamples. The lower bound of the number of labels needed for learning without counterexamples could be considerably improved, at least for some particular classes of automata.

In Chapter 5, we showed that besides the minimal corrections, there exist other types of corrections, such as maximal or random corrections. We think that further research on this subject, especially within the labeling context, could yield more results.

In Chapter 7, we allowed the random walks to go also “backwards” on transitions, by enhancing the structure of the target automaton with inverse transitions. Thus, we were able to estimate a polynomial upper bound for the number of trials needed to completely learn almost all automata in the on-line passive learning framework.

The following remarks are valid for the algorithms we discussed and we can interpret them metaphorically as well, for general learning methodologies.

- Learning is a process of gradually connecting new states with previously learned states.
- Some help could make a learning process faster.
- Too much help could constitute collusion.

- Learning with help is not necessarily working faster or giving better results.
- Helpful learning needs more resources.

The large number of experimental results presented in this thesis are in concordance with the theoretical background and could inspire future research.

PERSONAL RESULTS ABOUT LEARNING AUTOMATA

[9] Dana Angluin, Leonor Becerra-Bonache, Adrian-Horia Dediu, and Lev Reyzin. “Learning finite automata using label queries”. In: *Proceedings of the 20th International Conference on Algorithmic Learning Theory, Porto, Portugal, October 3–5, 2009*. Ed. by Ricard Gavaldà, Gábor Lugosi, Thomas Zeugmann, and Sandra Zilles. Vol. 5809. Lecture Notes in Computer Science. Springer-Verlag, 2009, pp. 171–185 (cit. on pp. 16, 20, 30, 45, 46, 95).

[14] Leonor Becerra-Bonache, Cristina Bibire, and Adrian-Horia Dediu. “Learning DFA from Corrections”. In: *Proc. Workshop on Theoretical Aspects of Grammar Induction*. Ed. by Henning Fernau. WSI-2005-14. Technical Report, University of Tubingen, 2005, pp. 1–11 (cit. on pp. 17, 45, 55, 95).

[16] Leonor Becerra-Bonache and Adrian-Horia Dediu. “Learning from a Smarter Teacher”. In: *Proceedings of the 10th International Conference on Intelligent Data Engineering and Automated Learning, Burgos, Spain, September, 2009*. Ed. by Emilio Corchado and Hujun Yin. Vol. 5788. Lecture Notes in Computer Science. Springer-Verlag, 2009, pp. 200–207 (cit. on pp. 63, 95).

[17] 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*. Vol. 4201. Lecture Notes in Computer Science. Tokyo, Japan: Springer-Verlag, 2006, pp. 281–292 (cit. on pp. 17, 30, 45, 55, 95).

[21] Adrian-Horia Dediu and Claudio Moraga. “Efficient Learning of Finite Automata from Undirected Random Walks”. Submitted to Theoretical Computer Science-A, Elsevier, August, 2014 (cit. on pp. 77, 81, 86, 88, 95).

BIBLIOGRAPHY

[1] András Ádám. *The behaviour and simplicity of finite Moore automata*. Budapest,Hungary: Akademiai Kiado, 1996 (cit. on p. 24).

[2] Romas Aleliunas, Richard M. Karp, Richard J. Lipton, Laszlo Lovasz, and Charles Rackoff. "Random walks, universal traversal sequences, and the complexity of maze problems". In: *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 1979, pp. 218–223. DOI: `10.1109/SFCS.1979.34`. URL: `http://dx.doi.org/10.1109/SFCS.1979.34` (cit. on pp. 84–86).

[3] Marco Almeida, Nelma Moreira, and Rogério Reis. *Enumeration and Generation of Initially Connected Deterministic Finite Automata*. Tech. rep. Series: DCC-2006-07, Universidade do Porto, 2010 (cit. on p. 89).

[4] Dana Angluin. "On the Complexity of Minimum Inference of Regular Sets". In: *Information and Control* 39.3 (1978), pp. 337–350 (cit. on p. 67).

[5] Dana Angluin. "A Note on the Number of Queries Needed to Identify Regular Languages". In: *Information and Control* 51.1 (1981), pp. 76–87 (cit. on p. 29).

[6] Dana Angluin. "Learning regular sets from queries and counterexamples". In: *Information and Computation* 75.2 (1987), pp. 87–106. DOI: `http://dx.doi.org/10.1016/0890-5401(87)90052-6` (cit. on pp. 12, 15, 16, 29, 30, 32, 33, 35).

[7] Dana Angluin. "Queries and Concept Learning". In: *Machine Learning* 2.4 (1988), pp. 319–342. DOI: `http://dx.doi.org/10.1023/A:1022821128753` (cit. on p. 29).

[8] Dana Angluin. "Negative Results for Equivalence Queries". In: *Machine Learning* 5.2 (1990), pp. 121–150. DOI: `http://dx.doi.org/10.1023/A:1022692615781` (cit. on p. 29).

[9] Dana Angluin, Leonor Becerra-Bonache, Adrian-Horia Dediu, and Lev Reyzin. "Learning finite automata using label queries". In: *Proceedings of the 20th International Conference on Algorithmic Learning Theory, Porto, Portugal, October 3–5, 2009*. Ed. by Ricard Gavaldà, Gábor Lugosi, Thomas Zeugmann, and Sandra Zilles. Vol. 5809. Lecture Notes in Computer Science. Springer-Verlag, 2009, pp. 171–185 (cit. on pp. 16, 20, 30, 45, 46, 95).

[10] José L. Balcázar, Josep Díaz, and Joaquim Gabarró. *Structural Complexity*. Berlin: Springer-Verlag, 1990 (cit. on p. 24).

[11] José L. Balcázar, Josep Díaz, and Ricard Gavaldà. “Algorithms for Learning Finite Automata from Queries: A Unified View”. In: *Advances in Algorithms, Languages, and Complexity*. 1997, pp. 53–72 (cit. on p. 30).

[12] Cyril Banderier and Robert P. Dobrow. “A Generalized Cover Time for Random Walks on Graphs”. In: *FPSAC’00*. Springer-Verlag, June 2000, 113–124 (cit. on p. 81).

[13] Frédérique Bassino and Cyril Nicaud. “Enumeration and random generation of accessible automata”. In: *Theoretical Computer Science* 381 (2007), pp. 86–104 (cit. on p. 89).

[14] Leonor Becerra-Bonache, Cristina Bibire, and Adrian-Horia Dediu. “Learning DFA from Corrections”. In: *Proc. Workshop on Theoretical Aspects of Grammar Induction*. Ed. by Henning Fernau. WSI-2005-14. Technical Report, University of Tubingen, 2005, pp. 1–11 (cit. on pp. 17, 45, 55, 95).

[15] Leonor Becerra-Bonache, Colin de la Higuera, Jean-Christophe Janodet, and Frédéric Tantini. “Learning Balls of Strings from Edit Corrections”. In: *Journal of Machine Learning Research* 9 (2008), pp. 1841–1870 (cit. on p. 55).

[16] Leonor Becerra-Bonache and Adrian-Horia Dediu. “Learning from a Smarter Teacher”. In: *Proceedings of the 10th International Conference on Intelligent Data Engineering and Automated Learning, Burgos, Spain, September, 2009*. Ed. by Emilio Corchado and Hujun Yin. Vol. 5788. Lecture Notes in Computer Science. Springer-Verlag, 2009, pp. 200–207 (cit. on pp. 63, 95).

[17] 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*. Vol. 4201. Lecture Notes in Computer Science. Tokyo, Japan: Springer-Verlag, 2006, pp. 281–292 (cit. on pp. 17, 30, 45, 55, 95).

[18] Jean Berstel. *Transductions and Context-free Languages*. Vol. 38. Leitfäden der angewandten Mathematik und Mechanik. Teubner, 1979 (cit. on p. 24).

[19] Gillea Brassard. “Crusade for a better notation”. In: *SIGACT News* 17.1 (June 1985), pp. 60–64. DOI: 10.1145/382250.382808. URL: http://doi.acm.org/10.1145/382250.382808 (cit. on p. 25).

[20] Colin Cooper and Alan M. Frieze. “Stationary distribution and cover time of random walks on random digraphs”. In: *Journal of Combinatorial Theory, Series B* 102.2 (2012), pp. 329–362 (cit. on p. 86).

[21] Adrian-Horia Dediu and Claudio Moraga. “Efficient Learning of Finite Automata from Undirected Random Walks”. Submitted to Theoretical Computer Science-A, Elsevier, August, 2014 (cit. on pp. 77, 81, 86, 88, 95).

[22] William Feller. *An Introduction to Probability Theory and its Applications. Vol. I*. 3rd ed. New York: Wiley, 1968 (cit. on p. 67).

[23] Yoav Freund, Michael J. Kearns, Dana Ron, Ronitt Rubinfeld, Robert E. Schapire, and Linda Sellie. "Efficient Learning of Typical Finite Automata from Random Walks". In: *Inf. Comput.* 138.1 (1997), pp. 23–48 (cit. on pp. 17, 20, 67–69, 71, 77).

[24] William I. Gasarch, Efim B. Kinber, Mark G. Pleszkoch, Carl H. Smith, and Thomas Zeugmann. "Learning via Queries with Teams and Anomalies". In: *Fundam. Inf.* 23.1 (Jan. 1995), pp. 67–89. URL: `http://dl.acm.org/citation.cfm?id=2383376.2383378` (cit. on p. 29).

[25] E. Mark Gold. "Language identification in the limit". In: *Information and Control* 10.5 (1967), pp. 447–474 (cit. on p. 15).

[26] E. Mark Gold. "Complexity of Automaton Identification from Given Data". In: *Information and Control* 37.3 (1978), pp. 302–320 (cit. on p. 67).

[27] Geoffrey R. Grimmett and David R. Stirzaker. *Probability and random processes*. Oxford science publications. Clarendon Press, 1985. ISBN: 9780198531852 (cit. on p. 84).

[28] Olga Grinchtein and Martin Leucker. "Learning finite-state machines from inexperienced teachers". In: *Proceedings of the 8th International Conference on Grammatical Inference: Algorithms and Applications*. Vol. 4201. Lecture Notes in Computer Science. Tokyo, Japan: Springer-Verlag, 2006, pp. 344–345. DOI: `10.1007/11872436_30`. URL: `http://dx.doi.org/10.1007/11872436_30` (cit. on p. 29).

[29] Charles M. Grinstead and J. Laurie Snell. *Grinstead and Snell's Introduction to Probability*. Orange Grove Texts Plus, 2009 (cit. on p. 83).

[30] Yuri Gurevich. "What does $O(n)$ mean". In: *SIGACT News* 17.4 (Mar. 1986), pp. 61–63. DOI: `10.1145/8307.8311`. URL: `http://doi.acm.org/10.1145/8307.8311` (cit. on p. 25).

[31] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979 (cit. on pp. 20, 22).

[32] Efim Kinber. "On Learning Regular Expressions and Patterns via Membership and Correction Queries". In: *Proceedings of the 9th International Colloquium on Grammatical Inference: Algorithms and Applications, Saint-Malo, France, September 22–24, 2008*. Ed. by Alexander Clark, François Coste, and Laurent Miclet. Vol. 5278. Lecture Notes in Computer Science. Berlin: Springer-Verlag, 2008, pp. 125–138 (cit. on p. 55).

[33] Aleksej D. Korshunov. "The number of automata, boundedly determined functions and hereditary properties of automata". In: *Kybernetika* 12.1 (1976), pp. 31–37 (cit. on pp. 75–77).

[34] George Lakoff and Mark Johnson. “Conceptual Metaphor in Everyday Language”. In: *The Journal of Philosophy* 77.8 (1980), pp. 453–486. DOI: 10.2307/2025464. URL: http://dx.doi.org/10.2307/2025464 (cit. on p. 15).

[35] Kevin J. Lang. “Random DFA’s Can Be Approximately Learned from Sparse Uniform Examples”. In: *COLT*. Ed. by David Haussler. ACM, 1992, pp. 45–52 (cit. on p. 89).

[36] Valery A. Liskovets. “Exact enumeration of acyclic deterministic automata”. In: *Discrete Appl. Math.* 154.3 (Mar. 2006), pp. 537–551. DOI: 10.1016/j.dam.2005.06.009. URL: http://dx.doi.org/10.1016/j.dam.2005.06.009 (cit. on p. 78).

[37] Victor Mitrana and Cristina Tirnăucă. “New bounds for the query complexity of an algorithm that learns DFAs with correction and equivalence queries”. In: *Acta Inf.* 48.1 (2011), pp. 43–50 (cit. on p. 55).

[38] Mehryar Mohri. “Finite-state transducers in language and speech processing”. In: *Comput. Linguist.* 23.2 (June 1997), pp. 269–311 (cit. on p. 24).

[39] Edward F. Moore. “Gedanken Experiments on Sequential Machines”. In: *Automata Studies*. Princeton U., 1956, pp. 129–153 (cit. on p. 48).

[40] Frederick Mosteller. *Fifty Challenging Problems in Probability with Solutions*. Dover, 1965 (cit. on p. 72).

[41] José Oncina, Pedro García, and Enrique Vidal. “Learning Subsequential Transducers for Pattern Recognition Interpretation Tasks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15.5 (1993), pp. 448–458 (cit. on p. 67).

[42] Rajesh Parekh and Vasant Honavar. “On the Relationship between Models for Learning in Helpful Environments”. In: *Proceedings of the 5th International Colloquium on Grammatical Inference: Algorithms and Applications, Lisbon, Portugal, September 11–13, 2000*. Ed. by Arlindo L. Oliveira. Vol. 1891. Lecture Notes in Computer Science. Springer-Verlag, 2000, pp. 207–220 (cit. on p. 16).

[43] Ronald L. Rivest and Robert E. Schapire. “Inference of Finite Automata Using Homing Sequences”. In: *Information and Computation* 103.2 (Apr. 1993), pp. 299–347. DOI: 10.1006/inco.1993.1021. URL: http://dx.doi.org/10.1006/inco.1993.1021 (cit. on p. 30).

[44] Géza Schay. *A Concise Introduction to Linear Algebra*. SpringerLink : Bücher. Springer, 2012. ISBN: 9780817683252 (cit. on p. 81).

[45] Cristina Tîrnăucă and Timo Knuutila. *Efficient Language Learning with Correction Queries*. Technical Report 822. Turku Center for Computer Science, May 2007 (cit. on pp. 45, 55).

[46] Boris A. Trakhtenbrot and Ya. M. Barzdin'. *Finite automata*. Vol. 1. Fundamental Studies in Computer Science. Behavior and Synthesis, Translated from Russian by D. Louvish, English translation edited by E. Shamir and L. H. Landweber. Amsterdam: North-Holland, 1973 (cit. on pp. 24, 68, 69, 75–77).

[47] Leslie Gabriel Valiant. "A theory of the learnable". In: *Communications of the ACM* 27.11 (1984), pp. 1134–1142. DOI: http://doi.acm.org/10.1145/1968.1972 (cit. on pp. 15, 29).

[48] Juan Miguel Vilar. "Query learning of subsequential transducers". In: *Proceedings of the Third International Colloquium on Grammatical Interference (ICGI-96): Learning Syntax from Sentences, Montpellier, France, September*. Ed. by Laurent Miclet and Colin de la Higuera. Vol. 1147. Lecture Notes in Computer Science. Springer-Verlag, 1996, pp. 72–83 (cit. on p. 30).

9 APPENDIX

9.1 HELPFUL LABELS, NUMERICAL RESULTS

Table 35: Comparative results, learning DFA with L^* and L_1^* with helpful labels

Test		L^*		L_1^* with helpful labels		
Id	States	EQ	MQ	LQ	Labels	$\|\lambda_{\ell_1}\|$
P1	2	1	5	11	1	2
P2	2	1	5	11	1	2
P3	2	1	5	11	1	2
P4	2	1	5	11	1	2
P5	2	1	5	11	1	2
P6	2	1	5	11	1	2
P7	2	1	5	11	1	2
P8	2	1	5	11	1	2
P9	2	1	5	11	1	2
P10	2	1	5	11	1	2
P11	2	1	5	11	1	2
P12	3	2	14	15	1	2
P13	3	2	14	15	1	2
P14	3	2	14	15	1	2
P15	3	2	11	15	1	2
P16	3	2	14	15	1	2
P17	3	2	14	15	1	2
P18	3	2	11	15	1	2
P19	3	2	14	15	1	2
P20	3	2	11	15	1	2
P21	3	2	14	15	1	2
P22	3	2	14	15	1	2
P23	4	3	27	19	2	3
P24	4	3	27	19	2	3
P25	4	2	19	19	2	3
P26	4	2	14	19	1	2
P27	4	2	14	19	1	2
P28	4	2	19	19	2	3
P29	4	2	19	19	2	3
P30	4	2	17	19	1	2
P31	4	3	27	19	2	3
P32	4	2	23	19	2	3
P33	4	2	19	19	1	2

Continued on next page

Test		L^*		L_1^* with helpful labels		
Id	States	EQ	MQ	LQ	Labels	$\lvert\lambda_{\ell_1}\rvert$
P34	5	3	39	23	2	3
P35	5	2	23	23	2	3
P36	5	3	31	23	2	3
P37	5	2	23	23	1	2
P38	5	2	23	23	2	3
P39	5	3	27	23	1	2
P40	5	4	49	23	2	3
P41	5	4	54	23	3	4
P42	5	2	34	23	2	4
P43	5	3	39	23	2	4
P44	5	2	23	23	2	3
P45	6	3	44	27	2	3
P46	6	3	35	27	1	2
P47	6	3	71	27	4	5
P48	6	4	44	27	2	3
P49	6	4	77	27	4	5
P50	6	3	65	27	4	5
P51	6	4	65	27	3	4
P52	6	4	71	27	4	5
P53	6	4	71	27	4	5
P54	6	3	31	27	2	3
P55	6	3	34	27	2	4
P56	7	4	71	31	2	4
P57	7	3	49	31	2	3
P58	7	3	39	31	3	5
P59	7	5	89	31	3	4
P60	7	4	76	31	2	3
P61	7	4	90	31	3	4
P62	7	4	65	31	2	4
P63	7	3	53	31	3	4
P64	7	4	83	31	2	4
P65	7	3	59	31	3	5
P66	7	4	69	31	4	5
P67	8	4	90	35	4	6
P68	8	5	83	35	3	4
P69	8	3	44	35	2	4
P70	8	4	59	35	2	3
P71	8	4	65	35	2	4
P72	8	5	111	35	4	5
P73	8	4	77	35	2	4
P74	8	4	77	35	2	4
P75	8	3	69	35	4	5
P76	8	4	90	35	3	4
P77	8	4	71	35	3	4

Continued on next page

Test		L^*		L^*_1 with helpful labels		
Id	States	EQ	MQ	LQ	Labels	$\|\lambda_{\ell_1}\|$
P78	9	5	125	39	4	5
P79	9	4	83	39	2	4
P80	9	4	97	39	3	4
P81	9	4	59	39	3	5
P82	9	4	77	39	3	5
P83	9	4	54	39	2	4
P84	9	4	95	39	5	7
P85	9	4	104	39	3	5
P86	9	4	54	39	2	4
P87	9	3	49	39	2	4
P88	9	4	77	39	3	4
P89	10	2	76	43	8	9
P90	10	5	132	43	4	6
P91	10	6	269	43	8	9
P92	10	4	90	43	3	5
P93	10	4	89	43	3	4
P94	10	6	143	43	3	6
P95	10	5	179	43	6	7
P96	10	6	170	43	6	7
P97	10	5	101	43	4	6
P98	10	4	97	43	3	5
P99	10	6	229	43	8	9
P100	11	5	229	47	9	10
P101	11	5	161	47	5	6
P102	11	5	135	47	3	5
P103	11	6	170	47	6	7
P104	11	5	104	47	3	5
P105	11	3	111	47	3	5
P106	11	4	83	47	3	5
P107	11	4	127	47	3	6
P108	11	4	97	47	3	5
P109	11	6	229	47	8	9
P110	11	3	77	47	2	3
P111	12	5	159	51	3	5
P112	12	5	186	51	10	11
P113	12	6	197	51	3	6
P114	12	5	161	51	6	8
P115	12	7	242	51	9	10
P116	12	6	199	51	4	8
P117	12	5	119	51	4	6
P118	12	5	170	51	3	6
P119	12	4	118	51	6	8
P120	12	4	90	51	3	5
P121	12	6	143	51	6	8

Continued on next page

Test		L^*		L_1^* with helpful labels		
Id	States	EQ	MQ	LQ	Labels	$\lvert\lambda_{\ell_1}\rvert$
P122	13	6	167	55	3	5
P123	13	5	132	55	4	6
P124	13	5	101	55	5	8
P125	13	4	101	55	3	6
P126	13	5	159	55	4	6
P127	13	7	197	55	3	6
P128	13	5	151	55	4	6
P129	13	5	167	55	5	8
P130	13	5	189	55	5	7
P131	13	6	274	55	11	12
P132	13	4	113	55	5	7
P133	14	4	208	59	10	12
P134	14	8	615	59	12	13
P135	14	4	135	59	7	9
P136	14	6	209	59	5	7
P137	14	5	151	59	4	6
P138	14	7	407	59	12	13
P139	14	5	189	59	4	8
P140	14	5	161	59	5	7
P141	14	5	181	59	3	6
P142	14	6	167	59	4	6
P143	14	6	199	59	8	10
P144	15	3	95	63	3	6
P145	15	6	219	63	7	9
P146	15	8	319	63	7	9
P147	15	4	101	63	3	6
P148	15	6	389	63	11	12
P149	15	6	233	63	7	9
P150	15	5	249	63	8	11
P151	15	4	151	63	5	8
P152	15	6	219	63	7	9
P153	15	5	167	63	3	6
P154	15	6	183	63	4	6
P155	16	7	359	67	7	10
P156	16	6	274	67	9	11
P157	16	5	179	67	4	7
P158	16	7	285	67	5	8
P159	16	6	259	67	7	10
P160	16	5	215	67	4	7
P161	16	9	463	67	11	13
P162	16	8	347	67	10	11
P163	16	6	175	67	3	6
P164	16	6	206	67	4	7
P165	16	6	224	67	6	8

Continued on next page

Test		L^*		L_1^* with helpful labels		
Id	States	EQ	MQ	LQ	Labels	$\|\lambda_{\ell_1}\|$
P166	17	5	191	71	9	12
P167	17	7	506	71	15	16
P168	17	6	285	71	8	10
P169	17	7	318	71	10	12
P170	17	8	340	71	8	11
P171	17	6	285	71	8	11
P172	17	5	239	71	7	9
P173	17	5	274	71	11	14
P174	17	6	206	71	6	9
P175	17	5	183	71	3	6
P176	17	6	233	71	4	7
P177	18	10	615	75	13	15
P178	18	6	335	75	10	12
P179	18	8	461	75	13	15
P180	18	7	340	75	9	12
P181	18	6	215	75	5	8
P182	18	7	269	75	7	11
P183	18	6	239	75	5	9
P184	18	8	601	75	12	14
P185	18	5	239	75	9	11
P186	18	7	323	75	4	7
P187	18	6	233	75	7	9
P188	19	6	405	79	12	13
P189	19	5	241	79	6	8
P190	19	6	278	79	6	10
P191	19	7	314	79	6	9
P192	19	4	175	79	4	7
P193	19	6	319	79	9	11
P194	19	6	371	79	14	15
P195	19	7	260	79	4	8
P196	19	5	188	79	5	7
P197	19	6	296	79	8	11
P198	19	7	233	79	4	7
P199	20	5	242	83	4	8
P200	20	8	419	83	16	17
P201	20	7	329	83	5	8
P202	20	5	279	83	5	8
P203	20	5	233	83	5	8
P204	20	9	873	83	16	17
P205	20	6	215	83	5	9
P206	20	7	323	83	8	10
P207	20	6	231	83	4	7
P208	20	6	274	83	8	11
P209	20	5	251	83	7	10

9.2 CORRECTION QUERIES, NUMERICAL RESULTS

Table 37: Comparative results for learning finite acceptors with L* and correction queries

				Correction					
		L*		Minimal		Random		Maximal	
Test	States	EQ	MQ	EQ	CQ	EQ	CQ	EQ	CQ
P1	2	1	5	1	4	1	4	1	4
P2	2	1	5	1	4	1	4	1	4
P3	2	1	5	1	4	1	4	1	4
P4	2	1	5	1	3	1	3	1	3
P5	2	1	5	1	4	1	4	1	4
P6	2	1	5	1	4	1	4	1	4
P7	2	1	5	1	4	1	4	1	4
P8	2	1	5	1	3	1	3	1	3
P9	2	1	5	1	4	1	4	1	4
P10	2	1	5	1	4	1	4	1	4
P11	2	1	5	1	4	1	4	1	4
P12	3	2	14	2	13	2	13	2	13
P13	3	2	14	1	5	1	5	1	5
P14	3	2	14	1	5	1	5	1	5
P15	3	2	11	1	5	1	5	1	5
P16	3	2	14	2	12	2	12	2	13
P17	3	2	17	1	5	1	5	1	5
P18	3	2	11	1	5	1	5	1	5
P19	3	2	17	2	14	1	5	1	5
P20	3	2	11	1	5	1	5	1	5
P21	3	2	14	1	5	1	5	1	5
P22	3	2	17	2	15	2	15	2	15
P23	4	3	27	2	12	2	12	2	12
P24	4	3	27	1	5	1	5	1	5
P25	4	2	19	2	18	2	18	2	18
P26	4	2	14	2	11	2	11	2	11
P27	4	2	14	2	11	2	11	2	11
P28	4	2	19	1	6	1	6	1	6
P29	4	2	19	1	6	1	6	1	6
P30	4	2	17	2	15	2	14	2	14
P31	4	3	27	1	5	1	5	1	5
P32	4	2	23	2	19	2	21	2	19
P33	4	2	19	1	6	1	6	1	6
P34	5	3	39	2	14	2	14	2	14
P35	5	2	23	2	21	2	21	2	21
P36	5	3	31	2	13	2	14	2	13
P37	5	2	23	2	22	2	19	2	22
P38	5	2	23	2	14	2	14	2	14

Continued on next page

				Correction					
		L*		Minimal		Random		Maximal	
Test	States	EQ	MQ	EQ	CQ	EQ	CQ	EQ	CQ
P39	5	3	27	2	16	2	14	2	12
P40	5	4	49	2	13	2	15	2	16
P41	5	4	54	4	52	4	52	4	52
P42	5	2	34	2	31	2	31	2	23
P43	5	3	39	2	10	2	10	2	10
P44	5	2	23	1	7	1	7	1	7
P45	6	3	44	3	25	2	15	2	15
P46	6	3	31	3	22	3	22	3	22
P47	6	3	71	1	8	1	8	1	8
P48	6	4	44	3	28	3	28	3	28
P49	6	4	77	1	8	1	8	1	8
P50	6	3	71	3	69	3	69	3	69
P51	6	4	71	3	36	3	29	2	16
P52	6	4	83	2	24	2	24	2	24
P53	6	4	71	2	16	1	8	1	8
P54	6	3	31	2	20	2	20	2	20
P55	6	3	34	2	21	2	17	3	26
P56	7	4	71	2	20	2	20	2	20
P57	7	3	49	3	32	3	32	2	18
P58	7	3	39	3	37	3	37	3	37
P59	7	5	89	3	38	4	63	3	38
P60	7	4	90	3	40	2	20	1	9
P61	7	4	83	4	77	4	77	4	77
P62	7	4	65	3	23	2	15	2	14
P63	7	3	53	2	14	2	14	2	16
P64	7	4	83	4	56	4	63	4	56
P65	7	3	59	3	35	3	33	3	34
P66	7	4	69	1	9	1	9	1	9
P67	8	4	90	3	80	4	77	3	80
P68	8	5	83	4	45	4	45	3	38
P69	8	3	44	3	35	3	30	3	32
P70	8	4	59	3	28	3	41	3	38
P71	8	4	65	3	30	3	30	2	24
P72	8	5	111	4	98	4	98	4	90
P73	8	4	77	3	45	3	34	4	51
P74	8	4	77	2	16	2	18	2	21
P75	8	3	69	2	21	2	24	2	21
P76	8	4	90	2	18	2	20	2	20
P77	8	4	71	4	57	3	28	3	28
P78	9	5	125	5	116	5	120	5	120
P79	9	4	83	2	26	2	21	2	26
P80	9	3	76	3	46	3	53	3	31
P81	9	4	59	4	58	3	43	4	55

Continued on next page

				Correction					
		L*		Minimal		Random		Maximal	
Test	States	EQ	MQ	EQ	CQ	EQ	CQ	EQ	CQ
P82	9	4	83	3	40	4	63	3	40
P83	9	4	77	3	46	3	50	3	47
P84	9	4	95	2	20	2	22	2	18
P85	9	5	97	5	83	2	44	2	44
P86	9	4	59	2	37	3	41	3	41
P87	9	3	49	3	36	4	50	3	53
P88	9	4	77	3	31	4	49	4	49
P89	10	2	76	2	75	2	74	2	75
P90	10	3	65	3	60	2	30	3	45
P91	10	6	269	1	12	2	23	2	23
P92	10	4	90	3	34	2	32	2	25
P93	10	4	95	4	55	3	36	2	27
P94	10	6	143	3	59	3	55	3	55
P95	10	5	189	2	32	2	28	1	12
P96	10	6	161	3	35	2	24	2	24
P97	10	5	101	5	76	4	64	4	72
P98	10	4	97	2	36	3	43	4	51
P99	10	6	229	2	38	2	29	1	12
P100	11	5	229	2	30	3	32	2	22
P101	11	5	161	6	124	6	97	6	97
P102	11	5	135	2	25	3	38	2	24
P103	11	6	170	3	42	3	58	3	37
P104	11	5	104	4	62	4	62	4	62
P105	11	4	111	3	61	3	55	3	50
P106	11	4	83	3	32	3	36	2	23
P107	11	5	143	4	66	2	34	3	44
P108	11	5	118	3	39	3	51	3	39
P109	11	5	199	5	176	5	195	5	195
P110	11	3	77	4	68	4	58	4	68
P111	12	5	167	5	94	5	150	3	56
P112	12	5	186	3	60	3	53	1	14
P113	12	6	215	3	55	3	38	2	31
P114	12	5	161	3	49	5	79	2	29
P115	12	6	197	4	63	3	49	2	24
P116	12	5	188	5	178	5	127	5	168
P117	12	5	119	3	51	3	51	4	43
P118	12	5	170	3	43	3	44	3	39
P119	12	4	118	4	43	3	38	3	37
P120	12	4	90	3	44	3	44	2	38
P121	12	6	143	3	39	2	29	2	26
P122	13	6	175	6	147	4	58	4	58
P123	13	5	132	4	67	3	40	3	51
P124	13	5	101	5	96	5	113	5	96

Continued on next page

				Correction					
		L*		Minimal		Random		Maximal	
Test	States	EQ	MQ	EQ	CQ	EQ	CQ	EQ	CQ
P125	13	4	101	4	74	5	91	3	50
P126	13	5	159	5	115	5	99	5	117
P127	13	5	118	3	64	4	64	4	59
P128	13	5	151	5	123	5	117	5	123
P129	13	5	167	4	96	5	123	5	151
P130	13	5	189	3	47	4	50	2	26
P131	13	6	263	3	47	3	37	2	33
P132	13	4	113	5	122	5	122	5	123
P133	14	4	208	4	204	4	204	4	204
P134	14	8	615	8	607	8	607	8	607
P135	14	5	143	2	38	2	32	2	31
P136	14	6	209	5	106	5	95	5	111
P137	14	5	143	2	49	4	69	3	45
P138	14	6	337	6	334	6	334	6	334
P139	14	5	199	3	55	3	47	3	59
P140	14	5	170	4	71	3	57	2	31
P141	14	5	188	4	82	5	109	4	81
P142	14	6	167	4	81	3	59	4	67
P143	14	6	199	3	42	3	44	3	41
P144	15	3	95	3	81	3	81	3	67
P145	15	6	219	6	182	4	130	4	144
P146	15	8	319	8	293	8	294	8	304
P147	15	4	101	4	77	4	76	4	87
P148	15	6	389	2	34	2	33	3	39
P149	15	6	224	3	48	4	74	4	83
P150	15	5	259	5	248	5	232	5	221
P151	15	4	167	4	153	4	99	5	116
P152	15	4	135	5	88	4	48	3	44
P153	15	5	167	4	80	5	115	3	71
P154	15	6	183	5	100	4	73	5	101
P155	16	7	383	6	204	4	70	3	66
P156	16	7	285	4	62	3	49	2	36
P157	16	5	167	3	74	3	68	4	82
P158	16	5	209	5	186	4	121	5	193
P159	16	6	259	4	61	2	38	2	36
P160	16	5	242	3	86	4	83	4	76
P161	16	9	509	2	32	2	33	2	33
P162	16	9	415	4	53	3	41	3	61
P163	16	6	167	4	99	4	77	5	108
P164	16	6	206	7	175	7	182	6	150
P165	16	4	215	4	147	4	143	5	120
P166	17	6	247	5	182	5	182	6	204
P167	17	7	506	3	58	2	42	2	36

Continued on next page

				Correction					
		L*		Minimal		Random		Maximal	
Test	States	EQ	MQ	EQ	CQ	EQ	CQ	EQ	CQ
P168	17	8	383	8	313	8	368	8	308
P169	17	8	351	3	46	3	44	2	35
P170	17	7	285	4	61	4	55	3	39
P171	17	7	351	7	329	5	239	5	229
P172	17	5	239	5	224	7	320	5	224
P173	17	5	274	3	54	2	44	3	44
P174	17	6	206	4	105	4	120	5	109
P175	17	5	175	4	103	5	134	4	95
P176	17	6	233	4	78	5	106	4	87
P177	18	10	629	10	608	10	608	10	608
P178	18	7	359	7	307	7	307	7	354
P179	18	7	402	3	64	2	32	2	35
P180	18	7	362	7	351	7	356	7	320
P181	18	6	231	6	120	5	110	5	135
P182	18	7	278	4	101	4	102	3	75
P183	18	6	269	3	51	4	70	3	67
P184	18	8	601	2	39	2	38	2	36
P185	18	5	239	5	214	5	217	5	221
P186	18	7	323	4	107	5	113	4	81
P187	18	6	233	4	75	4	87	4	77
P188	19	6	405	2	51	2	56	2	53
P189	19	5	241	4	79	2	52	4	80
P190	19	7	287	6	230	6	244	6	244
P191	19	6	263	6	212	6	207	6	207
P192	19	4	207	4	103	4	100	5	136
P193	19	6	319	6	268	6	304	6	283
P194	19	6	371	2	36	2	38	2	56
P195	19	7	260	5	128	3	63	5	123
P196	19	5	183	6	157	6	148	6	135
P197	19	6	296	6	243	6	249	7	276
P198	19	7	233	5	115	5	110	5	115
P199	20	5	242	6	251	5	208	7	273
P200	20	7	467	3	69	2	49	3	54
P201	20	5	224	6	230	6	199	4	178
P202	20	5	279	2	56	4	105	3	67
P203	20	6	269	5	157	4	116	5	153
P204	20	9	854	2	46	3	61	2	33
P205	20	6	215	5	184	6	210	4	133
P206	20	7	314	5	182	6	257	5	172
P207	20	6	231	6	167	5	130	4	73
P208	20	6	274	4	89	4	90	3	61
P209	20	5	269	4	188	4	143	4	187

9.3 RESET AND U-RESET$_p$, NUMERICAL RESULTS

For each number of states we generated 10 automata and for each automaton we made 10 experiments, thus every average value is the result over 100 experiments.

Table 38: The dependency on the number of states of the average number of trials and default mistakes for algorithms **Reset** and **u-Reset**$_p$

		Trials		Default Mistakes	
States	ρ	**Reset**	**u-Reset**$_p$	**Reset**	**u-Reset**$_p$
160	3.8	3225759.15	487963.22	15331.06	22840.39
161	3.7	1605061.02	454013.80	14358.36	21545.99
162	3.6	1321297.02	411024.46	13346.27	20034.94
163	3.6	3293322.77	428253.02	13435.02	20268.22
164	3.8	1461496.77	471152.77	15684.01	23462.52
165	3.5	1345003.01	392643.10	12556.01	18726.91
166	3.6	2049730.59	418400.13	13723.58	20446.87
167	3.9	2116047.56	541333.77	17035.24	25562.22
168	4.0	3792841.39	552036.72	18301.44	27228.84
169	3.7	3175057.03	476707.76	15095.73	22543.97
170	3.5	1921302.26	387371.61	12931.41	19475.92
171	3.9	2352590.86	527618.10	17481.51	25979.40
172	3.6	3210228.59	425197.24	14235.90	21435.21
173	3.6	401392485.40	454031.76	14089.28	21478.39
174	3.6	2065874.08	461259.44	14369.92	21471.06
175	4.0	1959710.88	592896.99	18992.04	28320.75
176	3.7	3064743.49	488223.67	15714.59	23489.01
177	3.6	2074627.97	454308.68	14622.12	21879.88
178	3.8	7184280.94	521656.14	16985.41	25532.73
179	3.8	3386415.87	547816.68	17137.22	25725.44
180	3.8	1473318.24	538866.74	17238.59	25833.59
181	3.5	2389289.77	441675.51	13802.08	20682.98
182	3.6	201909982.40	474731.41	14658.23	22425.80
183	3.8	2354465.51	524287.67	17493.37	26329.53
184	4.0	4691493.22	660954.89	20031.51	29803.02
185	3.4	1816213.73	419674.49	12857.01	19354.70
186	3.9	3103902.56	591045.79	19005.39	28381.73
187	3.4	2536153.38	416177.78	13009.01	19395.30
188	3.7	2574357.97	534494.64	16767.38	25188.98
189	3.6	2396955.95	480822.94	15649.62	23457.02
190	3.8	3508076.34	564830.62	18150.62	27399.38
191	3.7	3596472.32	519425.97	17015.18	25396.01

Continued on next page

*Average results for **Reset** and **u-Reset**$_p$*

		Trials		Default Mistakes	
States	ρ	**Reset**	**u-Reset**$_p$	**Reset**	**u-Reset**$_p$
192	3.7	3314483.28	533425.96	17142.46	25613.90
193	3.7	4741768.76	547656.81	17205.70	25635.93
194	3.7	1564162.76	535735.11	17234.36	25981.80
195	3.8	2394527.00	585075.99	18602.09	27977.70
196	3.8	201988965.30	631818.34	17561.76	28082.32
197	3.9	7827275.09	646010.87	20141.15	30128.26
198	3.7	2415728.35	553601.53	17675.17	26191.45
199	3.8	8152643.70	610000.43	19056.07	28590.87
200	3.7	1206167.92	560573.57	17780.49	26739.66
201	3.7	1505935.58	582977.46	17883.28	26850.18
202	3.8	1478967.02	618458.07	19264.33	28970.92
203	3.8	7383185.44	617197.96	19366.87	29255.97
204	3.5	2529460.38	483504.43	15529.15	23137.05
205	3.7	5341931.63	573653.13	18315.74	27343.84
206	3.8	2638599.40	609310.26	19723.35	29508.98
207	4.0	3306409.49	716714.48	22444.90	33722.40
208	3.7	2069852.29	583305.43	18534.77	27735.40
209	3.9	5755170.75	739197.65	23100.78	34739.80
210	3.9	2239504.30	777141.93	23291.84	34899.89
211	3.6	7395634.57	548158.53	17427.44	26063.62
212	3.7	3045121.06	595935.21	18855.58	28304.37
213	3.7	201470197.10	623553.58	18273.67	28467.50
214	3.9	2862977.23	724427.61	21810.17	32701.19
215	4.0	1831839.03	770838.30	23401.26	35092.65
216	3.7	4502989.05	618943.98	19316.70	28796.70
217	3.8	1673145.17	649936.27	20772.15	30923.51
218	3.8	2350931.85	655690.99	20766.78	31080.77
219	3.8	4379143.54	698168.38	20980.93	31404.67
220	3.7	202656760.40	640473.53	19424.02	29460.48
221	3.6	2673817.91	585965.05	18240.58	27187.80
222	3.8	2661508.76	682171.53	21166.29	31781.26
223	3.9	4353756.87	736009.84	22731.25	34024.13
224	3.7	201313685.50	659505.93	18430.06	29997.73
225	3.8	6607527.05	683412.22	21428.29	32121.84
226	3.7	2432660.94	647071.57	20111.93	30051.32
227	3.7	6027616.40	648305.46	20203.62	30401.93
228	3.7	3831202.08	693565.20	20252.34	30459.42
229	3.7	2641495.62	653268.93	20369.20	30595.12
230	3.7	1979366.54	748928.81	22462.41	33853.34
231	3.8	10119778.89	743711.14	22039.70	33064.33
232	3.8	6309661.94	726493.94	22110.11	33139.73
233	3.9	4442435.51	777454.64	23812.64	35702.47
234	4.0	2125158.36	831799.44	25416.97	37864.89

Continued on next page

*Average results for **Reset** and **u-Reset**$_p$*

		Trials		Default Mistakes	
States	ρ	**Reset**	**u-Reset**$_p$	**Reset**	**u-Reset**$_p$
235	3.8	3939086.74	730288.96	22448.07	33522.70
236	3.9	5235290.78	779274.89	24108.64	35669.26
237	3.7	10705245.22	673228.07	21115.60	31422.05
238	3.8	4959422.56	737159.37	22688.54	34160.59
239	3.8	5011833.83	734648.27	22813.72	34328.70
240	3.8	4136124.65	762349.38	22972.01	34468.56
241	3.9	6988195.27	831516.38	24522.12	36740.41
242	3.6	2503846.07	641125.16	19938.02	29873.42
243	3.9	2782307.47	809132.63	24776.52	37181.63
244	3.8	3188193.45	763582.48	23340.39	34892.33
245	3.8	204427892.50	824156.80	23231.22	35228.22
246	4.0	6466309.44	860047.72	26713.72	39906.47
247	3.8	5050543.59	743212.92	23610.74	35148.94
248	3.9	10458876.40	820570.06	25321.77	37728.88
249	3.5	2752615.80	607026.22	18948.31	28444.94
250	3.8	7793425.52	798595.71	23872.31	35539.22
251	4.0	6394471.88	892542.04	27332.68	40685.47
252	4.0	6163770.86	904043.99	27350.35	40955.60
253	3.9	4584967.12	848258.20	25803.82	38758.13
254	3.5	2893218.72	682788.62	19334.11	28862.73
255	4.0	4384857.30	883275.29	27659.10	41405.07
256	3.9	8902215.33	899222.89	26123.13	39464.32
257	3.9	4371363.64	856472.59	26262.85	39268.26
258	3.9	9000753.24	872662.36	26242.52	39384.46
259	4.0	207096867.50	957595.81	25978.33	41862.53
260	4.2	6771981.22	1177340.70	36104.47	54318.82
261	3.9	3350722.73	901239.62	26615.43	40106.78
262	3.8	16249242.60	835733.23	25037.06	37442.07
263	3.9	4670861.18	873823.86	26839.60	40235.52
264	4.1	21736861.78	1096740.35	32659.65	48716.76
265	3.9	3716401.86	900157.77	27008.13	40325.92
266	3.6	9567927.59	746364.52	22032.92	32637.22
267	3.6	3143397.61	735077.34	21990.23	33005.77
268	3.9	7495350.59	907493.55	27310.01	41030.56
269	3.7	91576813.38	792793.48	23958.28	36042.14
270	3.9	7477106.92	895554.41	27549.48	41261.96
271	3.9	9059258.99	909997.19	27631.29	41469.24
272	4.0	12433357.23	1011286.33	29526.93	44389.67
273	4.0	20130170.43	1040333.91	29606.49	44626.60
274	3.9	3711213.21	928706.94	27917.63	41887.75
275	3.9	2991109.71	930405.04	28023.25	41937.78
276	3.8	12981076.22	891823.76	26330.04	39501.74
277	3.9	7005703.73	981264.37	28267.40	42324.25

Continued on next page

*Average results for **Reset** and **u-Reset**$_p$*

		Trials		Default Mistakes	
States	ρ	**Reset**	**u-Reset**$_p$	**Reset**	**u-Reset**$_p$
278	3.9	19866795.68	925357.70	28278.96	42703.94
279	3.9	9712444.75	966917.91	28451.30	42569.73
280	4.0	25132533.75	1030403.93	30368.59	45701.01
281	3.8	9260768.50	889242.28	26795.16	40312.14
282	4.0	4220604.68	996894.01	30559.84	45442.04
283	3.9	4040927.00	1048490.49	31308.11	46846.16
284	3.9	3105644.34	989351.14	28940.63	43335.87
285	3.9	25589375.80	967967.05	29077.30	43501.16
286	4.0	13864264.92	1021932.62	31138.67	46468.35
287	3.9	5900294.80	983114.16	29350.38	43742.79
288	3.8	5698089.81	953231.50	27507.94	41136.14
289	4.0	7684203.74	1070355.74	31286.10	47018.39
290	4.0	6362297.94	1035057.67	31428.88	46983.61
291	3.9	7054778.17	998842.31	29718.07	44337.66
292	3.9	5480572.96	1010681.43	29773.14	44557.92
293	3.9	5588150.31	1027038.82	29881.74	44540.02
294	3.9	5215099.23	1033923.85	29976.29	44974.16
295	3.7	5663352.82	883022.90	26187.58	39236.25
296	4.0	13355695.92	1092415.92	32159.12	48254.75
297	4.0	6938859.30	1064523.63	32254.93	47907.97
298	3.9	5920746.17	1046166.53	30338.17	45470.85
299	4.0	9289344.15	1117959.71	32419.48	48620.51
300	3.9	6622445.18	1030060.91	30580.33	46116.52

10 INDEX