UNIVERSITÀ
DEGLI STUDI
DI PALERMO

NDS
NETWORKING AND DISTRIBUTED SYSTEMS
RESEARCH GROUP

GI-learning: an optimized framework for grammatical inference.

Article

Accepted version

P. Cottone, M. Ortolani, G. Pergola

In Proceedings of the 17th International Conference on Computer Systems and Technologies

GI-learning: an optimized framework for grammatical inference

Pietro Cottone, Marco Ortolani, Gabriele Pergola [1]

Abstract:
In this paper, we present a new open-source software library, GI-learning, for grammatical inference. The rise of new application scenarios in recent years has required optimized methods to address knowledge extraction from huge amounts of data and to model highly complex systems. Our library implements the main state-of-the-art algorithms in the grammatical inference field (RPNI, EDSM, L), redesigned through the OpenMP library for a parallel execution that drastically decreases execution times. To our best knowledge, it is also the first comprehensive library including a noise tolerance learning algorithm, such as Blue*, that significantly broadens the range of the potential application scenarios for grammar models. The modular design of our C++ library makes it an efficient and extensible framework for the design of further novel algorithms.*
Key words: *Grammatical inference; parallel algorithms; software library.*

INTRODUCTION

Knowledge extraction and data modeling are increasingly gaining importance in the larger field of machine learning [22, 26, 3]; in particular, a broad literature has been produced regarding the specific task of *grammatical inference* [15], an inductive process aiming at generating models able to describe the structures behind data. In this context, samples are considered strings of a language whose syntax needs to be guessed, hence models are provided in the form of *automata*.

The fundamental paradigm employed to infer the relations among data is due to Gold [13], and is the so-called *Identification in the limit*, which theoretically ensures the capability of an algorithm to produce a grammar from examples. In particular, regular languages and finite state automata (FSA) have been investigated in depth, and two main strategies have historically emerged, namely: offline strategies, including algorithms based on *state merging* [9], and online strategies, typically relying on *active learning* [6]. More recently, the field is being revitalized thanks to new application scenarios where a structural approach was proved particularly well suited to extract meaningful information [20, 23]. However, those scenarios gave rise to additional difficulties, related to dealing with embedded noise in data, that require proper learning algorithms in order to allow for noise tolerance in the inference process [2]; this is the case for instance of Blue* [27, 4] that exploits statistical techniques, such as hypothesis testing, to guide the selection of a final, more reliable, model.

To date, many of the mentioned algorithms have been implemented in publicly available frameworks; however, such tools usually focus on just one learning strategy [24], they do not include state-of-the-art algorithms [8], or are written in a prototypal language (e.g. Matlab) [1] not suitable for real application scenarios characterized by a high volume of data.

This work documents our proposal for a new, comprehensive library – *GI-learning* – which provides computationally efficient implementation for the main representative algorithms of the above-mentioned categories: RPNI, EDSM and Blue* for the state-merging approach and L^* [5] for the active learning. The library is written in C++ and aims to provide

[1]Work partially supported by the Italian Ministry of Education, University and Research on the "PAC" call funding the "SERVIFY (SERVIce FirstlY)" project, ref. PON03PE_00132_1, CUP: B72F14000300005

an optimized set of tools to cope with grammatical inference in the presence of complex systems and considerable data flows. In order to maximize performances, a parallelizable flow of execution was designed on top of the OpenMP library [11], so as to dynamically decrease the execution time by creating independent threads; at the same time, a modular structure eases the extension and maintenance process of the library.

In the rest of the paper, after a concise review of the main algorithms for regular grammar inference, a comparison of the existing libraries follows, focusing on their features with regard to efficiency, software architecture, and support to developers. The strategies and tools adopted in the proposed library – *GI-learning* – are described with particular regard to parallel execution, extensibility and the provided methods to manage automata. Finally, experimental assessments are reported to compare the performance of our library to existing ones in the scenarios of offline and online learning.

PRELIMINARIES: APPROACHES TO GRAMMATICAL INFERENCE

Before describing our library, we give here an overview of the theoretic foundations and properties of the main algorithms for grammatical inference.

State merging algorithms. Such methods rely on a specifically defined equivalence relation for selecting pairs of DFA states potentially candidate for merging [10]. They start by building a canonical automaton (the so-called PTA – Prefix Tree Acceptor), exactly matching all positive examples I_+, and proceed by repeatedly applying the equivalence criterion; the resulting automata have fewer states, hence they identify languages with a higher cardinality and effectively implement a generalization process; negative examples I_- are exploited to prevent overgeneralization. Such methods globally perform a search in the space of all possible sequences of state merging. In order to speed up such process, the *Red-Blue* framework [19] was proposed in the Abbadingo competition; it introduces a bland criterion of locality to limit the merges to assess at each step, thus reducing the overall computations.

Regular Positive and Negative Inference (RPNI) [21] is the first historically relevant state-merging algorithm, thanks to its ability to identify a language in the limit from a characteristic set for a target grammar. Its main limitation is due to performing an exhaustive depth-first search, which is bound to be computationally intensive; a natural evolution thus consisted in the introduction of a heuristic to drive the search, giving rise to the state-of-the-art algorithm known as *Evidence Driven State Merging (EDSM)* [19].

EDSM is an *exact learning algorithm* because the built Deterministic Finite state Automaton (DFA) models positive and negative examples without errors or approximations. However, in a real application scenario, the target automaton is rarely available, so checking whether the training set comprises a *characteristic* set of examples is not a viable option; moreover, the presence of noise in data may significantly compromise the final DFA, which becomes needlessly larger and would match the original, noise-free, data with low accuracy rate. An effective strategy would be to evaluate the relevance of examples by considering their frequency information, i.e. a distribution over the example set. To this end, the *Blue** algorithm was proposed [27]. Its key insight is a statistical distinction between relevant and irrelevant information, which is treated as noise. To process such data and deal with noise there are different approaches, frequently denoted through the "features selection" terminology [18]. Blue* borrows the so-called *wrapper* strategy, and deals with noise during the inference process, also preserving a deterministic final models. The structure of the Blue* algorithm resembles that of EDSM, where the only difference is the heuristic function which computes statistical scores.

Active learning A totally different learning paradigm is *Active learning*, defined according to the model known as "learning from queries" by Angluin [6]. It is a model based on the interaction between a *learner* and a *teacher*; instead of performing a state-merging proce-

dure, it defines learning by iterative queries, and *state-splitting* operations which generate hypotheses leading towards the minimum automaton.

The teacher is a device aware of the target automaton, and able to correctly answer some kind of queries asked by the learner. Unlike other inductive strategies, the role of the teacher allows to model a customized way to supply examples to the learner in the most effective way in order to infer the target automaton. The main algorithm in this context is called L^* designed by Angluin, a baseline algorithm for many others different methods[5]. It ensures termination of the process by producing a minimum consistent DFA recognizing the target regular language. Furthermore, denoted by n the number of states of the target DFA and by m an upper bound for the length of any counterexample provided by the teacher, the time complexity of the whole inference process is polynomially bounded in n and m. The teacher belongs to the class of *Minimally Adeguate Teachers (MAT)*, able to answer two kinds of queries, namely *membership* and *equivalence* queries. In the former type, learner proposes a sample to the teacher, which confirms or denies if the sample belongs to target language. In the latter type of query, the learner proposes a representation of the hypothesis language, typically by its automaton. The teacher answers positively if the hypothesis is equivalent to the target language; on the contrary, it returns a counterexample, i.e. a sample belonging to the symmetric difference set[2] of hypothesis and target language.

A marginal note regards which teachers are actually available for the inference task. Some instances are direct implementations of some known deterministic machines. Furthermore, different works pointed out several "teachers" other than automata. For instance, even though a trained recurrent neural network is just a "black box", it may still profitably be used as a teacher [12]. Another example comes from hardware testing, which might be carried out by feeding a sequence of operations to the device under test and using the output for membership queries [14]. On the learner side, some interesting works attempt to improve the learning task by advanced management of counterexamples [25].

OVERVIEW OF EXISTING LIBRARIES

Learning algorithms for grammar inference have been implemented into many software libraries, typically in order to provide a benchmark for performance and efficacy of specific tests. They are often prototypes, specifically designed for artificial use cases, where the involved data were synthesized to reproduce experimental scenarios about the inference process, or to assess modifications to existing algorithms. Due to lack of a modular and reusable structure, they are not immediately portable into different frameworks; for the same reason, also optimizing their execution is often challenging.

Among the few attempts to design carefully structured libraries, *LearnLib* [24], *libalf* [8] and *GI toolbox* [1] are worth mentioning (see Table 1).

Up until 2008, *LearnLib* was the only publicly available library. It was created at the University of Dortmund, was written in Java, and included the main algorithms for the active learning approach, such as L^* [5], and some of its subsequent extensions [25]. It allows inference of two types of automata, namely DFA and Mealy machines, i.e. finite state machines which produce an output string while processing the input [16].

With the aim to get better performance, a new library – *libalf* – was implemented, adopting the C++ language. It is open source, and unlike *LearnLib* contains algorithms for both state merging and active learning. In *libalf* parlance, it implements "passive offline" algorithms, such as Biermann's [7] and RPNI [21] and "active online" algorithms, such as L^*,

[2]The *symmetric difference* between two languages L_1 and L_2, denoted by $L_1 \oplus L_2$, is defined as:

$$L_1 \oplus L_2 = L_1 \backslash L_2 \cup L_2 \backslash L_1 = \{x \in \Sigma^* : (x \in L_1 \wedge x \notin L_2) \vee (x \notin L_1 \wedge x \in L_2)\}$$

	GI-learning	LearnLib	libalf	GI toolbox
Algorithm strategies	Offline/Online	Online	Offline/Online	Offline
Noise tolerance	Yes	No	No	No
Parallel impl.	Yes	No	No	No
Efficiency	Very High	High	Medium	Low
Progr. language	C++	Java	C++, Java (JNI)	Matlab
Hierarchical structure	Yes	Yes	Yes	No
Code documentation	Yes	Yes	No	Yes

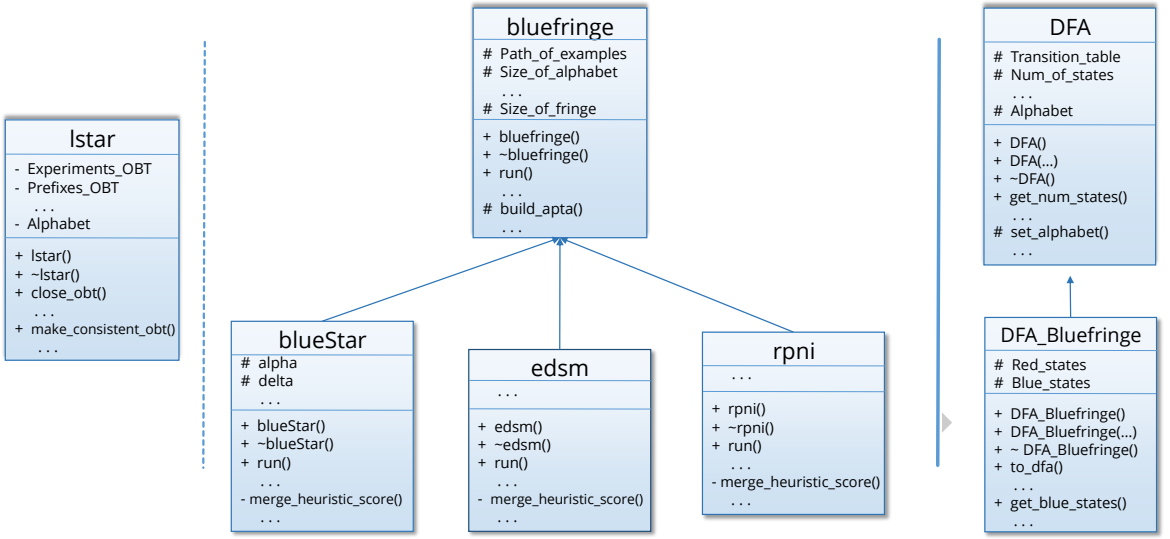
Table 1: A qualitative comparison of the main learning libraries with respect to *GI-learning*.

Rivest/Schapire [25], and others. Although *libalf* did outperform the original implementation of *LearnLib*, the latter in its latest versions obtained a significant efficiency increase, thus reversing the situation.

An open-source implementation of grammatical inference algorithms is also provided in Matlab, as the *GI toolbox*. Unlike the previous libraries, it focuses mainly on the state-merging approach, giving room to algorithms such as RPNI, EDSM [19], and algorithms for learning *k-testable* languages [15], which correspond to a specific subset of regular languages. Grammatical inference algorithms are implemented by employing matrices as the fundamental data structure to encode DFA, due to the optimization of Matlab processes with regard to these structures. The implemented methods follow the definition provided by de la Higuera [15], entailing the use of the *Red-Blue* framework. One of the advantages deriving from the Matlab framework is the ease to exploit its other toolboxes, such as for instance the one for genetic algorithms; however, despite the fact that the prototypical features of this programming language allow for a direct software implementation, performance optimization may be significantly hindered in terms of computations and memory management.

Unfortunately, for today's standards, the execution speed of both *LearnLib* and *libalf* is also again inadequate for most real application scenarios. For example, *LearnLib* was widely employed to improve software quality by a model-based testing approach, where a model of existing software (e.g. a finite state machine) is adopted to reduce software failure rates and to check the correctness of actual behavior. However, actually generating a software model still remains open research, and grammar inference has proven to be a promising strategy both through offline and online strategies. In the former case, learning starts from logs of software executions, whereas in the latter case, an active learning approach is specified, where a "learner" interacts with the software-under-learning (SUL), that plays the role of a "teacher"; a software model results from this cooperation, in terms of finite state machine, and it can be used to conduct software analyses. Several real-world software tools have been successfully modeled over *LearnLib* [17]; however, for complex systems, the time cost is still often prohibitive. For instance, a model for a control software of a printer required an inference process executing longer than 11 days, hence such scenarios may still benefit even from small performance improvements.

GI-learning, the library presented in this paper, was designed aiming to maximize the performance of the implemented inference algorithms by using deep parallelism, and efficient execution, thanks to the full control offered by C++ language and its libraries. In order to support as many application scenarios as possible, inference algorithms both for offline and online approaches have been implemented, as will be detailed in the following.



(a) Learning algorithms. (b) DFA classes.

Figure 1: Hierarchical structure of proposed library classes.

THE *GI-LEARNING* FRAMEWORK

GI-learning is an open-source and actively maintained library[3]. It was designed with efficiency, flexibility, and extensibility in mind, and it has been widely tested. Besides allowing inference of finite state automata, and particularly DFA, it also includes many support methods to make automata management easier.

In order to obtain satisfactory efficiency, *GI-learning* allows creating representations of huge amounts of data in the form of grammars. Therefore, inferential processes are optimized through dynamic parallelism involving simple and efficient data structures.

Dynamic parallelism is achieved by means of OpenMP [11], a multi-platform library for parallel applications in shared-memory systems. It provides support for parallelism in software, ensuring robustness, efficiency and compatibility across many platforms; furthermore, it dynamically sets the number of threads to spawn depending on the specific architecture, thus making the proposed *GI-learning* library suitable for cases when multiple cores are available, transparently to the users. More specifically, we built all the algorithms belonging to the *Red-Blue* framework (RPNI, EDSM and Blue*) on top of it; in this context, several threads are created to evaluate more than one merge at a time, and to carry on the pair-wise computation of the heuristic scores of red and blue states in parallel.

Our modular and hierarchical structure makes full use of the inheritance mechanism so as to allow easy maintenance, and straightforward addition of further algorithms, by extending the existing classes and data structures. A scheme of our hierarchy of classes both for data structures and algorithms is reported in Figure 1. The hierarchy of state-merging algorithms, in particular, derives from a `bluefringe` class that implements all the shared properties and methods, subsequently specialized by RPNI, EDSM and Blue*.

From the user's point of view, the library is highly flexible, allowing to set up the execution of an algorithms with just a few lines of code. It exploits encapsulation, by adopting simple interfaces to specify all the required parameters; for instance, the input of state-merging algorithms is a text file, composed by an alphabet of a customized number of symbols, and the strings over this alphabet, labeled as positive or negative examples. Additionally, both kinds of examples can have an associated weight (e.g. their frequency) for use by algorithms such as Blue*; this feature facilitates future extensions of the library to algorithms inferring

[3]Software available at: `https://github.com/piecot/GI-learning`

	RPNI	EDSM	L^*
GI-learning	✓	✓	✓
LearnLib			✓
Libalf	✓		✓
GI toolbox	✓	✓	

Table 2: Experiments with regard of supplied algorithms by libraries.

probabilistic automata [15]. Similarly, the L* algorithm may receive an input file with the DFA target known by the "teacher".

The fundamental class in the library is the `DFA` class, whose instances of course represent deterministic finite state automata and whose attributes and methods are initialized regardless of the learning algorithm.

Among them, the `Table-filling` method is worth mentioning as it detects the equivalent states of an automaton, and is used as a first step of the DFA minimization methods. Another method of the `DFA` class addresses the "equivalence query", that checks whether two automata are equivalent or not; this method is also based on the table-filling algorithm. If two automata are identified as not equivalent, a counter-example string can be generated as a proof of the difference among them; indeed, such string is accepted by one automaton and rejected by the other one. The "membership query" method returns the membership of a string belonging to the language identified by an automaton.

Finally, a relevant features of the `DFA` class is the graphic representation produced by a method encoding a DFA in *DOT*, a simple and effective language for graph description. The produced representations are acceptable both for visual inspection or for further automatic parsing by software; for instance, DFA can be readily visualized through *Graphviz*, an open-source tool managing DOT-encoded graphs.

EXPERIMENTAL ASSESSMENT

We devised several experiments in order to assess the impact of our optimized and parallel implementation of the learning algorithms in terms of execution speed.

The specific differences in the algorithms as implemented by the various software libraries were taken into account when measuring and comparing the time spent to infer DFAs. Table 2 summarizes the comparisons between the available algorithms. In the context of state-merging approaches, GI toolbox is worth considering, to date, as it is the only library providing a stable implementation of the state-of-the-art algorithm, EDSM. On the other side, the active learning approach is addressed by all the considered libraries, with the exception of GI toolbox, allowing a wide comparability of different implementations.

Tests were conducted on a Dell PowerEdge T620 server with 4 quad-cores Intel Xeon, 128 GB RAM, running CentOS 6 x64; the C++ compiler is g++ 4.8.2 with -O2 optimization parameter set; Java was version 1.8 of the Oracle VM. Times were computed from data submission to the production of the final DFA, in a 24h time window; an experiment not terminating within the allotted time is assumed to have failed. DFAs were randomly generated using the *LearnLib* generator, with a variable size ranging from 20 up to 100 states, and for three alphabet sizes (2, 10 and 100).

As regards state-merging algorithms, a graphical representation of the performance of *GI-learning* and *libalf* for RPNI is shown in Fig. 3; both libraries significantly outperformed GI toolbox by a factor of 10, which, for the larger alphabet did not even complete its execution, so we chose to not report it in the plot. A similar trend was obtained for EDSM, when GI-learning consistently outperformed GI toolbox; results are not shows due to space constraints. Finally, as regards active learning, an experimental assessment was conducted for the L^* algorithm.

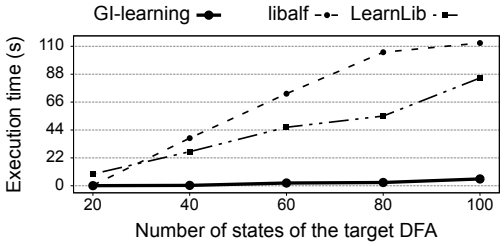
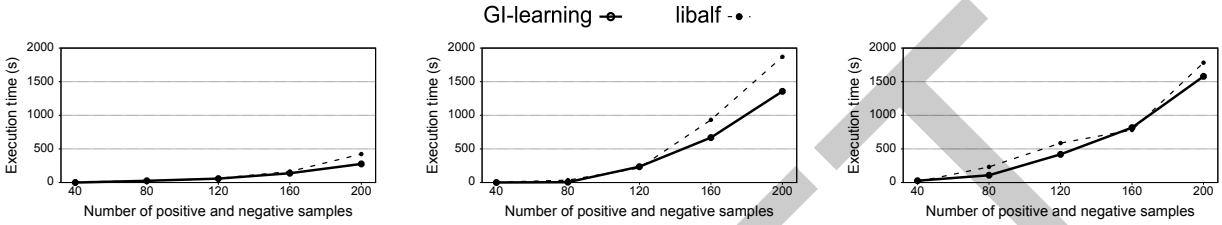


Figure 2: L^* performance for several size of target DFAs.

A representative case, with alphabet size 10, is reported in Fig. 2, for varying sizes of the target DFAs. For an alphabet of 100 symbols, all the libraries fail but *GI-learning*.



(a) Size of alphabet: 2 (b) Size of alphabet: 10 (c) Size of alphabet: 100

Figure 3: RPNI performance for several size of alphabet and number of examples.

CONCLUSION AND ON-GOING WORK

This work has presented a novel framework – GI-learning – which implements the state-of-the-art algorithms in the field of grammatical inference. It provides a ready-to-use software library suitable to extend existing algorithms, and to address new application scenarios; advanced applications, in principle prohibitive due to the involved complexity, become now feasible thanks to parallel execution of relevant computations, while the modular and hierarchical structure allows for a straightforward use and enhancement of supplied methods. Overall, experimental evaluation has shown that GI-learning gets more advantageous performances with respect to the main libraries adopted in literature so far, and those improvements become more significant as volume of data increases. Some interesting and on-going advancements concern the implementation of methods to estimate conformance between automata, as well as further optimization of algorithms in the active learning paradigm.

REFERENCES

[1] Hasan Ibne Akram et al. "Grammatical Inference Algorithms in MATLAB". In: *Proc. of the 10th Int. Conf. on Grammatical Inference*. ICGI'10. Valencia, Spain, 2010.

[2] Alessandra De Paola, Marco La Cascia, Giuseppe Lo Re, Marco Morana, Marco Ortolani. "Mimicking biological mechanisms for sensory information fusion". In: *Biologically Inspired Cognitive Architectures* 3 (2013), pp. 27–38.

[3] Alessandra De Paola, Salvatore Gaglio, Giuseppe Lo Re, Fabrizio Milazzo, Marco Ortolani. "Adaptive Distributed Outlier Detection for WSNs". In: *IEEE Transactions on Cybernetics* PP.99 (2014), pp. 1–12.

[4] Alfonso Farruggia, Giuseppe Lo Re, Marco Ortolani. "Detecting faulty wireless sensor nodes through Stochastic classification". In: *IEEE Int. Conf. on Pervasive Computing and Communications Workshops*. Mar. 2011, pp. 148–153.

[5] Dana Angluin. "Learning Regular Sets from Queries and Counterexamples". In: *Information and Computation* (Nov. 1987).

[6] Dana Angluin. "Queries and concept learning". In: *Machine learning* 2.4 (1988).

[7] A. W. Biermann et al. "On the Synthesis of Finite-State Machines from Samples of Their Behavior". In: *IEEE Trans. Comput.* 21.6 (June 1972).

[8] Benedikt Bollig et al. "libalf: The Automata Learning Framework". In: *Computer Aided Verification: 22nd International Conference, Edinburgh, UK*. Springer, 2010.

[9] Miguel Bugalho et al. "Inference of Regular Languages Using State Merging Algorithms with Search". In: *Pattern Recogn.* 38.9 (Sept. 2005).

[10] Matthew S. Collins et al. "Efficient Induction of Finite State Automata". In: *Proc. of the 13th Conf. on Uncertainty in Artificial Intelligence*. 1997.

[11] Leonardo Dagum et al. "OpenMP: An Industry-Standard API for Shared-Memory Programming". In: *IEEE Comput. Sci. Eng.* (Jan. 1998).

[12] Lee Giles et al. "Noisy time series prediction using recurrent neural networks and grammatical inference". In: *Machine learning* (2001).

[13] E. Mark Gold. "Language identification in the limit". In: *Information and control* (1967).

[14] Andreas Hagerer et al. "Model generation by moderated regular extrapolation". In: *Fundamental Approaches to Software Engineering*. Springer, 2002.

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

[16] Hopcroft et al. *Introduction to Automata Theory, Languages, and Computation*. 2007.

[17] Ramon Janssen et al. *Learning a state diagram of TCP using abstraction*. 2013.

[18] George H John et al. "Irrelevant features and the subset selection problem". In: *Machine learning: eleventh international conference*. 1994.

[19] Kevin J. Lang et al. "Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm". In: *Proc. of the 4th Intl Colloquium on Grammatical Inference*. ICGI '98.

[20] Tim Oates et al. "Motif discovery in spatial trajectories using grammar inference". In: *Proc. of ACM Int. Conf. on Information & Knowledge management*. 2013, pp. 465–468.

[21] Jose Oncina et al. "Identifying Regular Languages In Polynomial Time". In: *advances in structural and syntactic pattern recognition*. World Scientific, 1992.

[22] Pietro Cottone, Salvatore Gaglio, Giuseppe Lo Re, Marco Ortolani. "A machine learning approach for user localization exploiting connectivity data". In: *Engineering Applications of Artificial Intelligence* 50 (2016), pp. 125–134.

[23] Pietro Cottone, Salvatore Gaglio, Giuseppe Lo Re, Marco Ortolani. "User activity recognition for energy saving in smart homes". In: *Pervasive and Mobile Computing* 16, Part A (2015), pp. 156–170.

[24] Harald Raffelt et al. "LearnLib: A Library for Automata Learning and Experimentation". In: *Fund. Approaches to Software Engineering*. Springer, 2006.

[25] R. L. Rivest et al. "Inference of Finite Automata Using Homing Sequences". In: *Proc. of the 21st ACM Symposium on Theory of Computing*. ACM, 1989.

[26] Salvatore Gaglio, Giuseppe Lo Re, Marco Morana. "A Framework for Real-time Twitter Data Analysis". In: *Computer Communications* (Jan. 2016), pp. 236–242.

[27] Marc Sebban et al. *BLUE: a Blue-Fringe Procedure for Learning DFA with Noisy Data*.

ABOUT THE AUTHORS

PostDoc Pietro Cottone, Assistant professor Marco Ortolani, M.Sc. Gabriele Pergola.
DICGIM at University of Palermo: {firstname.lastname@unipa.it}