UNIVERSITÀ
DEGLI STUDI
DI PALERMO

Grammatical Inference for Multi-Scale Mobility Models

Tesi di Laurea Magistrale in Ingegneria Informatica

G. Pergola

Relatore: Prof. Salvatore Gaglio
Correlatori: Ing. M. Ortolani, Ing. P. Cottone

NDS LAB - Networking and Distributed Systems
http://www.dicgim.unipa.it/networks/

UNIVERSITÀ DEGLI STUDI DI PALERMO

SCUOLA POLITECNICA

Laurea Magistrale in Ingegneria Informatica

GRAMMATICAL INFERENCE FOR MULTI-SCALE MOBILITY MODELS

TESI DI LAUREA DI

Dott. Gabriele Pergola

RELATORE

Ch.mo Prof. Salvatore Gaglio

Anno Accademico 2014/2015

Sommario

L'estrazione di conoscenza rappresenta una delle sfide più stimolanti nel campo dell'Intelligenza Artificiale, come dimostrano gli svariati approcci che nel corso del tempo sono stati proposti in letteratura, che spaziano dal ragionamento simbolico all'apprendimento statistico. La capillare diffusione di dispositivi mobili, come *smartphone* e *tablet*, in grado di rilevare e memorizzare gli spostamenti degli utenti su vasta scala, ha dimostrato come "misurare" non significhi di per sé "comprendere", e che l'incremento di dati grezzi non corrisponda ad un immediato aumento di conoscenza.

Ad oggi, le tecniche più efficaci per l'estrazione di conoscenza sono state sviluppate nell'ambito della Statistical Learning Theory; tali metodi forniscono modelli poco esplicativi ad un'ispezione umana circa le caratteristiche salienti dei dati. Un approccio promettente è quello dell'*Algorithmic Learning Theory*, in cui i dati non sono assunti come campioni casuali, ma in quanto generati secondo una struttura nascosta, catturabile attraverso un processo di inferenza algoritmica. Tale processo è vincolato da precisi limiti teorici, come mostrato da E. Gold nel paradigma dell'*Identification in the limit* [37].

In questa tesi sono analizzate sia le potenzialità che i limiti dei *linguaggi formali* come strumenti di modellazione, dedicando particolare attenzione ai loro riconoscitori, gli *automi*. L'elaborazione dei dati simbolici attraverso di essi si avvale di approcci basati sia su euristiche che su metodi di ottimizzazione, nonché di tecniche sviluppate all'interno del paradigma dell'*Active Learning* [2].

L'obiettivo principale di questo lavoro è l'utilizzo dell'approccio algoritmico per l'estrazione di *modelli di mobilità* nel contesto delle *Smart Cities* [32]. In particolare, un modello di mobilità può essere definito come una rappresentazione concisa degli spostamenti di un utente, in cui sono sintetizzate i percorsi già osservati ed in grado di predire i futuri. Di fondamentale importanza, in questo senso, è stata la vasta diffusione di sensori di posizione in grado di raccogliere ingenti quantità di dati di posizione, necessari per ottenere un elevata affidabilità dei modelli inferiti; di contro, il volume di queste informazioni rende, ad oggi, l'estrazione di conoscenza un compito molto complesso.

I linguaggi formali, grazie alla loro natura ricorsiva, sono uno strumento che ben si presta alla gestione dell'inferenza di tali modelli, in quanto capaci di catturare a diversi fattori di scala le regolarità sottese al comportamento degli utenti monitorati. Parte integrante di questa tesi è stata la progettazione e lo sviluppo di una libreria *software* per l'inferenza grammaticale, su cui si basa il sistema proposto per l'estrazione di modelli di mobilità.

Le *performance* del sistema sono state testate su dati reali, ed in particolare si è scelto di adoperare il *dataset* fornito dal *GeoLife Project* [78], in cui è sono contenuti i tracciati GPS di oltre un centinaio di utenti monitorati per un periodo di cinque anni. Un ampio *set* di esperimenti è stato condotto al fine di testare la reale efficienza dell'inferenza e la precisione dei modelli ottenuti. I risultati mostrano come un approccio strutturale risulti efficace nell'estrazione di conoscenza, come confermato dai modelli inferiti, la cui intrinseca struttura ricorsiva si adatta alla descrizione delle abitudini di mobilità degli utenti.

UNIVERSITÀ DEGLI STUDI DI PALERMO
SCUOLA POLITECNICA
Laurea Magistrale in Ingegneria Informatica

GRAMMATICAL INFERENCE FOR MULTI-SCALE MOBILITY MODELS

Master's Thesis by
Gabriele Pergola

Supervisor
Prof. Salvatore Gaglio

2014/2015

Abstract

Knowledge extraction represents one of the most interesting challenges in Artificial Intelligence, and several approaches have been proposed, ranging from symbolic reasoning to statistical learning. The diffusion of mobile personal devices able to provide position information, such as smartphones and tablets, has demonstrated that "measuring" does not directly imply "understanding", hence an increased amount of raw data does not immediately produce more knowledge.

To date, the most effective techniques have been developed within *Statistical Learning Theory*; they provide the set of optimal parameters for models in the form of black-boxes, but are unable to give actual insight into the real nature of data. In this context, *Algorithmic Learning Theory* is a promising approach, because it does not assume that data are random samples, but rather regards them as generated by a hidden *structure*, which may be recovered through an algorithmic inferential process. Such inference is not a trivial task, due to the theoretic constraints pointed out by E. Mark Gold [37] within the *Identification in the limit* paradigm.

This thesis analyzes both the potentialities and the limits of *formal languages* as modelling tools, with specific regard to their recognizers, i.e. automata. The algorithmic approach deals with symbolic data, mainly by means of heuristics, and optimization methods; notably, however, an alternative approach based on the learning paradigm known as *Active learning* [2] has also been proposed in this context.

The main goal of the present work is an application of this approach in order to address the issue of *mobility model* extraction for *Smart Cities* [32]. A mobility model is a concise representation for coding user movements, synthesizing paths already observed and predicting future travels. The diffusion of wearable devices, equipped with position sensors able to collect a huge amount of spatial data, has proven fundamental for developing reliable mobility models; however, information extraction from this flow of raw data still remains an open issue. Formal languages, with their recursive structure, are a powerful tool to manage the whole process of mobility model extraction, thanks to their ability to capture regularities in users' behavior at different granularities. As part of this thesis, an inferential system based on regular languages was implemented, together with a complete software library for grammatical inference.

The performance of the proposed system was assessed on real data, collected by the *GeoLife Project* [78], which supplies a huge amount of spatial movements recorded via GPS devices. A set of experiments has been conducted aiming at computing both the efficiency of the inferential process, and the reliability of the obtained models. The presented results show that the structural approach is a valuable tool for knowledge extraction as confirmed by the obtained models, whose intrinsically recursive structure is well fit for describing user habits in the mobility scenario.

Contents

Acronyms

ALT	Algorithmic Learning Theory
DFA	Deterministic Finite state Automaton
EDSM	Evidence Driven State Merging
FSA	Finite State Automaton
GIG	Regular Grammatical Inference from Positive and Negative Samples by Genetic Search
MAT	Minimally Adeguate Teachers
MCA	Maximal Canonical Automaton
MDL	Minimum Description Length
NFA	Nondeterministic Finite state Automaton
POI	Points Of Interest
PTA	Prefix Tree Acceptor
RPNI	Regular Positive and Negative Inference
SLT	Statistical Learning Theory

Chapter 1

Introduction

"Computer Science is no more about computers than astronomy is about telescopes"

Edsger Dijkstra

Knowledge extraction has represented one of the most interesting challenges in Artificial Intelligence, and several approaches have been proposed, ranging from symbolic reasoning to statistical learning. The diffusion of devices such as smartphones and tablets, which typically embed position sensors, has shown clearly that an increasing amount of raw data does not entail more knowledge, hence "measuring" does not directly means "understanding".

Machine Learning is one of the main fields which aims to develop tools to manage knowledge. So far, the most effective techniques have been developed within *Statistical Learning Theory - SLT*, as the high rate of success and the portability to several application scenarios demonstrates. However, models obtained by such methods are provided in the form of optimal parameter settings for black-boxes, and does not satisfy the crucial demand about "the reasons why" for the inferred algorithm during the process of knowledge extraction.

A promising approach to address such weakness comes from *Algorithmic Learning Theory - ALT*, which does not assume that data are random samples, but rather regards them as generated by a shared hidden *structure*, which might be mined through an algorithmic inferential process.
The involved models are inherently oriented to process structural knowledge; nevertheless handling such information is not a trivial task, due to the theoretic constraintes pointed out by E. Mark Gold [37] within the *Identification in the limit* paradigm.

This work analyzes both the potentialities and the limits of *formal lan-*

guages, with specific regard to their recognizers, i.e.: automata; this is motivated by the obvious correspondence between the hierarchy of languages and that of automata, as defined by Noam Chomsky [17].

The algorithmic approach aims at processing symbolic data, a challenging task both because of the above mentioned theoretic limits, and for time complexity involved. A wide range of methods have been used to address these issues; among them, heuristic algorithms (Regular Positive and Negative Inference (RPNI) [62], Evidence Driven State Merging (EDSM) [50] and Blue* [70]), optimization algorithms (Regular Grammatical Inference from Positive and Negative Samples by Genetic Search (GIG) [27]), and learning algorithms (L* [1]) which are based on the alternative learning paradigm known as *Active learning* [2].

The study of these algorithms in the context of the present thesis has resulted in the creation of a software library, as well as in the implementation of an inferential system for mobility models, a key issue for *Intelligent Transportation Systems* that are becoming widespread in the context of *Smart Cities* [32]. A mobility model is a concise representation for user movements, synthesizing paths already observed and predicting future travels. Diffusion of personal devices with position sensors able to collect a huge amount of spatial data, has proved to be fundamental for developing reliable mobility models; however, the issue about information extraction from this flow of raw data still remains open [53].

The key insight of this thesis is to exploit formal languages in order to capture such information, thanks to their effectiveness as models for catching regularities about users' behavior. Several works, such as [38], have shown that inhabitants of a city naturally induce schemata from their daily routines, from residential places, to work and recreational places and back. These social patterns tend to characterize human behaviors and modeling those regularities is one of the main goals in mobility analysis.

Hence, the core of this thesis is the design and implementation of a system to manage the whole process of mobility model development. This includes geohash encoding, a symbolic and hierarchical representation of spatial coordinates, which are successively translated into a recursive structure fit for multi-level analysis.
The systems is composed by three modules, each one performing a different step of mobility model creation.
The first module is the *Mini Trajectory Database Manager* which transforms initial spatial information into symbolic strings, recording them in a database for future access.
A second module, the *Inference processor*, builds the mobility model for a

user. It relies on the Blue* algorithm which, despite being originally designed to manage noisy data, was slightly modified in order to cope with negative examples too; the asymmetry between positive and negative samples brings a computational burden due to the low information content of data. Blue* is able to detect such data and disregard them, leaving them out of the model, through a trade-off between frequency of occurrence of examples, and model simplicity.
The final module is the *Mobility Model Handler* devoted to expose mobility models as available services so as, for instance, to check compatibility between a user model and some new observed spatial behaviors. Inferred models are hierarchically arranged and robust, thanks to the adopted encoding; their flexibility also allows to employ them for multi-scale analysis of user movements, and finally they are characterized by being intrinsically predictive.

The assessment of the proposed system was conducted based on real data, namely the *GeoLife dataset* [78], collected by *Microsoft Research Asia*. Such dataset contains a huge amount of spatial movements recorded via GPS devices and smartphones over more than 5 years, and for more than a hundred users. Several experiments have been conducted in order to measure both the efficiency of the inferential processes, and the reliability of the obtained models.

An initial test has regarded the accuracy of the model prediction, through classical *k-fold cross validation*, in order to measure precision across different scale degrees, and to test the ability to correctly recognize previously unseen behaviors.

A second experiment was designed to compare automata inferred by two different algorithms, namely EDSM and Blue*, and showed that the latter results in an improvement in terms of efficacy (precision and size of the obtained automata) and efficiency (number of heuristic evaluations, and so on).

Finally, the possibility of using the proposed approach as a *recommender system* was tested; to this end, a simulation scenario was considered, and new users were modeled by exploiting existing mobility modes in order to provide improved suggestions about possible paths of interest.

The study of inference algorithms, and the analysis of the system for mobility support have shown that a structural approach may indeed be considered a valuable tool for knowledge extraction. The obtained results have confirmed the reliability of the proposed methods, whose intrinsically recursive nature is well fit for describing multi-scale models, which in turn emerge naturally in mobility scenarios.

Chapter 2

Knowledge and structural inference

"Too much faith should not be put in the powers of induction, even when aided by intelligent heuristics, to discover the right grammar. After all, stupid people learn to talk, but even the brightest apes do not."

- Noam Chomsky, *1963*

2.1 Learning

Being able to *learn* is one of the most prominent features of intelligent behavior, both for human and artificial agents. A software agent labeled as "smart" should be able to learn – hopefully in an automatic fashion – and its behavior should be more than the combination of its skills, allowing it to reason and come up with a plan in order to achieve some goal or to perform deductive inference and reasoning.

Different flavors of learning, together with the relative strategies, have been studied from several points of view within the fields of cognitive science; among these, Artificial Intelligence confines the question mainly within the *Machine Learning* subfield.

As pointed out by Simon [72], "learning denotes changes in systems that may be deemed adaptive in that they enable the system to perform the same task (or tasks drawn from the same population) more efficiently and more effectively over time". More precisely, Mitchell [60] defined what a well-posed learning problem in machine learning perspective is:

Definition. A computer program is said to *learn* from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

For instance, a computer program that learns to play a game might improve its performance, measured as its ability to win at the task defined by the rules of the game, given through the experience collected by playing several times.

More generally, an initial analysis of the targets of a learning process, i.e. what is possible to learn or not, points out at least two basic forms of learning [12], namely *skill refinement* and *knowledge acquisition*.

Skill refinement is usually performed by stepwise enhancement of motor and cognitive skills through practice, as for instance, in the case of tasks such as riding a bicycle or playing piano.

A further kind of learning is knowledge acquisition, which refers to gathering new symbolic information, together with the ability to make an effective use of that information. A typical instance is learning math, which requires to obtain information, to formulate concepts, grasping their meaning, and finally understanding how they are anchored to reality.

"Learning more" means that the newly acquired knowledge helps to explain a broader variety of situations, making more accurate predictions, and may be effectively applied in novel situations. Processing knowledge entities is triggered by internal or external events; managing already acquired information leads to the definition of new, more accurate, knowledge despite no addition from the external world.

In summary, the aim of such learning systems is to generate a refined knowledge about some aspects of reality, rather than directly improve their own skills. However, a likely side effect is an improvement of existing skills resulting from the intrinsic value of acquired knowledge.

2.1.1 Structural knowledge

Three basic forms of knowledge can be identified, according to knowledge theory [46, 58]:

- ***Declarative knowledge:*** it represents cognizance of an object, event or concept[1]. This type of knowledge allows to define and describe an

[1]Refer here to a definition of *concept* as equivalence class identified by a finite set of assertions, with an associated actual classification procedure. It is only a rough approximation of human notion of concept, but it has proven effective to drive concept learning research [55].

object or a concept, without necessarily entailing the ability to deal with it. For that reason, this type knowledge is referred to also as *knowing that* [68]. Representative instances of declarative knowledge are *schemas*, abstract constructs, defined by attributes inherited from others schemas.

- ***Procedural knowledge:*** it describes how to use or to make declarative knowledge effective; it is referred to as *knowing how*. Schemas and their relations (expressed by structural knowledge) are translated into *patterns*, which represent available system abilities. Some revealing examples are activities such as forming plans or solving problems, which need procedural knowledge. To accomplish these tasks, a learner needs to access schemas and their relations to finally extract relevant attributes. Therefore, declarative knowledge provides a base for procedural knowledge.

- ***Structural knowledge:*** this form is an intermediate type between declarative and procedural knowledge, and accommodates the translation from the former to the latter. Knowing *how* is not sufficient to know *that*, and it is necessary to understand *why*, which is the essence of structural knowledge. The explicit awareness and management of relations between schemas inside declarative knowledge denotes the structural one.

In some works in the literature [58], structural knowledge is conceived as part of the declarative one, which has two dimensions: content and structure. In this case, structure is the subset of declarative information about organization within a knowledge domain. Nevertheless, the difference between the two approaches remains a semantic distinction that does not exludes identifying structural knowledge as a distinct type or entity [46].

Additionally, it is worth recalling the implicit distinction between *structured* and *structural* knowledge. In the first case, what is pointed out is the arrangement in terms of entities and relationships, with an emphasis on *how knowledge is settled.* The latter case, instead, refers to the type of knowledge to be acquired, regardless of how it is arranged. Focus is on *patterns and structures of analyzed objects.*

2.2 Learning systems

Human as well as artificial learning systems may show different degrees of organization, ranging from the less complex to the more sophisticated ones; as a consequence, several taxonomies have been proposed, with respect to the considered features. Among them, three are worth mentioning and their respective criteria are the amount of available information when learning starts, the strategies adopted to generate new knowledge, and the internal representation employed.

2.2.1 Initial information

The information available to the learning system in its initial state is highly influential for its learning capabilities and performance.

At one extreme of an ideal classification scale are *expert systems*: information and instructions recorded at the beginning of the learning process remain prevalent while the system operates; this is all the knowledge for a particular application domain. Usually such systems need to rely on experts, who encode their own knowledge on it; this process is not straightforward or immune from errors, and typically requires many experts for each application area.

At the opposite side, there are systems like *artificial neural networks* (ANNs), a prominent representative of the connectionism paradigm . During the design phase, the information provided to the system is made up only of the input variables considered as representatives attributes for the examined objects, together with the initial weights of the connections into a built network, whose structure may be random or fixed by designer. Such systems learn to modify the weights of the connections during the incremental training stage.

Others instances belonging to within this category are evolutionary and genetic algorithms [42], aimed at numerical optimizations. All these learning methods are characterized by a high rate of numerical elaborations, as opposite to the symbolic processing performed by previously mentioned approaches.

2.2.2 Learning strategies

Learning strategies have been categorized according to the relevance of inference that takes place for knowledge management.

At one end of the range, methods which basically perform no information elaborations may be found; their knowledge can increase, but no modifications occur, except for the collection and indexing of new data. The cognitive effort is entirely delegated to the learning system designer. Conversely, methods developing new “theories” and discovering new concepts are found at the opposite end. They give much room to the inference process for the generation of new knowledge, as in the case of inductive inference. An intermediate approach is represented by systems that learn by analogy: they exploit solutions already available for well-known problems in order to infer customized solutions for new ones. Knowledge that bears strong similarity to the new desired skills or concepts can be easy rearranged for the new purpose.

A comprehensive taxonomy about learning strategies may be found in [56, 57], where methods are identified based on the kind of inference the learner performs to produce new information. Among the methods reviewed therein are:

1. ***Rote learning:*** the learner makes no inference or transformation on data. This strategy is widely used for software whose logic is completely embedded by the designer, and external information is readily available so it only needs to be recorded and no transformation is necessary.

2. ***Learning from instruction:*** the learning process is performed by a teacher or from an organized source, such as structured documents. Limited inference is required, as needed to detect and transform input data into an internally representation ready available, and integrated with previous knowledge. Those systems are able to receive and record instructions and eventually performs them.

3. ***Learning by deductive inference:*** this kind of systems also transform input data into an internal representation, but unlike previous ones, they carry out deductive (truth-preserving) inference, which restructures the given information in such a way that knowledge becomes more functional for application.

4. ***Learning by analogy:*** it allows to acquire new knowledge by extending or transforming the one that already exists. The specificity of this approach is the identification of the part of the current knowledge which may be better adjusted to new demands, and its transformation in an effective way. Once it is ready, the learner records it for future uses.

5. ***Inductive learning*** is the last learning strategy examined. It deserves particular attention due to its relevance during the inferential process generating structural knowledge, so it will be presented in greater detail in the following.

2.2.3 Inductive learning

The process of acquiring knowledge by inductive inference exploits facts originating from the environment or provided by teachers. It has a fundamental role for most of the techniques employed in the field of machine learning, and involves operations such as generalizing, correcting, transforming and refining the available knowledge so as to satisfy some constraints and match concepts with observations.

Despite its undeniable effectiveness, induction shows an intrinsic drawback: except for some special cases, the produced knowledge cannot be completely validated in principle, due to the fact that the produced hypotheses have infinite potential consequences, whereas only a finite number of them can be tested and checked, as eminently remarked by Hume [44]. In addition, given a set of observations and information, an infinite number of hypotheses explaining them can be generated. Therefore, in order to perform induction, additional criteria and knowledge (*background knowledge*) are needed to constrain generalization, and to guide the inference process toward the acceptable hypothesis [56].

Because of that, inductive inference is not “truth preserving”, but only “falsity preserving” [57]; in other words, from false premises only false generalizations will be inferred. On the other hand, hypotheses induced from true premises are only *potentially* true, unlike deductive inference which guarantees correct output.

To clarify this behavior, consider a classical statement: *“all observed swans are white”*. A deductive inference process would conclude that a swan inside a natural reserve is white, being an instance of an observed entity; given that the premise is true, then the conclusion must be too. In contrast, a typical inductive inference for this premise is *“all swans are white”*, and in spite of the huge amount of white swans observed, this conclusion resulted false after the first explorations in Australia during XVI century [35].

Therefore, inductive learning has a *inherently conjectural* nature, as observed by Popper and others [65, 57]: any knowledge generated by inductive inference from specific observations cannot in principle be proven definitely right, although it may be falsified.

Nevertheless, inductive inference remains one of the most powerful tools to generate new knowledge, and accurate choice of constraints and assumptions usually produces relevant results.

A deeper analysis of scenarios which take advantage from inductive inference leads to identify two different types of induction, namely *learning from examples* and *learning from observation and discovery* [12]:

- ***Learning from examples***: a learner induces a concept to describe a given set of examples and counterexamples. Inferred concepts are compatible with examples (positive samples) and reject every counterexamples (negative samples). Positive samples guide the generalization process avoiding data overfitting, whereas negative samples moderate hypothesis extensiveness, preventing overgeneralization.

In some application scenarios negative samples are not available, therefore constraints derived from the application domain are adopted to contain the side effect of overgeneralization; if unsuitable, an alternative strategy consists in choosing as little generalization as possible.

In both cases, there are two ways to provide samples to learners, either with *one trial*, or *incrementally*. In one-trial settings, all the examples are supplied at the beginning of the inference process, while in the latter case, several hypotheses compatible with available samples are produced at each step, resulting in incremental improving. Several studies examined the use of these methods within AI [37], thus building a solid framework to benchmark strategies for this kind of learning.

- ***Learning from observation and discovery***: also known as *unsupervised learning*, it is one of the most general forms of inductive learning. It involves discovering new theories, by developing new classification criteria or similar means, without assistance from any teacher. This strategy requires the learner to perform a larger amount of inference than any other learning approach.

  With respect to the degree of interaction with environment, a subclassification follows [12]:

  – *Passive observation:* the learner passively classify several aspects of environment.

  – *Active experimentation:* the learner perturbs the external environment aiming at observing the outcome of the interaction. This kind of experimentation may be random, dynamically focused or constraint-driven.

Internal knowledge representation directly influences the abilities of a system; a brief overview of the most common representations is provided in the following.

2.2.4 Internal knowledge representations

Internal knowledge representation is a key element in a learning system, because it constraints the tasks the system can carry out and the skills it can develop. In this thesis, the focus is on internal representations that can handle structural information, hence representations emphasizing the structural content of the acquired knowledge. A system has to deal with different kind of representations in order to accomplish its tasks, such as description of physical objects, rules of behavior, heuristics, classification taxonomies, and many more. Among them, *formal grammars* are worth particular attention considering their aptitude to model structural information [12].

To accomplish the assigned tasks requires developing many different types of knowledge, including the description of physical objects, rules of behavior, heuristics, classification taxonomies, and many more. Among them, *formal grammars* are worth particular attention considering their aptitude to model structural information [12].

Formal grammars are usually induced to recognize particular languages, especially artificially. Typically, they are inferred from sequences of expressions representing the target language, and may employ different kinds of representations such as regular expressions, (finite state) automata and generative grammars (Fig. 2.1).

This kind of representation is characterized by being intrinsically structural, which makes it especially interesting for the purposes of this work. *Syntactic pattern recognition* [29] is a classification method which exploits the objects' internal structure in order to define patterns and to carry on the inference process. Structural methods extract relationships between the basic components of patterns, rather than focusing on statistical properties of a phenomenon, like in statistical approaches.

In many works in literature, typical objects of pattern analysis are handwritten characters, fingerprints, chromosomes, spatial pictures composition, and so on [29]. The theoretical and practical tools examined in this work were tested in the context of automatic mining of *mobility patterns*. Users' spatial behaviors, at different scales, are automatically encoded into patterns based on regular grammars, obtained via state-of-the-art algorithms for grammar inference. User data are input of the inference system, that elaborates them in terms of formal languages. Such mobility patterns lay the ground for several applications inside the *smart-city* domain, providing support for user travels as, for instance, in a car pooling scenario [75].

This approach appears promising thanks to the straightforward analogy between the structure of patterns and the syntax of languages, which rests on a wide and well-founded literature [29]. Formal languages, automata and grammars, show at least two features worthy of attention: formal grammars directly lead to syntactic classifiers (parsers), employed as operative definition of inferred concepts; furthermore, in a setting characterized by high volume of data, the mined patterns represent the hidden structure behind information, returning an expressive representation for visual inspection, unlike statistical indicators.

Moreover, recent works support the effectiveness of structural approaches; for instance, in [25], researchers showed that the understanding of a connected speech gives rise to concurrent tracking of different timescales, for the identification of abstract linguistic structures at different hierarchical levels. Different neural processing timescales suggest a grammar-based internal construction of the linguistic structure, corroborating the insight that structure of thought, reasoning and language are strictly tied. Therefore, syntactic and grammar oriented approaches could provide a more human comprehensible model.

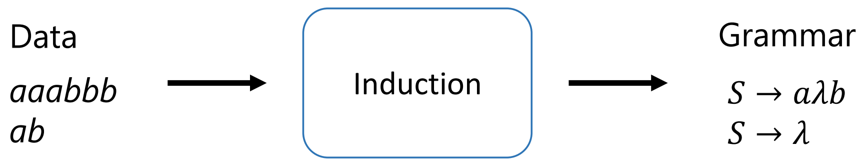


Figure 2.1: An inductive system for grammatical inference.

The most common alternative representations are reported in the following, according to the taxonomy presented in [12].

- **Algebraic expressions:** this internal representation is based on algebraic expression of a fixed functional form. The learning process consists of selecting the best set of numeric parameters or coefficients for the algebraic expressions in order to get the desired performance. For instance, *perceptrons* [67] belong to this category and their weight coefficients are the parameters to be tuned.

- **Decision trees:** they classify different item classes. The inner nodes correspond to selected attributes, while edges correspond to predetermined values that attributes may assumes. Leaves of tree define the set of objects belonging to the same classification.

- **Formal logic-based expressions and related formalisms:** they are general-purpose descriptions, employed to encode formal representations of individual items. They make use of formal logic expressions and they are composed by: propositions, predicates, variables and constraints.

- **Graphs and networks:** they are representations with the same expressive power of formal logic expressions; however they are more efficient and are usually exploited to compare knowledge performing graph matching or transformations.

- **Frames and schemas:** they are made of entities labelled, known as (*Slot*), with a fixed role within the representation. They provide larger units with respect to single formal logic expression. Usually, they are adopted in systems to acquire and manage plans in the form of compact entities, regardless of their inner complexity.
- **Taxonomies:** they consist of hierarchical representation of the elements of the application domain, built through learning by observations. A system needs to formulate valid criteria to classify objects descriptions in new categories, that represent the new acquired knowledge.
- **Multiple representations:** they are mixed sets of previous internal representations, used by several systems in order to produce new theories through their combinations, choosing the most suitable one to learn new concepts and handle them.

Chapter 3

Inference of regular languages

"Understanding is compression, comprehension is compression!"

Greg Chaitin

3.1 Definitions and notation

Formal language theory was developed by Noam Chomsky in the late '50s [17]; he defined the concepts of language and grammar as mathematical entities, providing a solid theoretic foundation for the studies in this field.

A ***formal language*** is traditionally defined as a *"a set (finite or infinite) of sentences*[1]*, each finite in length and constructed out of a finite set of elements".* [16].

Several representations of languages have been formulated inside the cognitive sciences, each suitable for different analysis and applications. Among these, two main descriptions have been developed inside the linguistic field and computer science respectively: one through *generative* approach and another one through a *recognition* approach [51].

Generative description

A generative description of a language is a finite set of symbols and rules, employed to build any sentence in the language. Such description states a ***formal grammar***, also known as *generative grammar.* From a mathematical

[1]For our purpose, sentences are generally equivalent to strings

point of view, it is a system identifying the language through rules that define all the well-formulas, i.e. language strings. Its input and output are strings built over the *alphabet* symbols; inputs begin with a *start symbol*, while the outputs are produced after that some *terminal symbols* have been processed. Depending on the type of rules composing the grammar, Chomsky developed a *hierarchicy of classes* of generative grammars [17], as described in the following.

Recognition description

A recognition description identifies a language by an *extensional enumeration* of each string belonging to language, or by a *intensional definition* describing the string structure identifying language; the latter description exploits the formalism of ***regular expressions***.

A further recognition description, with even more characterizing recognizing properties, exploits another kind of formalism: the ***automata***. They are abstract machines, whose input are made of strings. An automaton identifies (and describes) a language recognizing whether an input string belongs to languages, by accepting or rejecting it; the set of all accepted strings identifies the language described by an automaton.

For each class of grammars, within Chomsky's hierarchy, correspond a different kind of automata.

The two mentioned descriptions share many useful commons properties, even if they were independently developed. Nevertheless, they exhibit remarkable differences in terms of ease of use in handling languages, and implied computational complexity. For the aims of this thesis, analysis of these features lead to prefer automata as languages representation, due to theoretic tools developed to manage them, despite expressiveness of grammar description.

3.1.1 Alphabets, strings and languages

To begin with, let us introduce the definitions and formalisms adopted. Many of those are borrowed from literature and nowadays are widely used.

Alphabets

An ***alphabet*** Σ is a finite non-empy set of symbols called ***letters*** [41].

Strings

A ***string*** x over Σ is a finite sequence $x = a_1 \ . \ . \ . \ a_n$ of letters. $|x|$ denote the length of string x.

The ***empty string*** is denoted by ϵ, or alternatively with λ.

Σ^* denote the set of all finite strings over alphabet Σ. Also, Σ^+ is the the set $\Sigma^* - \epsilon$.

Languages

A ***language*** L is any set of strings, i.e. any subset of Σ^*.

The ***complement*** of a language L, taken with respect to Σ^*:

$$\overline{L} = \{w \in \Sigma^* : w \notin L\}$$

The ***complement of L_1 in L_2*** is denoted by $L_1 \backslash L_2$:

$$L_1 \backslash L_2 = L_1 \cap \overline{L_2} = \{x \in L_1 : x \notin L_2\}$$

Let us now define the ***symmetric difference*** between two languages L_1 and L_2, denoted by $L_1 \oplus L_2$:

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

The ***prefix set*** of a language L:

$$\mathit{Pref(L)} = \{u \in \Sigma^* : \exists v \in \Sigma^*, uv \in L\}$$

and ***suffix set*** of L:

$$\mathit{Suff(L)} = \{v \in \Sigma^* : \exists u \in \Sigma^*, uv \in L\}$$

A language is ***prefix-closed*** if $\forall u, v \in \Sigma^*, uv \in L \Rightarrow u \in L$, and ***suffix-closed*** if $\forall u, v \in \Sigma^*, uv \in L \Rightarrow v \in L$

Formal languages and grammars were hierarchically arranged by Chomsky [17]:

Definition (Recursively enumerable language)**.** A language is ***recursively enumerable*** or ***Turing-acceptable*** if there exists a Turing machine, which for each string $x \in \Sigma^*$ halts and returns 1 if $x \in L$, otherwise if $x \notin L$ it may either returns 0 or doesn't halt, looping forever.

Definition (Recursive language)**.** A language L is ***recursive*** or ***decidable*** if there is a Turing Machine that for each $x \in \Sigma^*$ halts and returns 1 if $x \in L$, 0 if $x \notin L$.

Definition (Primitive recursive language)**.** A language is ***primitive recursive*** if its characteristic function[2] is primitive recursive.

From the works over *effectively calculable* function [30]:

Definition (Recursive operation)**.** Given the functions $f : N^n \rightarrow N$ and $g : N^{n+2} \rightarrow N$, the function $z : N^{n+1} \rightarrow N$ is defined as:

$$\begin{cases} z(x_1, x_2, ..., x_n, 0) = f(x_1, x_2, ..., x_n) \\ z(x_1, x_2, ..., x_n, s(y)) = g(x_1, x_2, ..., x_n, y, z(x_1, x_2, ..., x_n, y)) \end{cases}$$

it is obtained from f and g by recursive operations.

Definition (Primitive recursive function)**.** A function is ***primitive recursive*** if built by applying, a finite number of time, composition and recursion operations over the basic functions:

1. Successor function: $S(x) = x + 1$;
2. Empty function: $N(x) = 0$;
3. Identity function: $U_i^n(x_1, x_2, ..., x_n) = x_i$ with $i \leq i \leq n$.

3.1.2 Grammars

Recall the informal grammar definition, as a generative device composed by a set of symbols and rules, that is able to build all the strings of a language. Formally:

Definition (Chomsky generative grammar)**.** A ***Chomsky generative grammar*** is a quadruple:

$$G = (\Sigma, V, S, P)$$

where:

- Σ is an alphabet, denoted as set of terminals symbols,

[2]Given a set S and an item x, a *characteristic function* returns 1 if $x \in S$, 0 if $x \notin S$.

- V is a disjointed alphabet over Σ, denoted as set of non terminals symbols,
- S is a particular symbol of V, known as the start symbol of the grammar,
- P is a finite set of production rules, defined as:

$$\Psi \rightarrow \Theta \text{ with } \Psi \in (\Sigma \cup V)^* V (\Sigma \cup V)^* \text{ and } \Theta \in (\Sigma \cup V)^*$$

Enforced constraints to the generation rules give rise to the differents classes of grammars[17]:

- **Type-0 grammars - Unrestricted:** it is the most general class of grammars, without any restriction, able to describe every classes of recursively enumerable languages. The automata recognizing and accepting these languages are the *Turing machines* [30].
- **Type-1 grammars - Context Sensitive:** for those grammars, production rules are defined as:

  $$\alpha_1 A \alpha_2 \rightarrow \alpha_1 \Psi \alpha_2$$

  $$\text{with } \alpha_1, \alpha_2, \Psi \in (\Sigma \cup V)^* \text{ and } A \in V$$

  Automata recognizing those languages are indicated as *Linear Bounded Automata.*
- **Type-2 grammars - Context Free:** in this case, production rules have the form:

  $$A \rightarrow \Psi$$

  $$\text{with } \alpha_1, \alpha_2, \Psi \in (\Sigma \cup V)^* \text{ and } A \in V$$

  Contex Free languages are recognized by automata denoted as *Push Down Automata (PDA).*
- **Type-3 grammars - Linear:** the production rules for the latter type of grammar have the form:

  $$A \rightarrow \Omega B \text{ or } A \rightarrow B\Omega$$

  $$\text{with } A \in V, B \in (V \cup \{\lambda\}) \text{ and } \Omega \in \Sigma^+$$

  Languages generated are namely *regular languages*, and automata recognizing them *finite automata.*

The classes of grammars establish a *nested hierarchy*, starting from type-0 grammars including all the others. The grammar hierarchy symmetrically induces an automaton hierarchy, made by the mentioned automata matching the different kinds of grammars (Fig. 3.1).

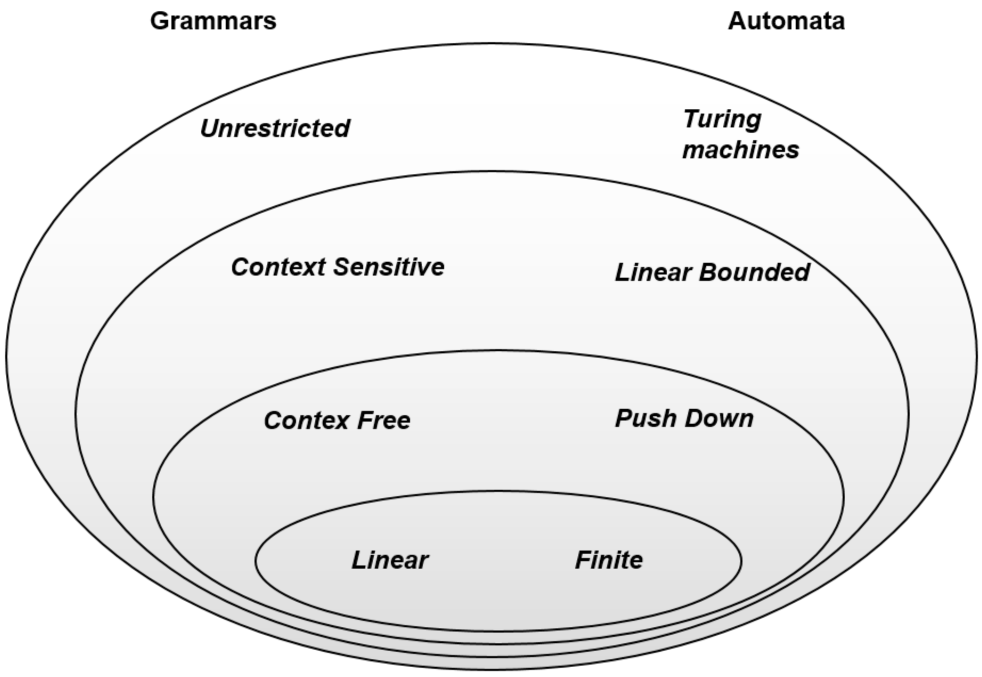


Figure 3.1: Chomsky's hierarchy for grammars and automata.

It is possible to strictly specify how grammars and their generated languages are related:

Definition. The *language generated by a grammar* G is the set of strings Σ^* produced from the start symbol S:

$$L(G) = x \in \Sigma^* : S \overset{G*}{\Rightarrow} x$$

with $\overset{G*}{\Rightarrow}$ transitive and reflexive closure[3] of the productions rules $\overset{G}{\Rightarrow}$.

Note that a grammar generates a unique language, while, asymmetrically, a language can be described by more than one grammar.

[3]Given two strings s_1 and s_2, if s_2 can be derived from s_1 in zero, one or more derivations, then $s_1 \overset{G*}{\Rightarrow} s_2$

3.1.3 Finite state automata

The representation by means of automata makes it easier to learn and manipulate language. For the purposes of this work, particular attention is given to ***Finite State Automaton (FSA)*** because of regular languages, a class of language deep examined for its properties and simplicity. Also, the obtained results for regular languages might be converted for higher classes of languages.

These automata are a restricted version of Turing machines, consisting of ***accepting*** and ***rejecting*** states, also denoted as ***final*** states, which enable automata to accept or reject the supplied strings. Further, there are states called ***neutral*** that confer a finer capability to discriminate strings.

Two variants of FSAs have been widely studied, the *Nondeterministic Finite state Automaton (NFA)* and *Deterministic Finite state Automaton (DFA)*. The main difference is the nondeterministic behavior of NFAs, due to the presence of more than one transitions for the same alphabet symbols or to ϵ-transitions. More precisely:

Definition (Nondeterministic finite automaton)**.** A ***Nondeterministic finite automaton*** (NFA) is a 6-tuple

$$\mathcal{A} = \langle \Sigma, Q, q_\lambda, F_A, F_R, \delta \rangle,$$

where:

- Σ is an alphabet;
- Q is a finite set of states;
- $q_\lambda \in Q$ is the start state;
- $F_A \subseteq Q$ and $F_R \subseteq Q$ are sets of final states denoted as *accepting states* e *rejecting states*;
- $\delta : Q \times \Sigma \to Q$ is the transition function, with arguments a state of Q and a symbol from Σ, returning a subset over Q.

The DFA is very similar, unless for the transition function:

Definition (Deterministic finite automaton)**.** A ***Deterministic finite automaton*** (DFA) is a 6-tuple

$$\mathcal{A} = \langle \Sigma, Q, q_\lambda, F_A, F_R, \delta \rangle,$$

where:

- Σ is an alphabet;
- Q is a finite set of states;
- $q_\lambda \in Q$ is the start state;
- $F_A \subseteq Q$ and $F_R \subseteq Q$ are sets of final states denoted as *accepting states* and *rejecting states*;
- $\delta : Q \times \Sigma \to Q$ is the transition function, with inputs a state of Q and a symbol from Σ, returning a value in Q (note the only difference with NFA is the type of returned value: for a DFA is a single value and not a subset of Q).

DFAs and NFAs are representations with equivalent expressiveness, and every language can be described indifferently with one of them [43]. An analogous relation to what highlighted for languages and grammars subsists between languages and automata. Indeed, one DFA induces one and only one language, conversely, for a language there is more than one DFA.

Let us focus an fundamental property for a finite set of positive examples related to a FSA:

Definition (Structurally complete finite set of positive examples)**.** A finite set of positive examples I_+ of a language L, is ***structurally complete*** with respect to an automaton A accepting L, if feeding the set I_+ into the automaton:

- every transition of A is used to recognize at least one string in I_+;
- each $q \in F_A$ is an accepting state for at least one string in I_+.

Structurally complete sets are fundamental for grammar inference algorithms because they encode all the needed information to successfully learn the target FSA.

Among all automata compatible with a set of examples, some of them have remarkable structure and properties. For instance, the ***minimum canonical automaton*** [41], is the automaton with the minimum number of states among automata recognizing the same language, and, furthermore, it is unique for that language (up to a different states labelling). Into the grammar inference process the minimum canonical automaton is interpreted as the target automaton of a search problem, in terms of a *state space search*, a typical problem in artificial intelligence.

Maximal Canonical Automaton

The ***Maximal Canonical Automaton (MCA)*** over (I_+) is the automaton accepting every string in I_+ and nothing else. Its structure is composed by a subtree for each examples, leading to a nondeterministic automaton due to strings sharing the same prefix. Formally:

Definition (Maximal Canonical Automaton)**.** Let $I_+ = \{u_1, ..., u_M\}$ a set of positive samples, where $u_i = a_{i,1}...a_{i,|u_i|}$, with $1 \leq i \leq M = |I_+|$. It is named ***MCA related to*** I_+ a 6-tuple:

$$MCA(I_+) = \langle \Sigma, Q, q_\lambda, F_A, F_R, \delta \rangle,$$

where:

- Σ is the alphabet used from I_+;
- $Q = \{v_{i,j} | 1 \leq i \leq M, 1 \leq j \leq |u_i|, v_{i,j} = a_{i,1}...a_{i,j}\} \cup \{\lambda\}$;
- $q_\lambda = \lambda$;
- $\delta(\lambda, a) = \{v | v = a_{i,1}, a = a_{i,1}, 1 \leq i \leq M\}$;
- $\delta(v_{i,j}, a) = \{v_{i,j+1} | v_{i,j+1} = v_{i,j}a_{i,j+1} \text{ e } a = a_{i,j+1}\}, 1 \leq i \leq M, 1 \leq j \leq |u_i| - 1$;
- $F_A = I_+$;
- $F_R = \emptyset$.

An instance of MCA is reported in Fig. 3.2, that is the $\mathcal{MCA}(\mathcal{I}_+)$ defined for the positive samples I_+:$\{a, aaa, abab, bba\}$, over the alphabet $\Sigma = a, b$. Observe that despite *a* ed *aaa* share a common prefix, they have separated and independent branches within the automaton. To address these redundancies, another canonical automaton is defined, namely Prefix Tree Acceptor (PTA)(I_+). In order to construct this DFA let us first introduce, the concept of *quotient automaton.*

Quotient automaton

Within the mathematical combinatorial set theory and the universal algebra, different properties and techniques have been examined about sets and their partitions [9]; some useful statements and results are borrowed for this work.

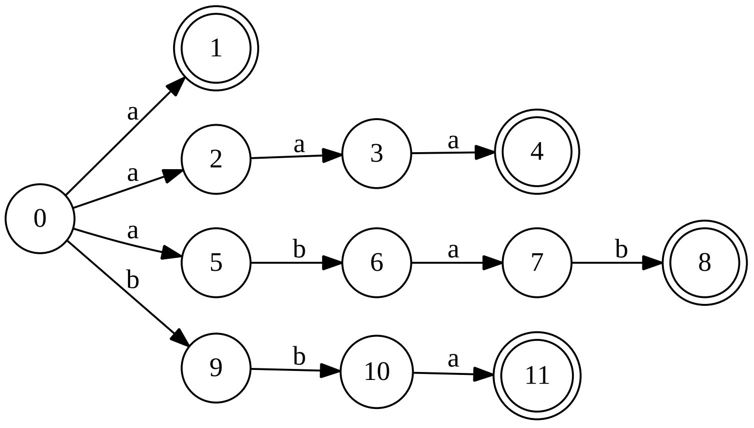


Figure 3.2: $\mathcal{MCA}(\mathcal{I}_+)$ for I_+:{a, aaa, abab, bba}

For a finite set S, a *partition* π is defined as the set of pairwise disjoint nonempty subsets of S, whose union is S [26].

Let s an element of S and $B(s, \pi)$ the function locating the unique *block*, or set, of π containing s. A ***quotient automaton A/π*** can be defined as a *partition* π over the automaton A [26]:

$Q' = Q/\pi = \{B(q,\pi)|q \in Q\}, F' = \{B \in Q'|B \cap F \neq \phi\}$,
$\delta' : Q'\text{x}\Sigma \to 2^{Q'} : \forall B, B' \in Q', \forall a \in \Sigma, B' \in \delta'(B,a)$ iff $\exists q, q' \in Q, q \in B, q' \in B'$ e $q' \in \delta(q,a)$

where $B(q,\pi)$ identify the unique block of π containing q.
Every state of Q, belonging to the same block B for the partition π, becomes the target element for a *merging* operation.

Prefix Tree Acceptor

The PTA of I_+, denoted PTA(I_+), is derived from MCA(I_+) by merging common prefixes; it is the quotient automaton MCA(I_+)/π_{I_+}, where the partition π_{I_+} is defined as follow:

$$B(q, \pi_{I_+}) = B(q', \pi_{I_+}) \text{ iff } Prefix(q) = Prefix(q')$$

The resulting automaton is a DFA, by construction; it is similar to MCA(I_+), with the difference that states sharing the same prefixes are merged.

Consequently, observe that in Fig. 3.3 the strings *a* ed *aaa*, now, have a common state with the same prefix, unlike to MCA(I_+).

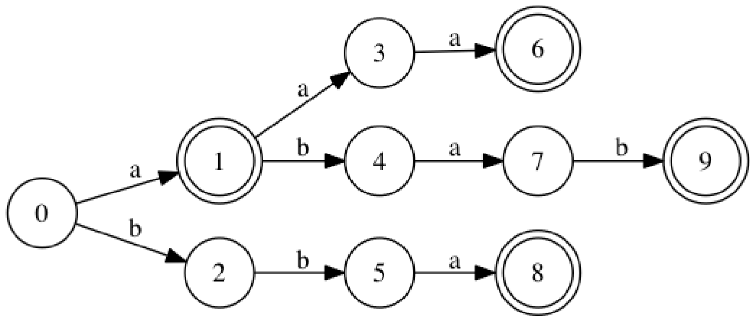


Figure 3.3: $\mathcal{PTA}(\mathcal{I}_+)$ for I_+:{a, aaa, abab, bba}

Universal Automaton

Finally, the UA denotes the ***universal automaton***, that accepts all the strings defined over the alphabet Σ: $L(UA) = \Sigma^*$, easily built by merging together all the states of PTA.

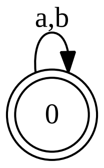


Figure 3.4: UA over alphabet *a,b*.

3.2 Grammar induction

Once introduced the necessary formalism and mathematical tools, let us proceed to pose a well-defined inductive problem and its main components. Each of them is directly related to the grammar induction context; in particular, the learning paradigms, the search space, and the techniques to explore it are examined.

To state an inductive inference problem, five items (or features) should be specified [3]:

1. Concept space;
2. Hypothesis space;
3. Presentation of examples;
4. Inference methods;
5. Success criteria.

Concept space

The concept space defines the implicit *target* space, and the solutions for the inductive problem. For grammar induction, it is a class of languages from Chomsky hierarchy; among these several works have adopted regular languages, or often the contex-free ones.

Hypothesis space

It is the space whose elements are manageable representations of concepts, therefore for each element of this set corresponds at least one element of concept space. Some instances of hypotheses are regular grammars, regular expressions or finite state automata. Usually the relation between representations (finite automata) and concepts (languages) is many-to-one.

Presentation of examples

Different kinds of examples are employed in inductive algorithms and their presentation leads to different approaches.

Meta-information about examples leads to distinguish two information presentations, *learning from text* and *learning from informant* [37]:

- ***from text:*** when the only supplied examples are positive, i.e. the examples to be accepted;
- ***from informant:*** when either positive and negative examples are used, i.e., examples to be accepted and rejected.

There are three main ways of providing examples to the inference algorithm;

- by ***complete presentation*** (or ***positive***): an infinite sequence of examples (complete or positive) is iteratively provided to the inference algorithm.
- by ***Given-Data***: inference methods retrieve a full finite set of examples, that are provided at the beginning of the process.
- by ***informant***: examples are iteratively chosen by the inferential algorithm itself by a so called ***teacher***, or ***oracle***.

  An Oracle is generally able to answer two kinds of questions: *membership query* and *equivalence query*; the former question asks whether a submitted string belongs to the target languages or not, while the latter asks whether an element of the hypotheses space, typically an automaton, is equivalent to target representation; if not, the oracle replies with a significant counterexample.

Inference methods

Since the beginning of the inductive inference as an autonomous field, two different lines of study arose: one concerning general and theoretic properties of the inference methods, and another which investigates practical inference strategies.

Tools coming from the former line of research are purely theoretical and allow to investigate the actual limits of inductive inference. However, they are not practical algorithms due to efficiency and complexity issues.

One of the most relevant algorithms in this field is the *identification by enumeration*, an abstract method which systematically explores the space of possible concepts compatible with the given samples. Its importance depends mainly on the following conjecture [77]:

Theorem 1 (Induction by enumeration)**.** *Every class of recursive languages, identifiable in the limit by any inductive inference algorithm, is also identifiable by an inference algorithm making use of induction by enumeration.*

Unfortunately, algorithms inspired to induction by enumeration are computationally infeasible due to the size of hypotheses space.

Rather than enumerating every hypothesis, practical strategies execute some symbolic or numeric manipulations in order to address time complexity issues. In particular, one of the most investigated approach propose a typical search problem inside a state space search [74] and exploits a conveniently featured automata space to drive exploration toward the target DFA, by heuristic evaluations [26][31].

Two predominant kinds of strategies have been used:

- greedy approaches select, iteratively, the smallest hypothesis compatible with analyzed examples, ensuring to achieve the target solution if the finite set of examples is structurally complete. Among those approaches, notable are *RPNI* and *L** algorithms;
- gn the contrary, heuristic algorithms conduct evaluations of slight changes on current hypothesis, in order to select the ones having higher (or lower) score. Some of them are *EDSM* algorithm, and the evolutionary inspired algorithms such as the genetic algorithm *GIG*. Formally:

Successful criteria

The most relevant learning paridigms is "**Identification in the limit**", defined by E. Mark Gold [37].

Learning is considered an infinite process. At each trial, an example is provided to the learner, that uses it to generate a new hypothesis; identification in the limit is achieved if, for any example presentation, the learner produces only a finite number of wrong hypotheses, and converges to the target solution in a finite number of steps. However, the learner is not able to identify the final correctness for the current hypothesis, since a counterexample might appear an arbitrarily time long after.

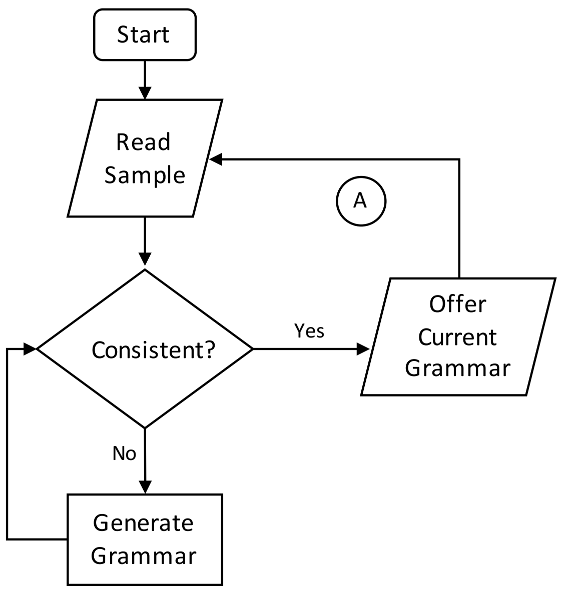


Figure 3.5: *Gold's "Identification in the limit" [63]. Note that no "stop" symbol is present, so the process is infinite. If at some point, learner enters into "A" loop and never leaves it, language has been identified in the limit.*

Definition (Identification in the limit)**.** Given a potentially infinite example sequence $e_1, e_2, e_3...$ of a target language $\mathcal{L}$, it is supplied to an inference algorithm denoted $\hat{I}$. If, after a finite numbers of trials, the generated hypothesis is always the same and is a description of $\mathcal{L}$, $\mathcal{L}$ is said to be *identifiable in the limit* by $\hat{I}$.
In others words, if h_i is the i-th guess, $\exists i \in \mathbb{N} : i_m$ is a description of L and $i_m = i_{m+1} = i_{m+2} \cdots$

In his seminal paper *"Language identification in the limit"* [37], Gold introduces by contrast what identification in the limit is not.

The two alternative stronger models are:

- *finite identification*: in this case the learner has to *signal* the correctness after a finite number of steps;
- *fixed-time identification*: in this latter paradigm correctness has to be achieved after a previously fixed number of steps.

Some alternative definition of identification have been proposed. One is the "**Behaviorally correct identification**"; unlike the already seen process, after a non specified trial it allows formal changes on hypothesis, but only if such modifications lead to equivalent representations. It is a less restrictive criterion than identification in the limit, introduced for the first time in [6, 13].

More recently the ***PAC-learning*** paradigm [76] was defined, in order to achieve only a partial identification of target language, under some accuracy and confidence parameters.

3.2.1 Limits of inductive inference

Exploiting the defined theoretical tools, several works investigated language inference limits and capabilities. Some remarkable studies by Gold [37], successfully demonstrates some theoretic results which affect every language inference approaches; thus most of the terminology nowadays used is borrowed from these studies.

To introduce some of these results, consider the class of *super-finite languages* [37] defined as:

Definition (Class of super-finite languages)**.** A class of languages is super-finite if contains all finite languages and at least one infinite language.

A first theoretic limit established by Gold [37] concerns the more constrained class of languages in a text presentation setting:

Theorem 2. *No super-finite class of languages can be identified in the limit, from text* [4]*.*

Therefore no regular language can be identified in the limit by every inferential algorithm, with a text presentation. Consequently, this theorem affects all the classes of languages due to the nested structure of the language hierarchy: indeed, all classes include regular languages, hence at least one infinite language.

A way to work around this limit is adding more information, such as negative examples. Unfortunately, also with the added information not all the classes of languages can be identified in the limit, as showed [37]:

Theorem 3. *The whole class of recursive language is not identifiable in the limit from an informant.* [5]

[4]Recall that a text presentation is composed only by positive examples

[5]Recall a *presentation from an informant* is a complete presentation, i.e. labelled positive and negative examples.

It remains to define the residual classes of languages identifiable from an informant:

Theorem 4. *The class of primitive recursive languages is identifiable in the limit from an informant.*

The class of primitive recursive languages includes context-sensitive languages, hence its proper subset of classes: contex-free and regular languages. Therefore, the previous theorem lays the theoretic foundations for inductive algorithms over the mentioned classes of languages.

Class	*Learnable from an informant?*	*Learnable from text?*
Recursively Enumerable	no	no
Primitive Recursive	yes	no
Context-Sensitive	yes	no
Context-Free	yes	no
Finite State	yes	no
Finite Cardinality	yes	yes

Table 3.1: Learnable classes of languages

One of the main issues in grammar inference is to find a minimum consistent DFA for supplied examples, due to its simplicity.

To build an automaton compatible with given examples is a trivial task; for instance, think of the PTA, a perfect separator between positive and negative sets of strings. On contrary, searching the *minimum* consistent automaton is hard task [36], which has given rise to a rich field of studies attempt to cope with this problem:

Theorem 5. *Identifying the minimum deterministic consistent automaton from informant is a NP-complete problem.*

Generalization criterion

Let us dwell on why attention has been directed toward minimum consistent automaton.

Since there are infinite DFA consistent with examples [41], an *inductive bias* needs to be introduced in order to (a) select the best solution between all acceptable DFAs and (b) reduce the numbers of elements in the automaton space to be checked. In order to cope with (a), an extra bias is added to select a target typology; later (Par. 3.3) a set characterization is pointed out to specify a well-posed automata search space (b).

One of the main inductive biases adopted is the ***Occam's razor principle*** [8, 18]; it states that "entities should not be multiplied unnecessarily". In an inference technique this means that the smallest hypothesis should be preferred; in particular, in grammar inference the Occam principle leads to choose the automaton with fewer states consistent with samples [64]. For instance, note that among the compatible DFAs there is the mentioned PTA, overfitting the examples, in which no kind of generalization took place; on the contrary, the Occam principle prompts to look for the smallest DFA, consistent with data, where a more compact hypothesis is a clue of patterns discovered behind examples.

A more sophisticated version of the Occam's razor principle is the ***Minimum Description Length (MDL)*** [39], with a more effective measure of simplicity. It states that simplicity depends also on the efficiency of hypothesis to explain data, other than its size. Thus, simplicity of a hypothesis H with respect to data D is the sum of the size of hypothesis $L(H)$ and the size of data D encoded by H, and principle of MDL select the H minimizing this sum:

$$argmin_{H \in \mathcal{H}} \quad L(H) + L(D|H),$$

where $\mathcal{H}$ is the hypothesis space.

MDL inserts the size of encoded data to avoid overgeneralization, which might lead to very compact hypotheses, but ineffective to detect the structure behind data. This is the case of universal automaton, able to generate all the strings of a languages and the ones that should not belong to it.

All the previous considerations, with regard to the difficulty to find the minimum consistent automaton, have led to conceive methods to achieve the best possible results.

3.3 Search space for automata

Grammar inference of minimum automaton consistent to $\langle I_+, I_- \rangle$ can be turned into a search problem for this DFA. Specified in this way, the target hypothesis search is a particular case of a more general machine learning setting known as *Version space approach*, defined by Mitchell in his paper "Generalization as search" [59, 20].

3.3.1 The Version Space approach

As stated before, hypotheses consistent with samples are infinite and constitute all plausible *versions* of target hypothesis. The key insight of the Version Space approach relies on a useful structure for the organization of the search space, namely a “general-to-specific” ordering of hypotheses; which is a natural structure existing for any concept learning problem, hence also for the DFA induction. Exploiting such ordering, learning algorithms can explore the infinite hypothesis space without explicitly enumerating every element of it [60].

Definition (Version space)**.** The **version space**, denoted as $VS_{H,I}$, with respect to hypotheses space H and training examples I, is the subset of hypotheses from H consistent with the training examples in I:

$$VS_{H,I} \equiv \{h \in H \mid \mathit{Consistent}(h, I)\}.$$

Intuitively, the general-to-specific ordering is based on constraints characterizing the hypotheses: fewer constraints entail more general hypotheses, and vice versa. In automaton search, there is an analogous structure composed by $MCA(I_+)$ as the most “constrained” consistent DFA, and the minimum consistent automaton as the one with fewer constraints.

Mitchell make a distinction between the set S of maximally specific solutions and the set G of maximally general solutions. In DFA induction, S is constituted by $MCA(I_+)$; while G (also known as *Border Set* [26]) is not directly identifiable due to the size of the search space. An example of the border set is highlighted in Fig. 3.11.

In grammar inference process, the general-to-specific ordering is defined in terms of relation between automata and order induced on languages (Fig. 3.6). A mathematical structure, known as *boolean lattice*, is generated from this kind of relation, and its features can be exploited in order to guide an efficient space exploration.

3.3.2 The partition lattice

The search toward targed automaton is carried out more efficiently if the search space is algebraically characterized [26] [31].

Some basic items can be specified for this space:

- **Initial state:** any DFA consistent to I_+;
- **Successor function:** pairwise state merging;

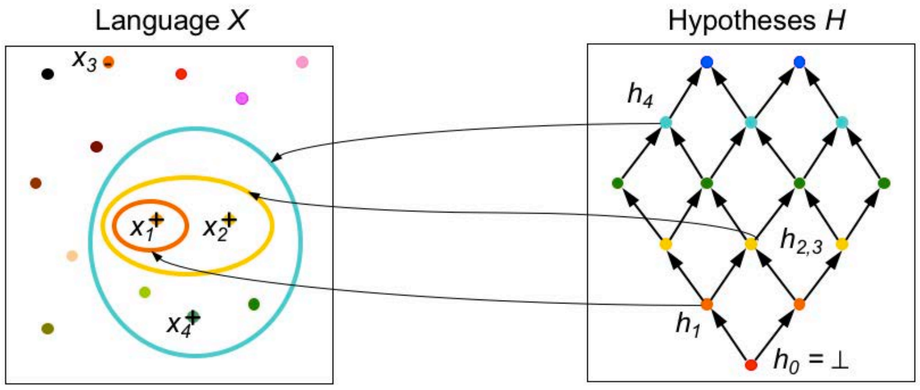


Figure 3.6: Ordering on hypothesis space and its relation to concepts space of language.

- **Goal state:** the minimum consistent automaton to I_+.

Many algebraic tools involved to characterize the state merging operation and the mentioned lattice have been studied.

To define the algebraical structure, consider two partitions[6] π_1 e π_2 over the set of states Q of an automaton $\mathcal{A} = \langle \Sigma, Q, q_\lambda, F_A, F_R, \delta \rangle$; r_i denotes the block number of the π_i partition. π_1 and π_2 are two generic partitions defined as $\pi_1 = \{B_{11}, ..., B_{1r1}\}$ and $\pi_2 = \{B_{21}, ..., B_{2r2}\}$

A *derivation operation* over these partitions is so defined:

Definition (Derivation operation)**.** The partition π_2 *directly derives from* π_1, if π_2 is constructed from π_1 as follows:

$$\pi_2 = \{B_{1j} \cup B_{1k}\} \cup (\pi_1 \backslash \{B_{1j}, B_{1k}\})$$

for some $j, k \in [0, r_1]$ with $j \neq k$.

In other words, a derivation operation is just a new arrangement of the set elements within its subset, often merging together some of these. When the set elements are the automaton states, adding some states to the same subset corresponds to merging these states, and adjusting transitions accordingly.

For instance, consider in Fig. 3.7 a general Boolean lattice generated with an initial four blocks partition, composed by all the combinations obtained merging the partition blocks.

[6]A partition of a set is a grouping of the set elements into non-empty subsets, called *blocks*, so that every element is included in one and only one of the blocks

Automata built by merging states are more general DFAs than the starting ones, reflecting a “general-to-specific ordering” between automata and thus between languages.

Formally, it is said that derivation operation establishes a *partial order relation*[7] $\preceq$ over the set of partitions of automaton $\mathcal{A}$, $\boldsymbol{P(\mathcal{A})}$, whose transitive closure are indicated with $\ll$. For the previous case $\pi_1 \preceq \pi_2$ (Def. 3.3.2), and for extension to the automata: $\mathcal{A}/\pi_j$ *derives from* $\mathcal{A}/\pi_i$, where $\mathcal{A}/\pi_j$ is also called the ***quotient automaton*** (Fig. 3.8).

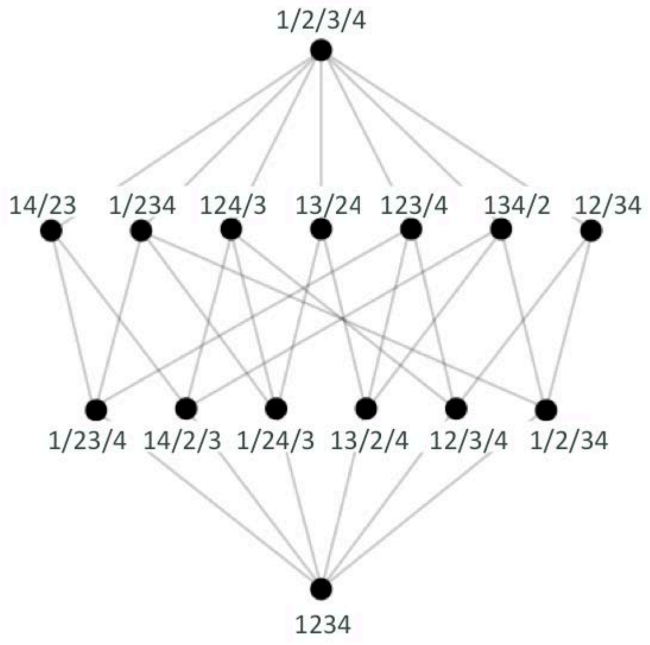


Figure 3.7: Boolean lattice with four initial partitions.

The generation of quotient automata, by construction satisfies the *inclusion language* property:

Theorem 6 (Language inclusion property)**.** *The quotient automaton A/π_j, built by application of $\ll$ on A/π_i, identify a more general language $L(A/\pi_j)$ including $L(A/\pi_i)$:*

$$A/\pi_i \ll A/\pi_j \quad \textit{if} \quad L(A/\pi_i) \subseteq L(A/\pi_j)$$

[7]In a set *partially ordered* S there are some elements $x, y \in S$ for which is not possible to state neither $x \preceq y$ or $y \preceq x$. In the automata lattice these elements are DFAs built by the same number of derivations.

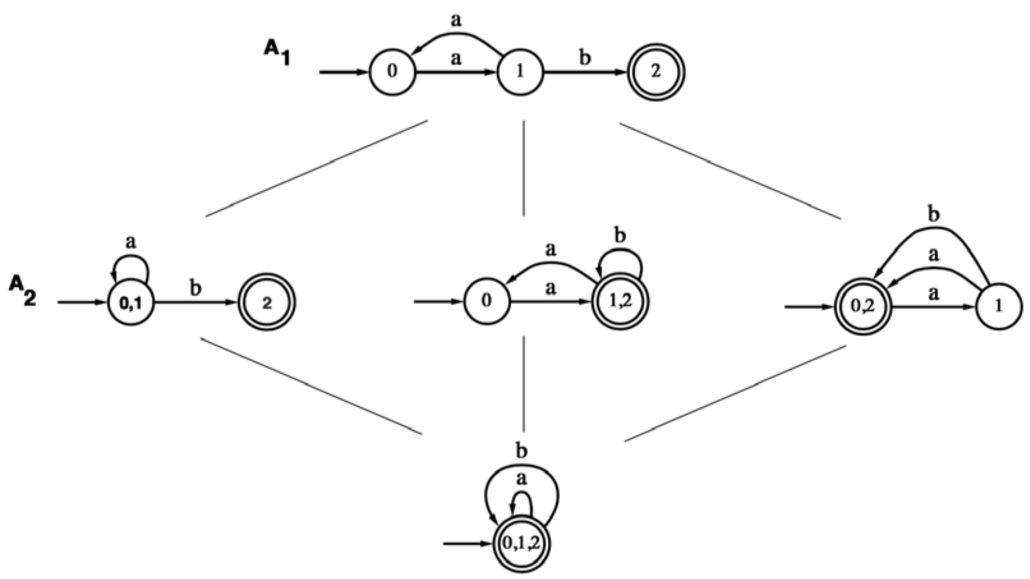


Figure 3.8: An examples of derived automata [26].

An example of initial automaton in the lattice is reported in Fig. 3.9; there is a one-to-one correspondence between partition blocks and automaton states. Applying a merging operation, two or more states are collapsed and transitions are properly rearranged generating the automaton in Fig.3.10; with partitions of more than one states. In terms of partitions, the starting automaton could be encoded as:{{0}{1}{2}{3}{4}{5}{6}{7}{8}{9}}; after the derivation operation, that is merging states 6 and 3, a new {3,6} block is generated. The final partition is: {{0}{1}{2}{3,6}{4}{5}{7}{8}{9}}.

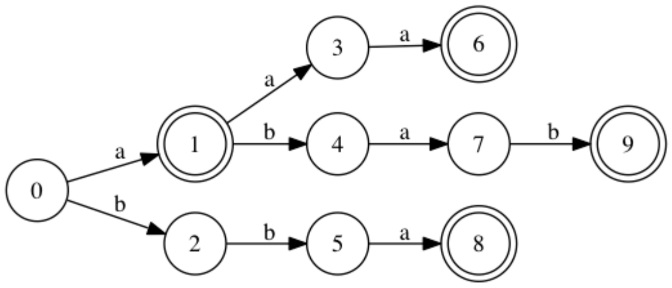


Figure 3.9: $\mathcal{PTA}(\mathcal{I}_+)$ for I_+:{a, aaa, abab, bba}.

The order relation established by state merging operations might have two different effects on automata. A first effect, is the generalization obtained by

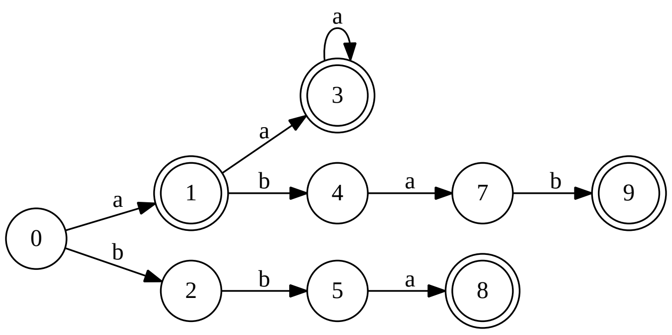


Figure 3.10: DFA $\mathcal{A}$ derived using I_+={a, aaa, abab, bba}.

merges that extends the recognized language, preserving the consistent with regard to $\langle I_+, I_- \rangle$ examples. A second effect is due to merges reducing the number of states but entailing no language generalization; this is the case of automaton minimization [43], that removes the equivalent states, leading to maintain the same language. In the former case, merges lead towards mini*mal* automaton, while in the latter case they lead towards minim*mum* automaton.

The set of all the possible partition for $\mathcal{A}$, *P($\mathcal{A}$)*, together with a partial order relation, define a ***boolean lattice Lat($\mathcal{A}$)***. Elements of lattice are the quotient automata obtained by merging the $\mathcal{A}$ states. $\mathcal{A}$ is called the null element, while the universal automaton UA is the universal element. A qualitative depiction of automaton lattice is reported in Fig. 3.11.

The depth of a general automaton A/π in $Lat(\mathcal{A})$ is $N - r(\pi)$, where N is the number of states of $\mathcal{A}$ and $r(\pi)$ number of block for the π partition. In particular, depth for $\mathcal{A}$ is 0 and for UA is equivalent to $N - 1$.

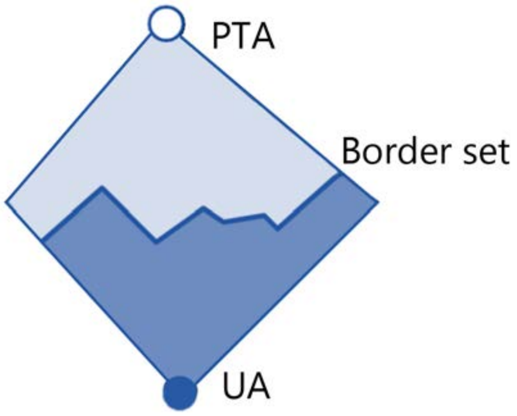


Figure 3.11: Boolean lattice.

3.3.3 An inductive bias for the lattice: structural completeness

In order to decline grammar induction as the discovery of an unknown automaton $\mathcal{A}$ for the observed samples $\langle I_+, I_- \rangle$, two issues have to be addressed: how to reduce the size of lattice without affecting the presence of target automaton $\mathcal{A}$? which are the necessary examples ensuring that $\mathcal{A}$ is in the lattice?

To deal with the former question, recall that there are infinite automata compatible with I_+; however, not all of these are meaningful. Consider for instance, a set of examples where the only observed letters are a and b whereas the alphabet is composed by a, b and c. Except for some external constraints, it is a legitimate bias to elaborate solutions adopting only used letters.

Furthermore, to deal with latter question, examples should implicitly encode all the needed information to induce states and transitions of target automaton, in such a way that the only and sufficient automata to be checked are those strictly consistent with the examples.

Therefore the candidate automata are (only) those for which the following conditions hold [41]:

- each transition is used at least once while parsing examples;
- are consistent with examples;
- each final state is involved as acceptor from at least one sample.

Formally, these features are achieved by *structurally complete* sets of examples (Par. 3.1.3).

Dupont *et al.* [26] demonstrated that such sets ensure several interesting lattice properties concerned with target automaton. For the lattice defined from $PTA(I_+)$ and a structural complete set of examples I_+ with respect to minimum consistent automaton $\mathcal{L}$, they showed that:

Theorem 7. *Given a set of positive examples I_+ for a regular language $\mathcal{L}$, let $\mathcal{A}_M(\mathcal{L})$ be the minimum consistent automaton recognizing $\mathcal{L}$. If I_+ is a structurally complete set with respect to $\mathcal{A}_M(\mathcal{L})$, then $\mathcal{A}_M(\mathcal{L})$ is in $Lat(PTA(I_+))$.*

Border Set

The target automaton can be defined through the concept of *Border set*, a peculiar set concerning the search within the automata lattice. Preliminarily, it is necessary to introduce the definitions of *antistring* and *automaton at maximal depth* [26]:

Definition (Antistring)**.** An antistring in a lattice is a set where every elements is not related by $\preceq$ with any other element in the set.

Definition (Automaton at maximal depth)**.** In a lattice of automata, an automaton $\mathcal{A}$ is at maximal depth if there is no automaton $\mathcal{A}'$ derivable from it, such that $\mathcal{A}' \cap I_- = \emptyset$

Definition (Border set)**.** The Border set $BS_{PTA}(I_+, I_-)$ is the antistring in $Lat(PTA(I_+))$, where each automaton is at a maximal depth.

Therefore the target automaton might also be defined as the smallest DFA in a border set $BS_{PTA}(I_+, I_-)$.

Note that $BS_{PTA}(I_+, I_-)$ divides the lattice into two regions, based on the compatibility of automata to negative samples: one composed by automata $\langle I_+, I_- \rangle$ consistent, the other one only $\langle I_+ \rangle$ consistent. These regions are graphically shown in Fig. 3.11.

Lattice size

Despite Theorem 7, that ensures the presence of minimum consistent automaton in $Lat(PTA(I_+))$, unfortunately, an exhaustive search remains prohibitive due to the size of lattice.

Indeed, the number of elements inside the lattice is finite and equal to the ***Bell number***. If E is a set of n elements, the number of partitions of E is the Bell number calculated as:

$$\omega(n+1) = \sum_{p=0}^{n} \binom{n}{p} \omega(n) \quad \text{with} \quad \omega(0) = 1$$

It grows very fast, e.g. the lattice related to $PTA(I_+)$ with 16 states is composed by $\omega(16) = 10.480.142.147$ elements.

3.3.4 Search strategies

Several strategies to explore the lattice have been developed to address the space search size issue.

State merging algorithms

The *state merging methods* (also called *equivalents-based methods* [19]) exploit a defined equivalent relation (or function) able to evaluate whether two states are equivalent or not [62]. They apply this equivalence repeatedly to a *canonical automaton* (usually the above mentioned $PTA(I_+)$) in order to reduce the number of states.

The main operations performed by these algorithms [19] are:

1. definition of equivalence relation;
2. building of canonical automaton (e.g.: $\mathcal{PTA}(\mathcal{I}_+)$);
3. automaton reduction applying equivalence criterion;
4. iteration of third step until required.

The canonical automaton is a DFA with no assumptions on data, which (over)fits the data exactly matching positive and negative examples; the generalization takes place by applying the equivalence criterion. New automata have fewer states and identify languages with a higher cardinality with respect to languages for the starting automaton.

The previous operations globally perform a search in the space of possible sequences of state merging. A sketch of this kind of search is in Fig. 3.12 and Fig. 3.13.

Among these methods, the *RPNI* algorithm [62] performs an exhaustive depth-first search; it was improved by adopting a heuristic to drive the search, giving rise to the state-of-the-art algorithm known as *EDSM* [50]. To deal with noise and distribution over strings, some statistical inference algorithm have been designed; among these the *Blue** [70] algorithm modifies EDSM to exploit sample frequencies.

Genetic algorithms

An optimization approach has been designed through *Genetic algorithms* [42]. One of the most relevant attempts is the *GIG* method [27]. Unlike previous algorithms, it begins the process with $MCA(I_+)$, and does not perform derivation operations systematically, but exploits genetic operators designed on purpose. Consequently, a search inside the lattice is performed, driven by a *fitness function* though the evolutionary paradigm.

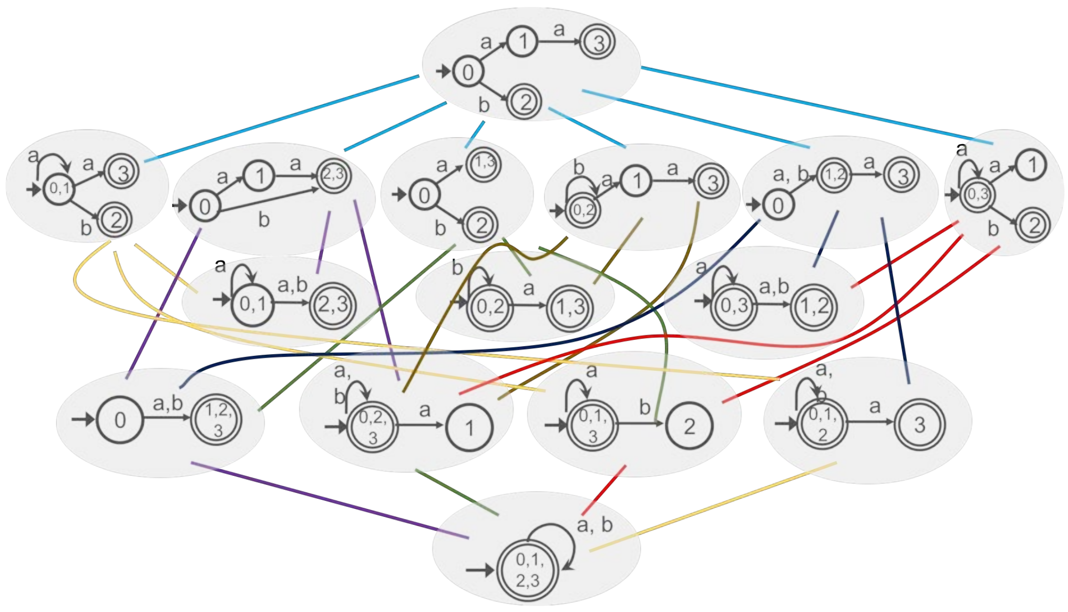


Figure 3.12: Lattice for $\mathcal{PTA}(\mathcal{I}_+)$ for I_+={aa, b}.

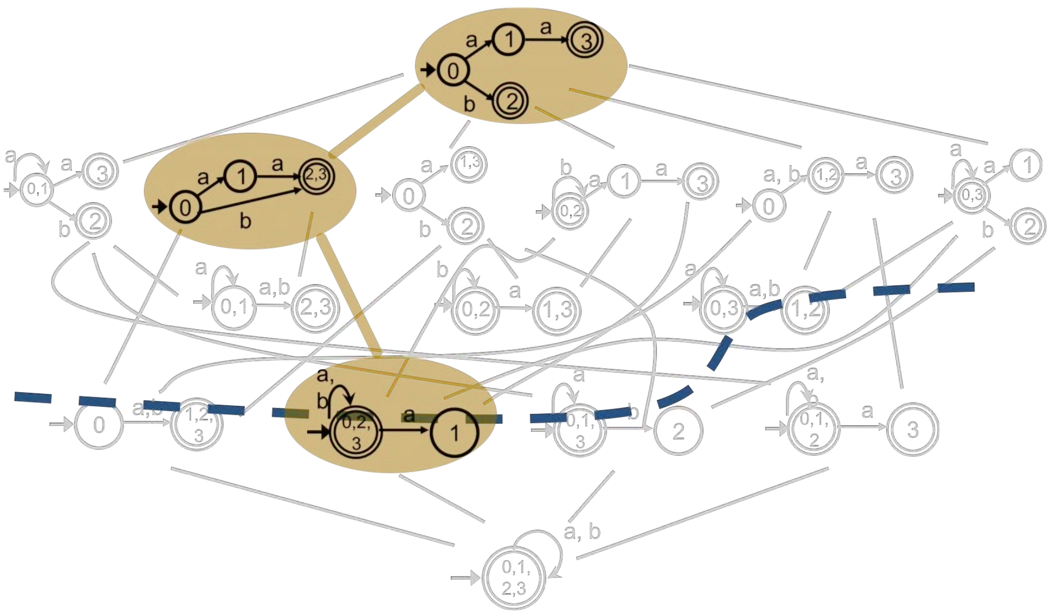


Figure 3.13: Sequence of merges towards smallest DFA into the border set, in a lattice for $PTA(I_+)$ for I_+={aa, b}.

Active Learning

Finally, an alternative paradigm is known as *Active Learning.* States and transitions are progressively added to the universal automaton UA, through *membership* and *equivalence query*, and counterexamples provided by an *oracle*; a fundamental algorithm for this paradigm is L^* [1].

3.4 Learning algorithms

A deeper analysis of inference algorithms shows how they depend on environment settings; in this sense, some relevant items are the features of examples such as type (positive or negative) or meta-information (counterexample), how they are supplied, and the data structures involved during automata inference.

Informed learners

All the inductive algorithms take as much advantage as possible from the type of examples.

A deeply investigated setting [19] is *learning from informant* (Par. 3.2), where data are positively or negatively labelled depending on whether they belong to the target language: $I = I_+ \cup I_-$. Every examined algorithm in this work is referred to this setting, theoretically supported by Theorem 4 which ensures identification in the limit, unless complexity issue in real scenarios occur.

Consider the following basic scenario as supporting for this setting. A first positive example presented to the inference algorithm is a, thus the generated hypothesis automaton is made of one state (Fig. 3.14a). Now, a second positive example is provided, aa: which should be the new automaton? (Fig. 3.14b) There are two possibilities: a) an automaton with an added state (Fig. 3.14c), or b) a more "risky" automaton using the existing state in a loop (Fig. 3.14d). If the preferred strategy is to add a state, coherently a new state will be added for any further examples, ending up with an enormous automaton able to parse only the learning data where no generalization has been done. Therefore the second strategy is crucial and the principle behind it is the mentioned Occam's razor (Par. 3.2.1). However, an issue arises if, in the previous scenario, the target language does not accept all the examples composed by more than a fixed number N of repeated *"a"*; with just one negative example (for instance, a negative strings with $N + 1$ repeated *"a"*), the overgeneralization towards an infinite number of misclassified examples is avoided.

Such considerations lead to prefer the settings with both positive and negative samples when achievable.

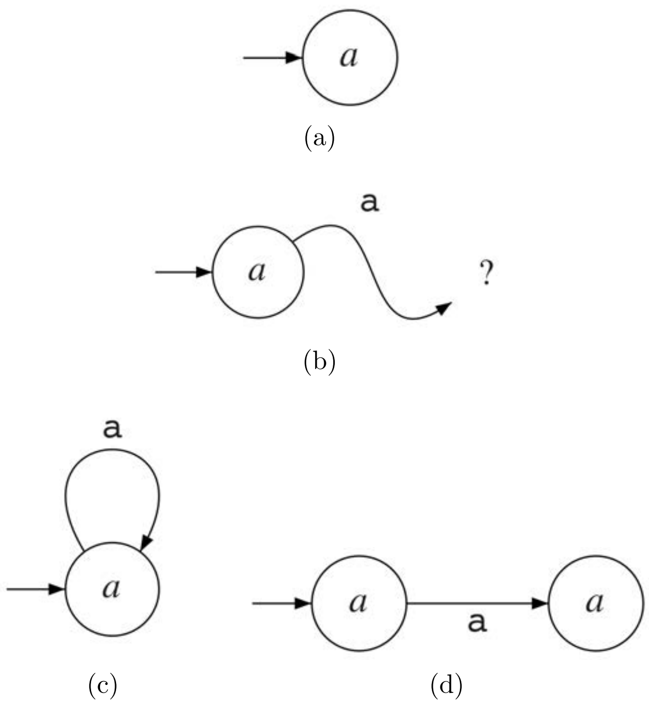


Figure 3.14: Different strategies to parse a set of strings.

Red-blue framework

Evaluating all the pairs of state candidates to be merged leads to a combinatorial explosion. Several approaches have been tried in order to limit the number of elaborated pairs without compromising the learning process.

A framework proved to be effective is the ***Red-Blue framework***, here employed in its most notable version defined during the Abbadingo competition [50]. The Red-blue framework was introduced as bland criterion of locality for the merges to be assessed. Such criterion has proved its effectiveness in automata with a large number of states, reducing the number of computations needed without irreparably affecting the inference search for the minimum consistent automaton.

During each iteration, the framework divides the states in three categories:

- The Red states which are the confirmed states for the final automaton; they will not be further processed.

- The BLUE states are the current candidates to be confirmed or collapsed into an existing RED state; they will be evaluated soon by the inference algorithm.
- Finally, the WHITE states which are currently ignored states; they will become BLUE or RED.

Only the pairs of states composed by a BLUE and a RED state are evaluated; whereas the WHITE states are ignored. The algorithm ends when the WHITE and BLUE states have been analysed and only RED states remain.

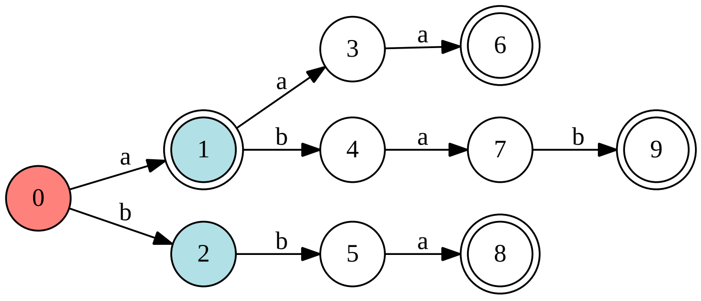


Figure 3.15: $PTA(I_+)$ in the Red-blue framework for I_+={a, aaa, abab, bba}.

DFAs produced by state merging algorithms within the Red-blue framework are composed by a set of RED states encompassed from a BLUE fringe: these BLUE states are candidates to become RED either by promotion or by merging with an already RED state. An instance of $PTA(I_+)$ in the Red-blue framework is reported in Fig. 3.15.

Several independently developed algorithms exploit the Red-blue framework to increase their efficiency; the most relevant of them are summarized in the following.

3.4.1 RPNI - Regular positive and negative inference

RPNI is one of the most notable state merging algorithms; several variants have been made, and among these the state-of-the-art algorithm EDSM.

RPNI belong to the paradigm of Identification in the limit, as demonstrated by its authors [62], if the training set encodes sufficient information to be a so-called *characteristic samples set* for the target language:

Definition (Characteristic samples set)**.** A complete set: $I = I_+ \cup I_-$ is ***characteristic*** for a language $\mathcal{L}$, if:

- positive samples set I_+ is structurally complete (def.3.1.3) for the target automaton A recognizing $\mathcal{L}$;
- negative samples set I_- prevents any merge between states of $PTA(I_+)$ which are not equivalent respect to the target.

The overall effect of RPNI merges is a depth-first search inside the lattice built on $PTA(I_+)$,$Lat(PTA(I_+))$. It starts from the minimum elements of lattice $PTA(I_+)$ adopting an equivalence relation between states, which leads to greedily accept the first merge consistent with $\langle I_+, I_- \rangle$.

Following the Red-blue framework, the policy is to evaluate only the merges between blue and red states; such evaluations are performed following the state creation order in the $\mathcal{PTA}$, that is a lexicographical order of I_+ examples . The adopted order is a significant feature of the learning algorithms, EDSM and other proposed variants improve their performance right through a different merge order.

If the DFA obtained from a merging results consistent with at least one negative example, it is rejected; and if for a blue state every possible merge is rejected, the state is promoted to red, thus becoming a definitive state of final automaton.

The algorithm can be outlined as in the following:

1. a canonical automaton $PTA(I_+)$ is generated; it is the starting elements for the lattice search;
2. for every blue state, every possible merge with a red state is evaluated, checking the consistency to I_-:
   - as soon as a compatible merge is detected, the process continue with the obtained automaton; a new set of current blue states is computed;
   - if no compatible merge have been discovered, current blue state is promoted to red state;
3. Repeat from step (2), until some blue state exists.

Note that, at step (2), it is sufficient to evaluate DFA consistence only with negative examples I_- due to the language inclusion property (Theorem 6); it ensures that every automaton obtained by state merging is consistent with the I_+ positive examples involved to built the starting canonical automaton.

RPNI Algorithm

```
Data: $I = I_+ \cup I_-$
Result: DFA $\mathcal{A}$
$\mathcal{A} \leftarrow$ Build-PTA($I_+$);
$\textsc{Red} \leftarrow \{q_\lambda\}$;
$\textsc{Blue} \leftarrow \{q_a : a \in \Sigma \cap \textsc{Pref}(I_+)\}$;
while $\textsc{Blue} \neq \emptyset$ do
    CHOOSE($q_0 \in \textsc{Blue}$) ;
    $\textsc{Blue} \leftarrow \textsc{Blue} \setminus \{q_b\}$ ;
    if $\exists q_r \in \textsc{Red}$ such that COMPATIBLE(MERGE($\mathcal{A}, q_r, q_b$), $I_-$) then
        $\mathcal{A} \leftarrow MERGE(\mathcal{A}, q_r, q_b)$;
        $\textsc{Blue} \leftarrow \textsc{Blue} \cup \{\delta(q,a) : q \in \textsc{Red} \wedge a \in \Sigma \wedge \delta(q,a) \notin \textsc{Red}\}$;
    else
        $\mathcal{A} \leftarrow$ PROMOTE($q_b, \mathcal{A}$);
    end
end
for $x \in S_-$ do
    $\mathbb{F}_\mathbb{R} \leftarrow \mathbb{F}_\mathbb{R} \cup \{\delta(q_\lambda, x)\}$;
end
return $\mathcal{A}$
```

Time complexity

Denoting with $\|I_+\|$ and $\|I_-\|$, respectively the sum of lengths of each example in I_+ and I_-, RPNI algorithm has to explore $\mathcal{O}$ ($\|I_+ + 1\|$) states of $\mathcal{PTA}$; that is an upper bound for the number of calls to the *CHOOSE* function.

For every selected state from *CHOOSE*, the time complexity depends on (a) merge evaluations and (b) consistency check with I_- examples. The numbers of merges to compute (a) is bounded above by the number of states $\| I_+ \| +1$. For each of them, checking the consistency (b) to I_- has a cost with an upper bound of $\mathcal{O}(\|I_+\|\|I_-\|)$[8].

Final time complexity is $\mathcal{O}$ ($\| I_+ \|^3 \| I_- \|$), that is polynomial to the length of examples [62].

[8] The first factor $\|I_+\|$ is an upper bound to the number of states of each quotient automaton. The product $\| I_+ \| \| I_- \|$ consider also the non-deterministic automata.

3.4.2 EDSM - Evidence-driven state merging

Let us introduce the EDSM algorithm starting from the implicit and optimistic criterion affecting RPNI performance:

- a first issue concerns the adopted *order* to select and evaluate the states to be merged. RPNI directly inherits the induced order from state labeling, following the creation state order within the PTA;
- a further issue arises from the greedy behavior that entails *total priority* to the first consistent merge discovered despite potential available promotions more appropriate in a long-term strategy.

EDSM aims at addressing such weaknesses, by introducing a priority for state promotions, and a new order for the merging of states through heuristic able to evaluate more promising merges. In fact, stress tests show that choosing the first available merge was a too optimistic strategy, and many heuristics have been developed in order to detect the best merge at every iteration; moreover, consider that selecting a bad merge leads to a snowball effect compromising all future merges. However, whereas RPNI identifies in the limit, this might not be true any longer due to a possible wrong lattice exploration induced from the heuristic; fortunately such heuristic is counter-intuitive and easy to avoid.

The decrease of the merging priority and the introduction of heuristic, derive from a paradigm whereby *if nothing stops, generalize* to a more careful *if there are some good reasons, generalize* [41]. "Generalize", in this context, leads to performing a merge.

The most effective heuristic rose in the Abbadingo competition [50]; it favors the pairs of states supported by the higher number of evidence, whence the name "*evidence-driven state merging*". There are different ways to consider a clue in favor of a merge, as the competition showed. Among the different approaches, the one developed from Price and Lang achieved the best results [50], that computes a score as the sum of the number of examples ending in a same state; this heuristic was showed to be representative of merge goodness, thus the merge with higher score is selected.

The EDSM *Reference algorithm*

A first EDSM version, namely ***Reference algorithm*** [50], does not adopt the Red-blue framework, therefore at each iteration all the possible pairs of states are evaluated in order to be eventually merged. It entails a larger pool of candidate merges to choose the best ones from.

The algorithm can be summarized as follows:

1. a canonical automaton $PTA(I_+)$ is generated;
2. for each pair of states a merge score is computed;
3. if all the evaluated merges are consistent the best one is performed, on contrary a promotion is executed;
4. repeat from step (2).

Take into consideration that without the Red-blue framework, when a merge is performed none of the states is necessarily a root of a tree, requiring a more sophisticated management of merge operation entailing an higher time complexity [50].

EDSM *Blue-fringe algorithm*

In order to reduce the number of merges to be evaluated and make a merge operation simpler, Hugues Jullie introduced the Red-blue framework inside the previous algorithm developing the so called ***Blue-fringe algorithm*** [50].

As in RPNI, the red states belong to the final automaton and its children define a blue fringe where each state is a root of a subtree. The pool of candidates to merge is now limited to the blue states; it leads to compute a significantly lower number of scores, and deals with simpler merge operations because of mentioned property of blue fringe.

The renovated algorithm might be outlined as follows:

1. a canonical automaton $PTA(I_+)$ is generated, the root state is red colored and its children of blue;
2. a score for every pair of blue-red states is computed ;
3. if a blue state has no consistent merge, it is promoted to red. If more than one blue state have no compatible merge, the shallowest is selected. Then restart from step (2);
4. if no blue node has to be promoted, the merge with higher score is performed. Then restart from step (2).

The process terminates when no blue state is available.

Follows the *Blue-fringe algorithm* pseudocode:

EDSM - Blue-fringe Algorithm

```
Data: $I = I_+ \cup I_-$
Result: DFA $\mathcal{A}$
$\mathcal{A} \leftarrow$ Build-PTA($I_+$);
Red $\leftarrow \{q_\lambda\}$;
Blue $\leftarrow \{q_a : a \in \Sigma \cap \text{Pref}(I_+)\}$;
while Blue $\neq \emptyset$ do
    promotion $\leftarrow$ false;
    bs $\leftarrow -\infty$;
    for $q_b \in$ Blue do
        if not promotion then
            atleastonemerge $\leftarrow$ false;
            for $q_b \in$ Red do
                $sc \leftarrow$ COUNT(MERGE($q_r, q_b, \mathcal{A}$), $S_+$, $S_-$);
                if $s > -\infty$ then atleastonemerge $\leftarrow$ true;
                if $s > bs$ then $bs \leftarrow s$; $\overline{q_r} \leftarrow q_r$; $\overline{q_b} \leftarrow q_b$;;
            end
            if not atleastonemerge then
                PROMOTE($q_b, \mathcal{A}$);
                promotion $\leftarrow$ true;
            end
        end
    end
    if not promotion then
        Blue $\leftarrow$ Blue $\{\overline{q_b}\}$;
        $\mathcal{A} \leftarrow$ MERGE($\overline{q_r}, \overline{q_b}, \mathcal{A}$);
    end
end
for $x \in S_+$ do $\mathbb{F}_\mathbb{A} \leftarrow \mathbb{F}_\mathbb{A} \cup \{\delta(q_\lambda, x)\}$;
for $x \in S_-$ do $\mathbb{F}_\mathbb{R} \leftarrow \mathbb{F}_\mathbb{R} \cup \{\delta(q_\lambda, x)\}$;
return $\mathcal{A}$
```

The new management of merge order and priority of promotions are direct observable comparing the EDSM and RPNI pseudocode.

In EDSM the state merging is not performed as soon as possible, but after an overall evaluation over every possible merge. Indeed, unlike RPNI, EDSM performs the definitive merge with the function *MERGE*[9], external to the

[9]It is that inside the last if-statement.

cycle on blue states where the best pair was selected.

The priority of state promotion despite available merges is observable by looking at the position of *PROMOTE* function; unlike RPNI, it is already inside the computation of only potential merges. If during the merge computations, a blue state is detected as incompatible with any red states it is immediately promoted and cycle stopped, then the cycle restart from the new DFA. Note, once again, that such promotion is performed in spite of the fact that some merges were available.

From the Abbadingo competition the "reference algorithm" resulted "slightly more effective" [50] than "blue-fringe algorithm", due to the larger pool of candidate merges, entailing an higher time complexity. However, some competition problems have received a better solution with the blue-fringe algorithm, denoting that a larger pool is not the only critical feature. In scenarios where time complexity is not a crucial component, previous considerations support a strategy that combines the two approaches to choose the best solution.

3.4.3 Blue*

Many algorithms have been developed in order to address these relevance issues in the case of *numerical* setting [60]; however, coping with learning from unbounded *symbolic* sequences remains an open challenge, as in the case of grammatical inference. Relevance of examples is achieved by considering their frequency information, i.e., a distribution over the example set.

EDSM is an *exact learning algorithm* because the built DFA fits positive and negative examples without errors or approximations. Furthermore, if adopted heuristic does not involve a wrong lattice exploration (Par. 3.4.2), the algorithm is able to identify in the limit the minimum consistent automaton if the learning sample is a *characteristic samples set* (Par. 3.4.1).
However, in a real application scenario some limitations become crucial. In fact, nearly always the target automaton is not available and it is not possible to verify if a characteristic set is within the learning sample. Secondly, if data are noisy, the final DFA is significantly compromised because automata are known to be sensitive to noise, which usually yields a bigger DFA with low accuracy rate. Such considerations lead to avoid overfitting examples, as well as to exploit their distribution to detect which are the relevant ones.

Therefore, EDSM algorithm has been further improved with an advanced management of relevant and noisy data; this gave rise to *Blue* algorithm* [70], presented in the following.
Blue* is based on a clever strategy to deal with a high amount of data. Its key insight is a statistical distinction between relevant and irrelevant information

which is treated as noise. Consider that presence of strongly irrelevant data can have a remarkable impact on performances of inferred classifiers, as well as noisy data, where overfitting further worsens classifier capabilities.

Among the many types of noise that can be observed (noisy labels, incomplete data, fuzzy data, etc.) the case of mislabeled data is addressed; this means that some positive examples could be regarded as negative and vice versa.
To process such data and deal with noise there are at least two different approaches, here denoted through the "features selection" terminology [45]. A first approach aims at removing the irrelevant examples *before* the inference process, also known as ***filter*** strategy; it entails an effective distance measure between symbolic samples, but it is difficult to determine and usually induces some bias within the produced set. In order to avoid such bias, a second approach called ***wrapper*** strategy aims at detect and deal with noise during the inference process; this is the case of the Blue* algorithm.

Let us analyse the new state merging strategy adopted and the used statistical test.

The statistical operators

Blue* authors firstly modify the RPNI algorithm devising the *RPNI** [69]. They changed the *merge operator* in order to make it statistically flexible to *misclassified* examples[10], then it was adapted to EDSM, adding the *statistical promotion* too.

This approach is based on the comparison of misclassified sample *proportions*, verifying that such proportions do not increase significantly after a state merging; the related size reduction of DFA is accepted when error does not increase significantly. Let *population* p denote the unknown complete set of strings identifying the target language, and *population sampling*, or *sampling*, $\hat{p}$ the set of positive and negative examples $I_+ \cup I_-$. Since the error variations might depend on the particular sampling of target language, a simple comparison of misclassified proportion is not sufficient and a statistical method is necessary to manage error variability.

The new merging rule considers a merge as ***statistically acceptable*** "if and only if the proportion p_2 of misclassified examples in a DFA after a merging is not significantly higher then the proportion p_1 computed before the merging" [70].

[10]A positive example is *misclassified* if feeding it to a DFA it ends in accepting state; conversely in the case of negative example.

In *statistical inference* there are many tests to evaluate two proportions, among them *hypothesis testing* best fits the current problem [7].

Hypothesis testing

Hypothesis testing is a useful statistical test to evaluate the probability that some conjecture is true. For instance, an interesting question about classifiers, such as DFA, might be: *what is the probability that error over examples, made by DFA*$_1$*,* p_1*, is greater than* p_2*, error made by DFA*$_2$*?*
Indeed, the difference $p_2 - p_1$ might be greater than zero also in the case where real errors are equal; this might happen due to the population sampling of target language. Thus, a high probability is needed to make such error comparison reliable.

The previous question can be stated as a ***formal statistical hypothesis*** [7]. Every formal hypothesis consists of two parts, defined to contains all possible outcomes of test. One is the ***null hypothesis*** $\mathcal{H}_0$, which states that after an event old conditions are still true; the other is the ***alternative hypothesis*** $\mathcal{H}_a$, on contrary it states that conditions are changed and something new happened.
If the two hypotheses are defined in terms of two proportion comparison the test is called *hypothesis test for difference in two population proportions* [7].
For instance, the null and alternative hypothesis for the previous example are:

$$\mathcal{H}_0 : p_1 = p_2$$
$$\mathcal{H}_a : p_2 > p_1$$

The $\mathcal{H}_0$ states that errors of p_1 and p_2 are equal in spite of such no significant difference; while $\mathcal{H}_a$ states that the new automaton introduces a significant error over examples.
A general process of testing hypotheses involves four main tasks [7]:

1. *Hypothesize*
   - STEP1: establish formal hypothesis (null and alternative one)
2. *Test*
   - STEP2: determine an appropriate statistical test
   - STEP3: set the value of alpha
   - STEP4: establish a decision rule
   - STEP5: gather samples data
   - STEP6: analyze data
3. *Take statistical action*
   - STEP7: make a statistical conclusion
4. *Determine the implications*
   - STEP8: make a business decision

1. Hypothesize: The above analyzed specification of formal hypothesis might be defined also with the alternative hypothesis $\mathcal{H}_a : p_2 < p_1$, or a more generic $\mathcal{H}_a : p_2 \neq p_1$. The null hypothesis $\mathcal{H}_0$ is always initially retained true.

The two proportions are assumed as random variables that follow a binomial distributions over independent trials; however, they are approximated to normal distributions to take advantage of continuous nature of such distribution.
This approximation is possible under certain conditions, i.e., for large size of sample set. More precisely, after stating the terms p, q and n, where p is the probability of getting a success on any one trial, $q = p - 1$ and n the number of trials, the necessary conditions to meet to approximate a binomial distribution with normal one are:

$$n \cdot p > 5 \quad \text{and} \quad n \cdot q > 5$$

2. Test: In this step an appropriate statistical test is selected and necessary estimators are computed.

Some of these estimators are mean, variance and proportions of the distributions for the random variables.

Furthermore, a parameter called *critical value* must be fixed to determine whether the null hypothesis is rejected or not. With regard to normal distribution, the critical value determines the *rejection* and *non rejection* areas, respectively, the region of values characterizing whether a null hypothesis is accepted or not.

A hypothesis can be *two-tailed* or *one-tailed*; two-tailed test defines null and alternative hypotheses as $p_2 = p_1$ and $p_2 \neq p_1$, while the one-tailed hypotheses are as $p_2 = p_1$ and $p_2 > p_1$ (or alternatively $p_2 < p_1$). In the latter case, critical value identifies only one area as rejection region for the null hypothesis (fig. 3.16).

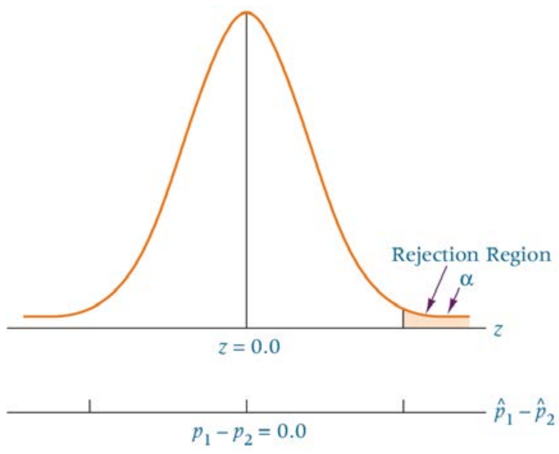


Figure 3.16: Rejection region for a one-tailed hypothesis in difference for two population proportions.

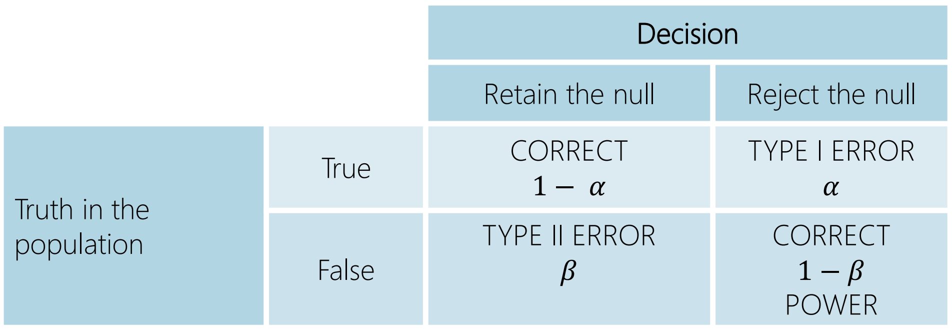
Information exploited during the process are only sample statistics, then it is possible to make incorrect decisions. Two kinds of errors can be made: *Type I error* and *Type II error* (fig. 3.17).

Type I error "is made by *rejecting a true null hypothesis*" [7]. It is the case, for instance, when values of proportions are the more extremes and its difference seems to support the alternative hypothesis while it is not true. E.g. a new DFA obtained by state merging entails an increased error on examples, but it may be promising for the future because it was an extreme case of a non significant error; then DFA merging is incorrectly rejected.

The rejection error represents the likelihood to made a Type I error; values falling over the critical value will be considered so extreme to be rejected. The "probability of making a Type I error is called α ***value*** or ***level of significance***" [7]; it is equal to the area under the curve in the rejection region, beyond the critical value. The typical values for α are 0.01, 0.025, 0.05.

Type II error is made by *failing to reject a false null hypothesis.* As opposed to previous error, null hypothesis is false but, for instance, some extreme values lead to accept it as true and reject the alternative hypothesis. For instance, a DFA resulting from a state merging introduces a relevant error with regard to the remaining elaborations, but current examples produce misclassified proportions not so different from previous DFA; then, the merge is incorrectly accepted.

The "probability of making a Type II error is called β, and $1-\beta$ is called ***power***[11] of the test" [7]; unlike α, it is not stated at the beginning of test. However, they are inversely related, if one is increased the other decrease, because α is made when the null hypothesis is rejected and conversely β.



		Decision	
		Retain the null	Reject the null
Truth in the population	True	CORRECT $1-\alpha$	TYPE I ERROR α
	False	TYPE II ERROR β	CORRECT $1-\beta$ POWER

Figure 3.17: Types of statistical errors.

The statistic used to evaluate the difference in two population proportions is the difference in sample proportions: $\hat{p_2}-\hat{p_1}$; $\hat{p_1}$ and $\hat{p_2}$ are also called the *empirical errors*, computed considering random samples from the population p of all the strings composing the language.

Binomial to normal distribution approximation is possible when some conditions about samples sets size are met; denoting by n_1 and n_2 the size of sample sets, the following conditions are known as necessary to achieve the approximation [7]:

$$\begin{array}{ll} n_1 \cdot \hat{p_1} > 5 & n_1 \cdot \hat{q_1} > 5 \\ n_2 \cdot \hat{p_2} > 5 & n_2 \cdot \hat{q_2} > 5 \end{array}$$

where $\hat{q} = 1-\hat{p}$.

[11]It is the "probability of a statistical test rejecting the null hypothesis when the null hypothesis is false" [7].

For the central limit theorem[12], for sufficiently large examples set, the difference in sample proportions is *normally distributed* with mean μ and standard deviation σ:

$$\mu_{\hat{p_2}-\hat{p_1}} = p_2 - p_1 \tag{3.1}$$

$$\sigma_{\hat{p_2}-\hat{p_1}} = \sqrt{\frac{p_1 \cdot q_1}{n_1} + \frac{p_2 \cdot q_2}{n_2}} \tag{3.2}$$

Before proceeding, it is worth noting that every change in μ and σ modify the normal distribution. Many characteristic values for a standard normal distribution are tabulated and to avoid dealing with so many tables, one for every possible normal distribution, a general formula allows to normalize every normal distribution to a standard one, called *z distribution* or *Standard Normal Distribution.*

The *standard score* (or σ) is “the number of standard deviations that a value, x, is above or below the mean” [7]; and for the a Standard Normal Distribution such values is called *z score* defined as:

$$z = \frac{x-\mu}{\sigma}, \qquad \sigma \neq 0 \tag{3.3}$$

Thus, the distance of every value x from mean can be easily converted in terms of distance in standard deviation units. Probabilities and percentiles related to *zscore* are tabulated for Standard Normal Distribution to easily compute the probability of a value to fall between the mean μ and x.

A formula for z can be defined by using the previous definitions for the difference in sample proportions:

$$z = \frac{(\hat{p_2} - \hat{p_1}) - (p_2 - p_1)}{\sqrt{\frac{p_1 \cdot q_1}{n_1} + \frac{p_2 \cdot q_2}{n_2}}} \tag{3.4}$$

where

- $\hat{p_1}$ and $\hat{p_2}$ are proportions from sample 1 and 2;
- p_1 and p_2 are proportions from sample 1 and 2;

[12]“If samples of size n are drawn randomly from a population that has a mean of μ and a standard deviation of σ , the sample means, $\overline{x}$, are approximately normally distributed for sufficiently large sample sizes ($n \geq 30$) regardless of the shape of the population distribution. If the population is normally distributed, the sample means are normally distributed for any size sample” [7].

- n_1 and n_2 are sizes of sample 1 and 2;
- $q_1 = 1 - p_1$ and $q_2 = 1 - p_2$.

Equation 3.4 computes a z score in the Standard Normal Distribution. It has an associated probability that a value falls in the interval between itself and the mean; such probability values are tabulated and directly available [7]. For given population proportions it represents the probability to get a particular difference in two sample populations.

It remains to analyze how to calculate p_1 and p_2 in denominator, although their difference $(p_2 - p_1) = 0$ by null hypothesis. In a real scenario, populations proportions p_1 and p_2 are usually unknown; thus, they can be estimated by sample proportions. An estimate value $\overline{p}$ is calculated by a weighted average defined as:

$$\overline{p} = \frac{n_1\, \hat{p_1} + n_2\, \hat{p_2}}{n_1 + n_2} \quad \text{and} \quad \overline{q} = 1 - \overline{p} \tag{3.5}$$

The resulting formula can be directly used for hypothesis testing about the difference of two proportions:

$$z = \frac{(\hat{p_2} - \hat{p_1}) - (p_2 - p_1)}{\sqrt{(\overline{p} \cdot \overline{q})(\frac{1}{n_2} + \frac{1}{n_2})}}. \tag{3.6}$$

Once such data are collected, the test can be performed.

3. Take statistical action: Depending on the test result (Eq. 3.6), a statistical conclusion is made, that is whether the null hypothesis is accepted or not.

4. Determine the implications: Statistical results imply the acceptance or rejection of a merge.

Hypothesis test in Blue*

Inference of DFA in Blue* is made through a hypothesis test, which leads to evaluate whether a merge introduces a significant error, in order to reject it. Further, an analysis of Type I and II error minimization leads to guide selection among all the acceptable merges or promotions.

1. Hypothesize:

Step 1. Let us define the population proportions p_1 and p_2 of misclassified strings by the $\mathcal{DFA}_1$, and $\mathcal{DFA}_2$ obtained by state merging operation.
Example set is composed by samples of population; its size is denoted by $N = n_1 = n_2$; while proportions are $\hat{p_1} = \frac{N_1}{N}$, $\hat{p_2} = \frac{N_2}{N}$, indipendent random variables and unbiased estimators of p_1 and p_2, with N_1 and N_2 indicating the misclassified examples for the two automata [70]. Furthermore, suppose the approximation conditions $Np > 5$ and $Nq > 5$ are satisfied (Par. 3.4.3).
The null and alternative hypotheses are:

$$\mathcal{H}_0 : p_2 = p_1 \tag{3.7}$$

$$\mathcal{H}_a : p_2 > p_1 \tag{3.8}$$

where:

- p_1 proportion of misclassified strings from $\mathcal{DFA}_1$;
- p_2 proportion of misclassified strings from $\mathcal{DFA}_2$.

The null hypothesis is supposed to be true and only a large value of $p_2 - p_1$ must lead to reject it.
Language inclusion property (Theorem 6) ensures that every obtained automaton by state merging is consistent to the I_+ positive examples; then if some of examples become statistically not acceptable, it will be never accepted again. Therefore, error can only remain the same or increase. In this case the test is also called a *one tailed test* due to one tail of these normal distributions.

2. Test:

Step 2. The appropriate test is Eq. 3.4.

Step 3. The α parameter is arbitrarily set to 0.01: $\alpha = 0.01$.

Step 4. The test is one-tailed. Then, for $\alpha = 0.01$, $Z_c = 0.5 - 0.01 = 0.49$ is the area under the curve between the mean and critical value. From tabulated value, critical value results $z_c = 2.33$.

If during test a z value greater than 2.33 is obtained, then null hypothesis will be rejected.

STEP 5. Suppose that $\hat{p_1}$, $\hat{p_2}$, n_1, and n_2 are known values.

STEP 6. $\overline{p}$ and $\overline{q}$ are directly calculated:

$$\overline{p} = \frac{n_1\,\hat{p_1} + n_2\,\hat{p_2}}{n_1 + n_2} \qquad \overline{q} = 1 - \overline{p}$$

At this point, all the needed parameters for computing the z value are known[13]:

$$z = \frac{(\hat{p_2} - \hat{p_1}) - (p_2 - p_1)}{\sqrt{(\overline{p}\cdot\overline{q})(\frac{1}{n_2} + \frac{1}{n_2})}} = \frac{(\hat{p_2} - \hat{p_1}) - (0)}{\sqrt{\frac{(\overline{p}\cdot\overline{q})}{N}}}$$

3. Take a statistical action:

STEP 7. If z is greater than z_c then null hypothesis is rejected, entailing that current merging is not statistically acceptable, with a risk of $\alpha\%$; on the contrary, the merging is added to the pool of candidate merges:

$$\begin{array}{llll} \text{if} & z > z_c & \text{then} & \textit{merge is rejected} \\ \text{if} & z \leq z_c & \text{then} & \textit{merge is compatible} \end{array}$$

4. Inference implications:

Among all acceptable blue states, a statistical criterion is adopted in order to select the best one. Therefore, the pair (blue, red) belonging to the pool of candidates C, $(b, r) \in C$ will be chosen, which minimizes the risk of having wrongly accepted it; this corresponds to minimize the Type II error β [70]. For a pair $(b, r) \in C$, the type II error $\beta_{b,r}$ is:

$$\beta_{b,r} = P(\mathcal{H}_0 \textit{ accepted} \mid \mathcal{H}_a \textit{ true}) \tag{3.9}$$

where P denotes the associated probability, represented by the identified area under the curve. However, the statistical law $\mathcal{H}_a : p_2 - p_1 > 0$ is unknown, so it is convenient to set a parameter $\delta > 0$ such that:

$$\beta_{b,r} = P(\mathcal{H}_0 \textit{ accepted} \mid p_2 - p_1 \ = \delta) \tag{3.10}$$

[13]Recall that for the null hypothesis $p_2 - p_1 = 0$

Eq 3.10 considers the normal distribution of alternative hypothesis, checking the intersection with the normal distribution of null hypothesis in the region of acceptance (fig. 3.18).

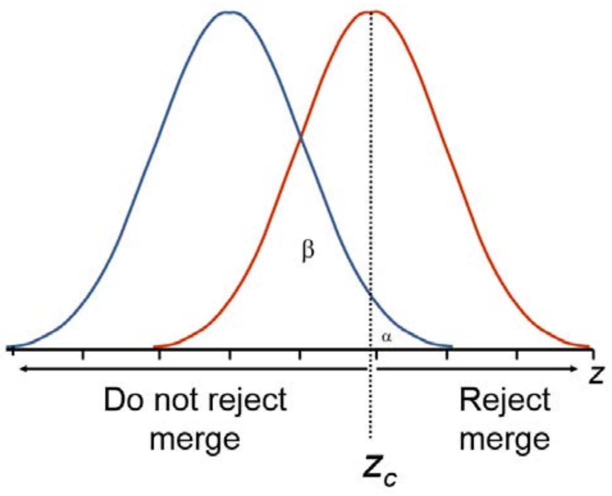


Figure 3.18: Regions defined by z_c critical value.

Let us rewrite such equation in terms of the critical value z_c:

$$\beta_{b,r} = P(z \ \leq \ z_c \mid p_2 - p_1 \ = \delta) \tag{3.11}$$

using Eq 3.4:

$$\beta_{b,r} = P\left(\frac{(\hat{p_2} - \hat{p_1}) - (p_2 - p_1)}{\sqrt{\frac{(\overline{p}\cdot\overline{q})}{N}}} \ \leq \ z_c \mid p_2 - p_1 \ = \delta\right) \tag{3.12}$$

Applying the alternative hypothesis condition:

$$\beta_{b,r} = P\left(\frac{(\hat{p_2} - \hat{p_1}) - \delta}{\sqrt{\frac{(\overline{p}\cdot\overline{q})}{N}}} \ \leq \ z_c \ - \frac{\delta}{\sqrt{\frac{(\overline{p}\cdot\overline{q})}{N}}}\right) \tag{3.13}$$

The first member of inequality is equivalents to Eq. 3.3, the equation used to transform the normal distribution to standard one. In this case, $\mu = \delta$ and $\sigma = \sqrt{\frac{(\overline{p}\cdot\overline{q})}{N}}$, then:

$$\beta_{b,r} = P\left(\mathcal{N}(0,1) \ \leq \ z_c \ - \frac{\delta}{\sqrt{\frac{(\overline{p}\cdot\overline{q})}{N}}}\right) \tag{3.14}$$

An arbitrary choice of a constant δ value is irrelevant because a relative comparison is performed in order to select the pair (b^*, r^*) minimizing the above quantity: $(b^*, r^*) = argmin_{(b,r \in C)}\ \beta_{b,r}(\delta)$.

However, the designed criterion favors merges so safe to be useless, i.e. merges with a null reduction of number of states. Therefore, a $e_{b,r}$ factor to be maximized is added, representing the number of states removed by a state merging operation:

$$(b^*, r^*) = argmin_{(b,r \in C)}\ \frac{\beta_{b,r}(\delta)}{e_{b,r}} \tag{3.15}$$

Similar considerations lead to evaluate the best possible *promotion*; it would be desiderable to promote the most state as the definitive one. A state can be promoted when all its merges results incompatible; however, they may result incompatible due to the presence of noise, and the risk of having wrongly rejected a merging of (b, r) can be easily measured as the Type I error $\alpha_{b,r}$:

$$\alpha_{b,r} = P(\mathcal{H}_0\ accepted \mid p_2 - p_1 = 0) \tag{3.16}$$

If a blue state is candidate to promotion, all its merges are incompatible because they have an associated z value: $z \geq z_c$ (Par. 3.4.3). Let us define the minimum z value that reject every red state for the analyzed blue: $z_b = min_r\ z_{b,r}$; the ones having the smaller z value is that with maximum probability to be wrongly rejected, with the maximum associated $\alpha_{b,r}$ area. The smaller z_b, the greater $\alpha_{b,r}$.
After computing the minimum z value for every blue state candidate to promotion, the greatest value among them is selected, which maximize z_b (and minimizes α_b), minimizing the risk of promoting a blue state that should be merged:

$$b^* = argmin_b\ \alpha_b \tag{3.17}$$

Finally, let us consider that Blue* can be considered as a more general case of EDSM with a naive heuristic function. Indeed, EDSM is like Blue* which must not accept misclassified examples. This behavior is achievable fixing $\alpha = 0.5$, refusing all merges where $z > 0$, i.e. $p_2 > p_1$. At the beginning of algorithm $p_1 = 0$ and no merging leading to $p_2 > 0$ will be accepted, obtaining the EDSM behavior with a naive heuristic evaluation.

3.4.4 L* - Active Learning

A totally different learning paradigm is *Active learning*, defined through the model of "learning from queries" by Dana Angluin [2]. It is a model based

on the interaction between a ***learner*** and a ***teacher***; instead of performing a lattice exploration, it attempts to exploit this interaction; despite the absence of a lattice to limit the search space, learning by queries is an effective process of state *splitting* towards the minimum consistent automaton.
The teacher is a device knowing the target automaton and able to correctly answer some kind of queries asked by the learner device. Unlike other inductive strategies, the role of the teacher allows to model a customized use of examples; the teacher might select which examples to supply to the learner, thus driving the development of DFA hypothesis by the examples more characterizing the target automaton.

The main algorithm in this context is called *L** designed by Angluin, a base algorithm for many other different algorithms [1] .
It ensures termination of the process by producing a minimum consistent DFA (or one isomorphic to it), recognizing the target regular language. Further, denoted by n the number of states of target DFA and 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 previous results of effectiveness and efficiency are synthesized in the following Theorem [1]:

Theorem 8. *"Given any minimally adequate Teacher presenting an unknown regular set U, the Learner L* eventually terminates and outputs an acceptor isomorphic to the minimum DFA accepting U. Moreover, if n is the number of states of the minimum DFA accepting U and m is an upper bound on the length of any counterexample provided by the Teacher, then the total running time of L* is bounded by a polynomial in m and n."*

Let us consider some observations about the different possible kinds of teacher and the support structures adopted by the learner.

The class of Minimillay Adeguate Teachers

The teacher involved in L* belongs to the class of ***Minimally Adeguate Teachers (MAT)***, able to answer two kinds of queries; *membership* and *equivalence query*, that are defined as follows:

- *Membership query*: learner proposes a sample to the teacher, which answers *YES* if the samples belong to target language, *NO* if not.

$$MQ : \Sigma^* \to \{YES, NO\}$$

- *Equivalence query*: the query proposes a representation of hypothesis language inferred, typically an automaton. The teacher answers positively if the hypothesis is equivalent to target language; on the contrary, it returns a counterexample, i.e., a sample belonging to the symmetrical difference set (Par. 3.1.1) of hypothesis and target language.

$$EQ : \mathcal{A} \to \{YES\} \cup \Sigma^*$$

A marginal note regards which teachers are actually available for the inference task. Some instances are direct implementations of some known deterministic machines. Furhtermore, different works pointed out several "teachers" quite different from an automaton. For instance, once a recurrent neural network is trained from a sequence it needs to be interpreted or it just remains a black box, that it could be used as a teacher [34]. Another example comes from requirements to test hardware, such devices could be tested entering a sequence of operations and use the output for membership queries [40].
On the learner side, some interesting works attempt to improve the learning task done with an improved management of counterexamples [66].

Observation table

The hypothesis DFA is encoded by a specific tabular representation called ***observation table***; a data structure managed during the inference algorithm [1]:

Definition (Observation table [41])**.** The *observation table* is a triple ⟨STA, EXP, OT⟩ where:
STA = Red ∪ Blue is a set of strings, denoting labels of states
Red $\in \Sigma^*$ is a finite set of states
EXP $\in \Sigma^*$ is the experiment set
Blue = Red $\cdot\Sigma\setminus$ Red is the set of states successors of Red, that are not Red
OT : STA × EXP → {0,1,∗} is a function such that:

$$OT[u][e] = \begin{cases} 1 & \text{if } ue \in L; \\ 0 & \text{if } ue \notin L; \\ * & \text{otherwise (not known).} \end{cases}$$

Counterexamples provided by equivalence queries, together with their suffixes, are exploited to generate a new hypothesis DFA, while membership queries allow to fill the holes derived from the introduction of new prefixes inside the observation table.

A DFA can be built from an observation table if some conditions are met: completeness, closure and consistency; these properties are now stated in the formalism adopted in [41]:

- **Completeness:** completeness ensures that for every prefix within the table all needed information is present, i.e. no partially, or totally, unknown behavior is present:

  Definition (Complete table)**.** A table is ***complete*** if there are no holes. "An *hole* within a table $\langle$STA, EXP, OT$\rangle$ is pair (u, e) such that $OT[u][e] = *$" [41].

  Once a table is checked as incomplete, the algorithm can fill it with membership queries.

- **Closure:** it ensures that every possible state reached from a transition is a final state for the automaton:

  Definition (Closed table)**.** "A table $\langle$STA, EXP, OT$\rangle$ is ***closed*** if given any row u of Blue there is some equals row v in Red", for every experiments in EXP [41].

  Whenever a table is checked as not closed, it can be closed with a "promotion", by adding the row $u \cdot \Sigma$ of Blue in Red.

- **Consistency:** it ensures that no inconsistency exists between rows representing the same state, which rises whenever, for the same input, the analysed rows execute different transitions:

  Definition (Consistent table)**.** "A table $\langle$STA, EXP, OT$\rangle$ is ***consistent*** if every equivalent pair of rows in Red remains equivalent in STA after appending any symbol: $OT[s_1] = OT[s_2] \Rightarrow \forall a \in \Sigma, OT[s_1a] = OT[s_2a]$" [41].

  An inconsistency is solvable by adding a column, which expands the set of experiments EXP with the string composed by suffix and experiment generating the inconsistency.

L* Algorithm

Once the table is complete, closed and consistent, an automaton is built from it. Such hypothesis automaton is proposed to the teacher by an equivalence query. If the teacher response is positive the process ends and the target automaton (or one equivalent) is found; otherwise, a counterexample is provided and the process continues exploiting such sample to update the observation table and checking the above mentioned properties again.
A sketch of this process is reported in Fig. 3.19, while a pseudocode follows [41] synthesizing the overall process:

L* Algorithm

```
Result: DFA $\mathcal{A}$
Initialization;
repeat
    while $\langle STA, EXP, OT \rangle$ is not closed or consistent do
        if $\langle STA, EXP, OT \rangle$ is not closed then
            $\langle$STA, EXP, OT$\rangle \leftarrow$ CLOSE($\langle$STA, EXP, OT$\rangle$);
        if $\langle STA, EXP, OT \rangle$ is not consistent then
            $\langle$STA, EXP, OT$\rangle \leftarrow$ CONSISTENT($\langle$STA, EXP, OT$\rangle$);
    end
    Answer $\leftarrow$ EQ($\langle$STA, EXP, OT$\rangle$);
    if $Answer \neq$ YES then
        $\langle$STA, EXP, OT$\rangle \leftarrow$ UseEQ($\langle$STA, EXP, OT$\rangle$, Answer);
until $Answer =$ YES;
return $BuildAutomaton(\langle STA, EXP, OT \rangle)$
```

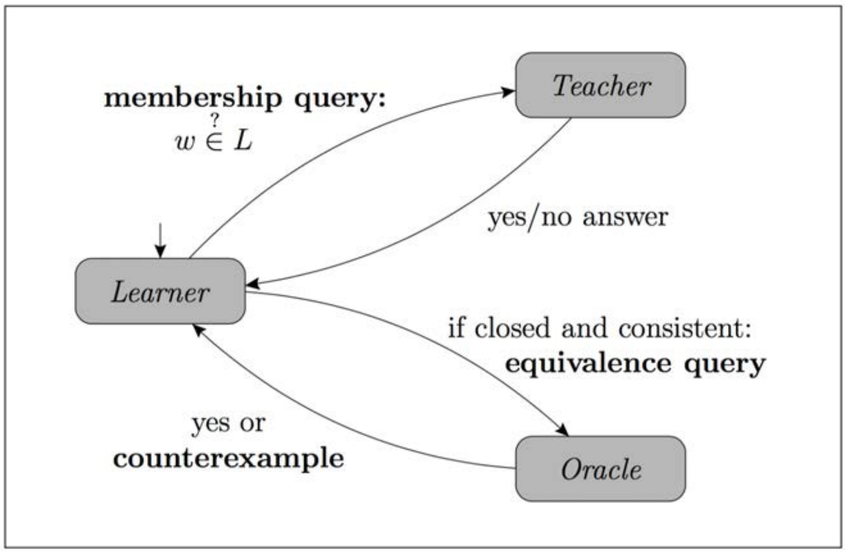


Figure 3.19: A schematic representation of L* execution.

Time complexity

Time complexity for the overall process is polynomially bounded, depending on the number of states of target automaton and the maximum length of counterexamples provided by the teacher.

Denote by n the number of states for the unknown target DFA, and m an upper bound for the length of any counterexample provided by teacher. At the end of the process, RED rows set must be at least n because the process generates a DFA from the observation table with at least a number of states equivalent to the number of unique rows in RED.
Properties og the inference algorithm L* ensure that, as soon as a closed and consistent table with at least n different rows in RED is built up, the algorithm ends because it must have developed a DFA equivalent to the target automaton.

The table can only grow "vertically", until n different rows in RED are added, thus, let us consider the operations generating a new row in RED. Every time the table is not closed, a different row is added to RED; as for a counterexample, at least one row is added in RED, and at most m. Also resolving an inconsistent generates a new row. Therefore, at least one unique row is added to set of RED state.

In brief:

- the number of experiments is upper bounded by n, the number of unique rows in RED needed to achieve the target automaton: $|EXP| < n$.

- equivalently, the number of equivalence queries is upper bounded by n.
- the number of membership queries is bounded by the total size of observation table: at most n columns, (i.e. the number of |EXP|), for at most nm rows (consider for every counterexample of length m at most m new rows, thus in the worst case m rows for each one of n experiments).

The total number of queries is equal to $\mathcal{O}(n^2m)$[41].

Table-filling algorithm

As seen, L* needs a teacher able to answer two kinds of query, equivalence and membership query.
To satisfy both possible types of question, i.e. provide a correct answer to queries, is not a trivial task due to efficiency issues. Several approaches have been developed in order to check the automata equivalence, or generate a counterexample.

Let us analyse in depth the *table-filling algorithm*[43], that meets both the previous requirements. It is an algorithm adopted to check if some states are equivalent within a DFA; thus, another task achieved from table filling is the minimization of automata.
It works recursively, by detecting pairwise distinct states; at the end of elaboration, pair of states not marked are equivalent. The concept of distinct states might be defined as in [43]:

Definition (Distinct states)**.** "A state p is distinguishable from state q if there is at least one string w such that one of $\hat{\delta}(p, w)$ and $\hat{\delta}(q, w)$ is accepting, and the other is not accepting."

The key insight of table-filling is that, if a pair of states for the same alphabet symbol executes a transition to a pair of distinct states, then the starting pair is distinguishable.
In the recursive algorithm, the firs step distinguishes pairs of states surely not equivalent: these composed by an accepting and a rejecting state. Recursively, from this base case the algorithm continues detecting all the distinguishable states:

BASE CASE: "If p is an accepting state and q is nonaccepting, then the pair {p,q} is distinguishable".

INDUCTIVE STEP: "Let p and q be states such that for some input symbol a, $r = \delta(p, a)$ and $s = \delta(q, a)$ are a pair of states known to be distinguishable. Then {p,q} is a pair of distinguishable states. The reason this rule makes sense is that there must be some string w that distinguishes r from s; that is, exactly one of $\hat{\delta}(p, w)$ and $\hat{\delta}(q, w)$ is accepting. Then string aw must distinguish p from q, since $\hat{\delta}(p, aw)$ and $\hat{\delta}(q, aw)$ is the same pair of states as $\hat{\delta}(r, w)$ and $\hat{\delta}(s, w)$".

Moreover, the criterion adopted in table-filling is further justified because whenever two states, for the same input, perform transitions towards undoubted distinct states, those states must be equivalent. If not, they could be merged into a unique state having, for the same symbol, two distinct transitions resulting in a nondeterministic automaton.

Theorem 9. *"If two states are not distinguished by the table-filling algorithm, then the states are equivalent". [43]*

Once the equivalence states identified, automaton can be minimized by merging these states.

In order to model a teacher, the table-filling should be used to check the equivalence between automata and generate a counterexample. To detect if two automata are equivalent, a new union DFA is generated from DFAs to be checked and it is provided to the table-filling; in such union DFA two start states exists, but it is sufficient to arbitrarily consider only one of them as starting state.
Automata are discovered as equivalent, if at the end of the process the two original starting states are showed as equivalent. To optimize the process it is possible to stop the algorithm as soon as the cell denoting the pair of start states is filled.

Time complexity

The table-filling fills the table in polynomial time with respect to number of states n of analyzed automaton.

In the equivalence discovery task, the total number of states is the sum of the number of states of the two original automata. Distinct pairs considered at every table elaboration are $n * \frac{n-1}{2}$, thus $\mathcal{O}(n^2)$. The table elaboration is upper bounded by the number of all state pairs, in the worst case only one pair is distinguishable; on contrary algorithm ends earlier. Therefore an upper bound will be surely $\mathcal{O}(n^4)$.

However, it is possible to decrease such limit to $\mathcal{O}(n^2)$ with a more careful algorithm, handling dependencies among the pair of states analyzed during the table-filling. For each pair a list is handled; when a pair is discovered distinguishable all pairs "depending on" it are also updated as distinct. In turn, the list associated to that pairs are examined and updated.

Chapter 4

Multi-scale mobility models

"To iterate is human, to recurse divine."

Peter Deutsch

A mobility model is a concise representation of user movements, synthesizing paths already observed and predicting future paths.

Nowadays, an increasing amount of data of real users movements is readily available, thanks to the diffusion of devices such as smartphones and tablets, which embed position sensors [24]. Even though shaping accurate and reliable models requires a big amount of raw data at high resolution, mining useful information from this wealth of observations remains an open issue [32, 21].

To achieve this goal, an effective way is to induct smart mobility models over these data. It has been shown [38] that behind movements of inhabitants of cities, there are schema naturally induced from their daily routine: from residential places, to work and recreational places and back people tend to repeat similar courses of actions. These social patterns tend to characterize human behaviors [23], and modelling these regularities is one of the main goals in mobility analysis. Usually, regularities involve some degree of predictability, as pointed out in many works [38, 73]: they are detected as pattern changes between weekdays and holidays, or depending on the adopted scale to recognize paths.

On this basis, many research works studied the portability to several application scenarios in order to support human mobility.

4.1 Exploiting mobility models

Reliable mobility models con be exploited in many interesting application scenarios [33], such as mobility pattern and user activity recognition, travel prediction, extraction of Points Of Interest (POI) and location recommendation, anomaly detection or the generations of synthetic samples, and so on.

These and many other applications require to adopt non-trivial techniques to mine information from big data collections of user travels, and several works gave rise to methods able to achieve such results.

A first classification shows two kinds of mobility models [11]: synthetic and trace models.

The *synthetic models* are designed to mimic the required behavior for nodes. They are useful when networks environment or protocol to test are not available, and offer a direct mathematical management. They aim at automatically generate new traces by graph models [10], or take advage of veichle profiles [47]. A survey for this type of models is [11].

Instead, *trace-based models* provide more realistic representations since they involv real traces, however they need a large amount of data to be collected during a long period [48]. This is the case of human mobility models and allows to perform mobility pattern recognition, infer future spatial behavior, check if observed behaviors are compatible or not with a user [14] [4] [15] [54]; stricly linked to pattern recognition is the ability to detect anomalies [71].
Comparing mobility models leads to highlight the *POI* [81], in order to detect the most relevant places and eventually reccommend them; such suggestions might be more effective if they are based on measures of similarity between users [52]. A recommender system could suggest also *Paths of Interest*: possible travels interesting a user, based their similarity with other user travels.
Furthermore, accurate mobility models are ready to generate any amount of synthetic trajectories [33]; thus, they are usually involved in network simulations when realistic traces are not available or too expensive to collect.

Often mined information is related to **mobility phenomena** as traffic statistics or frequent itineraries, rather than focus on **mobility users** even if the user-driven approach let a direct support to user choices.
Moreover, most works are focused on extraction of aggregation indexes at most employing geometry tools and analysis [75], despite the inherently structural nature of trajectories. A structural approach seems more promising, even though managing structural information remain a complex task, due to combinatorics explosion.

Some approaches have attempted to infer a structural model through a syntactic approach and a symbolic encoding, which exploit formal grammars

as a formalism for target models. Among them two different methods have been proposed in [49], one where the alphabet is composed by “status” of an agent (e.g. healthy, sick) and states are its “locations” (e.g. at home, at work); another symmetrically where states are the agent “status” where alphabet symbols are the “location”. Status is inferred by an analysis of temporal sequences of travels, while locations rise from an unsupervised classification algorithm over the most visited places.
In [33], *Probabilistic Context-free Grammars - PCFG* are involved in a framework to analytically extract measures for networks; the paper proposed an inference algorithm, and a slightly different definition of mobility data oriented PCFG.

A grammar induction approach to discovering motifs in trajectories and identify anomalies is in [61]. The *mSEQUITUR* algorithm was developed to classify travels and discover its motifs; it was exploited in a related framework (*STAVIS*) designed to analyze and visualize trajectories.

However, in all the previous works, symbolic encoding and noisy care remain intricate, as well as trajectory frequency processing; in addition, they show poor flexibility to analysis at different scales.
In this thesis a different approach is proposed, in order to achieve a multi-scale mobility user model via regular languages. Paths are initially translated into a new symbolic representation accomplished by geohash encoding; then, symbolic data are used to infer a hierarchy of DFAs, constituting a hierarchical mobility model describing at each level of hierarchy different spatial habits of user.

Let us introduce the *Geohash encoding*, which has a key role in inferred mobility models.

4.2 A symbolic encoding: Geohash

Geohash is a geocode system, that represents a pair of *(latitude,longitude)* coordinates by *hash* value [5]. It was created to easily add spatial coordinates inside web *Uniform Resource Locator - URL*; then it was proved to be an effective system to encode geographical locations, making storage and management in a database easy.

The *geohash algorithm* parts the world in a hierarchical structure: a grid of 32 cells (4 rows by 8 columns), each of them denoted by a letter or a number. Every cells is then split into 32 cells, which in turn is possible to split in 32 more cells, and so on, entailing a gradual precision enhancement for identified locations.

In a first phase of encoding, the geohash string is a binary string; each bit represents the alternate splitting of global rectangle in longitude-latitude $[-180, 180] \times [-90, 90]$. The first subdivision is vertical and generates two rectangles: $[-180, 0] \times [-90, 90]$ and $[0, 180] \times [-90, 90]$. Locations to the left of these partitions have a first bit set to '0', while the ones on the right has the first bit set to '1', as showed in Fig. 4.1.

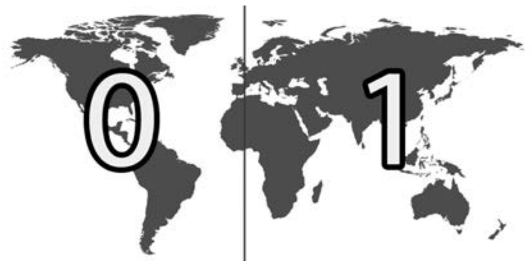


Figure 4.1: First longitudinal subdivision [-180,0] × [0,180].

Each of obtained rectangle is split again but horizontally, giving rise to two other rectangles; locations over the divisor line have an assigned second bit equal to '0', while those below '1', as in Fig. 4.2.

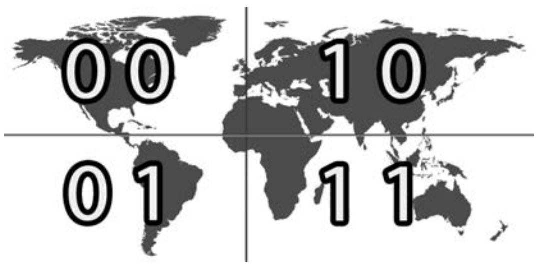


Figure 4.2: First longitudinal subdivision [-90,0] × [0,90].

Adding bits produces new spatial partitions, improving precision. The implicit structure induced by these divisions is a recursive quadtree [28].
By construction, bits in even position encode longitude information, while those in odd refer to the latitude. For instance, inside string 01011, "001" bits encode longitude values, while "11" latitude ones.

In order to make the geohash encoding suitable for the web, final encoding is based on a 32 symbol alphabet, one for each cell: *"0123456789bcdefghjkmnpqrstuvwxyz"*. Binary string can have an arbitrary length, whereas in the alphabetic they are usually multiple of five; indeed, binary values in groups of five are used as index for alphabet symbols [22]. For instance, the group 01011_2 is index for '*b*' symbols; and a whole string *sqc2zgw* denotes:

s	q	c	2	z	g	w
11000	10110	01011	00010	11111	01111	11100

Table 4.1: Bit groups and associated alphabet symbols.

The generated string, called ***geohash string***, identifies a precise cell in the hierarchical space.

A rough estimate of covered area by a geohash cell, related to geohash lenght used, is showed in Table 4.2. To compute the cell extension consider the area of initial rectangle equal to Earth surface, i.e. about 510.072.000 km^2. Each character reduces by a factor $2^5 = 32$ area of an higher order cell, hence the covered area for a geohash of length i:

$$area(i) = \frac{510.072.000 km^2}{(2^5)^i}$$

Geohash length	$\approx$ Covered area km^2
1	16.000.000
2	500.000
3	15.000
4	500
5	15
6	0.5
7	0.02

Table 4.2: Geohash areas.

One of the features making this encoding highly flexible and attractive is that an arbitrary number of further characters can be added to geohash string to refer to a smaller geographical space. It might be exploited to carry out search to different granularity level of grid with a direct control on length of strings.

Moreover, two other features are worthy of attention: the *inclusion property* and the *locality property*; both related to hierarchical nature of geohash.

A geohash string obtained by adding a new character to a previous one, identifies a new cell inside the bigger cell recognized from the old string; such property is called the *inclusion property*. Consider for instance, the point determined by the pair (38.120281, 13.357278), it is inside the cell identified by *sqc2zg*, as well as inside the cell of *sqc2zg**w***, or *sqc2zg**wk*** and so on.

The *locality property* depends on the geohash string structure: strings sharing common prefix recognize adjacent spatial locations. The converse is not always true; two adjacent places might have completely different geohash strings due to the recursive structure of quadtrees. Indeed, a higher level discontinuity might be reflected in a lower level although considered cells are adjoining; for instance, the geohash *u19hbp* is inside an higher level cell *u*, but some spatial neighbors are inside the higher level cell *g*.

4.3 Formal languages for mobility models

Let us show how an artificial learning process for the mobility model is defined by the more abstract theoretic tools (knowledge type, inference strategy, etc.) to the concretely handled items (hypothesis representation, inference algorithm, and so on).

The general target of inference is a mobility model for a user; however entities and tools involved at each level also depends on higher level.

To present the inductive abstraction levels, the learning process sketched so far is analyzed, showing choices made to each level, from more abstract management of structural knowledge towards the lower level of automata inference algorithms (Fig. 4.3).

$$\begin{cases} \textit{Initial information} \rightarrow \text{Positive and negative examples} \\ \textit{Internal representation} \rightarrow \text{Regular languages} \\ \textit{Learning strategy} \rightarrow \text{Inductive learning} \begin{cases} \textit{Concept space} & \rightarrow & \text{Regular languages} \\ \textit{Hypothesis space} & \rightarrow & \text{DFA} \\ \textit{Presentation of examples} & \rightarrow & \text{Given-Data} \\ \textit{Inference method} & \rightarrow & \text{Blue* algorithm} \\ \textit{Success criterion} & \rightarrow & \text{Identification in the limit} \end{cases} \end{cases}$$

Figure 4.3: Learning process, from abstract to practical tools.

A first level of abstraction is concerned with learning strategies (see Par. 2.2.2) to manage structural knowledge.

Learning strategies → Inductive learning

Inductive learning is a strategy feasible to fit behaviors, exposing high predictive power.

Initial information (Par. 2.2.1) for the learning system might be viable routes, thus streets and highway, and in general viable spaces, and eventually obstacles as mountains or seas. This knowledge should consider also transportation mode, indeed obstacles for an airplane are very different from those for a car.

Learning from examples (Par. 2.2.3) exploits the numerous datasets available today, and a comparison among examples provided by user and counterexamples from other users is sufficient to learn a *mobility pattern*.

Instead *learning from observation and discovery* is ruling out because a user or environment interaction is not easy to address in an reliable way entailing relevant management costs.

Examples are provided in *one-trial* way for the offline process. This does not compromise the possibility to improve the inferred models, because the system might be able to include new knowledge within existing models.

Internal representation → Regular languages

A convenient representation (Par. 2.2.4) to manage trajectories, and exploit their structural nature, is *regular languages*. It entails a syntactical approach to treat data, effective to capture regularities embedded into paths.

Obviously, paths are not only characterized by regular behaviors, but regularities partially grasped also in irregular paths can be sufficient to uniquely denote a user behavior and predict it. Long paths are often planned and repetitive, i.e., airplanes or ships routes; further, as previous mentioned studies [38, 73] pointed out human spatial behaviors are mostly regular, as for daily routines where typical patterns depend on weekday or holidays, and are recognizable at different scales.

Analyses at different scales allow to classify paths observed with the granularity which better highlights their regularities: small paths by a finer granularity, longer paths by a coarser one.

The exploited description to encode regular language is *DFA*, the introduced recognizer for regular languages (Par. 3.1). In this context its graphics representation is a meaningful characteristic to allow human inspection in scenarios where the high amount of data lead to usually adopt only synthetic index, not always effective to understand data.

4.4 The language of paths: DFAs as mobility models

Once high level learning task is fixed, the inductive problem can be characterized adopting the classification in five items (Par. 3.2):

1. Concept space;
2. Hypothesis space;
3. Presentation of examples;
4. Inference methods;
5. Success criteria.

Concept space $\longrightarrow$ Mobility models

The elements of concept space are all the possible mobility models; one of them is selected as mobility model for the analyzed user. This model is a recursive multilevel model, made by a pool of 32 regular languages for each different granularity level of spatial grid induced from geohash. These are the ***language of paths*** for a user.

The alphabet is composed by geohash symbols (Par. 4.2) used to identify the cells of grid.
All the user trajectories are encoded by a set of geohash strings:

Definition (Trajectory)**.** A *trajectory* is a nonempty finite set of ordered geohash strings.

Trajectories have a *multiscale* nature, the same path can be observed at different scales to mine characterizing features. For instance, in a coarse scale, in the order of country size, a user seems to stay prevalently in same place, it is not usual to go over one's own country. By adopting a finer scale where travels between city are detected, a user can move across different cities to get to his workplace daily. In turn, to perform latter travel he should take some vehicles, e.g. a subway, and movements from house to the closer subway station might be analyzed.
This kind of analysis is directly addressed thanks to geohash *inclusion property* (Par. 4.2); every user movement can be inspected at different scale considering different lengths of geohash prefixes for the same location.

Path 1	Path 2	Path 3	Path 4
wt**g**	wt**u**	ww**d**	wx**4**
wt**s**	wt**v**	ww**7**	wx**5**
wt**t**	wt**w**	ww**5**	-
-	wt**v**	wt**g**	-
-	-	wt**u**	-
-	-	ww**5**	-

Table 4.3: Examples of geohash paths.

w	wt	ww	wx
ttt	gst	d75	45
tttt	uvwv	5	-
wwwttw	gu	-	-
xx	-	-	-

Table 4.4: Examples within a "language of paths" for prefixes of length $L = 1$ and $L = 2$.

For instance, if the scale factor is denoted by strings of length $n = 3$, some recorded paths for a user are reported in Table 4.3.
Every column in the table describes a path of geohash strings[1], an ordered sequence of spatial cells. These paths are also called "*trajectories*".
A *geohash prefix* is a shorter geohash string identifying a cell to a coarser scale factor. In the example, prefixes highlighted for *Path 3* with length $L_1 = n-2 = 1$ and $L_2 = n-1 = 2$ are reported in Table 4.5.

Prefixes L_1	Prefixes L_2
w	wt
-	ww

Table 4.5: Prefixes at different length from Path 3 in Table 4.3.

Among them, consider the *ww* prefix. It detects a cell where a part of *Path 3* lies. In particular, this path is composed by cells identified by *d, 7, 5* symbols; hence, the string *d, 7, 5* represents a strings of the "language of paths" for user in the *ww* geographic area, and is called a *mini trajectory*:

Definition (Mini trajectory)**.** A *mini trajectory* is a string of ordered geohash alphabet symbols describing a path (or part of it) lying entirely in a cell.

Mini trajectories (Table 4.4) can be extracted for each cell by looking at all the contiguous subsequence of suffixes (bold in Table 4.3) of geohash strings sharing a common prefix inside a trajectory.

[1]Recall that every geohash denote, and it is equivalent, to a pair (latitude, longitude)

To sum up:

- user path, namely *trajectories*, are a sequence of points;
- each points is represented by its geohash encoding;
- each path can be regarded as a coarse- or fine-grained, depending on the length of the geohash at each point;
- for each cell, a regular language of the (sub)paths, namely *mini trajectories*, in that cell is inferred. It is the *language of paths* for that cell;
- the mobility model for a user is the union of these inferred languages (namely *"languages of paths"*).

The hierarchical structure of a multi-scale mobility models are graphically sketched in Fig. 4.4.

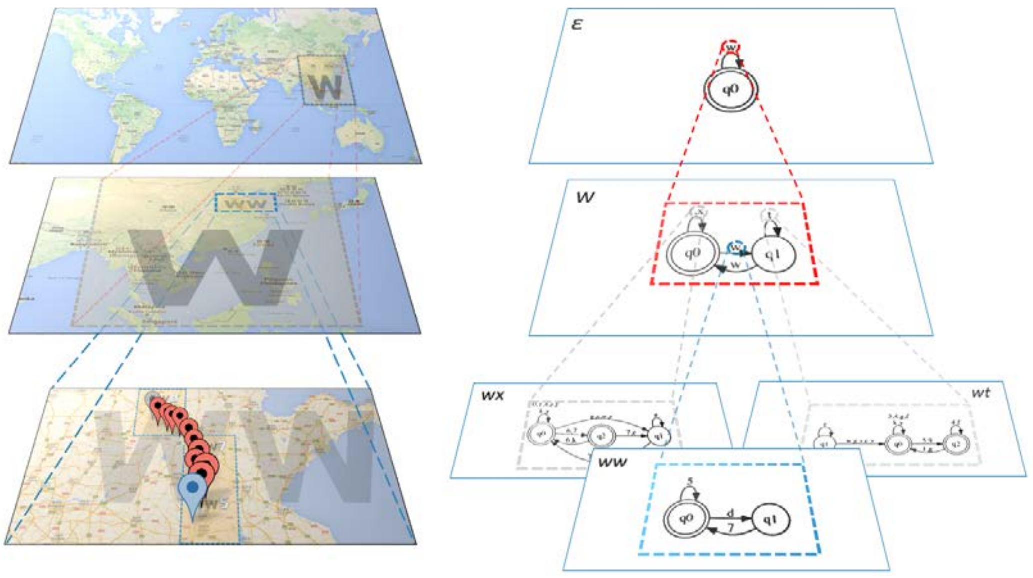


Figure 4.4: A "Hierarchical" graphic representation of path *d, 7, 5*.

Hypothesis space ⟶ Deterministic Finite state Automata - DFAs

The described structure of models in concept space is reflected in the arrangement of DFAs describing the languages of paths. *"Language of paths"* are managed by a pool of DFAs denoting elements of hypotheses space; these pools are representatives of past and future spatial behaviors in the areas identified by each automaton. Recall that a prefix identified a cell in the geohash grid, and paths lying in it are defined over 32 sub-cells. A single DFA is associated to the geohash cell (and thus to a prefix) and describes local user model.

Generated automata are *hierarchical automata* meaning that for every state of a DFA corresponds one or more automata hierarchically lower, able to describe with dynamic precision the paths for selected sub-area.

A simple navigation can be conducted between the different spatial scales of model; indeed, to achieve a better resolution it is sufficient to perform "hierarchical" navigation through the pool of automata, substituting for each symbols in a transition the correspondent automaton, describing in with more details spatial behaviors for a smaller area (Fig. 4.5). This automaton is denoted by the geohash prefix obtained by concatenating the strings identified the parents automaton and the symbols of traversed transition.

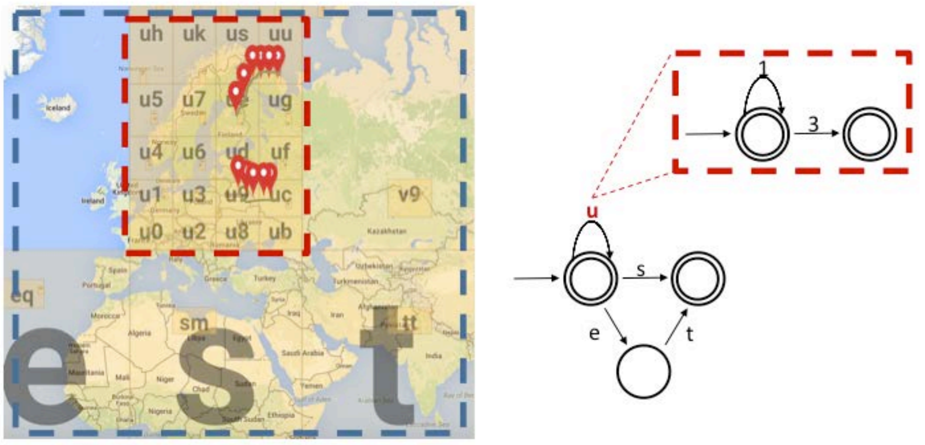


Figure 4.5: A "Hierarchical" navigation.

For instance, consider the automaton related to the cell *w*; it was inferred from the paths of the user and those of other ones, e.g. the path *wwwttw* (Table 4.3). This *wwwttw* path is internally composed by the previous analysed path *d 7 5*, however it is recognizable with a finer-granularity achieved switching to the smaller cell *ww* that contains *d 7 5* and *5*, paths described by a correspondent automaton.

Presentation of examples ⟶ *Given-data*

Every inferred mobility model is developed over the comparison between a user and other ones sharing common places.

Travelled user mini trajectories are treated as *positive samples* I_+, while the ones travelled by other users as *negative samples* I_-. If some of them is shared from both sets then only the positive one is retained.
Note that a user might stay in the same area for a long time, this happens when observation scale make cells covering entire countries and most of user travels not overcame such borders; as a result, a lot of repetitions of the same symbols appear, increasing a type of complexity unmanageable by regular languages. Therefore a pre-processing is needed in order to remove repetitions within paths.

Mobility models might include some paths inside neither positives or negative examples, these are paths originated by model ***generalization*** (or ***generative capability***) (Par. 3.3.1). In this setting, a new path becomes admissible when (a) it is not forbidden by negative examples, and (b) model simplicity is improved by it.

Generalization process is effectively managed in the *Learning from informant* setting (3.2); it was preferred to *learning from text* because there is no ground truth to contain generalization when only positive examples are available. In the latter case, two scenarios arose, one with an overgeneralization where too many paths became admissible, another with an overfitting of data. Instead, it is possible to preserve generalization process - avoiding overfitting - exploiting the paths of other users as negative examples, which bound the process before it becomes meaningless.

This is not cost free due to two issues: one is the imbalance between positive and negative samples (the negatives come from many users, while the positives only from one), the other one concern to the question: which are the relevant positive and negative paths?

A successful approach leads to consider the path *frequencies*; indeed, a path travelled many times is more representative of users behaviors than one performed just few times. Unfortunately, regular languages do not incorporate a mechanism to treat examples frequency, than all the examples have the same occurrence in a language.

In order to maintain advantage of deterministic models, but also consider frequency of examples, such information is handled into the inference process (and not inside the model), entailing an approach to leaving out some examples if they are not *statistically relevant*, i.e. with a low frequency, and if they lead to a simpler model. Hence, new paths developed from the generalization process come from the trade-off between model simplicity and the relevance mini trajectories.

Success criteria $\longrightarrow$ Identification in the limit

Once the position data are encoded by geohash algorithm, they are implicitly filtered by the introduced approximation. Unless it is explicitly required, a very high spatial precision is not so useful for mobility pattern applications, entailing a relevant introduction of noise to deal with.

Instead, such noise-free data lead to use exact learning paradigm as the *Identification in the limit* (Par. 3.2).

An alternative criterion, as the *PAC-learning* (see Par. 3.2), is generally more effective for noise data; in such case introduces an approximation at the end of the process entail more flexible model. However, the loss of precision introduced by *PAC-learning* has no relevant effects in a setting as the geohash one.

Inference methods $\longrightarrow$ Blue* algorithm

The huge amount of data and the absence of some kind of teacher make Blue* algorithm [70] the most suitable for grammatical inference of mobility models. In particular, the high volume of data to deal with favours the algorithms in the Red-blue framework due to its efficiency [50].

Blue* reflects all the choices made so far modelling the learning process. It involves a learning from informant ($\langle I_+, I_- \rangle$), without any teacher, and identification in the limit as successful criterion. Statistical criteria within the inference process give flexibility related to examples frequency and model simplicity, as showed in Table 4.6 and Fig. 4.6, where obtained DFAs by EDSM and Blue* are compared.
For each spatial cell of geohash grid an execution of Blue* is performed, building up the local automaton.

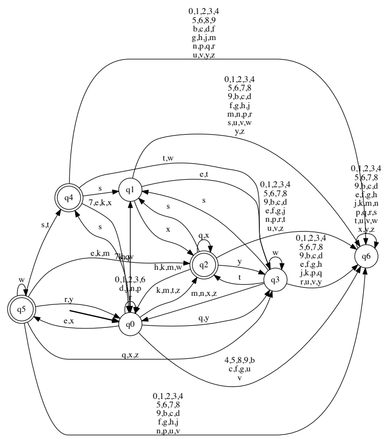
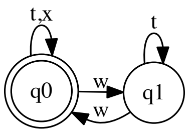
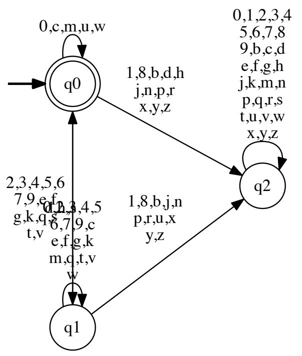
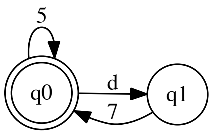
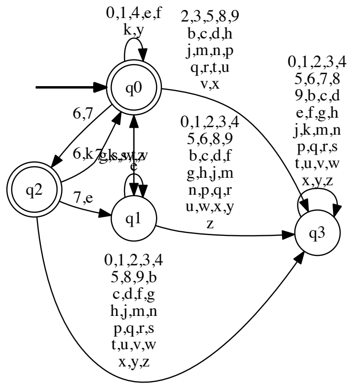
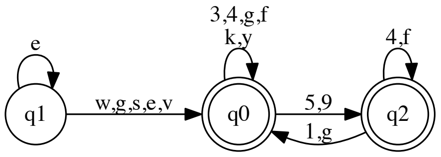
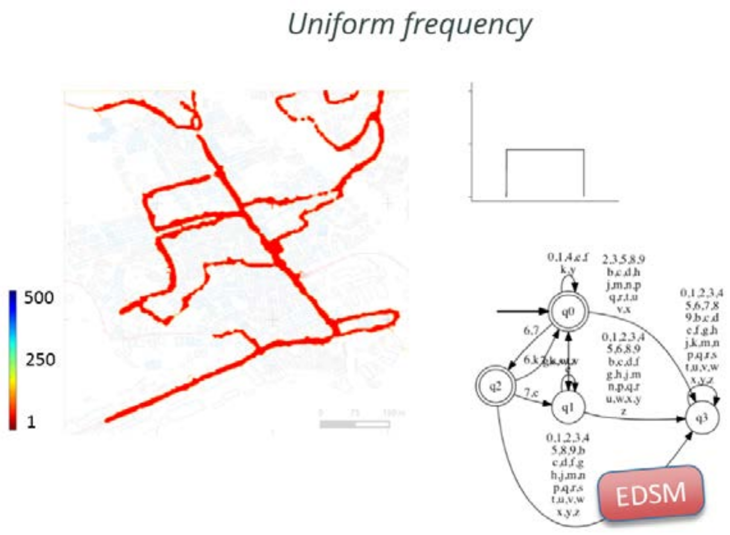
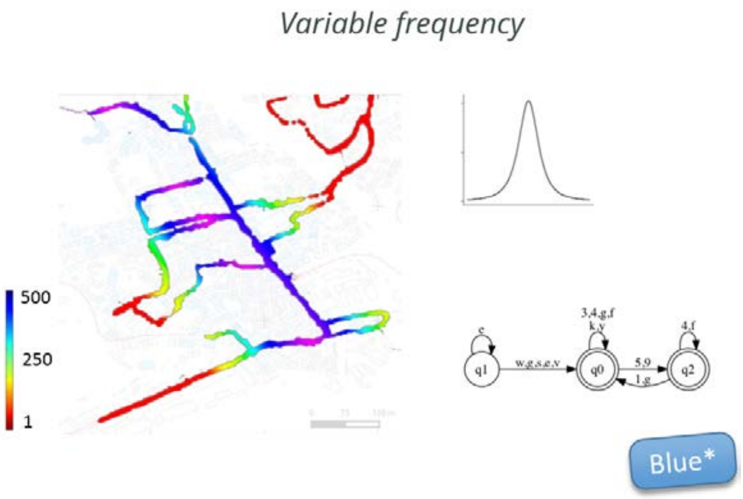
Prefix	*EDSM*	*Blue**
w		
ww	 0,c,m,u,w q0 1,8,b,d,h j,n,p,r x,y,z q2 q1 1,8,b,j,n p,r,u,x y,z	
wwm		

Table 4.6: Some inferred DFAs by EDSM and Blue*, representing the languages of paths for a user.



(a) Model inferred by EDSM.



(b) Model inferred by Blue*.

Figure 4.6: Frequency information used by EDSM and Blue*.

Chapter 5

Experimental assessment

"In theory, there is no difference between theory and practice. But, in practice, there is."

Jan van de Snepscheut

5.1 A modular inference system for mobility models

Whole inference process from trajectory dataset to the final pool of DFAs establishes a modular system with independent modules:

- **Mini Trajectory Database Manager:** it is appointed to turn the ellipsoidal coordinates into geohash strings and to create a record inside a database for every prefix of variable length from 1 to 6. For each prefix correspond a set of mini trajectories performed from users. Therefore, for a given prefix, the database provides all the mini-trajectories for a user and a cell.
- **Inference Processor:** it makes requests for user data, used by Blue* algorithm. At the end of computation it returns a mobility model for the user, consisting of a hierarchical pool of DFAs.
- **Mobility Model Handler:** it exploits mobility model to provide several services. It recognizes whether a trajectory belongs to a user, or produces a number of random paths from the user model that can be used synthetic data for simulations.

"GI-learning" - A framework for learning grammar

Analysis and study of inductive algorithms for regular grammars lead to develop a framework to test and compare performances. Our purpose was to observe some feedback about the studied theoretical statements of grammar inference, performed over usually biased environments.

In particular, such goals was achieved through a development of two software: a library collecting the main grammar inference algorithms, and a system exploiting the learning library to address a real application of grammar inference for mobility models.

The "GI-learning" software library was implemented in C++ programming language, due to its high performances in processing a huge amount of data, thanks to its nature of pure compiled language. Furthermore, *Object-oriented paradigm - OOP* allows to set up a modular structure for an easy maintenance and to integrate others inference algorithms.

The main items within the framework (Fig. 5.1) are:

- DFA;
- Blue-fringe algorithm;
  - EDSM algorithm;
  - Blue* algorithm;
- L* algorithm.

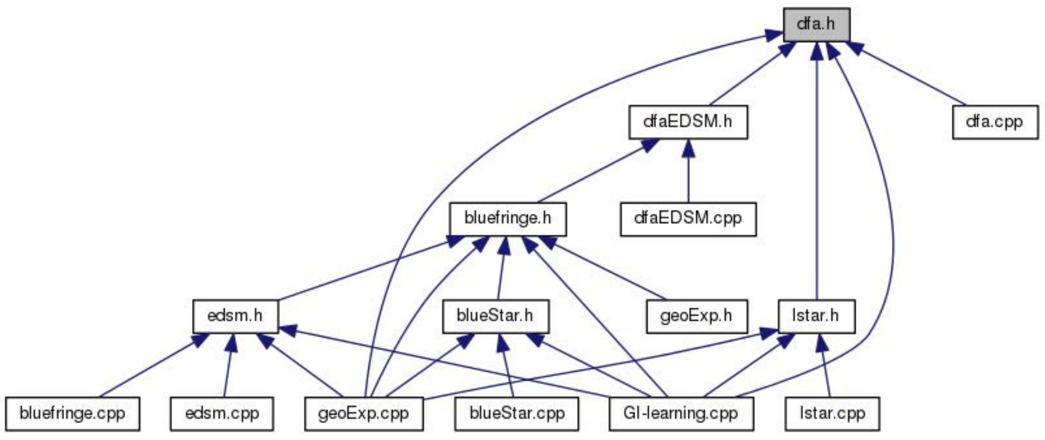


Figure 5.1: Hierarchical view of "GI-learning" framework.

5.2 Geolife dataset

The proposed approache has been tested on the *Geolife dataset* [81, 79], a collection of real users spatial behaviors in a city, collected during the *Geolife project* by *Microsoft Research Asia* [80].

The project aimed at creating a *social network* service based on positioning information, allowing users to share visited places and connect their travelers profile [78]. The system needs to measure the users similarity and suggests potential friends and interesting locations according to their mutual affinity; thus, the system exploits similar friends to suggest the most interesting places not visited.

During the project, travels information have been collected for over five years (Aprile 2007 - August 2012) among 182 users, for 17.621 total trajectories and a temporal extensions of over 50.000 hours of movements.

Several GPS devices was used, namely GPS loggers and GPS-phones, with a high sampling rate (1∼5 seconds or 5∼10 meters) for more then 90% of trajectories within dataset. An heat-map representing dataset distribution is reported in Fig. 5.2.

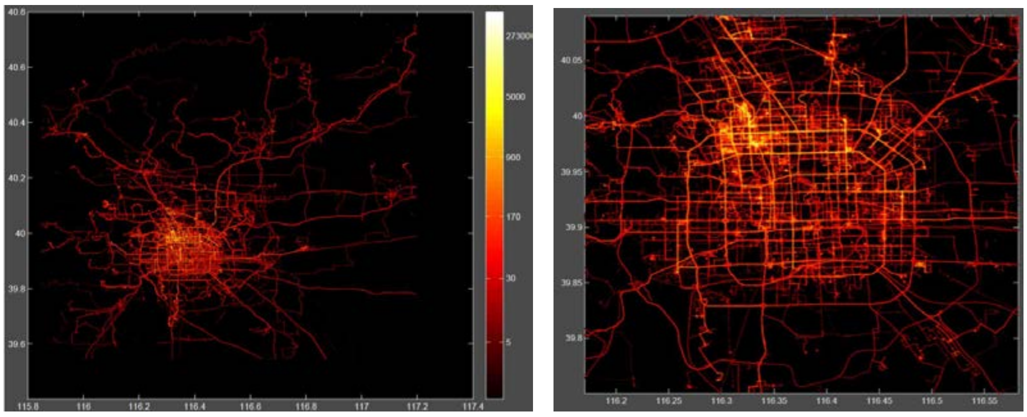
	Statistics
Time span	04/2007 - 8/2012
Number of users	182
Number of trajectories	18.670
Number of points	24.876.978
Total distance	1.292.951 *km*
Total duration	50.176 *hours*

Table 5.1: Some statistics of Geolife dataset [80].

A wide range of outdoors movements are recorded, several daily activities, such as going to work, or school, coming back home, going to gym and so on. Moreover, different kinds of transportation modes are logged such as walk, bike, bus, car, subway, train, airplane and boat.

Over than 30 cities in China are monitored, among these, Beijing is the most frequent one. However, sometimes there are trajectories involving USA and European cities, as Italy.

Data are structured in “trajectories”: a set of GPS sample points, recorded with time (date and hour) and spatial (latitude, longitude and altitude) coordinates.



(a) (b)

Figure 5.2: Geolife dataset heat-map.

5.3 Experimental assessment of mobility model extraction

A first scenario was set up in order to assess the accuracy of mobility models inferred for users, a typical procedure adopted [50] to analyse inferred DFA. All the experiments were performed on Geolife dataset, and the lengths of geohash strings are up to 7 symbols.

Tests were conducted via ***k-fold Cross Validation*** [60], a statistical model evaluation which divides the training set in several subsets used to achieve a non-biased training process.

Some examples are initially removed from the *training set* and inserted into a *test set*. Then, when model is inferred, its accuracy is evaluated using the test set, composed by examples never seen before. However, the accuracy evaluation can depend on which data have been selected as test set, leading to high variance of accuracy score among the different runs.
To address this issue, dataset is parted into k disjoint subsets; hence, k runs are executed, and for each run one subset is used as test set and the residuals $k-1$ subsets are joined to form the training set. Finally, an average score is calculated over the k trial results.

Obviously, to run k trials entails an increasing time complexity; however, accuracy estimations are more reliable, due to a less addiction on how dataset gets parted during the training process.

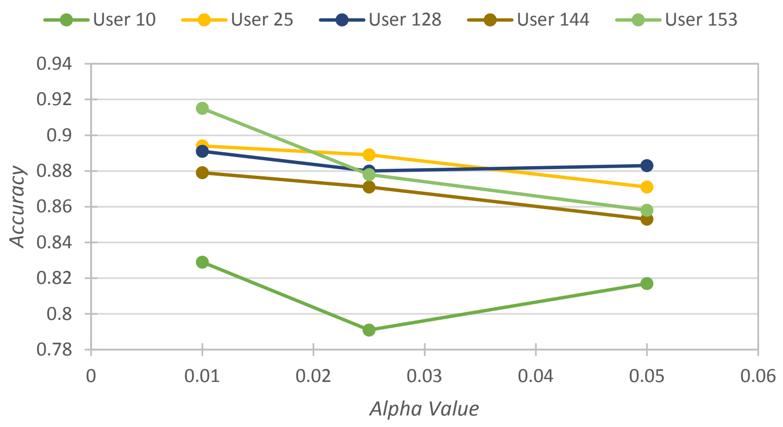
5.3.1 Blue* parameter

A k-fold cross validation procedure has been adopted also to check the best value for the α parameter of Blue* (see Par. 3.4.3) with respect to accuracy and size of inferred automata.

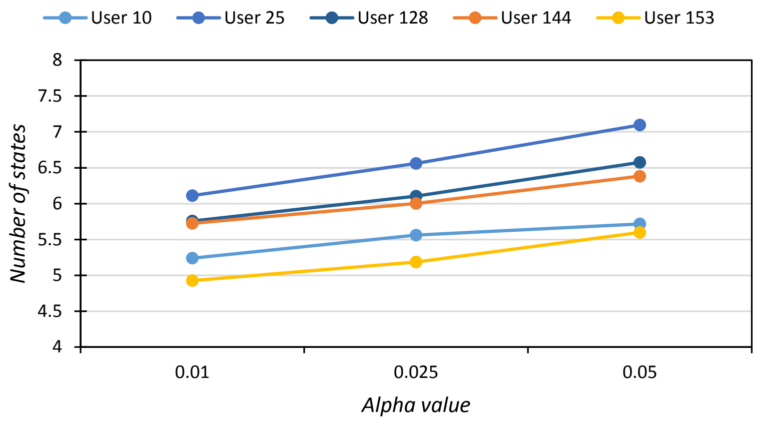
Some typical values, suggested by empirical evaluations [7], are 0.01, 0.025 and 0.05. Among them the value 0.01 resulted the best one because it performs algorithm executions that accept a higher number of merges, despite a higher error on training set.

In particular, two experiments have been conduct setting $k = 5$, thus at each run the ratio between the training set and test set at 80/20 for each users.

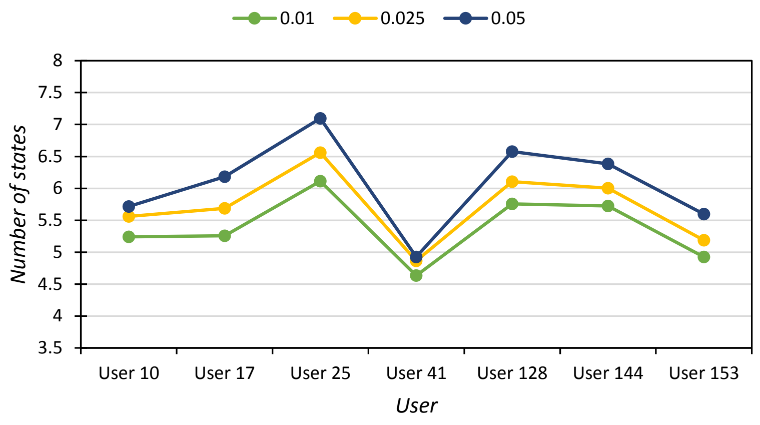
In the following graphs are showed results for the 3 different alpha values, for five representative users. Obtained automata adopting $\alpha = 0.01$ introduces and higher error on training set (Fig. 5.4) but entails an higher accuracy on test (Fig. 5.3a) and smaller size in terms of number of states (Fig. 5.3b).
Analysis of performances of learning processes show that Blue* with $\alpha = 0.01$ need to execute a smaller number heuristic evaluations (Fig. 5.6) and a smaller number of merges (Fig. 5.5).



(a) Accuracy

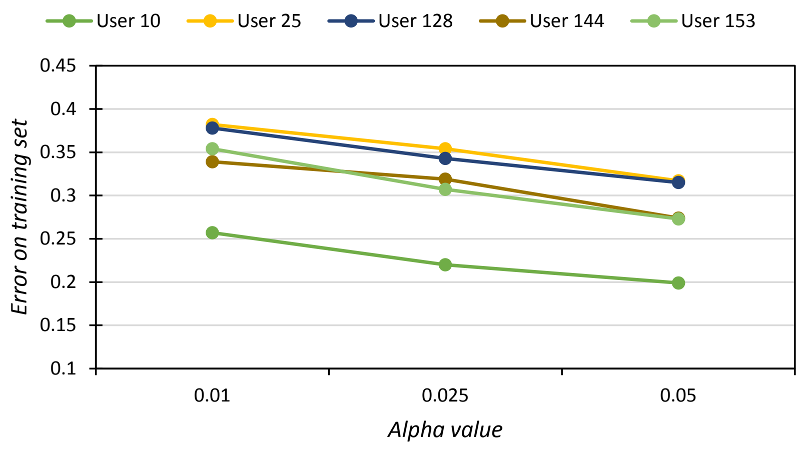


(b) Size of DFA

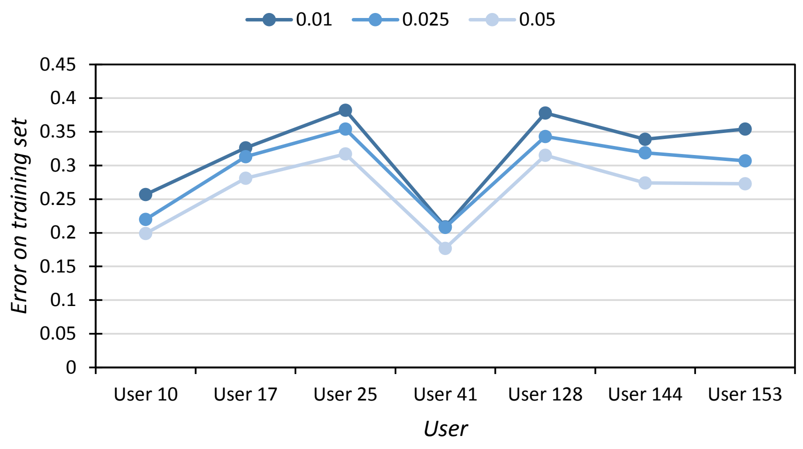


(c) Size of DFA

Figure 5.3: Accuracy and size of DFA for different values of α (0.01, 0.025, 0.05).

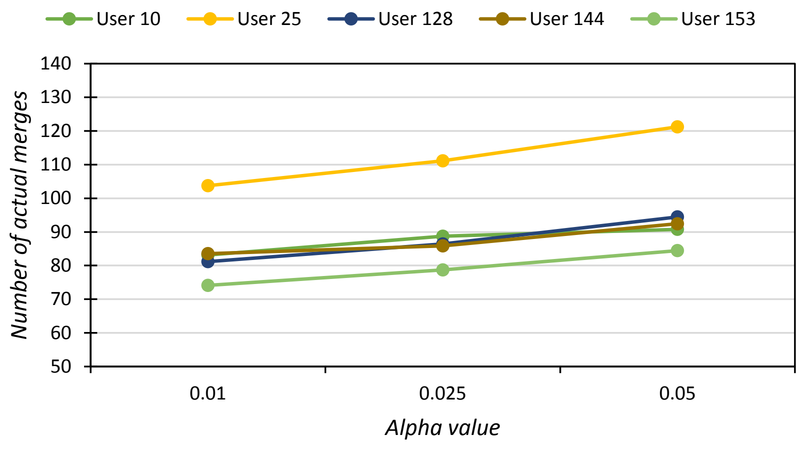


(a) Accuracy

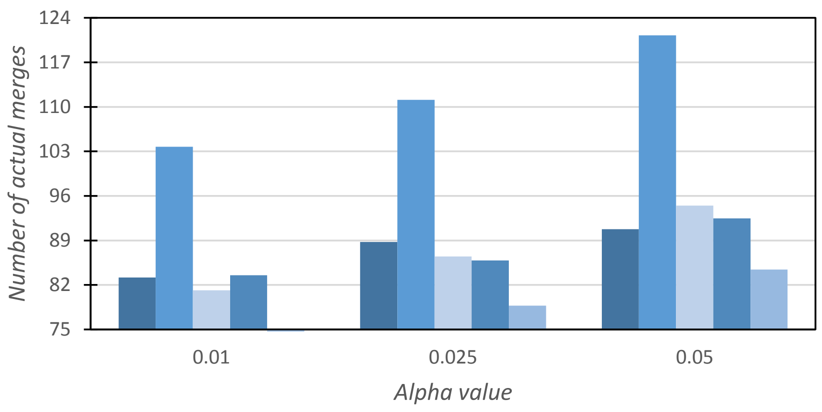


(b) Size of DFA

Figure 5.4: Error on training set for different values of α (0.01, 0.025, 0.05).

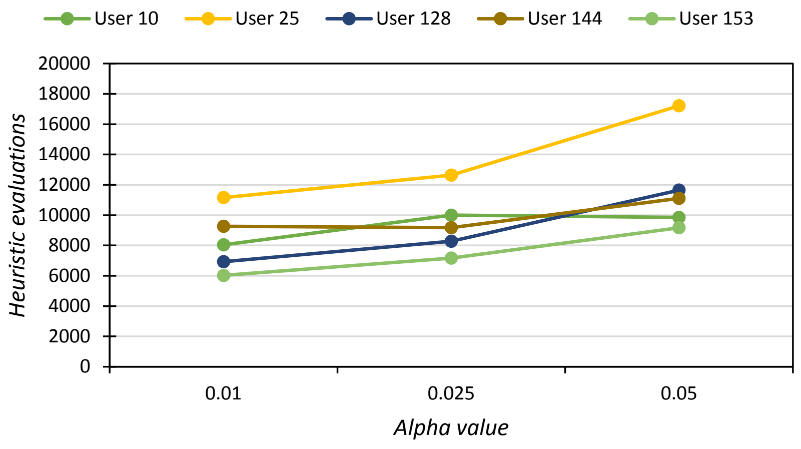


(a) Merges

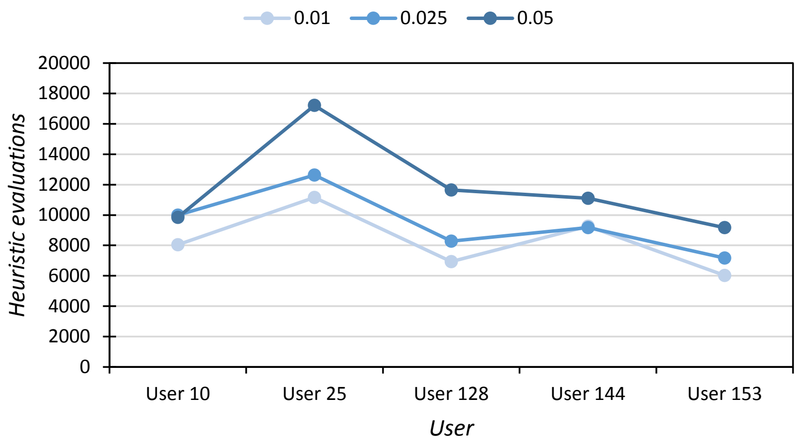


(b) Merges

Figure 5.5: Actual merges for different values of α (0.01, 0.025, 0.05).



(a) Accuracy



(b) Size of DFA

Figure 5.6: Number of heuristic evaluations for different values of α (0.01, 0.025, 0.05).

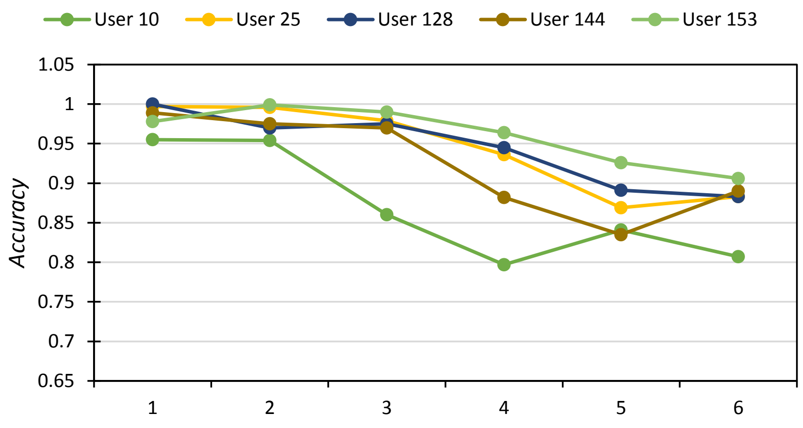
5.3.2 Multi-scale performances

Mobility models for users were tested at different scales of geohash, which entails to work with several lengths of geohash strings.
In particular, prefixes from length 1 up to 6 have been tested; hence, geographical locations were identified by geohash strings with maximum length of 7, detecting geographical areas of about 153 m^2.

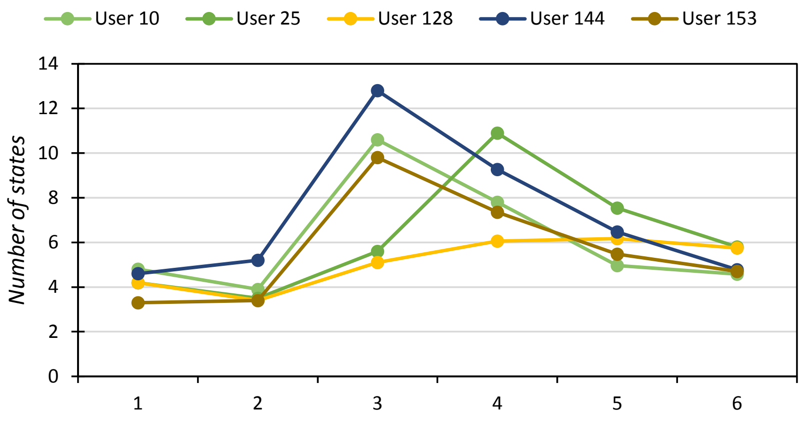
Accuracy results presented (Fig. 5.7a) shows that models are very effective with coarse granularity and this seems reasonable due to high regularities of people in movements observed in the scale of country. Such accuracy show an interesting decreasing on prefixes of length 4 and 5, for areas of abut 5 km, where users shows less regularities; however, it does not significantly affects the efficacy of predictions.

Size of DFAs has a slight different trend (Fig. 5.7b), which seems to increase for strings of length 4 and 5 but decrease again in strings of length 6 and 7. A similar trend has the required number of heuristic evaluations (Fig. 5.8).

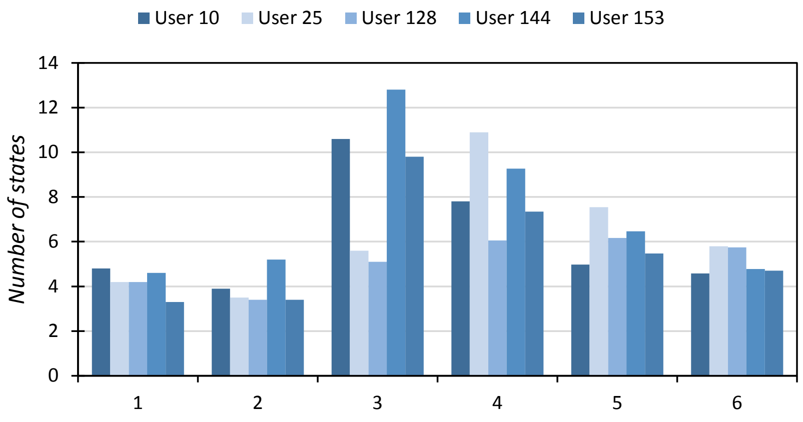
Merges at levels 4 ad 5 are not so effective and entail more algorithm iterations. Several graphs of error rate of Blue* on training set are reported in Fig. 5.8.



(a) Accuracy for different levels

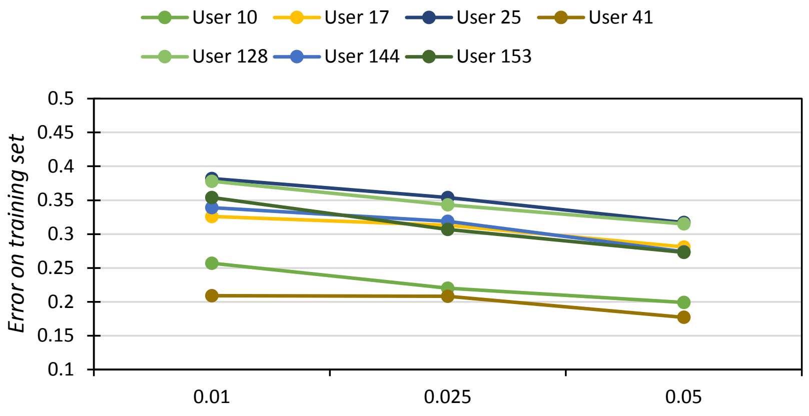


(b) Size of DFA for different levels

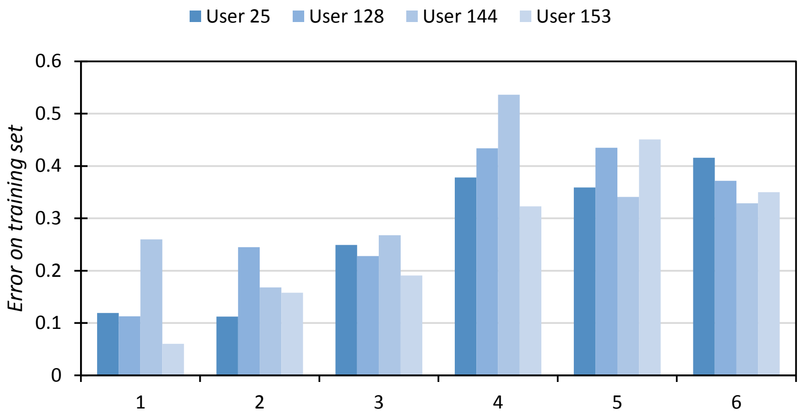


(c) Size of DFA for different levels

Figure 5.7: Accuracy and size of DFA for 6 different levels of scale.

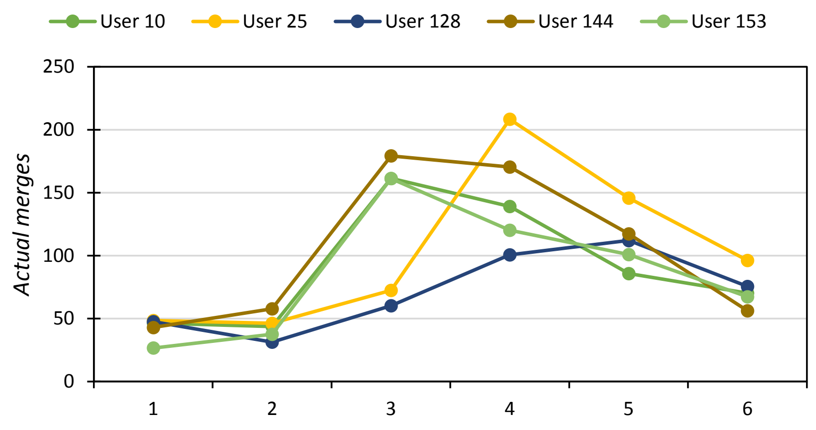


(a) Error on training set

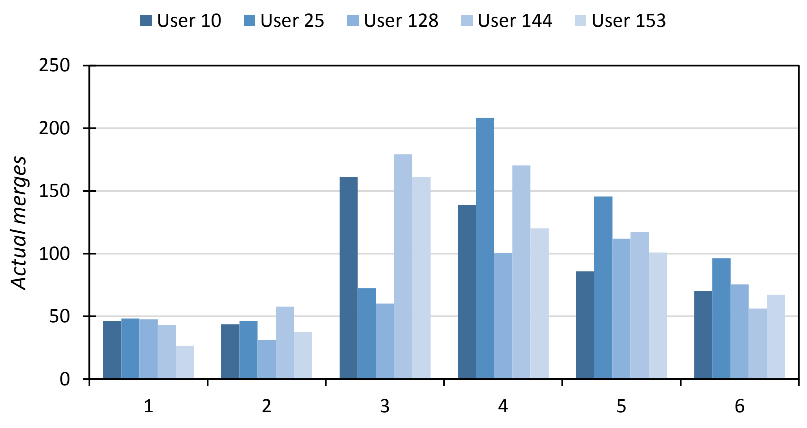


(b) Error on training set

Figure 5.8: Blue* error on training set for 6 different levels of scale.

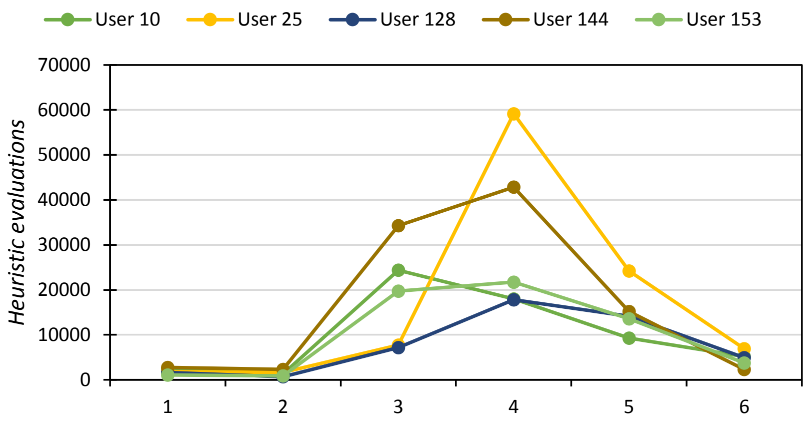


(a) Number of actual merges

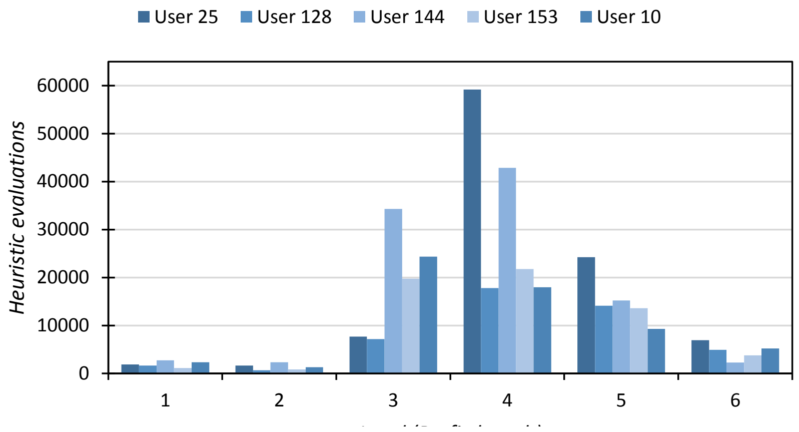


(b) Number of actual merges

Figure 5.9: Blue* number of actual merges for 6 different levels of scale.



(a) Number of heuristic evaluation



(b) Number of heuristic evaluation

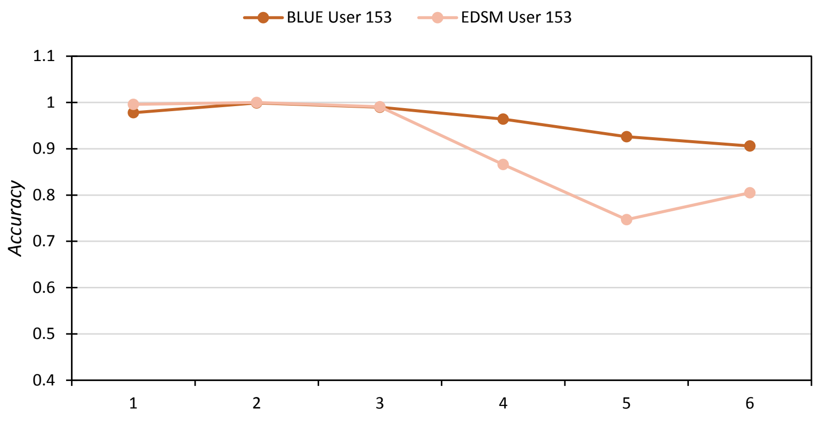
Figure 5.10: Blue* number of heuristic evaluations for 6 different levels of scale.

5.3.3 EDSM vs Blue*

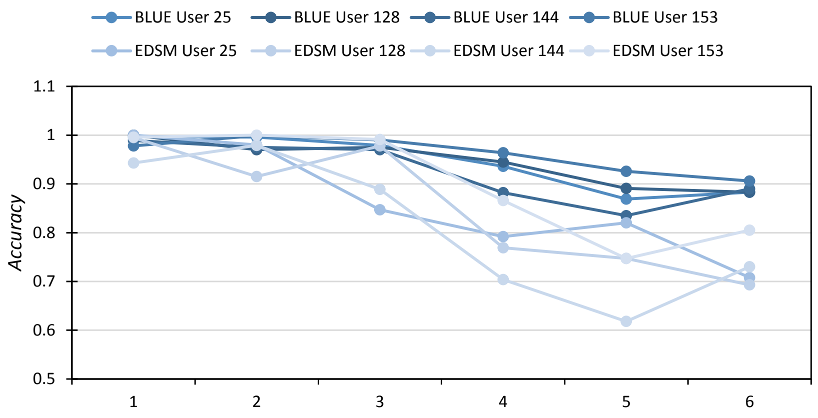
One of the key insight behind the design of the system for mobility models was to exploit Blue* algorithm, to address the unbalance between positive and negative examples, using the frequency information outside the model, in the inference algorithm.

To assess this choice with empirical evidences several comparison tests have been conducted.

In Fig. 5.11a, the obtained accuracy through difference scales is reported for one of the users; it is representative of a common trend between many users, as showed in Fig. 5.11b.
Performances of EDSM and Blue* are similar for coarse granularity, but when dealing with level of analysis where more characterizing behaviors appear, Blue* shows significantly best performances. A better accuracy (Fig. 5.11b) was achieved by more compact models (Fig. 5.12a and 5.12b), and more efficient inference processes are executed, with less heuristic evaluations (Fig. 5.13a and 5.13b) and merges performed (Fig. 5.14a and 5.14b).

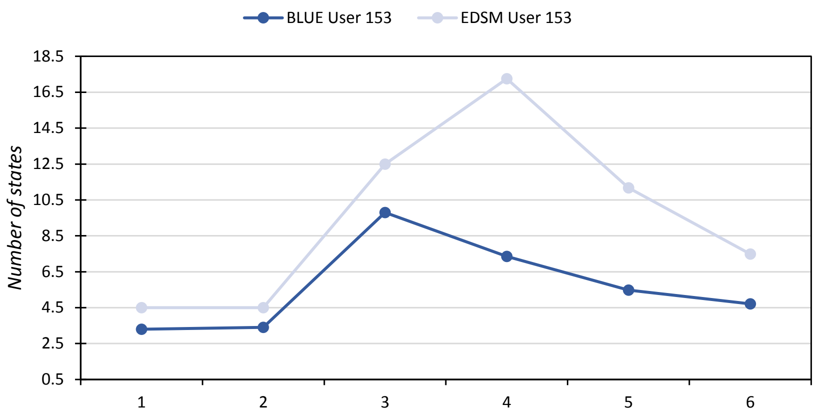


(a) Accuracy

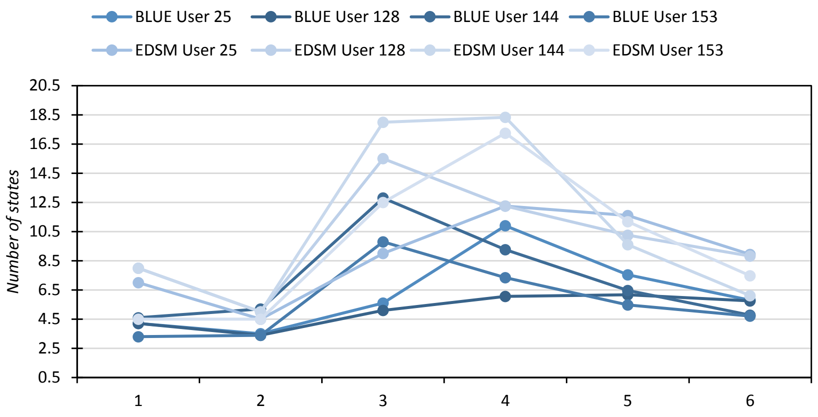


(b) Accuracy

Figure 5.11: EDSM versus Blue* accuracy test.

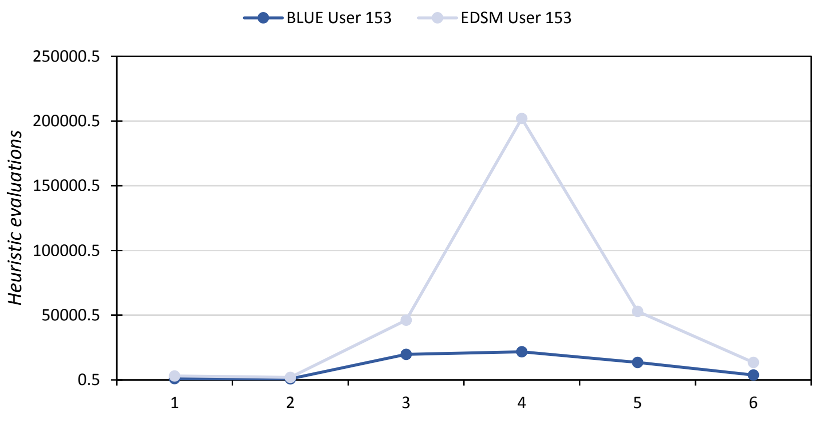


(a) Number of states

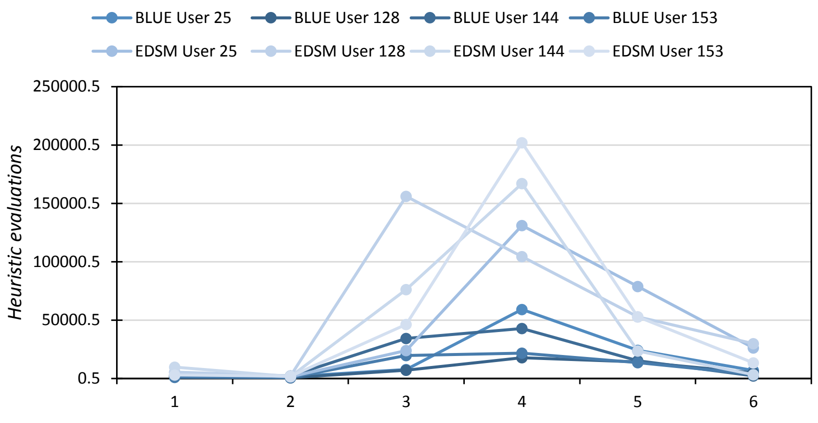


(b) Number of states

Figure 5.12: Number of states for DFA inferred by EDSM and Blue*.

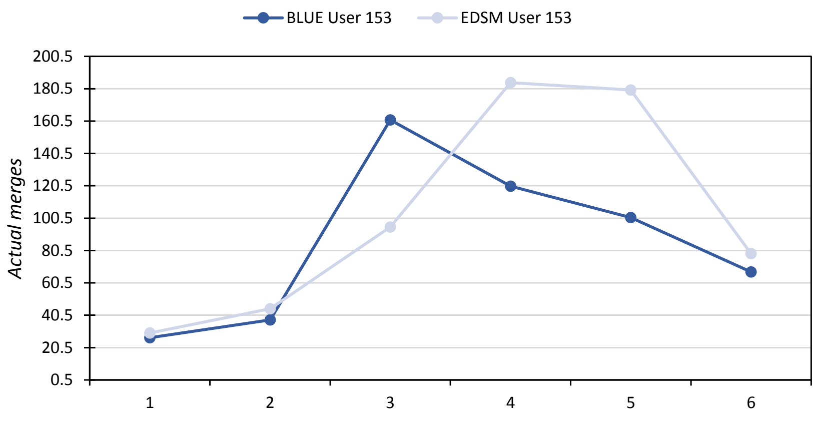


(a) Heuristic evaluations

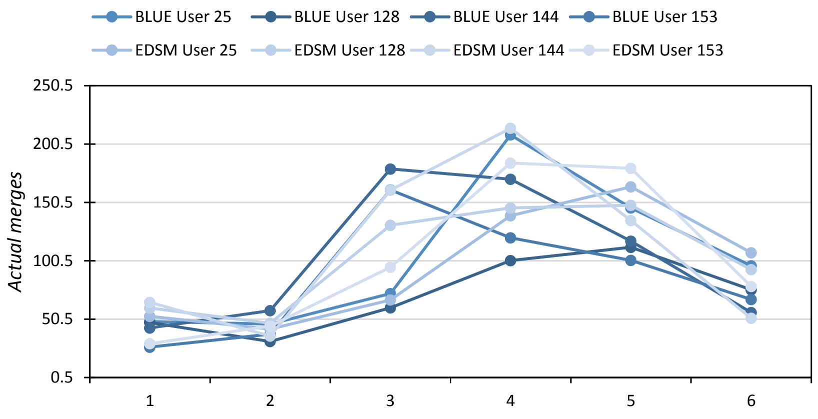


(b) Heuristic evaluations

Figure 5.13: Number of heuristic evaluations performed by EDSM and Blue*.



(a) Number of merges



(b) Number of merges

Figure 5.14: Number of actual merges performed by EDSM and Blue*.

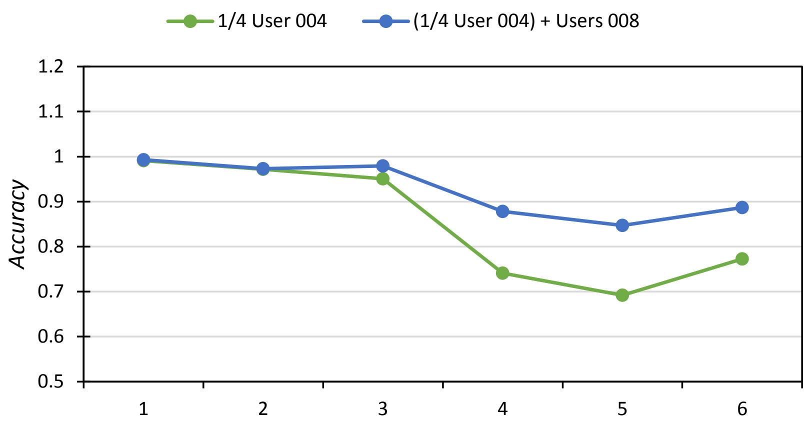
5.3.4 A “cold start” scenario

Finally, the possibility of using the proposed approach as a *recommender system* was tested; to this end, a simulation scenario was considered: a new user, with few information, were modeled by the mobility models of others users to provide improved suggestions about possible paths of interest.

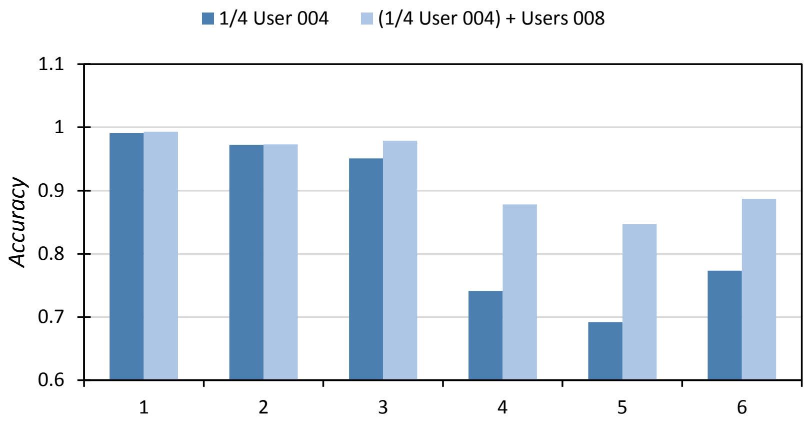
Specifically, the scenario simulates that a user, namely *User 4*, was recently inserted inside the system. Therefore, not so much information are available to make a mobility model. In particular, it was simulated considering only 1/4 of really available data; while a test set was built with the remaining data.

Two models have been inferred: one for the user with 1/4 of data, another using this quarter of data together to all information available for the $User8$, a user resulted similar to $User4$ by visual inspection.
Once inferred the two models, the built test set from $User4$ was fitted in them and accuracy results (Fig. 5.15a and 5.15b) indicate some improvements due to the exploited model of similar user.

This is only a little clue in favor of possible future applications of syntactical models, in order to grasp and exploit structures between observed phenomenons.



(a) Accuracy at different levels



(b) Accuracy at different levels

Figure 5.15: Accuracy at different levels in a *cold start* scenario.

Conclusion

"Perhaps the most profound impact of AI will be the effects that artificial intelligences will have on our understanding of ourselves. Copernicus and later astronomers moved us from the center of the universe to a small planet in one of countless galaxies. Darwin and later evolutionists moved us from the center of creation to our present place among countless DNA-based life forms. [..] What changes await us if we are successful in building machines as intelligent as we?"

Nils Nilsson

This thesis has investigated the use of a symbolic approach to deal with the extraction of structural knowledge, aiming at revealing the hidden structures embedded in a huge amount of raw data.

In fact, traditional methods typically fail in providing actual insight into the real nature of data, because the models they provide consist only in a set of optimal parameters, unable to give a human comprehensible representation of the main features emerging from the data themselves.

A shift in perspective may be of help to tackle with the unaddressed goal of representing knowledge by means of the *structure*, a more natural representation; in particular, concepts and methods borrowed from Algorithmic Learning Theory (ALT), which relies on formal languages and automata, can be very useful in the task of knowledge extraction.

In this context, the learning process can be modelled as grammatical inference, an inductive process able to select the best grammar that is consistent with the available samples, according to the learning model known as *identification in the limit* [37]. Indeed, grammars are a valuable tool to explain and encode the relations embedded into data, thanks to their recursive nature, that allows to perform multi-scale analyses.

In order to highlight the potential of grammatical inference, the method applied to the problem of inferring mobility models for *Smart Cities* [32].

A system based on regular language was implemented; it elaborates position data, and employs a new software library for grammatical inference task, designed and implemented as part of this thesis.

The performances of the proposed system was tested in a real life scenario, using data provided by the *GeoLife project* [78], consisting of a wealth of spatial movements recorded via GPS loggers. The system was thoroughly assessed through a complete set of experiments in order to measure the reliability of the models by different algorithms.

An initial test has regarded the accuracy of the model prediction, through classical *k-fold cross validation*, in order to measure precision across different scale level, and to test the ability to correctly recognize previously unseen behaviors.

A second experiment was conducted to compare automata inferred by two different algorithms, namely EDSM and Blue*, and showed that the latter results in an improvement in terms of efficacy (precision and size of the obtained automata) and efficiency.

Finally, models provided by the proposed approach were tested in the context of a *recommender system*, in order to verify their ability to provide suggestions about possible paths of interest.

Results have confirmed the reliability of the proposed methods, whose intrinsically recursive nature is well fit for describing multi-scale models, which in turn emerge naturally in mobility scenarios.

Bibliography

[1] Dana Angluin. “Learning Regular Sets from Queries and Counterexamples”. In: *Inf. Comput.* (Nov. 1987), pp. 87–106.

[2] Dana Angluin. “Queries and concept learning”. In: *Machine learning* (1988), pp. 319–342.

[3] Dana Angluin and Carl H. Smith. “Inductive Inference: Theory and Methods”. In: *ACM Comput. Surv.* (Sept. 1983), pp. 237–269.

[4] Daniel Ashbrook and Thad Starner. “Using GPS to learn significant locations and predict movement across multiple users”. In: *Personal and Ubiquitous Computing* (2003), pp. 275–286.

[5] Zoran Balkić, Damir Šoštarić, and Goran Horvat. “GeoHash and UUID identifier for multi-agent systems”. In: *Agent and Multi-Agent Systems. Technologies and Applications*. Springer, 2012, pp. 290–298.

[6] Janis Barzdinš. “Two theorems on the limiting synthesis of functions”. In: *Theory of algorithms and programs* (1974), pp. 82–88.

[7] Ken Black. *Business statistics: for contemporary decision making*. John Wiley & Sons, 2011.

[8] Alselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. “Occam’s Razor”. In: *Inf. Process. Lett.* (Apr. 1987), pp. 377–380.

[9] Stanley Burris and Hanamantagida Pandappa Sankappanavar. *A course in universal algebra*. Springer, 1981.

[10] Roberta Calegari, Mirco Musolesi, Franco Raimondi, and Cecilia Mascolo. “CTG: a connectivity trace generator for testing the performance of opportunistic mobile systems”. In: *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM. 2007, pp. 415–424.

[11] Tracy Camp, Jeff Boleng, and Vanessa Davies. “A survey of mobility models for ad hoc network research”. In: *Wireless communications and mobile computing* (2002), pp. 483–502.

[12] Jaime G Carbonell, Ryszard S Michalski, and Tom M Mitchell. “An overview of machine learning”. In: *Machine learning*. Springer, 1983, pp. 3–23.

[13] John Case and Carl Smith. “Comparison of identification criteria for machine inductive inference”. In: *Theoretical Computer Science* (1983), pp. 193–220.

[14] Ling Chen, Mingqi Lv, and Gencai Chen. “A system for destination and future route prediction based on trajectory mining”. In: *Pervasive and Mobile Computing* (2010), pp. 657–676.

[15] Francisco Chinchilla, Mark Lindsey, and Maria Papadopouli. “Analysis of wireless information locality and association patterns in a campus”. In: *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*. Vol. 2. IEEE. 2004, pp. 906–917.

[16] Noam Chomsky. *Syntactic Structures*. Mouton, 1957.

[17] Noam Chomsky. “Three models for the description of language”. In: *Information Theory, IRE Transactions on* (1956), pp. 113–124.

[18] Orlando Cicchello and Stefan C. Kremer. “Inducing grammars from sparse data sets: a survey of algorithms and results”. In: *The Journal of Machine Learning Research* (2003), pp. 603–632.

[19] Matthew S. Collins and Jonathan J. Oliver. “Efficient Induction of Finite State Automata”. In: *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*. UAI’97. Providence, Rhode Island: Morgan Kaufmann Publishers Inc., 1997, pp. 99–107.

[20] François Coste and Jacques Nicolas. “How considering incompatible state mergings may reduce the DFA induction search tree”. In: *Grammatical Inference*. Springer, 1998, pp. 199–210.

[21] Pietro Cottone, Salvatore Gaglio, Giuseppe Lo Re, and Marco Ortolani. “User activity recognition for energy saving in smart homes”. In: *Pervasive and Mobile Computing* (2014).

[22] Jakob Aarøe Dam. “A Web-Based Weather Service for Wind Sports”. PhD thesis. Aarhus Universitet, Datalogisk Institut, 2009.

[23] A. De Paola, M. Ortolani, G. Lo Re, G. Anastasi, and S.K. Das. “Intelligent Management Systems for Energy Efficiency in Buildings: A Survey”. In: *ACM Computing Surveys* 47.1 (2014), p. 38.

[24] Alessandra De Paola, Salvatore Gaglio, Giuseppe Lo Re, and Marco Ortolani. “Multi-sensor Fusion through Adaptive Bayesian Networks”. In: *AI*IA 2011: Artificial Intelligence Around Man and Beyond*. Vol. 6934. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 360–371.

[25] Nai Ding, Lucia Melloni, Hang Zhang, Xing Tian, and David Poeppel. “Cortical tracking of hierarchical linguistic structures in connected speech”. In: *Nature neuroscience* (2016), pp. 158–164.

[26] P. Dupont, L. Miclet, and E. Vidal. “What is the Search Space of the Regular Inference?” In: *In Proceedings of the Second International Colloquium on Grammatical Inference (ICGI’94)*. Springer Verlag, 1994, pp. 25–37.

[27] Pierre Dupont. “Regular grammatical inference from positive and negative samples by genetic search: the GIG method”. In: *Grammatical Inference and Applications*. Springer, 1994, pp. 236–245.

[28] Anthony Fox, Chris Eichelberger, John Hughes, and Skylar Lyon. “Spatiotemporal indexing in non-relational distributed databases”. In: *Big Data, 2013 IEEE International Conference on*. IEEE. 2013, pp. 291–299.

[29] Menahem Friedman and Abraham Kandel. “Introduction to pattern recognition”. In: *Series in Machine Perception Artificial Intelligence* 32 (1999).

[30] M. Frixione and D. Palladino. *La computabilità: algoritmi, logica, calcolatori*. Le bussole. Carocci, 2011.

[31] King-Sun Fu and Taylor L. Booth. “Grammatical Inference: Introduction and Survey-Part I”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (1986), pp. 343–359.

[32] W. Zhang G. Pan G. Qi. “Trace analysis and mining for smart cities: issues, methods, and applications”. In: *IEEE Communications Magazine* 121 (2013).

[33] Sahin Cem Geyik, Eyuphan Bulut, and Boleslaw K Szymanski. “Grammatical inference for modeling mobility patterns in networks”. In: *Mobile Computing, IEEE Transactions on* (2013), pp. 2119–2131.

[34] Lee Giles, Steve Lawrence, and Ah Chung Tsoi. “Noisy time series prediction using recurrent neural networks and grammatical inference”. In: *Machine learning* (2001), pp. 161–183.

[35] Donald Gillies and Giulio Giorello. *La filosofia della scienza nel XX secolo*. GLF editori Laterza, 2010, pp. 10–11.

[36] E. Mark Gold. "Complexity of automaton identification from given data". In: *Information and Control* (1978), pp. 302–320.

[37] E. Mark Gold. "Language identification in the limit". In: *Information and control* (1967), pp. 447–474.

[38] Marta C. Gonzalez, Cesar A. Hidalgo, and Albert-Laszlo Barabasi. "Understanding individual human mobility patterns". In: *Nature* (2008), pp. 779–782.

[39] Peter Grünwald. "A tutorial introduction to the minimum description length principle". In: *Advances in minimum description length: Theory and applications* (2005), pp. 23–81.

[40] Andreas Hagerer, Hardi Hungar, Oliver Niese, and Bernhard Steffen. "Model generation by moderated regular extrapolation". In: *Fundamental Approaches to Software Engineering*. Springer, 2002, pp. 80–95.

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

[42] J.H. Holland. *Adaptation in natural and artificial systems*. The University of Michigan Press, Ann Arbor, 1975.

[43] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Pearson/Addison Wesley, 2007.

[44] David Hume. *A treatise of human nature*. Courier Corporation, 2012.

[45] George H John, Ron Kohavi, Karl Pfleger, et al. "Irrelevant features and the subset selection problem". In: *Machine learning: proceedings of the eleventh international conference*. 1994, pp. 121–129.

[46] David H Jonassen, Katherine Beissner, and Michael Yacci. *Structural knowledge: Techniques for representing, conveying, and acquiring structural knowledge*. Psychology Press, 1993.

[47] Feliz Kristianto Karnadi, Zhi Hai Mo, and Kun-chan Lan. "Rapid generation of realistic mobility models for VANET". In: *Wireless Communications and Networking Conference*. IEEE. 2007, pp. 2506–2511.

[48] Minkyong Kim, David Kotz, and Songkuk Kim. *Extracting a Mobility Model from Real User Traces*. 2006.

[49] Zoltán Koppányi. *Individual Human Mobility Modeling with Probabilistic Automata*. 2012.

[50] Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. "Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm". In: *Proceedings of the 4th International Colloquium on Grammatical Inference*. ICGI '98. London, UK, UK: Springer-Verlag, 1998, pp. 1–12.

[51] Willem JM Levelt. *An introduction to the theory of formal languages and automata*. John Benjamins Publishing, 2008.

[52] Quannan Li, Yu Zheng, Xing Xie, Yukun Chen, Wenyu Liu, and Wei-Ying Ma. "Mining user similarity based on location history". In: *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*. ACM. 2008, p. 34.

[53] G. Lo Re, M. Morana, and M. Ortolani. "Improving User Experience via Motion Sensors in an Ambient Intelligence Scenario." In: *Pervasive and Embedded Computing and Communication Systems (PECCS), 2013*. 2013, pp. 29–34.

[54] Wesley Mathew, Ruben Raposo, and Bruno Martins. "Predicting future locations with hidden Markov models". In: *Proceedings of the 2012 ACM conference on ubiquitous computing*. ACM. 2012, pp. 911–918.

[55] Ryszard S Michalski. "Concept learning". In: *Encyclopedia of artificial intelligence* 1 (1987), pp. 185–194.

[56] Ryszard S Michalski. "Learning strategies and automated knowledge acquisition". In: *Computational models of learning*. Springer, 1987, pp. 1–19.

[57] Ryszard S Michalski. "Understanding the nature of learning: Issues and research directions". In: *Machine learning: An artificial intelligence approach* 2 (1986), pp. 3–25.

[58] AA Mitchell and MT Chi. "Measuring knowledge within a domain". In: *The representation of cognitive structure* (1984), pp. 85–109.

[59] Tom M Mitchell. "Generalization as search". In: *Artificial intelligence* (1982), pp. 203–226.

[60] Tom M Mitchell et al. *Machine learning*. 1997.

[61] Tim Oates, Arnold P Boedihardjo, Jessica Lin, Crystal Chen, Susan Frankenstein, and Sunil Gandhi. "Motif discovery in spatial trajectories using grammar inference". In: *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*. ACM. 2013, pp. 1465–1468.

[62] Jose Oncina and Pedro Garcia. “Identifying Regular Languages In Polynomial Time”. In: *advances in structural and syntactic pattern recognition, volume 5 of series in machine perception and artificial intelligence*. World Scientific, 1992, pp. 99–108.

[63] Steven Pinker. *Language, cognition, and human nature: Selected articles*. Oxford University Press, 2013.

[64] Leonard Pitt. “Inductive inference, DFAs, and computational complexity”. In: *Analogical and inductive inference* (1989), pp. 18–44.

[65] Karl R Popper. “The logic of scientific discovery”. In: *London: Hutchinson* (1959).

[66] R.L. Rivest and R.E. Schapire. “Inference of Finite Automata Using Homing Sequences”. In: *Information and Computation* (1993), pp. 299–347.

[67] F. Rosenblatt. *The perceptron: A perceiving and recognizing automaton*. 1957.

[68] Gilbert Ryle. *The concept of mind*. Routledge, 2009.

[69] Marc Sebban and Jean-Christophe Janodet. “On State Merging in Grammatical Inference: A Statistical Approach for Dealing with Noisy Data”. In: *Machine Learning, Proceedings of the Twentieth International Conference USA*. 2003.

[70] Marc Sebban, Jean-Christophe Janodet, and Frédéric Tantini. *BLUE: a Blue-Fringe Procedure for Learning DFA with Noisy Data*.

[71] Pavel Senin, Jessica Lin, Xing Wang, Tim Oates, Sunil Gandhi, Arnold P Boedihardjo, Crystal Chen, and Susan Frankenstein. *Time series anomaly discovery with grammar-based compression*. 2015.

[72] Herbert A Simon. “Why should machines learn?” In: *Machine learning*. Springer, 1983, p. 28.

[73] Chaoming Song, Zehui Qu, Nicholas Blumm, and Albert-László Barabási. “Limits of predictability in human mobility”. In: *Science* (2010), pp. 1018–1021.

[74] Peter Norvig Stuart Russell. *Artificial Intelligence: A Modern Approach*. 3rd. Prentice Hall Series in Artificial Intelligence. Prentice Hall, 2010.

[75] Roberto Trasarti, Fabio Pinelli, Mirco Nanni, and Fosca Giannotti. “Mining mobility user profiles for car pooling”. In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2011, pp. 1190–1198.

[76] Leslie G Valiant. “A theory of the learnable”. In: *Communications of the ACM* (1984), pp. 1134–1142.

[77] Thomas Zeugmann and Steffen Lange. “A Guided Tour Across the Boundaries of Learning Recursive Languages”. In: *Algorithmic Learning for Knowledge-Based Systems, GOSLER Final Report*. London, UK, UK: Springer-Verlag, 1995, pp. 190–258.

[78] Yu Zheng, Yukun Chen, Xing Xie, and Wei-Ying Ma. “GeoLife2. 0: a location-based social networking service”. In: *Mobile Data Management: Systems, Services and Middleware, 2009. MDM’09. Tenth International Conference on.* IEEE. 2009, pp. 357–358.

[79] Yu Zheng, Quannan Li, Yukun Chen, Xing Xie, and Wei-Ying Ma. “Understanding mobility based on GPS data”. In: *Proceedings of the 10th international conference on Ubiquitous computing.* ACM. 2008, pp. 312–321.

[80] Yu Zheng, Xing Xie, and Wei-Ying Ma. “GeoLife: A Collaborative Social Networking Service among User, Location and Trajectory.” In: *IEEE Data Eng. Bull.* (2010), pp. 32–39.

[81] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. “Mining interesting locations and travel sequences from GPS trajectories”. In: *Proceedings of the 18th international conference on World wide web.* ACM. 2009, pp. 791–800.