Probabilistic Grammatical Inference System for Finite State Automata -
The P-GIFSA System

CHAFIA KARA-MOHAMED (alias HAMDI-CHERIF)¹,², ARWA ALBELAIHI¹,³, MUNEERA AL-SHAMRI¹,³, RE’AM AL-HUSSAN¹,³, ABOUBEKEUR HAMDI-CHERIF¹,⁴

¹Computer College
Qassim University
PO Box 6688 – 51452 Buraydah
SAUDI ARABIA
(²) email: smhmd@qu.edu.sa
³armnre@gmail.com
⁴Permanent Address: Université Ferhat Abbas Setif (UFAS)
Faculty of Science
Computer Science Department
19000 Setif
ALGERIA
(⁴) email: shrief@qu.edu.sa

Abstract: - A probabilistic grammatical inference system for finite state automata (P-GIFSA) is designed, developed and tested. P-GIFSA represents an integrated system for inferring finite state automata (FSA) implementing probabilistic algorithms. Instead of traditionally defining languages over strings, P-GIFSA considers a distribution over strings via the inference of probabilistic finite automata (PFA). Whether probabilistic or non-probabilistic, there is no freely-available portable software that addresses the GI of FSA. As a result, researchers are entitled to write ad hoc independent programs to solve a specific problem - programs that might face portability issues. Using the unified modeling language (UML), P-GIFSA offers a continuously-upgradable software system initially implementing two probabilistic algorithms, namely ALERGIA, and distinguishing strings automata inference (DSAI) algorithms. For portability reasons, Java™ programming language is used for development, enhanced by a friendly graphical user interface (GUI).

Key-Words: - Grammatical inference (GI), Finite state automata (FSA), Probabilistic finite automata (PFA), Deterministic probabilistic finite automata (DPFA), Deterministic frequency finite automata (DFFA), Unified modeling language (UML)-based design, Web-based applications.

1. Introduction
With the deluge of data and information provided by the World Wide Web (WWW), data mining techniques and the corresponding embedded machine learning algorithms have gone through a profound mutation for the last two decades, or so. As a branch of artificial intelligence (AI), machine learning is concerned with the design and development of algorithms and tools that allow computers to learn from experience or by analogy. As a field of machine learning, grammatical inference (GI) refers to the elaboration of an unknown formal language by processing a set of correct and eventually erroneous sentences, supplied to the machine as training examples. The goal of inference is to discover an unknown pattern or language from a relatively tiny sample of available training examples. As an extension of a previous work dealing with the inference of FSA in the non-probabilistic case, our work now addresses one important subject in the field of GI, namely the inference of FSA using probabilistic methods [1]. Instead of defining a language as a set of strings, one can define a distribution over strings. In the sequel, we confine ourselves to the automatic inference of an automaton, responsible for generating one probabilistic language, from positive and eventually negative examples. In general, there are two kinds of learning, namely inductive and deductive. Because of its conventional usage in GI, we focus on the inductive learning only. Indeed, the main task for automated learning boils down to extracting valuable information or knowledge from data on the basis of the inductive conclusions that are drawn from a given set of data. Despite the huge efforts made by researchers in algorithms design and analysis including context-free grammar
learning, software implementation in the GI area remains very limited [2]. The latest available GI tools are the MATLAB GI-Toolbox™ [3] and EMILE system [4]. Particularly, there is no freely-available integrated and portable software that addresses GI of FSA whether in the probabilistic or non-probabilistic settings. In the present work, in order to infer an FSA from a set of examples, we use two main probabilistic algorithms in GI, namely ALERGIA [5] and distinguishing strings automata inference (DSAI) algorithms [6]. As it stands now, P-GIFSA is a tool initially implementing these two algorithms and easily upgradable. Traditionally, GI has been applied in various domains such as information theory, formal languages, language acquisition, computational linguistics, pattern recognition, computational learning theory, bioinformatics, and in control systems, among others [7].

In the next section, we describe the theoretical background related to the proposed system. In Section 3, we summarize the algorithms implemented in P-GIFSA. Section 4 is devoted to P-GIFSA design implementation and testing, showing the results in its GUI. The paper ends with a conclusion summarizing the main results and designating some possible future enhancements.

2. P-GIFSA Background

2.1 Motivations

The main motivation behind the development of P-GIFSA is that, although FSAs are the simplest representation of formal languages in the Chomsky hierarchy, there is a deficiency in the availability of free and portable software for GI in general and for GI as applied to FSA, in particular. The shortage is greater when it comes to probabilistic algorithms for FSA inference.

It is now customary that researchers write ad hoc independent and usually non-portable programs to solve specfic problems. Thus, there is an urgent need for a computational system that integrates in a comprehensive and organized fashion, well-known off-the-shelf probabilistic algorithms for inferring FSA. We believe that such an environment will save precious time for researchers by providing them with “all-in-one” software while providing an open framework for further development. Based on these arguments, we set out the objective of initially extending the so-called GIFSA [1] into the P-GIFSA as a system for FSA inference; the extension being done for the probabilistic case with a friendly graphical user interface (GUI) as a prelude to a more general system.

2.2 P-GIFSA characteristics

2.2.1 Existing systems

(i) GIFSA

GIFSA (grammatical inference for finite state automata) is a system implementing tabu search algorithm and minimum description length (MDL) owing to their popularity. Tabu search method [8], as any search algorithm, requires the definition of a space and the definition of chosen local operators to move around this space. Each neighbors’ solution should be measured. The best of these neighbors, according to a given fitness function, is kept for the next iteration. The idea is to iteratively try to find a neighbor of the current solution that improves it, i.e., that obtains the highest fitness value. The goal of using this particular search algorithm is to learn regular languages defined by non-deterministic finite automata (NFAs).

Minimum description length (MDL) is the second chosen algorithm for implementation in GIFSA. It is one representative of data compression techniques. The intuitive principle of simplicity or parsimony proposed in Occam’s razor is at the root of the formalization of the minimum description length (MDL) principle. This latter states that the most likely model to describe the data is the one that requires the fewest bits to encode. The description length is defined as the number of bits required for the description of the model [9], [10].

(ii) MATLAB™ GI Toolbox

MATLAB™ GI Toolbox represents one of the rare available systems is GI [3]. It implements some well-known GI algorithms such as evidence driven state merging (EDSM) algorithm, k-testable machine and regular positive and negative inference (RPNI) algorithm. The main motivation behind this software is to assess the suitability of the MATLAB™ platform in the implementation of GI algorithms. Specifically, the developed software experimentally shows that MATLAB™ implementation of state merging algorithms can effectively be applied and deployed using MATLAB™ in a comparable manner as other machine learning tools. In addition to the above-cited algorithms, an updated version of GI Toolbox is now available and incorporates additional algorithms such as MDI (minimum divergence inference) algorithm [11] and OSTIA (onward subsequential transducer inference algorithm) [12]. Although the main functions used in this software
are available as text or m-files over the Internet\textsuperscript{1}, the MATLAB\textsuperscript{TM} GI Toolbox still requires the costly MATLAB\textsuperscript{TM} platform.

(iii) EMILE Toolbox

The other system available is EMILE\textsuperscript{2} (entity modeling intelligent learning engine). It is based on the idea that similar concepts can be substituted if they are in the same context without destroying the structure of the sentence. EMILE has been applied in biomedical and semantic learning, among others. However, EMILE is a closed system in that users have to conform to the algorithm embedded in it and they have no possibility to upgrade it. Therefore, EMILE cannot be considered as a toolbox per se [4].

2.2.2 Context of P-GIFSA development

The goal of P-GIFSA is to make available to GI researchers a set of implemented algorithms that help them in probabilistic finite automata (PFA) and deterministic probabilistic finite automata (DPFA) inference issues. The software inductively builds an automaton that can recognize any sentence that belongs to the language to be learned by this FSA. On the theoretical side, GI algorithms are ordered in classes of similar characteristics. Because all these algorithms require advanced study of some principles of formal language theory, we consider only two of these to start with.

On the implementation level, the available GI software tools cited above such as GIFSA, MATLAB\textsuperscript{TM} GI-Toolbox and EMILE do not consider the same algorithms implemented in P-GIFSA. Following the same lines of development as in GIFSA, and in order to fill this gap, P-GIFSA offers the following characteristics:

- **Intuitive input GUI**: P-GIFSA is GUI-supported and allows input to be easily and intuitively introduced to the system in addition to various general utilities such as help and other basic definitions of formal languages.
- **Readability**: P-GIFSA provides outputs that are intuitively offered to the user in various forms such as text and graphical forms.
- **Probabilistic Algorithms**: Algorithms are chosen from the probabilistic techniques, not implemented in previous similar systems.
- **Upgradability**: Since it is based on state-of-the-art software engineering techniques, P-GIFSA is easily upgradable and users can add their own algorithms.

\textsuperscript{1}http://gtoolbox.googlecode.com/svn/trunk/Beta
\textsuperscript{2}http://turing.wins.uva.nl/~pietera/ALS/

- **Free**: no license is required to use P-GIFSA for non-commercial purposes.

2.2 Theoretical tools [13]

2.2.1 Probabilistic finite automata (PFA)

The concept of probabilistic finite automata (PFA) naturally extends the concept of non-deterministic finite automata (NFA) with the addition of probabilities.

**Definition 1 - Probabilistic finite automata (PFA)**

As an extension of NFA, a probabilistic finite automata (PFA) is a tuple defined by

\[ A = (\Sigma, Q, I_p, F_p, \delta_p) \]

where:

- \( \Sigma \) is the input alphabet,
- \( Q \): a finite set of states, labeled \( q_1, \ldots, q_{|Q|} \),
- \( I_p: Q \rightarrow Q' \cap [0,1] \), the initial-state probabilities,
- \( F_p: Q \rightarrow Q' \cap [0,1] \), the final-state probabilities,
- \( \delta_p: Q \times (\Sigma \cup \{\lambda\}) \times Q \rightarrow Q' \) is a transition function,
- \( \delta_p \): The function is complete: \( \delta_p(q, a, q') = 0 \) means “no transition between \( q \) and \( q' \) is labeled with \( a \).” The transition is denoted \((q, q', P)\) instead of \( \delta_p(q, a, q') = P \), where \( P \) is the probability.

To comply with the axioms of a probability, the functions \( \delta_p \), \( I_p \) and \( F_p \) are such that:

\[ \sum_{q \in Q} I_p(q) = 1 \]

2.2.2 Deterministic probabilistic finite automata (DPFA)

Deterministic probabilistic finite automata (DPFA) are the probabilistic counterpart of deterministic finite automata (DFA).

(i) **Definition 2 - DPFA**

A probabilistic finite automata (PFA) denoted by \( A = (\Sigma, Q, I_p, F_p, \delta_p) \), with symbols as defined above, is a deterministic probabilistic finite automata (DPFA) if and only if it meets the following conditions:

- \( \exists q_i \in Q \), unique initial state such that \( I_p(q_i) = 1 \);
- \( \delta_p \subseteq Q \times \Sigma \times Q \) has no \( \lambda \)-transition
- \( \forall q \in Q, \quad \forall a \in \Sigma, \quad \|q'': \delta_p(q, a, q') > 0 \| \leq 1 \)

(ii) **Difference between PFA and DPFA**

Transitions in DFA

In a DPFA, a transition \((q, a, q', P)\) is totally defined by \( q \) and \( a \). For more clarity, we associate with a DPFA a non-probabilistic transition function defined below.
Definition 3
Let $A$ be a DPFA. $\delta_A : Q \times \Sigma \rightarrow Q$ is the transition function with $\delta_A(q, a) = q'$; $\delta_P(q, a, q') = 0$.

In practice, when the issue is to learn a PFA, we are given frequencies and not probabilities. By frequency, we mean the number of times an event occurs, and by relative frequency we mean the number of times an event occurs over the number of times the event could have occurred. This leads us to the use of a deterministic frequency finite automaton (DFFA).

2.2.4 Example of PFA
Figure 1 shows an example of PFA. There are 4 states: $q_1$, $q_2$, $q_3$, $q_4$. Arrows from one state to another are labeled by two elements: the first is the character of the input string and the second is the probability of taking this transition.

For each state, inside the node, is written in sequence from the left to right: the probability (eventually zero) to be an initial state, the name of the state and the probability (eventually zero) to be in the final state. When the probabilities are zero (whether related to initial or final states), they are dropped from the diagram.

Note that the sum of different probabilities of all outgoing edges and the probability of being final, for a given state is equal to 1. Sum of initial states’ probabilities is also equal to one.

3. P-GIFSA implemented algorithms
We concentrate on two probabilistic algorithms, namely ALERGIA and DSAI.

3.1 ALERGIA [5], [13]
The ALERGIA algorithm is a state merging algorithm that makes a predefined ordering of the states, tests for compatibility and then executes a merging operation. In order to follow the automaton inference, the RED-BLUE framework is used. By RED, we mean identified parts of the automaton while BLUE designates the unidentified ones. The remaining uncolored nodes illustrate the pieces that are parts of the augmented prefix tree acceptor (APTA). The ALERGIA steps are defined below.

**ALERGIA**

Input: sample $S$, $\alpha > 0$
Output: an FFA $A$

Compute $t_0$, threshold on the size of the multiset needed for the test to be statistically significant;

$A \leftarrow \text{FPTA}(S)$;

$\text{RED} \leftarrow \{q_\lambda\}$;

$\text{BLUE} \leftarrow \{q_a : a \in \Sigma \cap \text{PREF}(S)\}$;

while CHOOSE $q_b$ from BLUE such that $\text{FREQ}(q_b) \geq t_0$
do

if $\exists q_r \in \text{RED} : \text{ALERGIA-COMPATIBLE}(A, q_r, q_b, \alpha)$ then

$A \leftarrow \text{STOCHASTIC-MERGE}(A, q_r, q_b)$
else

$\text{RED} \leftarrow \text{RED} \cup \{q_b\}$
end

BLUE $\leftarrow \{q_{ua} : ua \in \text{PREF}(S) \land q_u \in \text{RED} \backslash \text{RED}\}$;

end

return $A$

ALERGIA algorithm is supported with ALERGIA-TEST described below.

**ALERGIA-TEST**

Input: an FFA $A$, $f_1, n_1, f_2, n_2$, $\alpha > 0$
Output: a Boolean result specifying whether the frequencies $f_1/n_1$ and $f_2/n_2$ are suitably near

$\gamma < \left| \frac{f_1}{n_1} - \frac{f_2}{n_2} \right|$

Return $\gamma < \left( \frac{1}{\sqrt{n_1}} + \frac{1}{\sqrt{n_2}} \right) \left( \frac{1}{2} \ln(\frac{2}{\alpha}) \right)$
ALERGIA algorithm is also supported with ALERGIA-COMPATIBLE as described below.

### ALERGIA-COMPATIBLE

**Input**: an FFA $A$, 2 states $q_u$, $q_v$, $\alpha > 0$  
**Output**: compatibility of $q_u$ and $q_v$?

Correct $\leftarrow$ true;  
if ALERGIA-TEST($FPA(q_u)$, $FREQ_A(q_u)$, $FPA(q_v)$, $FREQ_A(q_v)$, $\alpha$) then  
Correct $\leftarrow$ false;  
for $a \in \Sigma$ do  
if ALERGIA-TEST($\delta_{fr}(q_u, a)$, $FREQ_A(q_u)$, $\delta_{fr}(q_v, a)$, $FREQ_A(q_v)$, $\alpha$) then  
Correct $\leftarrow$ false  
end  
return Correct

### 3.2 DSAI [6], [13]

The second algorithm implemented in P-GIFSA is the DSAI algorithm. The basic idea behind this algorithm is the $\mu$-distinguishability which is based on the evaluation of a threshold ($\mu$) distinguishing between two states $q$ and $q'$. This boils down to finding a string $x$ such that:

$$\left| Pr_{A,q}(x) - Pr_{A,q'}(x) \right| = \mu \geq L_{\infty}(D_{A,q}, D_{A,q'}) \quad (1)$$

where $Pr_{A,q}(x)$ is the probability of generating $x$ by $A$ when taking $q$ as the initial state.  
Given a BLUE state $q_u$, visited a minimum number of times and RED state $q_v$, using Equation $1$, DSAI computes the value of $L_{\infty}(D_{A,q'}, D_{A,q_b})$. The condition of compatibility and merging is given by:

$$L_{\infty}(D_{A,q'}, D_{A,q_b}) < \mu/2 \quad (2)$$

In order to avoid an accumulation of small errors, a state that is not visited enough times is retained until the end. The main functions of the DSAI algorithm are DSAI-COMPATIBLE and STOCHASTIC-MERGE with a profound similarity in structure between ALERGIA and DSAI.

### 4. P-GIFSA design and test

On the design level, P-GIFSA is based on unified modeling language (UML) software, used as an engineering standard modeling method. Indeed, UML, as a modeling tool, plays an important role in the readability of the software, its continuous improvement and, above all, in its future easy maintenance and upgradability.

#### 3.1 UML Class Diagram

Figure 2, drawn at the end of the paper, shows the overall unified modeling language (UML) class diagram of P-GIFSA. It is self-explained. In the UML, a class diagram is a static structural representation of the system showing its classes, their attributes, the operations they can perform and the relationships between objects [14].

#### 3.2 P-GIFSA Test

As a first step, Eclipse IDE for Java Developers is used for the new project building, where the new classes are created. We show how a user can interact with P-GIFSA via the interface. The figures are self-explanatory.

Figure 4 shows the ALERGIA running on an example with results given in Figure 5. Testing whether an unknown string belongs to a language or not is described in Figure 6 (for positive example) and Figure 7 (for negative example).

---

3 https://www.eclipse.org/
5. Conclusion

We have developed P-GIFSA as a novel environment for grammatical inference as applied to FSA for the probabilistic case. We have described all the important methodological steps needed to design and implement a GI environment. In this respect, the proposed system offers off-the-shelf probabilistic GI methods for solving practical FSA problems in the same vein as GIFSA, its non-probabilistic counterpart. Specifically, P-GIFSA implements different and important probabilistic algorithms such as ALERGIA and DSAI. Although P-GIFSA is relatively simple, it is powerful enough to allow FSA probabilistic-based inference from a small number of examples. As it stands now, P-GIFSA offers helpful assistance to both experts and users alike in addressing any FSA probabilistic inference related issues. The proposed environment is ready for practical use to solve problems in many sequence-related domains such as bioinformatics and engineering, for instance. It is recommended to use P-GIFSA and develop it further by integrating additional relevant GI probabilistic algorithms to make it more general.

References


Websites accessed as of May 2014
- EMILE additional information available at: http://turing.wins.uva.nl/~pietera/ALS/

Figure 2 – P-GIFSA Class Diagram

DFA
- Σ: char
- Q: [ ] State
- I: State
- δ: [] Transition

+DFA()
+DFA(String[], String[])
+Build_BTDA(String[], String[])
+Calculate_alphabets(String[])
+Calculate_pREF(String[])
+Calculate_transit(String[], String[] , State[], String[])
+setchar (char)
....
+getchar0

DFFA
+ Build_FPTA (String[], String[])
+setstate(State)
....
+getstate()
....
+STOCHASTIC_MERGE()
+STOCHASTIC_FOLD()
....

State
- Name: String
- ft: int
initial:Boolean
final:boolean

+State()
....
+successor()
+predessessor()
....
+get_FT()
....
+setfrequency (int)
....

Transition
- IS: State
- FS: State
- car: char
- tfr:int

+Transition()
....
+setlS(State)
....
+getlS()
....

ALERGIA
-S: Sample
-α: int

+ALERGIA(String[], double)
+ALERGIA_COMPATIBLE()
+ALERGIA_TEST()

Sample
-S: [ ] String

+GetSize()
+Print()
....

DSAI
-S: Sample
-μ:int

+DSAI(String[], double)
+DSAI_COMPATIBLE()