Encyclopedia of Artificial Intelligence

Juan Ramón Rabuñal Dopico
University of A Coruña, Spain

Julián Dorado de la Calle
University of A Coruña, Spain

Alejandro Pazos Sierra
University of A Coruña, Spain

Information Science REFERENCE

INFORMATION SCI
Hershey · New York

Director of Editorial Content: Kristin Klinger
Managing Development Editor: Kristin Roth
Development Editorial Assistant: Julia Mosemann, Rebecca Beistline
Senior Managing Editor: Jennifer Neidig
Managing Editor: Jamie Snavely
Assistant Managing Editor: Carole Coulson
Typesetter: Jennifer Neidig, Amanda Appicello, Cindy Consonery
Cover Design: Lisa Tosheff
Printed at: Yurchak Printing Inc.

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue, Suite 200
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com/reference

and in the United Kingdom by
Information Science Reference (an imprint of IGI Global)
3 Henrietta Street
Covent Garden
London WC2E 8LU
Tel: 44 20 7240 0856
Fax: 44 20 7379 0609
Web site: http://www.eurospanbookstore.com

Copyright © 2009 by IGI Global. All rights reserved. No part of this publication may be reproduced, stored or distributed in any form or by any means, electronic or mechanical, including photocopying, without written permission from the publisher.

Product or company names used in this set are for identification purposes only. Inclusion of the names of the products or companies does not indicate a claim of ownership by IGI Global of the trademark or registered trademark.

Library of Congress Cataloging-in-Publication Data

Encyclopedia of artificial intelligence / Juan Ramon Rabunal Dopico, Julian Dorado de la Calle, and Alejandro Pazos Sierra, editors.
p. cm.
Includes bibliographical references and index.
Summary: "This book is a comprehensive and in-depth reference to the most recent developments in the field covering theoretical developments, techniques, technologies, among others"--Provided by publisher.
ISBN 978-1-59904-849-9 (hardcover) -- ISBN 978-1-59904-850-5 (ebook)
1. Artificial intelligence--Encyclopedias. I. Rabunal, Juan Ramon, 1973- II. Dorado, Julian, 1970- III. Pazos Sierra, Alejandro.
Q334.2.E63 2008
006.303--dc22
2008027245

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

All work contributed to this encyclopedia set is new, previously-unpublished material. The views expressed in this encyclopedia set are those of the authors, but not necessarily of the publisher.

If a library purchased a print copy of this publication, please go to http://www.igi-global.com/agreement for information on activating the library's complimentary electronic access to this publication.

Evolutionary Grammatical Inference

Ernesto Rodrigues
Federal University of Technology, Brazil

Heitor Silvério Lopes
Federal University of Technology, Brazil

INTRODUCTION

Grammatical Inference (also known as grammar induction) is the problem of learning a grammar for a language from a set of examples. In a broad sense, some data is presented to the learner that should return a grammar capable of explaining to some extent the input data. The grammar inferred from data can then be used to classify unseen data or provide some suitable model for it.

The classical formalization of **Grammatical Inference** (GI) is known as Language Identification in the Limit (Gold, 1967). Here, there are a finite set S_+ of strings known to belong to the language L (the positive examples) and another finite set S_- of strings not belonging to L (the negative examples). The language L is said to be identifiable in the limit if there exists a procedure to find a grammar G such that $S_+ \subseteq L(G)$, $S_- \not\subset L(G)$ and, in the limit, for sufficiently large S_+ and S_-, $L = L(G)$. The disjoint sets S_+ and S_- are given to provide clues for the inference of the production rules P of the unknown grammar G used to generate the language L.

Grammatical inference include such diverse fields as speech and natural language processing, gene analysis, pattern recognition, image processing, sequence prediction, information retrieval, cryptography, and many more. An excellent source for a state-of-the art overview of the subject is provided in (de la Higuera, 2005).

Traditionally, most work in GI has been focused on the inference of regular grammars trying to induce finite-state automata, which can be efficiently learned. For context free languages some recent approaches have shown limited success (Starckie, Costie & Zaanen, 2004), because the search space of possible grammars is infinite. Basically, the parenthesis and palindrome languages are common test cases for the effectiveness of grammatical inference methods. Both languages are context-free. The parenthesis language is deterministic but the palindrome language is nondeterministic (de la Higuera, 2005).

The use of evolutionary methods for context-free grammatical inference are not new, but only a few attempts have been successful.

Wyard (1991) used Genetic Algorithm (GA) to infer grammars for the language of correctly balanced and nested parentheses with success, but fails on the language of sentences containing the same number of *a*'s and *b*'s ($a^n b^n$ language). In another attempt (Wyard, 1994), he obtained positive results on the inference of two classes of context-free grammars: the class of *n*-symbol palindromes with $2 \leq n \leq 4$ and a class of small natural language grammars.

Sen and Janakiraman (1992) applied a GA using a pushdown automata to the inference and successfully learned the $a^n b^n$ language and the parentheses balancing problem. But their approach does not scale well.

Huijsen (1994) applied GA to infer context-free grammars for the parentheses balancing problem, the language of equal numbers of *a*'s and *b*'s and the even-length 2-symbol palindromes. Huijsen uses a "markerbased" encoding scheme with has the main advantage of allowing variable length chromosomes. The inference of regular grammars was successful but the inference of context-free grammars failed.

Those results obtained in earlier attempts using GA to context-free grammatical inference were limited. The first attempt to use Genetic Programming (GP) for grammatical inference used a pushdown automata (Dunay, 1994) and successfully learned the parenthesis language, but failed for the $a^n b^n$ language.

Korkmaz and Ucoluk (2001) also presented a GP approach using a prototype theory, which provides a way to recognize similarity between the grammars in the population. With this representation, it is possible to recognize the so-called building blocks but the results are preliminary.

Copyright © 2009, IGI Global, distributing in print or electronic forms without written permission of IGI Global is prohibited.

Javed and his colleagues (2004) proposed a Genetic Programming (GP) approach with grammar-specific heuristic operators with non-random construction of the initial grammar population. Their approach succeeded in inducing small context-free grammars.

More recently, Rodrigues and Lopes (2006) proposed a hybrid GP approach that uses a confusion matrix to compute the fitness. They also proposed a local search mechanism that uses information obtained from the sentence parsing to generate a set of useful productions. The system was used for the parenthesis and palindromes languages with success.

BACKGROUND

A formal language is usually defined as follows. Given a finite alphabet $\sum$ of symbols, we define the set of all strings (including the empty string ε) over $\sum$ as $\sum^*$. Thus, we want to learn a language $L \subset \sum^*$. The alphabet $\sum$ could be a set of characters or a set of words. The most common way to define a language is based on grammars which gives rules for combining symbols and to produce the all sentences of a language.

A grammar is defined by a quadruple $G = (N, \sum, P, S)$, where N is an alphabet of nonterminal symbols, $\sum$ is an alphabet of terminal symbols such that $N \cap \sum = \phi$, P is a finite set of production rules of the form $\alpha \rightarrow \beta$ for $\alpha, \beta \in (N \cup \sum)^*$ where * represents the set of symbols that can be formed by taking any number of them, possibly with repetitions. S is a special nonterminal symbol called the start symbol.

The language $L(G)$ produced from grammar G is the set of all strings consisting only of terminal symbols that can be derived from the start symbol S by the application of production rules. The process of deriving strings by applying productions requires the definition of a new relation symbol $\Rightarrow$. Let $\alpha X \beta$ be a string of terminals and nonterminals, where X is a nonterminal. That is, α and β are strings in $(N \cup \Sigma)^*$, and $X \in N$. If $X \rightarrow \varphi$ is a production of G, we can say $\alpha X \beta \Rightarrow \alpha \varphi \beta$. It is important to say that one derivation step can replace any nonterminal anywhere in the string. We may extend the $\Rightarrow$ relationship to represent one or many derivation steps. We use a * to denote more steps. Therefore, we formally define the language $L(G)$ produced from grammar G as $L(G) = \{ w \mid w \in \sum^*, S \Rightarrow^* w \}$.

More details about formal languages and grammars can be found in textbooks such as Hopcroft et al (2001).

The Chomsky Hierarchy

Grammars are classified according to the form of the production rules used. They are commonly grouped into a hierarchy of four classes, known as the **Chomsky hierarchy** (Chomsky, 1957).

- *Recursively enumerable languages:* a grammar is unrestricted, and its productions may replace any number of grammar symbols by any other number of grammar symbols. The productions are of the form $\alpha \rightarrow \beta$ with $\alpha, \beta \in (N \cup \sum)^*$.
- *Context-sensitive languages:* they have grammars with productions that replace a single nonterminal by a string of symbols, whenever the nonterminal occurs in a specific *context*, i.e., has certain left and right neighbors. These productions are of the form $\alpha A \gamma \rightarrow \alpha \beta \gamma$, with $A \in N$ and $\alpha, \beta, \gamma \in (N \cup \sum)^*$. A is replaced by β if it occurs between α and γ .
- *Context-free languages:* in this type, grammars have productions that replace a single nonterminal by a string of symbols, regardless of this nonterminal's context. The productions are of the form $A \rightarrow \alpha$ for $A \in N$ and $\alpha \in (N \cup \sum)^*$; thus A has no context.
- *Regular languages:* they have grammars in which a production may only replace a single nonterminal by another nonterminal and a terminal. The productions are of the form $A \rightarrow B\alpha$ or $A \rightarrow \alpha B$ for $A, B \in N$ and $\alpha \in \sum^*$.

It is sometimes useful to write a grammar in a particular form. The most commonly used in grammatical inference is the Chomsky Normal Form. A CFG G is in Chomsky Normal Form (CNF) if all production rules are of the form $A \rightarrow BC$ or $A \rightarrow \alpha$ for $A, B, C \in N$ and $\alpha \in \sum$.

The Cocke-Younger-Kasami Algorithm

To determine whether a string can be generated by a given context-free grammar in CNF, the Cocke-Younger-Kasami (CYK) algorithm can be used. This

algorithm is efficient and it has complexity $O(n^3)$ where n is the sentence length .

In the CYK algorithm, first a triangular table that tells whether the string w is in $L(G)$ is constructed. The horizontal line corresponds to the positions of the string $w = a_1 a_2 .. a_n$. The table entry V_{rs} is the set of variables A $\in P$ such that $A \Rightarrow^* a_r a_{r+1} ... a_s$. We are interested in whether the start symbol S is in the set V_{1n} because that is the same as saying $S \Rightarrow^* a_1 a_2 ... a_n$ or $S \Rightarrow^* w$, i. e., $w \in L(G)$.

To fill the table, we work row-by-row upwards. Each row corresponds to one length of substrings; the bottom row is for strings of length 1, the second-from-bottom row for strings of length 2 and so on, until the top row corresponds to the one substring of length n which is w itself. The pseudocode is in Figure 1.

Genetic Programming

Genetic Programming (GP) is an evolutionary technique used to search over a huge state space of structured representations (computer programs). Each program represents a possible solution written in some language. The GP algorithm can be summarized in Figure 2 (Koza, 1992).

The evaluation of a solution is accomplished by using a set of training examples known as fitness cases which, in turn, is composed by sets of input and output data. Usually, the fitness is a measure of the deviation between the expected output for each input and the computed value given by GP (Banzhaf, Nordin, Keller & Francone, 2001).

Figure 1. The CYK algorithm

```
For r = 1 to n do
  V_r1 = { A | A → a_r ∈ P }

For s = 2 to n do
  For r = 1 to n-s+1 do
    V_rs = ∅
    For k = 1 to s-1 do
      V_rs = V_rs ∪ { A | A → BC ∈ P, B ∈ V_rk and C ∈ V_r+k, s-k
```

Figure 2. The GP algorithm

```
Generate Initial Population Randomly
While not (Stopping Condition)
  begin
     Evaluate the fitness of each individual
     Select individuals according to their fitness
     Modify them by applying genetic operators
  end
Returns the best individual found
```

There are two main selection methods used in GP: fitness proportionate and tournament selection. In the fitness proportionate selection, programs are selected randomly with probability proportional to its fitness. In the tournament selection, a fixed number of programs are taken randomly from the population and the one with the best fitness in this group is chosen. In this work, we use the tournament selection.

Reproduction is a genetic operator that simply copies a program to the next generation. Crossover, on the other hand, combines parts of two individuals to create two new ones. Mutation changes randomly a small part of an individual.

Each run of the main loop of GP creates a new generation of computer programs that substitutes the previous one. The evolution is stopped when a satisfactory solution is achieved or a predefined maximum number of generations is reached.

A GRAMMAR GENETIC PROGRAMMING APPROACH

We present how a GP approach can be applied to the inference of context-free grammars. First, we discuss the representation of the grammars. The modification needed in the genetic operators are also presented. In the last section, the grammar evaluation are discussed.

Initial Population

It is possible to represent a CFG as a list of structured trees. Each tree represents a production with its left-hand side as a root and the derivations as leaves. Figure 3 shows the grammar $G = (N, \Sigma, P, S)$ with $\Sigma = \{a, b\}$, $N = \{S, A\}$ and P = $\{S \rightarrow AS$; $S \rightarrow b$; $A \rightarrow SA$; $A \rightarrow a \}$.

The initial population can be created with random productions, provided that all the productions are reachable direct or indirectly starting with S.

Genetic Operators

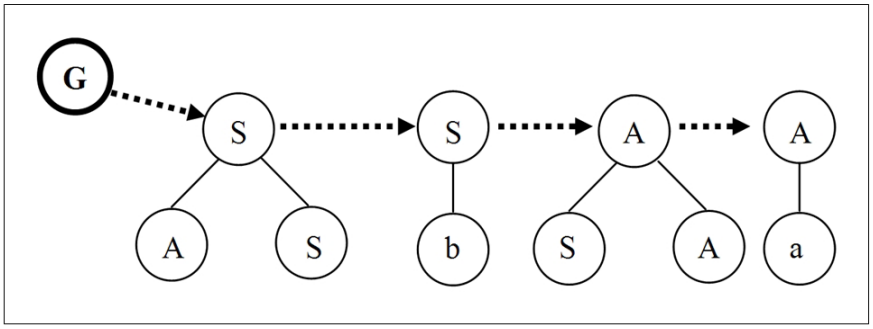
The crossover operator is applied over a pair of grammars and works as follows. First, a production is chosen using a tournament selection. If the second grammar has no production with the same left-hand side of the production chosen, crossover is rejected. Otherwise, the productions are swapped.

The mutation operation is applied to a single selected grammar. A production is then chosen using the same mechanism of crossover. A new production, with the same left-hand side and with a randomly right-side, replaces the production chosen.

The crossover probability is usually high (≈90%) and the mutation probability is usually low (≈10%).

Unfortunately, using only the genetic operators mentioned, the convergence of the algorithm is not guaranteed. In our recently work, we demonstrated that the use of two local search operators is needed: an incremental learning operator (Rodrigues & Lopes, 2006) and an expansion operator (Rodrigues & Lopes, 2007). The first uses the information obtained from a CYK table to discover which production is missing to cover the sentence. The latter can expand the set of productions dynamically providing diversity.

Figure 3. An example of a CFG represented as a list of structured trees



The Incremental Learning Operator

This operator is applied before the evaluation of each grammar in the population. It uses the CYK table obtained from the parsing of positive examples to allow the creation of an useful new production. The pseudocode is in Figure 4.

Once this process is completed with success, hopefully, there will be a set of positive examples (possible all) recognized by the grammar. Although, there is no warranty that some negative examples will still remain being rejected by the grammar.

The Expansion Operator

This operator adds a new nonterminal to the grammar and generates a new production with this new nonterminal as a left-side. This new approach allows grammars to grow dynamically in size. To avoid a new useless production, a production with another non-terminal in the left-side and the new non-terminal in the right-side is generated. It is important to emphasize that the new operator adds two productions to the grammar.

This operator promotes diversity in the population that is required in the beginning of the evolutionary process.

Grammar Evaluation

In grammatical inference, we need to train the system with both positive and negative examples to avoid overgeneralization. Usually the evaluation is done counting the positive examples covered by a grammar in proportion to the total of positive examples. If the grammar cover some negative examples, it is penalized in some way.

In our recently work, we use a **confusion matrix** that is typically used in supervised learning (Witten & Frank, 2005). Each column of the matrix represents the number of instances predicted either positively or negatively, while each row represents real classification of the instances. The entries in the confusion matrix have the following meaning in the context of our study:

- TP is the number of positive instances recognized by the grammar.
- TN is the number of negative instances rejected by the grammar.
- FP is the number of negative instances recognized by the grammar.
- FN is the number of positive instances rejected by the grammar.

Figure 4. The incremental learning operator pseudocode

```
For each positive example
  Construct the CYK table with V_rs
  If the example is not recognized
     If V_1n is not empty
        Then clone the root production changing
             the left-hand side by S.
     Else
        If V_2,n-1 is not empty
           Then add production S → V_11 V_2,n-1
        Else
          For s = n-1 to n/2 do
          begin
            If V_1s is not empty
               Then If V_s+1, n-s is not empty
                        Then add production S → V_1s V_s+1,n-s
          end
```

There are a several measures that can be obtained from the confusion matrix. The most common is total accuracy that is obtained from the total of correct classified examples divided by the total number of instances. In this paper we used two other measures: specificity (Equation 1) and sensitivity (Equation 2). These measures evaluate how positive and negative examples are correctly recognized by the classifier.

$$\text{specificity} = \frac{TN}{TN + FP} \quad (1)$$

$$\text{sensitivity} = \frac{TP}{TP + FN} \quad (2)$$

The fitness is computed by the product of these measures leading to a balanced heuristic. This fitness measure was proposed by (Lopes, Coutinho & Lima, 1998) and widely used in many classification problems.

The use of confusion matrix provides a better evaluation of the grammars in the population, because grammars with the same accuracy rate usually has different values for specificity and sensitivity.

FUTURE TRENDS

The GP approach for the grammatical inference is based on the CYK algorithm and the confusion matrix. The preliminary results are promising but there are two problems that must be addressed.

The first is the that the solution found is not necessarily the smallest one. Depending on the run, the grammar inferred varies in size and, sometimes, it can be difficult to understand and may have useless or redundant production rules. Further work will focus on devising a mechanism able to favor shorter partial solutions.

The second is called "bloat", the uncontrolled growth of the size of an individual in the population (Monsieurs & Flerackers, 2001). The use of an expansion operator may cause this undesirable behavior. Nevertheless, this behavior was not detected in the experiments because all useless productions are eliminated during the search.

CONCLUSION

This article proposes a GP approach for context-free grammar inference. In this approach, an individual is a list of structured trees representing their productions with their left-hand side as the root and the derivations as leaves. It uses a local search operator, named Incremental Learning, capable of adjusting each grammar according to the positive examples. It also uses an expansion operator which adds a new production to the grammar allowing the grammars to grow in size. This operator promotes diversity in the population that is required in the earlier generations.

The use of a local search mechanism that is capable of learning from examples promotes a fast convergence. The preliminary results demonstrated that the approach is promising.

REFERENCES

Banzhaf, W., Nordin, P., Keller, R.E. & Francone, F.D. (2001) Genetic Programming: an introduction. San Francisco: Morgan Kaufmann.

Chomsky, N. (1957) Syntactic Structures. Paris: Mouton.

de la Higuera, C. (2005) A bibliographical study of grammatical inference. Pattern Recognition. 38(9), 1332-1348.

Dunay, B.D. (1994) Context free language induction with genetic programming. Sixth International Conference on Tools with Artificial Intelligence (ICTAI '94), IEEE Computer Society, 828-831.

Gold, E. M. (1967). Language identification in the limit. Information and Control. 10(5), 447-474.

Hopcroft, J. E., Motwani, R. & Ullman, J. D. (2001) Introduction to Automata Theory, Languages, and Computation. Addison-Wesley.

Huijsen, W. O. (1994) Genetic grammatical inference. CLIN IV. Papers from the Fourth CLIN Meeting, 59–72.

Javed, F. Bryant, B., Crepinsek, M., Mernik, M. & Sprague, A. (2004) Context-free grammar induction

using genetic programming. Proceedings of the 42nd. Annual ACM Southeast Conference '04, 404-405.

Korkmaz, E.E. & Ucoluk, G. (2001) Genetic programming for grammar induction. Proceedings of 2001 Genetic and Evolutionary Computation Conference Late Breaking Papers, 245-251.

Koza, J.R. (1992) Genetic Programming: On the Programming of Computers by Natural Selection. Cambridge: MIT Press.

Lopes, H.S., Coutinho, M.S. & Lima, W.C. (1998) An evolutionary approach to simulate cognitive feedback learning in medical domain. Proceedings of the Genetic Algorithms and Fuzzy Logic Systems: Soft Computing Perspectives, 193-207.

Monsieurs, P. & Flerackers, E. (2001) Reducing bloat in genetic programming. Proceedings of the International Conference, 7th Fuzzy Days on Computational Intelligence, Theory and Application, LNCS v. 2206, Springer-Verlaq, 471-478.

Rodrigues, E. & Lopes, H.S. (2007) Genetic Programming for induction of context-free grammars. Proceedings of the 7th. International Conference on Intelligent Systems Design and Applications (ISDA'07) (*to be published*).

Rodrigues, E. & Lopes, H.S. (2006) Genetic programming with incremental learning for grammatical inference. Proceedings of the 6th. International Conference on Hybrid Intelligent Systems (HIS'06), IEEE Press, Auckland, 47.

Sen, S. & Janakiraman, J. (1992) Learning to construct pushdown automata for accepting deterministic context-free languages. Proceedings of the Applications of Artificial Intelligence X: Knowledge-Based Systems, SPIE v. 1707, 207–213.

Starckie, B,. Costie, F. & van Zaanen, M. (2004) The Omphalos context-free grammar learning competition. Proceedings of the International Colloquium on Grammatical Inference, 16-27.

Witten, I. H. & Frank, E. (2005) Data Mining. San Francisco: Morgan Kaufmann.

Wyard, P. (1994) Representational issues for context free grammar induction using genetic algorithms. Proceedings of the Second International Colloquium in Grammatical Inference (ICGI-94). LNAI n 862. Springer-Verlag, 222–235.

Wyard, P. (1991) Context free grammar induction using genetic algorithms. Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA'91). Morgan Kaufmann, 514-518.

KEY TERMS

CYK: A Cocke-Younger-Kasami algorithm used to determine whether the sentence can be generated by the grammar.

Evolutionary Computation: Large and diverse class of population-based search algorithms that is inspired by the process of biological evolution through selection, mutation and recombination. They are iterative algorithms that start with an initial population of candidate solutions and then repeatedly apply a series of the genetic operators.

Finite Automata: A model of behavior composed of a finite number of states, transitions between those states, and actions. They are used to recognize regular languages.

Genetic Algorithm: A type of evolutionary computation algorithm in which candidate solutions are represented typically by vectors of integers or bit strings, that is, by vectors of binary values 0 and 1.

Heuristic: Function used for making certain decisions within an algorithm; in the context of search algorithms, typically used for guiding the search process.

Local Search: A type of search method that starts at some point in search space and iteratively moves from position to neighbouring position using heuristics.

Pushdown Automata: A finite automaton that can make use of a stack containing data. They are used to recognize context-free language.

Search Space: Set of all candidate solutions of a given problem instance.