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

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.

Grammar-Guided Genetic Programming

Daniel Manrique
Inteligencia Artificial, Facultad de Informatica, UPM, Spain

Juan Ríos
Inteligencia Artificial, Facultad de Informatica, UPM, Spain

Alfonso Rodríguez-Patón
Inteligencia Artificial, Facultad de Informatica, UPM, Spain

INTRODUCTION

Evolutionary computation (EC) is the study of computational systems that borrow ideas from and are inspired by natural evolution and adaptation (Yao & Xu, 2006, pp. 1-18). EC covers a number of techniques based on evolutionary processes and natural selection: evolutionary strategies, genetic algorithms and genetic programming (Keedwell & Narayanan, 2005).

Evolutionary strategies are an approach for efficiently solving certain continuous problems, yielding good results for some parametric problems in real domains. Compared with genetic algorithms, evolutionary strategies run more exploratory searches and are a good option when applied to relatively unknown parametric problems.

Genetic algorithms emulate the evolutionary process that takes place in nature. Individuals compete for survival by adapting as best they can to the environmental conditions. Crossovers between individuals, mutations and deaths are all part of this process of adaptation. By substituting the natural environment for the problem to be solved, we get a computationally cheap method that is capable of dealing with any problem, provided we know how to determine individuals' fitness (Manrique, 2001).

Genetic programming is an extension of genetic algorithms (Couchet, Manrique, Ríos & Rodríguez-Patón, 2006). Its aim is to build computer programs that are not expressly designed and programmed by a human being. It can be said to be an optimization technique whose search space is composed of all possible computer programs for solving a particular problem. Genetic programming's key advantage over genetic algorithms is that it can handle individuals (computer programs) of different lengths.

Grammar-guided genetic programming (GGGP) is an extension of traditional GP systems (Whigham, 1995, pp. 33-41). The difference lies in the fact that they employ context-free grammars (CFG) that generate all the possible solutions to a given problem as sentences, establishing this way the formal definition of the syntactic problem constraints, and use the derivation trees for each sentence to encode these solutions (Dounias, Tsakonas, Jantzen, Axer, Bjerregard & von Keyserlingk, D. 2002, pp. 494-500). The use of this type of syntactic formalisms helps to solve the so-called closure problem (Whigham, 1996). To achieve closure valid individuals (points that belong to the search space) should always be generated. As the generation of invalid individuals slows down convergence speed a great deal, solving this problem will very much improve the GP search capability. The basic operator directly affecting the closure problem is crossover: crossing two (or any) valid individuals should generate a valid offspring. Similarly, this is the operator that has the biggest impact on the process of convergence towards the optimum solution. Therefore, this article reviews the most important crossover operators employed in GP and GGGP, highlighting the weaknesses existing nowadays in this area of research. We also propose a GGGP system. This system incorporates the original idea of employing ambiguous CFG to overcome these weaknesses, thereby increasing convergence speed and reducing the likelihood of trapping in local optima. Comparative results are shown to empirically corroborate our claims.

BACKGROUND

Koza defined one of the first major crossover operators (KX) (1992). This approach randomly swaps subtrees in both parents to generate offspring. Therefore, it tends to disaggregate the so-called building blocks across the trees (that represent the individuals). The building blocks are those subtrees that improve fitness. This over-expansion has a negative effect on the fitness of the individuals. Also, this operator's excessive exploration capability leads to another weakness: an increase in the size of individuals, which affects system performance, and results in a lower convergence speed (Terrio & Heywood, 2002). This effect is known as bloat or code bloat.

There is another important drawback: many of the generated offspring are syntactically invalid as the crossovers are done completely at random. These individuals should not be part of the new population because they do not provide a valid solution. This seriously undermines the convergence process. Figure 1 shows a situation where one of the two individuals generated after Koza's crossover breaches the constraints established by a hypothetical grammar whose sentences represent arithmetic equalities.

The strong context preservative crossover operator (SCPC) avoids the problem of desegregation of building blocks (also called context) across the trees by setting severe (strong) constraints for tree nodes considered as possible candidates for selection as crossover nodes (D'haesler, 1994, pp. 379-407). A system of coordinates is defined to univocally identify each node in a derivation tree. The position of each node within the tree is specified along the path that must be followed to reach a given node from the root. To do this, the position of a node is described by means of a tuple of *n* coordinates $T = (b_1, b_2, \ldots b_n)$, where n is the node's depth in the tree, and b_i indicates which branch is selected at depth i (counting from left to right). Figure 2 shows an example representing this system of coordinates.

Only nodes with the same coordinates from both parents can be swapped. For this reason, a subtree may possibly never migrate to another place in the tree. This limitation can cause serious search space exploration problems, as the whole search space cannot be covered unless each function and terminal appears at every possible coordinate at least once in any one individual in the population. This failure to migrate building blocks causes them to evolve separately in each region, causing a too big an exploitation capability, thereby increasing the likelihood of trapping in local optima (Barrios, Carrascal, Manrique & Ríos, 2003, pp. 275-293).

As time moves on, the code bloat phenomenon becomes a serious problem and takes an ever more prominent role. To avoid this, Crawford-Marks &

Figure 1. Incorrect operation of Koza's crossover operator

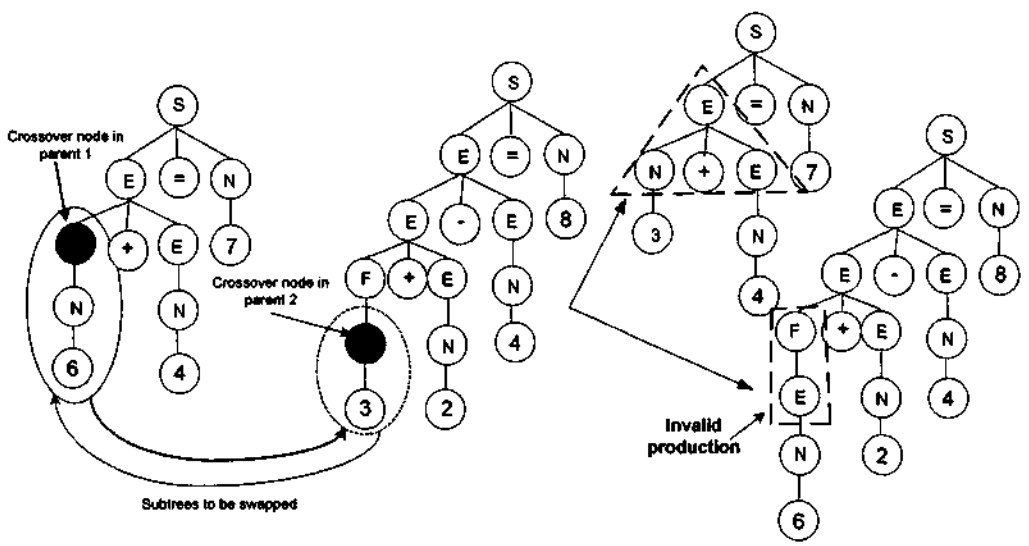
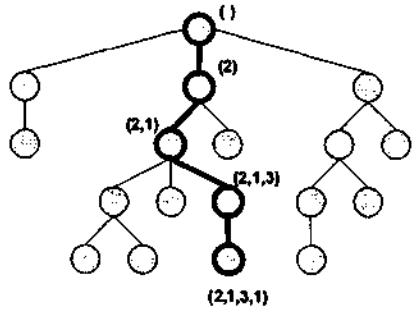


Figure 2. The system of coordinates defined in SCPC



Spector (2002) developed the Fair crossover (pp. 733-739). This is a modified version of the approach proposed by Langdon (1999, pp. 1092-1097). Tree size is controlled as follows. First, a crossover node in the first parent is selected at random and the length, l, of the subtree extending from the node to the leaves is calculated. Then, a node is also selected at random in the second parent, and the length, l_2, for this second subtree is calculated. If l_2 is within the range $[l - l/4, l + l/4]$, then the crossover node for the second parent is accepted, and the two subtrees are swapped. If not, another crossover node is selected at random for the second parent and the check is run again. This way, the size of the subtree in the second parent to be swapped is controlled and limited, so the code bloat phenomenon is avoided. Another aspect to comment here is that the range in which l_2 must be included can be modified to afford specific problems more efficiently, but the range originally proposed works fine for most of them.

Whigham proposed one of the most commonly used operators (WX) in GGGP (1995, pp. 33-41). Because of its sound performance in such systems, it has become the de facto standard and is still in use today (Rodrigues & Pozo, 2002, pp. 324-333), (Hussain, 2003), (Grosman & Lewin, 2004, pp. 2779-2790). The algorithm works as follows. First, as all the terminal symbols have at least one non-terminal symbol above them, then, without loss of generality, the crossover nodes can be confined exclusively to locations on nodes containing non-terminal symbols. A non-terminal node belonging to the first parent is selected at random. Then a non-terminal node labeled with the same non-terminal symbol as in the first-chosen crossover node is selected from the second parent. This assures that generated individuals belong to the grammar-generated language, as the crossed nodes share the same symbol. This operator's main flaw is that there are other possible choices of node in the second parent that are not explored and that could end in the target solution (Manrique, Marquez, Ríos & Rodríquez-Patón, 2005, pp. 252-261).

THE PROPOSED CROSSOVER OPERATOR FOR GGGP SYSTEMS

The proposed operator is a general-purpose operator designed to work in any GGGP system. It takes advantage of the key feature that defines a CFG as ambiguous: the same sentence can be obtained by several derivation trees. This implies that there are several individuals representing the solution to a problem. It is therefore easier to find. This operator consists of eight steps:

1. Choose a node, except the axiom, with a non-terminal symbol randomly from the first parent. This node is called crossover node and is denoted CN1.
2. Choose the parent of CN1. As we are working with a CFG, this will be a non-terminal symbol. The right-hand sides of all its production rules are stored in the array R.
3. The derivation produced by the parent of CN1 is called main derivation, and is denoted A ::= C. Calculate the derivation length l as the number of symbols in the right-hand side of the main derivation. Having l, the position (p) of CN1 in the main derivation and C, define the three-tuple T(l, p, C).
4. Delete from R all the right-hand sides with different lengths from the main derivation.
5. Remove from R all those right-hand sides in which there exists any difference between the symbols (except the one located in position p) in each right-hand side and the symbols in C.
6. The set X is formed by all the symbols in the right-hand sides of R that are in position p. X contains all the non-terminal symbols of the second parent that can be chosen as a crossover node (CN2).
7. Choose CN2 randomly from X, discarding all the nodes that will generate offspring trees with a size greater than a previously established value D.

8. Calculate the two new derivation trees produced as offspring by swapping the two subtrees whose roots are CN1 and CN2.

The underlying idea of this algorithm consists on calculating which are the non-terminal symbols that can substitute the symbol contained in CN1, bearing in mind that the production rule that contains CN1 keeps being valid. Since all non-terminal symbols that can generate valid production rules are taken into account in the crossover process, this operator takes advantage of ambiguous grammars.

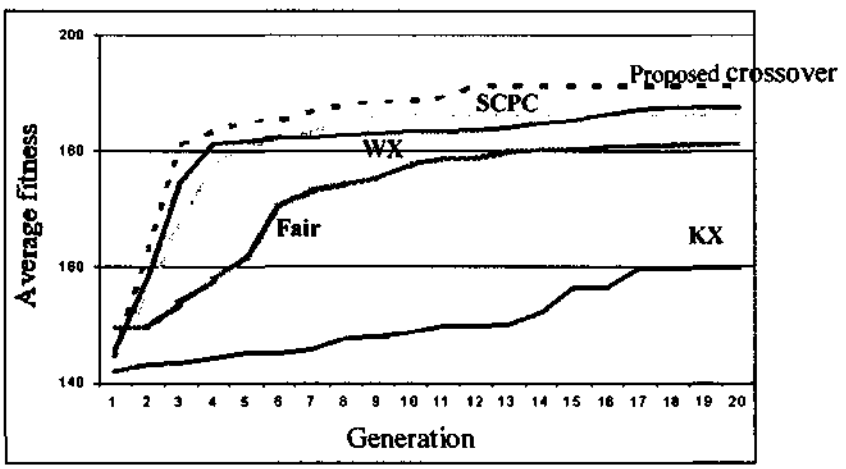
The proposed crossover operator has primarily three attractive features: a) step 7 states a code bloat control mechanism, b) the offspring produced are always composed of two valid trees and c) step 6 indicates that all the possible nodes of the second parent that can generate valid individuals are taken into account, not only those nodes with the same non-terminal symbol as the one chosen for the first parent. This third feature increases the GGGP system's exploration capability, which avoids trapping in local optima and takes advantage of there being more than one derivation tree (potential solution to the problem) for a single sentence.

Results

We present and discuss the results achieved by the crossover operators described in the background section and the operator that we propose. To do so, we have tackled a complex classification problem: the real-world task of providing breast cancer prognosis (benign or malignant) from the morphological characteristics of microcalcifications. Microcalcifications are small mineral deposits in breast tissue that could constitute cancer. This experiment involved searching a knowledge base of fuzzy rules that could give such a prognosis.

The data employed for giving a disease prognosis are: patient's age, lesion size, lesion location in the breast, and particular features of the microcalcifications: number, distribution and type. Number indicates the quantity of existing clustered microcalifications, distribution shows how they are clustered and type reflects the individual morphology of the microcalcifications. To run the tests, 365 microcalcifications were selected at random. Of these, 315 lesions were randomly selected for use as genetic programming system training cases with the different crossover operators described. After training, the fittest individual was selected to form a knowledge base with the fuzzy rules encoded by this individual. Then, the knowledge base was tested with

Figure 3. Average convergence speed for each crossover operator



the 50 remaining lesions not chosen during the training phase to output the number of correctly classified patterns in what we have called the testing phase.

The CFG employed was formed by 19 non-terminal symbols, 54 terminals and 51 production rules, some of them included to obtain an ambiguous grammar. The population size employed was 1000, the upper bound for the size of the derivation trees was set to 20. The fitnesss function consisted of calculating the number of well-classified patterns. Therefore, the greater the fitness, the fitter the individual is, with the maximum limit of 315 in the training phase and 50 in the test.

Figure 3 shows the average evolution process for each of the five crossover operators in the training phase after 100 executions.

It is clear from Figure 3 that KX yields the worst results, because it maintains an over-diverse population and allows invalid individuals to be generated. This prevents it from focusing on one possible solution. The effect of Fair is just the opposite, leading very quickly to one of the optimal solutions (this is why it has a relatively high convergence speed initially), and slowing down if convergence is towards a local optimum (which happens in most cases). WX and SCPC produce good results, bettered only by the proposed crossover. Its high convergence speed evidences the benefits of taking into account all possible nodes of the second parent that can generate valid offspring.

Table 1 shows examples of fuzzy rules output in one of the executions for the best two crossover operators —WX and the proposed operator— once the training phase was complete.

Table 2 shows the average number (rounded up or down to the nearest integer) of correctly classified patterns after 100 executions, achieved by the best individual in the training and test phases, and the percentage of times that the system converged prematurely.

KX again yields the worst results, correctly classifying just 57.46% (181/315) of patterns in the training phase and 54% (27/50) in the testing phase. SCPC and Fair crossovers also return insufficient results: around 59% in the training phase and 54%-56% in the testing phase, although, as shown in Figure 3, SCPC has a higher convergence speed. Finally, note the similarity

Table 1. Some knowledge base fuzzy rules output by two GGGP systems

Crossover operator	Rule 1	Rule 2
WX	IF NOT (type=branched) OR (number=few) THEN (prognosis=benign)	
Proposed	IF NOT (age=middle) AND NOT (location=subaerolar) AND NOT(type=oval) THEN (prognosis=malignant)	IF (type=heterogeneous) THEN (prognosis=malignant)

Table 2. Average number of correctly classified patterns and unsuccessful runs

Crossover operator	Training	Testing	Unsuccessful runs
KX	181/315 (57.46%)	27/50 (54%)	36%
SCPC	186/315 (59.04%)	28/50 (56%)	14%
Fair	185/315 (58.73%)	27/50 (54%)	15%
WX	191/315(60.63%)	30/50 (60%)	8%
Proposed	191/315(60.63%)	31/50 (62%)	2%

between WX and the proposed operator. However, the proposed operator has higher speed of convergence and is less likely to get trapped in local optima, as it converged prematurely only twice in 100 executions.

FUTURE TRENDS

The continuation of the work described in this article can be divided into two main lines of investigation in GGGP. The first involves finding an algorithm that can estimate the maximum sizes of the trees generated throughout the evolution process to assure that the optimal solution will be reached. This would overcome the proposed crossover operator's weakness of not being able to reach a solution because the permitted maximum tree size is too restrictive for it to be able to reach a good solution, whereas this solution could be found if individuals were just a little larger.

The second interesting line of research derived from this work is the use of ambiguous grammars. It has been empirically observed that using the proposed operator combined with ambiguous grammars in GGGP systems benefits convergence speed. However, "too much ambiguity" is damaging. The idea is to get an ambiguity measure that can answer the question of how much ambiguity is needed to get the best results in terms of efficiency.

CONCLUSION

This article summarizes the latest and most important advances in GGGP, paying special attention to the crossover operator, which (alongside the initialization method, the codification of individuals and, to a lesser extent, the mutation operator, of course) is chiefly responsible for the convergence speed and the success of the evolution process.

GGGP systems are able to find solutions to any problem that can be syntactically expressed by a CFG. The proposed crossover operator provides GGGP systems with a satisfactory balance between exploration and exploitation capabilities. This results in a high convergence speed, while eluding local optima as the reported results demonstrate. To be able to achieve such good results, the proposed crossover operator includes a computationally cheap mechanism to control bloat, it always generates syntactically valid offspring and it can choose any node from the second parent to generate the offspring, rather than just those nodes with the same non-terminal symbols as the one chosen in the first parent.

REFERENCES

Barrios, D., Carrascal, A., Manrique, D. & Ríos, J. (2003). Optimization with real-coded genetic algorithms based on mathematical morphology. *International Journal of Computer Mathematics*, (80) 3, 275-293.

Couchet, J., Manrique, D., Ríos, J. & Rodríguez-Patón, A. (2006). Crossover and mutation operators for grammar-guided genetic programming. *Softcomputing*, DOI 10.1007/s00500-006-0144-9.

Crawford-Marks, R. & Spector, L. (2002). Size control via size fair genetic operators in the pushGP genetic programming system. *In proceedings of the genetic and evolutionary computation conference*, New York, 733-739.

D'haesler, P. (1994). Context preserving crossover in genetic programming. *In IEEE Proceedings of the 1994 world congress on computational intelligence*, Orlando, (1) 379-407

Dounias, G., Tsakonas, A., Jantzen, J., Axer, H., Bjerregard, B., & von Keyserlingk, D. (2002). Genetic Programming for the Generation of Crisp and Fuzzy Rule Bases in Classification and Diagnosis of Medical Data. *Proceedings of the 1st International NAISO Congress on Neuro Fuzzy Technologies*, Havana, Cuba, 494-500.

Grosman, B. & Lewin, D.R. (2004). Adaptive Genetic Programming for Steady-State Process Modeling. *Computers and Chemical Engineering*, 28 2779-2790.

Hussain, T.S. (2003). *Attribute grammar encoding of the structure and behaviour of artificial neural networks*. PhD Thesis, Queen's University. Kingston, Ontario, Canada.

Keedwell, E., & Narayanan, A. (2005). *Intelligent bioinformatics*. Wiley & Sons.

Koza, JR. (1992). *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge.

Langdon, WB. (1999). Size fair and homologous tree genetic programming crossovers. *In proceedings of genetic and evolutionary computation conference*, GECCO'99, Washington DC, 1092-1097.

Manrique, D. (2001). *Diseño de redes de neuronas y nuevas técnicas de optimización mediante algoritmos genéticos [Artificial neural networks design and new optimization techniques using genetic algorithms]*. PhD Thesis, Facultad de Informática, Universidad Politécnica de Madrid.

Manrique, D., Márquez, F., Ríos, J. & Rodríguez-Patón A. (2005). Grammar-based crossover operator in genetic programming. *Lecture Notes in Artificial Intelligence*, 3562 252-261.

Rodrigues, E. & Pozo, A. (2002). Grammar-Guided Genetic Programming and Automatically Defined Functions. *In proceedings of the 16th Brazilian symposium on artificial intelligence*, Recife, Brazil, 324-333.

Terrio, MD., & Heywood, MI. (2002). Directing crossover for reduction of bloat in GP. *In IEEE proceedings of Canadian conference on electrical and computer engineering*, (2) 1111-1115.

Whigham, P.A. (1995). Grammatically-based genetic programming. *In proceedings of the workshop on genetic programming: from theory to real-world applications*, California, 33-41.

Whigham, P.A. (1996). *Grammatical bias for evolutionary learning*. PhD Thesis, School of Computer Science, Australian Defence Force (ADFA), University College, University of New South Wales.

Yao, X., & Xu, Y. (2006). Recent advances in evolutionary computation. *Journal of Computer Science & Technology*, (21) 1 1-18.

KEY TERMS

Ambiguous Grammar: Any grammar in which different derivation trees can generate the same sentence.

Closure Problem: Phenomenon that involves always generating syntactically valid individuals.

Code Bloat: Phenomenon to be avoided in a genetic programming system convergence process involving the uncontrolled growth, in terms of size and complexity, of individuals in the population

Convergence: Process by means of which an algorithm (in this case an evolutionary system) gradually approaches a solution. A genetic programming system is said to have converged when most of the individuals in the population are equal or when the system cannot evolve any further.

Fitness: Measure associated with individuals in an evolutionary algorithm population to determine how good the solution they represent is for the problem.

Genetic Programming: A variant of genetic algorithms that uses simulated evolution to discover functional programs to solve a task.

Grammar-Guided Genetic Programming: The application of analytical methods and tools to data for the purpose of identifying patterns, relationships or obtaining systems that perform useful tasks such as classification, prediction, estimation, or affinity grouping.

Intron: Segment of code within an individual (subtree) that does not modify the fitness, but is on the side of convergence process.