

---

# **GRAMMATICAL EVOLUTION**

*Evolutionary Automatic Programming  
in an Arbitrary Language*

---

# GENETIC PROGRAMMING SERIES

Series Editor

**John Koza**  
*Stanford University*

*Also in the series:*

**GENETIC PROGRAMMING AND DATA STRUCTURES: Genetic Programming + Data Structures = Automatic Programming!** *William B. Langdon*; ISBN: 0-7923-8135-1

**AUTOMATIC RE-ENGINEERING OF SOFTWARE USING GENETIC PROGRAMMING,** *Conor Ryan*; ISBN: 0-7923-8653-1

**DATA MINING USING GRAMMAR BASED GENETIC PROGRAMMING AND APPLICATIONS,** *Man Leung Wong and Kwong Sak Leung*; ISBN: 0-7923-7746-X

---

*The cover image was generated using Genetic Programming and interactive selection. Anargyros Sarafopoulos created the image, and the GP interactive selection software.*

---

# GRAMMATICAL EVOLUTION

*Evolutionary Automatic Programming  
in an Arbitrary Language*

*by*

**Michael O'Neill**

*University of Limerick, Ireland*

**Conor Ryan**

*University of Limerick, Ireland*



SPRINGER SCIENCE+BUSINESS MEDIA, LLC

### **Library of Congress Cataloging-in-Publication Data**

Michael O'Neill, Conor Ryan

GRAMMATICAL EVOLUTION: Evolutionary Automatic Programming in an Arbitrary Language

ISBN 978-1-4613-5081-1      ISBN 978-1-4615-0447-4 (eBook)

DOI 10.1007/978-1-4615-0447-4

---

**Copyright** © 2003 Springer Science+Business Media New York

Originally published by Kluwer Academic Publishers in 2003

Softcover reprint of the hardcover 1st edition 2003

All rights reserved. No part of this work may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, microfilming, recording, or otherwise, without the written permission from the Publisher, with the exception of any material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work.

Permission for books published in Europe: [permissions@wkap.nl](mailto:permissions@wkap.nl)

Permissions for books published in the United States of America: [permissions@wkap.com](mailto:permissions@wkap.com)

*Printed on acid-free paper.*

# Contents

Preface	ix
Foreword	xi
Acknowledgments	xv
1. INTRODUCTION	1
1 Evolutionary Automatic Programming	1
2 Molecular Biology	2
3 Grammars	2
4 Outline	3
2. SURVEY OF EVOLUTIONARY AUTOMATIC PROGRAMMING	5
1 Introduction	5
2 Evolutionary Automatic Programming	6
3 Origin of the Species	8
4 Tree-based Systems	10
4.1 Genetic Programming	11
4.2 Grammar based Genetic Programming	13
4.2.1 Backus Naur Form	13
4.2.2 Cellular Encoding	16
4.2.3 Bias in GP	16
4.2.4 Genetic Programming Kernel	16
4.2.5 Combining GP and ILP	16
4.2.6 Auto-parallelisation with GP	17
5 String based GP	17
5.1 BGP	18
5.2 Machine Code Genetic Programming	19
5.3 Genetic Algorithm for Deriving Software	20

5.4	CFG/GP	21
6	Conclusions	21
3.	LESSONS FROM MOLECULAR BIOLOGY	23
1	Introduction	23
2	Genetic Codes & Gene Expression Models	24
3	Neutral Theory of Evolution	26
4	Further Principles	28
5	Desirable Features	29
6	Conclusions	32
4.	GRAMMATICAL EVOLUTION	33
1	Introduction	33
2	Background	34
3	Grammatical Evolution	35
3.1	The Biological Approach	36
3.2	The Mapping Process	36
3.2.1	Backus Naur Form	37
3.2.2	Mapping Process Outline	39
3.3	Example Individual	40
3.4	Genetic Code Degeneracy	42
3.5	The Search Algorithm	44
4	Discussion	45
5	Conclusions	47
5.	FOUR EXAMPLES OF GRAMMATICAL EVOLUTION	49
1	Introduction	49
2	Symbolic Regression	49
2.1	Results	51
3	Symbolic Integration	52
3.1	Results	52
4	Santa Fe Ant Trail	55
4.1	Results	56
5	Caching Algorithms	57
5.1	Results	60
6	Conclusions	62
6.	ANALYSIS OF GRAMMATICAL EVOLUTION	63
1	Introduction	63

2	Wrapping Operator	64
2.1	Results	64
2.1.1	Invalid Individuals	64
2.1.2	Cumulative Frequency of Success	65
2.1.3	Genome Lengths	65
2.2	Discussion	67
3	Degenerate Genetic Code	67
3.1	Results	69
3.1.1	Diversity Measures	70
3.2	Discussion	72
4	Removal of Wrapping and Degeneracy	72
4.1	Results	72
5	Mutation Rates	74
5.1	Results	76
6	Conclusions	77
7.	CROSSOVER IN GRAMMATICAL EVOLUTION	79
1	Introduction	79
2	Homologous Crossover	81
2.1	Experimental Approach	81
2.2	Results	83
2.3	Discussion	91
3	Headless Chicken	92
3.1	Experimental Approach	93
3.2	Results	94
3.3	Discussion	95
4	Conclusions	98
8.	EXTENSIONS & APPLICATIONS	99
1	Translation	99
2	Alternative Search Strategies	101
3	Grammar Defined Introns	102
4	GAUGE	103
4.1	Problems	105
4.1.1	Onemax	106
4.1.2	Results	107
4.2	Mastermind - a deceptive ordering version	108
4.2.1	Results	109

4.3	Discussion	112
4.4	Conclusions and Future Work	112
5	Chorus	113
5.1	Example Individual	114
5.2	Results	116
6	Financial Prediction	117
6.1	Trading Market Indices	117
6.1.1	Experimental Setup & Results	119
7	Adaptive Logic Programming	121
7.1	Logic Programming	121
7.2	GE and Logic Programming	123
7.2.1	Backtracking	124
7.2.2	Initialisation	125
7.3	Discussion	125
8	Sensible Initialisation	125
9	Genetic Programming	127
10	Conclusions	128
9.	CONCLUSIONS & FUTURE WORK	129
1	Summary	129
2	Future Work	130
	Index	143

# Preface

Since man began to dream of machines that could automate not only the more mundane and laborious tasks of everyday life, but that could also improve some of the more agreeable aspects, he has turned to nature for inspiration. This inspiration has taken all sorts of forms, with inventors producing everything from Icarus-like, bird-inspired flying machines to robots based on some of the more mechanically useful human appendages.

Another inspiration that can be taken from nature is to employ its *tools*, rather than necessarily employing its *products*. In this way, the field of evolutionary computation has taken stock of the power of evolution, and applied it, albeit at a very coarse level, to problem solving. Genetic Programming, a powerful incarnation of evolutionary computation uses the artificial evolutionary process to automatically generate programs. The adoption of evolution to automatic generation of programs represents one of the most promising approaches to that holy grail of computer science, automatic programming, that is, a computer that can automatically generate a program from scratch given a high-level problem description.

Research in Genetic Programming has explored a number of program representations beyond the original Lisp S-expression syntax trees, and some of the more powerful of these incorporate a developmental strategy that transforms an embryonic state into a fully fledged adult program.

The form of Genetic Programming presented in this book, Grammatical Evolution, delves further into nature's processes at a molecular level, embracing the developmental approach, and drawing upon a number of principles that allow an abstract representation of a program to be evolved.

This abstraction enables firstly, a separation of the search and solution spaces that allow the EA search engine to be a plug-in component of the system, facilitating the exploitation of advances in EAs by GE. Secondly, this allows the evolution of programs in an arbitrary language with the representation of a program's syntax in the form of a grammar definition.

Thirdly, the existence of a degenerate genetic code is enabled, giving a many-to-one mapping, that allows the exploitation of neutral evolution to enhance the search efficiency of the EA. Fourthly, we can adopt the use of a wrapping operator that allows the reuse of genetic material during a genotype-phenotype mapping process.

This book is partly based on the Ph.D. thesis of Michael O'Neill, and reports a number of new directions in Grammatical Evolution research that are being conducted both within the confines of the University of Limerick's Biocomputing-Developmental Systems Centre where the book's authors reside, and also developments that are occurring through collaborations around the globe.

M. O'NEILL & C. RYAN

# Foreword

This book is based on the PhD thesis of Michael O'Neill, but is written in a very accessible language intended for a wider audience. The authors succeed to elucidate aspects of Genetic Programming that have hitherto not been examined in such detail.

Genetic Programming (GP) was first applied seriously in expression trees of LISP-like program structures. I do not rule out the possibility that these first steps were due to the clean nature of LISP and its derivatives which made it possible to focus attention on the mechanical problem of getting evolution to go, instead of being forced to constantly observe the behavior of the underlying representation.

After the formalization of grammars through Chomsky, both the Linguistics and the Computer Science community have taken advantage of the new tools. Notably Computer Science was able to exploit Chomsky's formalism for the systematic generation of a multitude of computer languages that since populate our machines. Up to this day it is a key activity of Computer Scientists to invent new languages. It is thus very natural to ask whether Genetic Programming which intends to apply artificial evolution to the determination of computer behavior would not be best used in the realm of those structures and rules which make up computer programs.

Once it was shown that GP is possible in principle, a variety of representations appeared which exploited different aspects of the paradigm. Among these were grammars and linear strings of code. Both aspects are an active area of research in Evolutionary Computation. The use of grammatical structures for GP has recently gained much prominence through the work of the authors. In a variety of applications O'Neill and Ryan show convincing evidence that Grammatical Evolution, GE as it is abbreviated, is a lively branch of Genetic Programming.

In the distinctive words of the authors, with grammars "one is effectively using the weapon of the enemy against them". Complexity of behavior is approached from a perspective of building it up from simple rules, much as the

complexity of meaning is built up from the use of grammars in natural language. At the heart of GE lies the fact that genes are only used to determine which rule is applied when, not what the rules are. Those are fixed beforehand as the grammar and provide the environment from which genes choose their "words".

Other important aspects of this work are development and linearity of code. By development in Grammatical Evolution, the process of constructing the phenotype is meant, a process strictly under control of the chosen (arbitrary) grammar. Linearity is an aspect of GE analogous to what is found in natural evolutionary systems. Instead of nucleotides, the linear sequence of genotypes is written in simple bit patterns which are interpreted appropriately. The process generates a number of phenomena also found in natural evolution, for instance the appearance (and productive use) of neutral code.

In my mind, we are just at the beginning of a thorough exploration of possibilities for artificial evolution through these processes. A great deal of *Terra Incognita* lies before us and is waiting to be explored. "Grammatical Evolution" is an important contribution to this undertaking. I am very glad that this book, written by two specialists, Michael O'Neill and Conor Ryan, has appeared.

Wolfgang Banzhaf

*To John & Jane, and  
Gráinne  
Michael*

*To Katy, my favourite  
distraction  
Conor*

# Acknowledgments

It has been five years since the first Grammatical Evolution publication, and in that time, a large number of people have helped us out in all sorts of ways. We are very fortunate that many of these people have also become very good friends because of, and, in some cases, despite, working with us.

**J.J. Collins** was a co-author on the first GE paper and contributed a lot of GA code to the original system, while **Maarten Keijzer** has been involved in a number of our recent papers, was responsible for the work on Ripple Crossover and ALP, and contributed to many intense discussions, some of which were related to Evolutionary Computation. **Tony Brabazon** introduced us to the delights of Financial Prediction, and was involved in some of the papers that contributed to Chapter 8. **Miguel Nicolau** contributed to a number of different aspects of this work, and is currently developing GAUGE, as well as pretending that he can play the guitar. **Atif Azad** and **Ali Ansari** co-operated with the work on Chorus, as did **Alan Sheahan**, who also advised us on statistics in the book, as well as pretending that he can't play the guitar. **Mike Cattolico** was involved in the work on crossover, and made sure that we ate properly while in the US. **John O'Sullivan** was responsible for the work on comparing search strategies, and **Vladan Babovic** was a collaborator on both the Ripple Crossover and Logic Programming sections, while **Robin Matthews** collaborated on the Financial Prediction work.

Other people not directly involved in the work, but to whom we are thankful for their support and help include **Wolfgang Banzhaf**, **John Koza**, **Forrest H. Bennett III**, **Bill Langdon**, **Una-May O'Reilly**, **Norman Paterson**, **Tony Cahill** and **Jennifer Willies**.

We are also thankful to the continued support of the **Computer Science and Information Systems** department at the **University of Limerick**, past and present members of the **Biocomputing-Developmental Systems Centre** there and the staff at Kluwer, particularly the ever-patient **Melissa Fearon**.

We would like to thank our parents, families and friends for showing that there is life beyond GE.