Lee Spector, W. B. Langdon, Una-May O'Reilly, Peter Angeline

Welcome to the third volume of *Advances in Genetic Programming* series. The Genetic Programming (GP) field has matured considerably since the first two volumes were produced in conjunction with workshops held at the International Conference on Genetic Algorithms (ICGA) [Kinnear, Jr., 1994; Angeline and Kinnear, Jr., 1996]. During the 1993 and 1995 ICGA conferences the interest in GP was strong but within this broader community there were only a few presentation and publication slots for GP work. The *Advances* volumes therefore provided an important outlet for creative new work in GP. Interest in GP continued to grow and there is now a separate annual conference on GP in the USA, as well as a European workshop on GP and several other conferences that regularly feature GP themes (see Section 1.2).

Since annual conferences now serve the function of quickly disseminating concise descriptions of current research, it is appropriate to update the role of the *Advances* series. As editors, we decided that this volume should be highly selective and focus solely on genuine advances that we feel are outstanding and will have broad impact on those interested in GP.

The volume consists of a combination of invited and contributed chapters. A call for contributions to the book was circulated electronically on the genetic programming mailing list[1] and in several other forums. The call for contributions stated that the "primary criterion by which submissions will be evaluated is the likely impact on the future work of other theoreticians and practitioners." We also invited a number of active researchers who previously demonstrated significant contributions to submit chapters that they felt best addressed the criteria in the call for contributions. All chapters were reviewed by the editors, sometimes with the aid of additional specialists, and only those deemed to be true advances in the field were selected for inclusion in the book. We turned away several otherwise solid and interesting papers that we felt fell short of our standards for anticipated impact. We gave the accepted chapters more pages than in previous volumes, and we conducted an intensive, iterative review/revision process in order to produce chapters of the highest possible quality. We believe that the solicitation procedure, the review criteria, and the editing process were all successful, producing an outstanding collection of chapters that will endure as being critical and influential to practitioners of GP.

For those new to GP we start in Section 1.1 with a description of the basic genetic programming algorithm. Sections 1.2 and 1.3 contain additional sources of information and public domain software implementations of GP that we hope will be of wide interest. Finally, for all readers, Section 1.4 provides an overview of the remaining chapters of the book that we hope will encourage the simply curious reader to browse creatively and the more deliberate reader to immediately find sought-after material.

[1] The address for the list is `genetic-programming@cs.stanford.edu`. To subscribe send an e-mail message with the body "`subscribe genetic-programming` *your@email.address*" to `genetic-programming-request@cs.stanford.edu`.

1.1 A brief overview of genetic programming

This section treats GP as a specialization of the genetic algorithm (GA) technique [Holland, 1975]. This is complementary to the perspective in *Advances II*, which showed how GP relates to Evolutionary Programming [Angeline, 1996]. The basic procedure of a GA can be abbreviated as follows:

1. Create an initial population of random individuals.

2. Test the "fitness" of each individual.

3. If an individual is sufficiently good then stop and report success.

4. Otherwise create a new population from the more fit individuals, using "genetic operators" such as reproduction, mutation, and crossover.

5. Replace the old population with the new population and return to Step 2.

This algorithm characterizes GAs in general, and one can view GP as a specialization of GA for the evolution of executable programs. The primary differences between GP systems and other GA systems are:

- In a GP system individuals are generally executable structures such as computer programs, while in other GA systems individuals take various forms (typically bit or character strings).

- In a GP system fitness is normally assessed by *executing* the individuals, while in other GA systems fitness is assessed in ways that depend on the structure of individuals and the problem being solved.

To apply GP to a particular problem one must first specify a set of primitive elements, called *functions* and *terminals*, out of which candidate and complete solutions may be constructed. In the simplest case one chooses these elements in a way that ensures that every possible combination of elements will execute without generating an error condition, typically by restricting all values to be of a single data type. One must then provide a *fitness function* that assesses the quality of individuals in the population, and set values for a range of parameters covering details of structure representation, selection method, initialization procedures, genetic operators, and so on. Once these preparatory steps are taken one can run the genetic programming system to start the algorithm outlined above, producing, in the best case, a program that accomplishes the required task in a short amount of time. Variations are possible and in many cases advantageous; many of these are described in detail in this and earlier volumes of this series [Kinnear, Jr., 1994; Angeline and Kinnear, Jr., 1996].

1.2 Other GP Resources

More detail on the basic GP technique, its relation to other work in machine learning and treatment of advanced topics in GP can be found in several sources. [Banzhaf et al., 1998a] is particularly recommended for newcomers to the field as it contains clear introductory examples and also extensive references to the GP literature.

- Books and Journals

– *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, along with followup volumes 2 and 3 (forthcoming) [Koza, 1992; Koza, 1994; Koza et al., 1999].

– *Genetic Programming, An Introduction* [Banzhaf et al., 1998a].

– The first two volumes of *Advances in Genetic Programming* [Kinnear, Jr., 1994; Angeline and Kinnear, Jr., 1996] and (of course) this volume, III.

– *Genetic Programming and Data Structures* [Langdon, 1998]

– *Evolutionary Induction of Binary Machine Code* [Nordin, 1997]

– *Journal of Genetic Programming and Evolvable Machines*, Kluwer (forthcoming)

- Proceedings

– *Genetic Programming '96, '97, '98* [Koza et al., 1996; Koza et al., 1997; Koza et al., 1998], and late breaking papers collections [Koza, 1996; Koza, 1997; Koza, 1998]

– *EuroGP '98, '99* [Banzhaf et al., 1998b; Poli et al., 1999], late breaking papers [Poli et al., 1998]

Many other conference and workshop proceedings have GP related papers. Some examples are the International Conference on Genetic Algorithms (ICGA), Foundations of Genetic Algorithms (FOGA), Parallel Problem Solving from Nature (PPSN), International Conference on Evolutionary Computation (ICEC), Artificial Life (ALife), and European Conference on Artificial Life (ECAL). A good way to find a specific paper or conference volume is to use the GP bibliography (see below).

- [Langdon, 1996] contains a GP bibliography organised by subject. A non indexed (but current) version is available at `http://www.cs.bham.ac.uk/~wbl/biblio/gp-bibliography.html`

- Mailing lists

∗ GPlist – *the* GP discussion list. Subscribe by sending email to `genetic-programming-REQUEST@cs.stanford.edu` with a single line `subscribe` in the body of

the message. Beware the FAQ (Frequently Asked Questions) is seriously out of date, especially ftp addresses. A nearly complete archive of previous messages can be found at http://adept.cs.twsu.edu/~thomas/gpmail.html

∗ GA Digest – Moderated list on Genetic Algorithms, approximately monthly bulletins. Subscribe by sending email to ga-list-REQUEST@aic.nrl.navy.mil An archive of previous messages, etc. is located at http://www.aic.nrl.navy.mil/galist

- WWW and FTP sites

∗ Many of our authors have extensive Internet World Wide Web (www) pages. For example, John Koza's www pages include information on teaching, publications, GP, www links, etc. See http://smi-web.stanford.edu/people/koza. Information about conferences on GP, and on evolutionary computation in general, plus general information on GP can also be found at http://www.genetic-programming.org

∗ GP ftp site, code and early papers ftp://ftp.mad-scientist.com/pub/genetic-programming

∗ GP Notebook — multiple sites e.g. http://www.geneticprogramming.com

∗ ENCORE — The Hitch-Hiker's guide to evolutionary computation — multiple sites, e.g. ftp://ftp.de.uu.net/pub/research/softcomp/EC

∗ Bias within GP tutorial, GP98, http://www.cs.rochester.edu/u/rosca/GPTutorialPage.html

∗ Additional internet resources are documented in [Tufts, 1996].

1.3 Public Domain GP Implementations (unsupported)

C/C++

lilgp — Douglas Zonger, Michigan State University, GARAGe, Genetic ALgorithm Research and Application Group http://garage.cse.msu.edu/software/software-index.html 325kb

DGPC — Dave's [Andre] Genetic Programming code in C, (only 2 bytes per tree node) http://www.cs.berkeley.edu/~dandre/gp.tar.gz 211kb

GPQUICK/GPdata — Andy Singleton/W. B. Langdon, C++ (only 1 byte per tree node) http://www.cs.bham.ac.uk/~wbl/ftp/gp-code 172kb

SGPC — Simple GP in C, Walter Tackett and gpc++ — C++, by Adam Fraser may be found at ftp://ftp.mad-scientist.com/pub/genetic-programming/code

Lisp

Lisp code for [Koza, 1992] and [Koza, 1994] is also stored in `ftp://ftp.mad-scientist.com/pub/genetic-programming/code`

Java

GPsys — Adil Qureshi `http://www.cs.ucl.ac.uk/staff/A.Qureshi`

GP across the Internet — Chong `ftp://ftp.cs.bham.ac.uk/pub/authors/W.B.Langdon/gp-code/DGP`

Others

Implementations in other languages, for example SmallTalk, Mathematica and Prolog can also be found at `ftp://ftp.mad-scientist.com/pub/genetic-programming/code`

1.4 The work in this volume

1.4.1 Part I: Applications

Perhaps the most dramatic change in the field since the previous volumes of *Advances* has been the growth in the number of real-world applications. This is an important sign of the maturity of the field; early GP work included many applications, but many of these were simple and it was rare that the applied work solved hard problems that workers in the application areas really needed to have solved. In several recent cases, however, GP researchers have teamed up with engineers or researchers in other fields and the resulting teams have applied GP tools to significant real-world problems. In other cases the transition to real-world applications has been made without the aid of GP specialists, a sign that GP technology is ready for more widespread use. We chose to start this volume with the applications in part because we would like to show those outside GP that it is feasible to apply it to outstanding problems in their fields. We also feel that GP researchers working on theoretical issues or on extensions to GP technique should keep one eye trained on the real-world uses of GP.

Chapter 2, "An Automatic Software Re-Engineering Tool based on Genetic Programming" by Conor Ryan and Laur Ivan, attacks the difficult and commercially important problem of automatically transforming serial programs into functionally identical parallel programs written in a C-like syntax. The authors argue that their Paragen system extracts parallelism at least as well as human experts in much less time, and that it serves an important need as no other fully automatic parallelization techniques are currently available. They also argue that GP is particularly well-suited to this application area because it is not

only able to spot and exploit code patterns but also to explore the possibilities of moving and transforming code in ways that humans would not.

In Chapter 3, "CAD Surface Reconstruction from Digitized 3D Point Data with a Genetic Programming/Evolution Strategy Hybrid" by Robert E. Keller, Wolfgang Banzhaf, Jörn Mehnen and Klaus Weinert, we see GP applied in another commercially important area. In many modern manufacturing processes, after development, prototypical physical workpieces are digitized to obtain their descriptions. Unfortunately, a digitized representation is not sufficient as a computer-aided design object and it is necessary to derive from it an appropriate construction-oriented CAD surface representation before it can be provided as input to rapid-prototype systems or used in other design or construction processes. The authors present a system, SURREAL, that uses a hybrid of GP and Evolution Strategies [Rechenberg, 1994] to perform the requisite "surface reconstruction" task. SURREAL simultaneously performs pattern recognition and structure evolution. It exploits the variable-length representation of genetic programming to explore search spaces without pre-defined dimensionality. The authors note that SURREAL has potential uses beyond the obvious manufacturing applications; for example, it should perform better than existing reconstruction approaches for highly irregular surfaces such as human faces, with possible applications in cosmetics, medicine, entertainment, and person identification.

Carolyn Penstein Rosé provides a promising new GP-based approach to a classical artificial intelligence problem in Chapter 4, "A Genetic Programming Approach for Robust Language Interpretation." The problem of robust language interpretation is to construct a representation of the meaning of a sentence which is independent of its words by using a predefined set of meaning-encoded primitives in relation to each other. A parsing grammar first handles the task, but, because it is necessarily incomplete, it fails on *extragrammatical* sentences that fall outside its domain producing only partially-parsed subexpressions. ROSE uses genetic programming to search the space of possible relationships between the meaning representation of these grammatical sub-expressions in order to construct an accurate and complete representation of the entire sentence. The ROSE system depends on GP to be efficient. Using GP obviates the need for an expensive, maximally flexible parser and, as a repair strategy, it avoids the need for hand-coded repair rules. Overall, the author states that ROSE "yields a significantly better time/quality trade-off than previous non-GP approaches".

In Chapter 5, "Time Series Modeling Using Genetic Programming: An Application to Rainfall-runoff Models," P. A. Whigham and P. F. Crapper apply a novel grammar-based variant of GP to a hydrological modeling problem. Their system discovers rainfall-runoff relationships for two different catchments on different continents and with different climates. They show it is more robust than previous methods when rainfall-runoff correlation are poor and when the assumptions built into other methods do not hold. The authors also note that while other machine learning techniques have been applied to rainfall-runoff modeling, their GP-based system produces models that are more readily interpretable in terms of processes and behaviors used in the application area.

In Chapter 6, “Automatic Synthesis, Placement, and Routing of Electrical Circuits by Means of Genetic Programming,” John R. Koza and Forrest H Bennett III present the latest enhancements to their work on GP applications to electrical circuit design. In this chapter they demonstrate for the first time how a single GP process can automatically create the topology, component sizing, placement, and routing of analog electrical circuits, a process that normally requires several human engineers with specialized design skills. In addition to a general description of their technique they provide a detailed example of the evolution of an analog low pass filter. They also point out that GP has now produced many results that are competitive with human-produced results, including many for which patents have previously been awarded, and they discuss GP as an “invention machine”.

Chapter 7, “Quantum Computing Applications of Genetic Programming” by Lee Spector, Howard Barnum, Herbert J. Bernstein and Nikhil Swamy, presents work on applying GP to computers that don’t yet exist—“quantum computers” that will manipulate the states of atomic-scale objects to gain efficiencies not achievable with digital computers based on classical physics. Even though large-scale quantum computers do not yet exist GP can be used in conjunction with a simulated quantum computer both to discover new quantum computer algorithms and to help explore the the potential power of quantum computing. The authors demonstrate the evolution of several better-than-classical quantum algorithms including both previously discovered and new examples.

1.4.2 Part II: Theory

As the applications of GP have expanded it has become more urgent that we understand how and why the process works when it does, and how and why it fails when it fails. Only on the basis of a solid theoretical understanding of these issues can we intelligently apply or enhance the technique. Early theoretical approaches, based for example on the schema theorem from traditional genetic algorithms, made little progress because of several factors including GP’s richer representations, variable length genotypes, and execution semantics for individuals. Neither have attempts to avoid theory, relying for example on blind faith in the power of recombination and natural selection, produced sufficiently satisfying answers to questions about how and why GP works or doesn’t work.

Chapter 8, “The Evolution of Size and Shape” by W. B. Langdon, Terry Soule, Riccardo Poli and James A. Foster investigates the phenomenon of growth in program size in GP populations. This has been known for some time but previous explanations have not been completely satisfactory. The authors start from the basics — What is the nature of the search space upon which GP operates? That is, what are program search spaces like? How big are they, i.e. how many possible programs are there? What shapes can they have? How are program shapes (particularly tree shapes) distributed in the search space? How is program performance (i.e. fitness) distributed in the space? They present a maximum likelihood (maximal entropy) model of bloat, which simply suggests programs evolve towards

the part of the search space containing the most programs of the current best fitness level. This not only explains the evolution of program length but also program shape (which had not previously been considered). The chapter also considers various GP specific mechanisms in bloat and proposes two new genetic operators which considerably reduce bloat. The keenly interested reader should also read Chapter 11 where Rosca and Ballard also propose a bloat-reducing genetic operator which is derived from an analysis with a different perspective. As well, in Chapter 16, Ito, Iba and Sato propose new genetic operators that foster the protection and growth of building blocks. Their evaluation shows that their self-tuning mechanism also can address bloat.

Chapter 9, "Fitness Distributions: Tools for Designing Efficient Evolutionary Computations" by Christian Igel and Kumar Chellapilla, is an important landmark in the theoretical analysis of subtree mutation operators. Recently there has been impressive experimental work on mutation-only GP [Chellapilla, 1997], and in this chapter the authors start to lay theoretical ground work for this work by analysing the effectiveness of a total of 11 different subtree operators within evolving populations on a total of four benchmark problems. Their analysis yields important conclusions about these operators and recommendations for their future use.

Chapter 10, "Analysis of Single-Node (Building) Blocks in Genetic Programming" by Jason M. Daida, Robert R. Bertram, John A. Polito 2, and Stephen A. Stanhope, addresses the crux of the crossover versus mutation debate in automatic program generation (cf. Chapter 9). That is the question of "building blocks". Do they exist in program parse trees? If so, are the current mechanisms effective in finding them and building complete solutions from them? If not, can more effective mechanisms be devised? The authors empirically analyze the evolutionary process as it combines and exploits the most primitive potential building block in GP — terminals which are ephemeral random constants. Their investigation provides qualified support for the notion of simple building blocks within the current GP framework, but they warn that GP dynamics are complex. They further suggest that the notions of genotype and phenotype need careful rethinking when used in the context of GP.

Chapter 11, "Rooted-Tree Schemata in Genetic Programming" by Justinian P. Rosca and Dana H. Ballard, turns GP on its head. Instead of viewing the GP process as forming solutions in a "bottom up" fashion it suggests GP solutions evolve from the root down in a "top down" fashion. They present detailed quantitative analysis of several important aspects of the dynamics of GP. Following a review of work on Genetic Algorithms (GAs) population dynamics in terms of the schema theorem and Price's Theorem they present a GP schema theorem based on rooted-tree schema. They then extend this to consider the phenomenon of increase in program size in GP and a common response to it, adding a parsimony bias to the fitness function. Their theoretical analysis suggests values for the strength of the parsimony bias which they experimentally verify using the parity and Pac-Man problems. In addition, further analysis indicates that schema growth needs to be controlled so that the size of a program does not influence the likelihood of a tree-schema's

survival during crossover. Two methods of exercising this control are delineated and one, which adapts the probability of disruption of a tree as a function of its size, is shown to be effective on the parity and Pac-Man problems.

1.4.3 Part III: Extensions

The final part of the book contains eight chapters that push the genetic programming framework in new directions, either to allow for new classes of applications or to produce qualitative improvements in performance.

Chapter 12, "Efficient Evolution of Machine Code for CISC Architectures using Instruction Blocks and Homologous Crossover" by Peter Nordin, Wolfgang Banzhaf and Frank D. Francone, describes two major advances for the world of linear machine code GP. Both are aimed at significantly extending their commercial GP system Discipulus. The first advance enables machine code GP to evolve machine code for complex instruction set computers (CISC, such as INTEL X86 chips used in most personal computers PCs, JAVA and many embedded processors) whilst retaining the speed and performance advantages of machine code evolution available up to now on reduced instruction set computers (RISC) such as the SUN-Sparc. The second advance is a description of homologous crossover in machine code GP. By ensuring that like parts of programs are crossed over the authors are able to produce a more productive crossover operator in which the offspring are more related to their parent programs. Chapter 12 also contains comparisons between tree and machine code GP and discussions of future applications of machine code GP.

Modern computers typically act on 32 or 64 bits simultaneously. Chapter 13, "Sub-machine-code Genetic Programming" by R. Poli and W. B. Langdon, describes several ways in which GP can exploit this internal parallelism. Their technique is described and demonstrated via several Boolean problems and an optical character recognition (OCR) problem. The technique is easy to implement (pointers to publicly available code are included) and can readily speed up any existing GP implementation dramatically (one to nearly two orders of magnitude).

Astro Teller describes in Chapter 14, "The Internal Reinforcement of Evolving Algorithms", a new approach in which the behaviour of a parent program is used to create a credit-blame map locating the useful and not-so-useful parts within it. The credit-blame map is used to guide the location of crossover points so that the useful parts of the program are more likely to preserved in its offspring. Each program is represented as a directed graph with similarities to artificial neural networks and data flow machines with indexed memory. PADO evolves programs which correctly classify their input into classes. The new system of internal reinforcement, which is demonstrated within PADO on problems of classifying complex real images and sounds, is shown to improve performance.

Chapter 15, "Inductive Genetic Programming with Immune Network Dynamics" by Nikolay I. Nikolaev, Hitoshi Iba and Vanio Slavov, describes a major advance in which previously unconnected streams of GP research (Stroganoff and Immune Genetic Program-

ming) are brought together and shown to be highly effective for example on time-series prediction problems using small populations. Immune Genetic Programming is presented using an analogy of an immune system (GP) evolving to overcome a disease by binding to a number of disease causing antigens (the test cases). Perhaps one of the most important aspects of the new approach is that it provides a principled mechanism for dynamically changing the fitness function (i.e. the active test cases) which improves GP learning efficiency.

Chapter 16, "A Self-Tuning Mechanism for Depth-Dependent Crossover" by Takuya Ito, Hitoshi Iba and Satoshi Sato, proposes an improved crossover operator for tree based GP. They note that traditional crossover tends to swap relatively small subtrees and, while in some problems exchanging larger subtrees appears to improve performance, this is not always the case. Therefore they propose a new operator which takes into account this problem dependency and learns, via a self-tuning mechanism, the best size to use for crossover as the GP run proceeds. The mechanism depends on the assumption that, if depth selection probability is efficiently assigned to a tree, the fitness of the structure's offspring will improve and the depth selection probability will be inherited by subsequent generations. That is, their subtree crossover operator coevolves with the GP population and so can adapt to the problem and achieve improved results. They also note a reduction in population "bloat".

Chapter 17, "Genetic Recursive Regression for Modeling and Forecasting Real-World Chaotic Time Series" by Geum Yong Lee, presents several improved techniques whereby GP can evolve non-linear models of time-varying statistics based upon a history of several GP runs. The improvements are demonstrated on a wide range of different time series data and evolve improved models with small populations and therefore low computational effort.

Chapter 18, "Coevolutionary Fitness Switching: Learning Complex Collective Behaviors Using Genetic Programming" by Byoung-Tak Zhang and Dong-Yeon Cho, tackles one of the important unsolved problems: how to get multiple independent programs/agents/robots to learn to co-operate together to solve complex problems. They present improved learning based upon coevolution and demonstrate it on Robot Soccer games. These games are known for their difficulty and the manual coding of winning teams is far from trivial.

Our final chapter, Chapter 19, "Evolving Multiple Agents by GP" by Hitoshi Iba, continues the multi-agent theme. Here Iba uses the difficult robot navigation problem to demonstrate a new multi-agent learning scheme in which the agents learn to communicate with each other. He compares this with more traditional Q-learning approaches (which scale badly). However the chapter also describes efficient means of combining GP and Q-learning.

Bibliography

Angeline, P. J. (1996), "Genetic programming's continued evolution," in *Advances in Genetic Programming 2*, P. J. Angeline and K. E. Kinnear, Jr. (Eds.), Chapter 1, pp 1–20, Cambridge, MA, USA: MIT Press.

Angeline, P. J. and Kinnear, Jr., K. E. (1996), *Advances in Genetic Programming 2*, Cambridge, MA, USA: MIT Press.

Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. (1998a), *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*, Morgan Kaufmann, dpunkt.verlag.

Banzhaf, W., Poli, R., Schoenauer, M., and Fogarty, T. C. (1998b), *Genetic Programming*, volume 1391 of *LNCS*, Paris: Springer-Verlag.

Chellapilla, K. (1997), "Evolving computer programs without subtree crossover," *IEEE Transactions on Evolutionary Computation*, 1(3):209–216.

Holland, J. H. (1975), *Adaptation in natural artificial systems*, Ann Arbor: University of Michigan Press.

Kinnear, Jr., K. E. (1994), *Advances in Genetic Programming*, Cambridge, MA: MIT Press.

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

Koza, J. R. (1994), *Genetic Programming II: Automatic Discovery of Reusable Programs*, Cambridge Massachusetts: MIT Press.

Koza, J. R. (1996), *Late Breaking Papers at the Genetic Programming 1996 Conference Stanford University July 28-31, 1996*, Stanford University, CA, USA: Stanford Bookstore.

Koza, J. R. (1997), *Late Breaking Papers at the 1997 Genetic Programming Conference*, Stanford University, CA, USA: Stanford Bookstore.

Koza, J. R. (1998), *Late Breaking Papers at the 1998 Genetic Programming Conference*, University of Wisconsin, Madison, WI, USA: Omni Press.

Koza, J. R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M. H., Goldberg, D. E., Iba, H., and Riolo, R. (1998), *Genetic Programming 1998: Proceedings of the Third Annual Conference*, University of Wisconsin, Madison, WI, USA: Morgan Kaufmann.

Koza, J. R., David Andre, Bennett III, F. H., and Keane, M. (1999), *Genetic Programming 3*, Morgan Kaufman, Forthcoming.

Koza, J. R., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M., Iba, H., and Riolo, R. L. (1997), *Genetic Programming 1997: Proceedings of the Second Annual Conference*, Stanford University, CA, USA: Morgan Kaufmann.

Koza, J. R., Goldberg, D. E., Fogel, D. B., and Riolo, R. L. (1996), *Genetic Programming 1996: Proceedings of the First Annual Conference*, Stanford University, CA, USA: MIT Press.

Langdon, W. B. (1996), "A bibliography for genetic programming," in *Advances in Genetic Programming 2*, P. J. Angeline and K. E. Kinnear, Jr. (Eds.), Appendix B, pp 507–532, Cambridge, MA, USA: MIT Press.

Langdon, W. B. (1998), *Data Structures and Genetic Programming: Genetic Programming + Data Structures = Automatic Programming!*, Boston: Kluwer.

Nordin, P. (1997), *Evolutionary Program Induction of Binary Machine Code and its Applications*, PhD thesis, der Universitat Dortmund am Fachereich Informatik.

Poli, R., Langdon, W. B., Schoenauer, M., Fogarty, T., and Banzhaf, W. (1998), *Late Breaking Papers at EuroGP'98: the First European Workshop on Genetic Programming*.

Poli, R., Nordin, P., Langdon, W. B., and Fogarty, T. C. (1999), *Genetic Programming, Proceedings of EuroGP'99*, LNCS, Goteborg, Sweden: Springer-Verlag, forthcoming.

Rechenberg, I. (1994), *Evolutionsstrategie'94*, volume 1 of *Werkstatt Bionik und Evolutionstechnik*, Stuttgart: Friedrich Frommann Verlag (Günther Holzboog KG).

Tufts, P. (1996), "Genetic programming resources on the world-wide web," in *Advances in Genetic Programming 2*, P. J. Angeline and K. E. Kinnear, Jr. (Eds.), Appendix A, pp 499–506, Cambridge, MA, USA: MIT Press.