Genetic Programming Theory and Practice 2010: An Introduction

Trent McConaghy[1], Ekaterina Vladislavleva[2], and Rick Riolo[3]

[1]*Solido Design Automation Inc., Canada;* [2]*Department of Mathematics and Computer Science, University of Antwerp, Antwerp, Belgium;* [3]*Center for Study of Complex Systems, University of Michigan.*

Abstract The toy problems are long gone, real applications are standard, and the systems have arrived. Genetic programming (GP) researchers have been designing and exploiting advances in theory, algorithm design, and computing power to the point where (traditionally) hard problems are the norm. As GP is being deployed in more real-world and hard problems, GP research goals are evolving to a higher level, to *systems* in which GP algorithms play a key role. The key goals in GP algorithm design are reasonable resource usage, high-quality results, and reliable convergence. To these GP algorithm goals, we add GP system goals: ease of system integration, end-user friendliness, and user control of the problem and interactivity. In this book, expert GP researchers demonstrate how they have been achieving and improving upon the key GP algorithm *and* system aims, to realize them on real-world / hard problems. This work was presented at the GP Theory and Practice (GPTP) 2010 worshop. This introductory chapter summarizes how these experts' work is driving the frontiers of GP algorithms and GP systems in their application to ever-harder application domains.

Keywords: genetic programming, evolutionary computation

1. The Workshop

In May 2010 the Center of Studies of Complex Systems at the University of Michigan – with deep historical roots in evolutionary computation tracing back to Holland's seminal work – opened its doors for the invitees of the workshop on Genetic Programming in Theory and Practice 2010. Over twenty experienced and internationally distinguished GP researchers gathered in Ann Arbor to close themselves in one room for two and a half days, present their newest (and often controversial) work to the critical attention of their peers, discuss the challenges of genetic programming, search for common traits in the field's development, get a better understanding of the global state-of-the-art and share the vision on the "next big things" in GP theory and practice.

The atmosphere at the workshop has always been enjoyable, with every participant trying to get a deep understanding of presented work, provide constructive comments on it, suggest links to the relevant topics in the broad field of computing, and question generality, scalability of the approach. The workshop fosters a friendly atmosphere wherein inquiring minds are genuinely trying to understand not only what they collectively know or can do with GP, but also

Introduction to *Genetic Programming Theory and Practice VIII*, R. Riolo, T. McCongahy, K. Vladislavleva (Eds.)

what they collectively do not yet know or cannot yet do with GP. The latter understanding is a major driving force for further developments that we have observed in all workshops.

We are grateful to all sponsors and acknowledge the importance of their contributions to such an intellectually productive and regular event. The workshop is generously founded and sponsored by the University of Michigan Center for the Study of Complex Systems (CSCS) and receives further funding from the following people and or organizations: Michael Korns of Freeman Investment Management, State Street Global Advisors, Third Millenium, Bill and Barbara Tozier of Vague Innovation, Evolved Analytics, the Computational Genetics Laboratory of Dartmouth College and the Biocomputing and Developmental Systems Group of the University of Limerick.

We also thank Jürgen Schmidhuber for an enlightening and provocative keynote speech, which covered his thoughts on what makes a scientific field mature, a review of his work in solving difficult real-world problems in pragmatic ways, and his theoretical work in GP- and non-GP-based program induction.

2. Summary of Progress

Last year, GPTP 2009 marked a transition wherein the aims of GP algorithms – reasonable *resource* usage, high *results* quality, and *reliable* convergence – were being consistently realized on an impressive variety of "real-world" *applications* by skilled practitioners in the field. This year, for GPTP 2010, researchers have begun to aim for the next level: for *systems* where GP algorithms play a key role. This was evident by the record number of GPTP demos, and by a renewed emphasis on system usability and user control. Also reflecting this transition, discsussions had a marked unity and depth of questions on the philosophy and future of GP, on the need to re-think the algorithms and re-design systems to solve conceptually harder problems.

This chapter is organized accordingly. After a brief introduction to GP, Section 4 describes goals for design of GP algorithms and systems. Then the contributions of this volume (from the workshop) are summarized from two complementary perspectives: section 5 describes the "best practice" techniques that GP practitioners have invented and deployed to achieve the GP algorithm and system aims (including the improvements of GPTP 2010), and section 6 describes the application domains in which success through best practices has been reported. We conclude with a discussion of observations that emerged from the workshop, challenges that remain and potential avenues of future work.

To make the results of the workshop useful to even a relative novice in the field of GP, we first provide a brief overview of GP.

3. A Brief Introduction to Genetic Programming[2]

Genetic programming is a search and optimization technique for executable expressions that is modeled on natural evolution. Natural evolution is a powerful process that can be described by a few central, general mechanisms; for an introduction, see (Futuyma, 2009). A population is composed of organisms which can be distinguished in terms of how fit they are with respect to their environment. Over time, members of the population breed in frequency proportional to their fitness. The new offspring inherit the combined genetic material of their parents with some random variation, and may replace existing members of the population. The entire process is iterative, adaptive and open ended. GP and other evolutionary algorithms typically realize this central description of evolution, albeit in somewhat abstract forms. GP is a set of algorithms that mimic of survival of the fittest, genetic inheritance and variation, and that iterate over a "parent" population, selectively "breeding" them and replacing them with offspring.

Though in general evolution does not have a problem solving goal, GP is nonetheless used to solve problems arising in diverse domains ranging from engineering to art. This is accomplished by casting the organism in the population as a candidate program-like solution to the chosen problem. The organism is represented as a computationally executable expression (aka structure), which is considered its genome. When the expression is executed on some supplied set of inputs, it generates an output (and possibly some intermediate results). This execution behavior is akin to the natural phenotype. By comparing the expression's output to target outputs, a measure of the solution's quality is obtained. This is used as the "fitness" of an expression. The fact that the candidate solutions are computationally executable structures (expressions), not binary or continuous coded values which are elements of a solution, is what distinguishes GP from other evolutionary algorithms (O'Reilly and Angeline, 1997).

GP expressions include LISP functions (Koza, 1992; Wu and Banzhaf, 1998), stack or register based programs (Kantschik and Banzhaf, 2002; Spector and Robinson, 2002), graphs (Miller and Harding, 2008; Mattiussi and Floreano, 2007; Poli, 1997), programs derived from grammars (Gruau, 1993; Whigham, 1995; O'Neill and Ryan, 2003), and generative representations which evolve the grammar itself (Hemberg, 2001; Hornby and Pollack, 2002; O'Reilly and Hemberg, 2007). Key steps in applying GP to a specific problem collectively define its search space: the problem's candidate solutions are designed by choosing a representation; variation operators (mutation and crossover) are selected (or specialized); and a fitness function (objectives and

[2] Adapted from (O'Reilly et al., 2009).

Introduction to *Genetic Programming Theory and Practice VIII*, R. Riolo, T. McConghay, K. Vladislavleva (Eds.)

constraints) which expresses the relative merits of partial and complete solutions is formulated.

For a more detailed overview we refer the reader to the book (Poli et al., 2008), which is available for free online.

4. GP Challenges and Goals

In the early days of GP, the challenge was simply to "make it work" on small problems. As the field of GP research has matured, to be able to solve challenging real-world problems GP experts have strived to improve GP *algorithms* in terms of efficient computational *resource* usage, ensuring better quality *results*, and attaining more *reliable* convergence. With the maturation of "best practice" approaches, researchers are starting to create whole *systems* using GP which present its own challenges: ease of system integration, end-user friendliness, user control of the problem (perhaps interactively). This section elaborates on these GP algorithm and system goals and challenges.

GP Algorithm Goals and Challenges

A successful GP algorithm has at least the following attributes.

Efficent Use of Computational Resources includes shorter runtime, reduced usage of processor(s), and reduced memory and disk usage, for a given result. Achieving efficent use of computer resources has traditionally been a major issue for GP. A key reason is that GP search spaces are astronomically large, multi-modal, epistatic (e.g., variable interactions), have poor locality[3], and other nonlinearities. To handle such challenging search spaces, significant exploration is needed (e.g. large population sizes). This entails intensive processing and memory needs. Exacerbating the problem, fitness evaluations (objectives and constraints) of real-world problems tend to be expensive. Finally, because GP expressions have variable length, there is a tendency for them to "bloat"— to grow rapidly without a corresponding increase in performance (*cf.* Poli's Chapter 5 in this book). Bloat can be a significant drain on available memory and CPU resources.

Ensuring Quality Results. The key question is: "can a GP result *be used* in the target application?" This may be more difficult to attain than evident at first glance because the result may need to be human-interpretable, trustworthy, or predictive on dramatically different inputs— attaining such qualities can be

[3]Poor locality means that a small change in the individual's genotype often leads to large changes in the fitness and introducing additional difficulty into the search effort. For example, the GP "crossover" operation of swapping the subtrees of two parents might change the comparison of two elements from a "less than" relationship to an "equal to" relationship. This usually gives dramatically different behavior and fitness.

challenging. Ensuring quality results has always been perceived as an issue, but the goal is becoming more prominent as GP is being applied to more real world problems. Practitioners, not GP, are responsible for deploying a GP result in their application domain. This means that the practitioner (and potentially their client) must trust the result sufficiently to be comfortable using it. Human-interpretability (readability) of the result is a key factor in trust. This can be an issue when deployment of the result is expensive or risky, when customers' understanding of the solution is crucial; when the result must be inspected or approved; or to gain acceptance of GP methodology.

Reliable convergence means that the GP run can be trusted to return reasonable, useful results, without the practitioner having to worry about premature convergence or whether algorithm parameters like population size were set correctly. GP can fail to capably identify sub-solutions or partially correct solutions and successfully promote, combine and reuse them to generate good solutions with effective structure. The default approach has been to use the largest population size possible, subject to time and resource constraints. This invariably implies high resource usage, and still gives no guarantee of hitting useful results even if such results exist. Alternative approaches to increase the number of iterations with smaller population sizes still lack robust scenarios for computing resource allocation.

Goals for GP Incorporated in larger Systems

These are necessary attributes of GP for successful "GP systems," i.e., systems in which GP plays a key role[4]. A successful GP system must no doubt have many other attributes particular to the context in which it is deployed, but each of the following factors certainly have high impact on the system's success.

Ease of system integration is how easy the GP algorithm is to deploy as part of the entire system, by the person or a team building the system. Even if a GP algorithm does well on the algorithm challenges, its may be hard for system integrators (or other researchers) to deploy because of high complexity or many parameters. Simple algorithms with few parameters are worth striving for; and if this is not possible, then readily available software with simple application programming interfaces and good documentation is a reasonable solution.

End-user friendliness is the end-user's perspective of how easy the system is to use when solving the problem at hand, when GP is only a subcomponent of the overall system. The user wants to solve a problem economically, with

[4]GP may not even be the centerpiece of the system—that's fine!

quality results, reliably. The user task should be smooth and efficient, not tedious and time consuming.

User (Interactive) Control of the Problem. The system (and its subsystems) should not be solving a problem any harder than it needs to be, especially when it makes a qualitative difference to resource usage, result quality, or convergence. To meet this goal, users should be able to specify problems to be solved with as much resolution as appropriate. In some cases, this also means interactivity with results so far, to further guide exploration according to the user's needs, intuitions or subjective tastes. And it specifically does *not* mean user-level control of the GP algorithm itself: the end-user should not have to be a GP expert to use GP to solve a problem, just as GP experts do not have to be experts on electronics in order to use computers.

For more book-length texts on applying GP to industrial problems, we refer the reader to recent books on the subject – by GPTP participants themselves: (Kordon, 2009), (Iba et al., 2010), and (McConaghy et al., 2009).

5. GP Best Practices

First, we describe general best practices that GP practitioners use to achieve GP algorithm goals. Then, we review advances made at GPTP 2010 toward attaining those GP algorithm goals, followed a review of GPTP 2010 work that addresses GP system goals.

In general, GP computational resource use has been made more efficient by improved algorithm design, improved design of representation and operators in specific domains. The importance of high demands of GP for computational resources has been lessened by Moore's Law and increasing availability of parallel computational approaches, meaning that computational resources become exponentially cheaper over time. Results quality has improved for the same reasons. It is also due to a new emphasis by GP practitioners on getting interpretable or trustworthy results. Reliability has been enhanced via algorithm techniques that support continuous evolutionary improvement through a systematic or structured fashion, so that the practitioner no longer has to "hope" that the algorithm isn't stuck. Implicit or explicit diversity maintenance also helps. Finally, thoughtful design of expression representation and genetic operators, for general and specific problem domains, has led to GP systems achieving human-competitive performance. Techniques along these lines include evolvability, self-adaptiveness, modularity and bloat control.

At GPTP 2010, the following papers demonstrated advances in GP algorithm aims (efficient computational resource usage, results quality, or reliable convergence):

- Poli (Chapter 5) draws on recently developed theory to construct a very simple technique that manages bloat.

- Harding *et al.* (Chapter 6) and Spector (Chapter 2) illuminate the state of the art in using self-modifying individuals to achieve highly scalable GP.
- Pattin, Moore *et al.* (Chapter 12) also uses self-adaptation and demonstrates how to incorporate expert knowledge in novel ways, for highly scalable GP.
- Lichodzijewski and Heywood (Chapter 3) and Soule *et al.* (Chapter 4) make further advances in GP scalability through evolution of teams.
- Orlov and Sipper (Chapter 1) is an excellent example of best-practice operator design to maintain evolvability in a highly constrained space.
- Smits *et al.* (Chapter 9) points towards evolution in the "compute cloud," by exploring massively parallel evolution.
- Iba and Aranha (Chapter 13) exploits the structure of the resource-allocation problem in operator and algorithm design to improve GP scalability and results quality.
- Bergen and Ross (Chapter 14) explores how to handle problems with $\gg 2$ objectives yet maintain convergence.
- Korns (Chapter 7) and McConaghy (Chapter 10) aggressively transform and simplify their respective problems for GP as much as possible, to greatly reduce GP resource needs.

At GPTP 2010, the following papers demonstrated advances in GP system goals (system-integrator usability, user-level usability, or user control of the problem and interactivity).

For system integrator usability: Schmidt and Lipson (Chapter 8) shows an approach that achieves the reliable convergence of the popular ALPS algorithm (Hornby, 2006), but with a simpler algorithm having fewer parameters. Harding *et al.* and Spector (Chapter 2) are also examples of relatively simple algorithms, algorithms that have been simplified over the years as their designers gained experience with them. In his keynote address, Jürgen Schmidhuber described the achievement of best-in-class results using simple backpropagation neural networks but with modern computational resources.

For user-level usability: Castillo *et al.* (Chapter 11) prescribes a flow for industrial modeling people where they can use GP as part of their overall *manual* flow in developing trustworthy industrial models. In the special demos session, many researchers presented highly usable GP systems, including Kotanchek's DataModeler (symbolic regression and data analysis package for Mathematica), Schmidt and Lipson's Eureqa (symbolic regression), Bergen and Ross's Jnetic Textures (art), and Iba and Aranha's CACIE (music).

For user control of the problem / interactivity: Korns (Chapter 7) describes an SQL-style language to specify symbolic regression problems, so that function search only changes subsections of the overall expression. Bergen and Ross (Chapter 14) and Iba and Aranha (Chapter 13) describe systems that emphasize usability in interactive design of art and music, respectively.

What is equally significant in these papers is that which is not mentioned or barely mentioned: GP algorithm goals that have already been solved sufficiently for particular problem domains, allowing researchers to focus their work on the more challenging issues. For example, there are several papers that do some form of symbolic regression (SR), which historically has had major issues with interpretability or bloat. Yet in these pages, the SR papers barely discuss interpretability or bloat, because best practices avoid the issue in one or more ways, most notably: pareto optimization using an extra objective of minimizing complexity, templated functional forms like McConaghy's CAFFEINE or Korns' abstract expressions or simply using the GP system to generate promising subexpressions in a manual modeling flow. Other off-the-shelf techniques that solve specific problems well have been around for years and are being increasingly adopted by the GPTP community. These include grammars to restrict program evolution (Whigham, 1995; O'Neill and Ryan, 2003), competent algorithms to handle multiple objectives and/or constraints e.g. (Deb et al., 2002), and meta-algorithms providing diversity and continuous improvement like ALPS (Hornby, 2006). Finally, significant compute resources are available to most: in an informal survey at the workshop, we found that most groups use a compute cluster, and two groups are already using "the compute cloud."

6. Application Successes Via Best Practices

One of the fascinating aspects of GP research is that GP is so general, i.e. "search through a space of (program or structure) entities," that it can be used to attack an enormous variety of problems, including many problems that are currently unapproachable by any other technique. This year's batch of applications is no exception. This section briefly reviews the applications.

One of the long-standing aims of AI, and GP, has been evolution of software in the most general sense possible. GPTP this year was fortunate to have three groups present work directly on this. Orlov and Sipper (Chapter 1) present FINCH, a system to evolve Java bytecode, an evolutionary substrate that has evolvability close to machine code, yet returns interpretable Java code thanks to industry-standard bytecode decompilers. Spector (Chapter 2) presents an autoconstructive version of PUSH, a GP system which evolves stack-based programs. Finally, Harding *et al.* (Chapter 6) presents a self-modifying Cartesian GP which evolves graphs that can be interpreted as software, circuits, equations, and more.

Two chapters introduce wholly new problems for GP. McConaghy (Chapter 10) introduces the problem of building density models at a distribution's tails (and dusts off the general problem of symbolic density modeling), for the application of SRAM memory circuit analysis. Lichodzijewski and Heywood

(Chapter 3) introduce the problem of solving a Rubik's cube with GP, taking the perspective of temporal sequence learning.

GP continues to help the artistic types. Bergen and Ross (Chapter 14) describe a sophisticated interactive system for interactive evolutionary art, and Iba and Aranha (Chapter 13) describe an advanced system for interactive evolutionary music. Both systems have been already used extensively by artists and musicians.

In a biology application, Pattin, Moore *et al.* (Chapter 12) describe the use of GP for disease susceptibility modeling.

GP remains popular in financial applications. Korns (Chapter 7) ups the ante on a set of symbolic regression and classification problems that are representative of financial modeling problems to aid stock-trading decisionmaking. Iba and Aranha (Chapter 13) describes a system for portfolio allocation.

For the problem of industrial modeling (e.g. of inferential sensors at Dow), Castillo *et al.* (Chapter 11) focuses on a structured approach to exploit GP results within industrial modelers' model development flows. Undoubtedly, the symbolic regression approach in Smits *et al.* (Chapter 9) will find end usage in Dow's industrial modeling environment as well.

Other approaches used standard problems in (symbolic) classification or regression as their test suites, though the emphasis was not the application. This includes work by Soule *et al.* (Chapter 4), Poli (Chapter 4), and Schmidt and Lipson (Chapter 8).

7. Themes, Summary and Looking Forward

The toy problems are gone; the GP systems have arrived. No doubt there will continue to be qualitative improvements to GP algorithms and GP systems for years to come. But is there more? We posit there is.

Despite these achievements, GP's computer-based evolution does not demonstrate the potential associated with natural evolution, nor does it always satisfactorily solve important problems we might hope to use it on. Even when using best practice approaches to manage challenges in resources, results, and reliability, the computational load may still be too excessive and the final results may still be inadequate. To achieve success in a difficult problem domain takes a great deal of human effort toward thoughtful design of representations and operators.

Many questions and challenges remain:

- What does it take to make GP a science? (Is this even a realistic question?) How can work on applications facilitate the continued development of a GP theory?
- What does it take to make GP a technology? (Is this even a realistic question?) What fundamental contributions will allow GP to be adopted into broader

use beyond that of expert practitioners? For example, how can GP be scoped so that it becomes another standard, off-the-shelf method in the "toolboxes" of scientists and engineers around the world? Can GP follow in the same vein of linear programming? Can it follow the example of support vector machines and convex optimization methods? One challenge is in formulating the algorithm so that it provides more ease in laying out a problem. Another is determining how, by default – without parameter tuning, GP can efficiently exploit specified resources to return results reliably.

- How do we get 1 million people using GP? 1 billion? (Should they even know they're using GP?)
- Success with GP often requires extensive human effort in capturing and embedding the domain knowledge. How can this up-front human effort be reduced while still achieving excellent results? Are there additional automatic ways to capture domain knowledge for input to GP systems?
- Scalability is always relative. GP has attacked fairly large problems, but how can GP be improved to solve problems that are 10x, 100x, 1,000,000x harder?
- What opportunities await GP due to new computing architectures and substrates, with potentially vastly richer processing resources? This includes massively multicore processors, GPUs, and cloud computing; but it also includes digital microfluidics, modern programmable logic, and more.
- What opportunities await GP due to massive memory and storage capacity, coupled with giant databases? For example, this has already profoundly affected machine learning applied to speech recognition, not to mention web search. Massive and freely available databases are coming online, especially from biology.
- What "uncrackable" problems await a creative GP approach? The future has many challenges in energy, health care, defence, and more. For many fields, there are lists of "holy grail" problems, unsolved problems, even problems with prize money attached.

These questions and their answers will provide the fodder for future GPTP workshops. We wish you many hours of stimulating reading of this volume's contributions.

References

Deb, Kalyanmoy, Pratap, Amrit, Agarwal, Sameer, and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6:182–197.

Futuyma, Douglas (2009). *Evolution, Second Edition*. Sinauer Associates Inc.

Gruau, Frederic (1993). Cellular encoding as a graph grammar. *IEE Colloquium on Grammatical Inference: Theory, Applications and Alternatives*, (Digest No.092):17/1–10.

Hemberg, Martin (2001). GENR8 - A design tool for surface generation. Master's thesis, Department of Physical Resource Theory, Chalmers University, Sweden.

Hornby, Gregory S. (2006). ALPS: the age-layered population structure for reducing the problem of premature convergence. In Keijzer, Maarten, Cattolico, Mike, Arnold, Dirk, Babovic, Vladan, Blum, Christian, Bosman, Peter, Butz, Martin V., Coello Coello, Carlos, Dasgupta, Dipankar, Ficici, Sevan G., Foster, James, Hernandez-Aguirre, Arturo, Hornby, Greg, Lipson, Hod, McMinn, Phil, Moore, Jason, Raidl, Guenther, Rothlauf, Franz, Ryan, Conor, and Thierens, Dirk, editors, *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 1, pages 815–822, Seattle, Washington, USA. ACM Press.

Hornby, Gregory S. and Pollack, Jordan B. (2002). Creating high-level components with a generative representation for body-brain evolution. *Artificial Life*, 8(3):223–246.

Iba, Hitoshi, Paul, Topon Kumar, and Hasegawa, Yoshihiko (2010). *Applied Genetic Programming and Machine Learning*. CRC Press.

Kantschik, Wolfgang and Banzhaf, Wolfgang (2002). Linear-graph GP—A new GP structure. In Foster, James A., Lutton, Evelyne, Miller, Julian, Ryan, Conor, and Tettamanzi, Andrea G. B., editors, *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002*, volume 2278 of *LNCS*, pages 83–92, Kinsale, Ireland. Springer-Verlag.

Kordon, Arthur (2009). *Applying Computational Intelligence: How to Create Value*. Springer.

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

Mattiussi, Claudio and Floreano, Dario (2007). Analog genetic encoding for the evolution of circuits and networks. *IEEE Transactions on Evolutionary Computation*, 11(5):596–607.

McConaghy, Trent, Palmers, Pieter, Gao, Peng, Steyaert, Michiel, and Gielen, Georges G.E. (2009). *Variation-Aware Analog Structural Synthesis: A Computational Intelligence Approach*. Springer.

Miller, Julian Francis and Harding, Simon L. (2008). Cartesian genetic programming. In Ebner, Marc, Cattolico, Mike, van Hemert, Jano, Gustafson, Steven, Merkle, Laurence D., Moore, Frank W., Congdon, Clare Bates, Clack, Christopher D., Moore, Frank W., Rand, William, Ficici, Sevan G., Riolo, Rick, Bacardit, Jaume, Bernado-Mansilla, Ester, Butz, Martin V., Smith, Stephen L., Cagnoni, Stefano, Hauschild, Mark, Pelikan, Martin, and Sastry,

Kumara, editors, *GECCO-2008 tutorials*, pages 2701–2726, Atlanta, GA, USA. ACM.

O'Neill, Michael and Ryan, Conor (2003). *Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language*, volume 4 of *Genetic programming*. Kluwer Academic Publishers.

O'Reilly, Una-May and Angeline, Peter J. (1997). Trends in evolutionary methods for program induction. *Evolutionary Computation*, 5(2):v–ix.

O'Reilly, Una-May and Hemberg, Martin (2007). Integrating generative growth and evolutionary computation for form exploration. *Genetic Programming and Evolvable Machines*, 8(2):163–186. Special issue on developmental systems.

O'Reilly, Una-May, McConaghy, Trent, and Riolo, Rick (2009). GPTP 2009: An example of evolvability. In Riolo, Rick L., O'Reilly, Una-May, and McConaghy, Trent, editors, *Genetic Programming Theory and Practice VII*, Genetic and Evolutionary Computation, chapter 1, pages 1–18. Springer, Ann Arbor.

Poli, Riccardo (1997). Evolution of graph-like programs with parallel distributed genetic programming. In Back, Thomas, editor, *Genetic Algorithms: Proceedings of the Seventh International Conference*, pages 346–353, Michigan State University, East Lansing, MI, USA. Morgan Kaufmann.

Poli, Riccardo, Langdon, William B., and McPhee, Nicholas Freitag (2008). *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`. (With contributions by J. R. Koza).

Spector, Lee and Robinson, Alan (2002). Genetic programming and autoconstructive evolution with the push programming language. *Genetic Programming and Evolvable Machines*, 3(1):7–40.

Whigham, P. A. (1995). Grammatically-based genetic programming. In Rosca, Justinian P., editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41, Tahoe City, California, USA.

Wu, Annie S. and Banzhaf, Wolfgang (1998). Introduction to the special issue: Variable-length representation and noncoding segments for evolutionary algorithms. *Evolutionary Computation*, 6(4):iii–vi.

Index