Genetic Programming Acquires Solutions by Combining Top-Down and Bottom-Up Refinement

Justinian Rosca
Siemens Corporate Research
Princeton, NJ 08540
rosca@scr.siemens.com

1 Solution Acquisition: Classical View

Does Genetic programming (GP) merely guess solutions? We know this is not true, therefore it is important to understand the process of solution refinement in GP. Such an understanding would facilitate the design of better representations and tools to make the evolutionary process more transparent. The classical view, in line with the building block hypothesis in GA/GP, is that GP solutions are refined *bottom-up*, i.e. by starting with small pieces of useful code and then by combining them to compose higher and higher order components. Such pieces of code should be protected from crossover and other disruptive genetic operators [2] or their functionality is lost. For problems where the semantics of the representation facilitates the creation and repeated use of quasi independent components, the bottom-up solution acquisition hypothesis is viable. However, for most problem representations this is not true. They do not satisfy the "independent component" criterion. There is little evidence of what building blocks actually are in this case, and consequently the bottom-up acquisition hypothesis is not the main explanation for GP's success.

The GP representation using Automatically Defined Functions [4] or Adaptive Representation [12] make unambiguous what could possibly be building blocks: subroutines. Subroutines can be hierarchically invoked in order to build solutions and are subject to genetic variation. The bottom-up solution acquisition view appears justified when using subroutines. Subroutines (ADFs) should be *stable* components. [11] presented experimental evidence that they are indeed refined or acquired bottom-up. It considered representations for the parity problem with three components: two subroutines and the main program body. It computed the percentage of genetic changes in a component (e.g. an ADF), taken over all programs in the genealogic tree of a solution created by GP, as a function of generation. The analysis showed that subroutines are learned in the order ADF_0 first, ADF_1 second, etc. Components at the bottom of the hierarchy (such as ADF_0) are learned first, and are subsequently used to define higher level components and so on.

2 New View: Top-Down Refinement

Efforts for understanding solution refinement in GP were reported in [3, 9], who looked at how independent sub-solutions are acquired as a function of their fitness contributions. We are interested in structural biases for solution construction, when it is unknown what constitutes a subsolution.

The insight is offered by rooted-tree schemata, i.e. rooted fragments of parse trees [13]. In contrast to the classical view, the rooted-tree schema theory suggests that solutions are refined *top-down* starting with the root part of the tree and refining the problem decomposition in a top-down manner.

Two rooted-tree schemata are different if they differ in shape or if they have the same shape but at least one different point. Different rooted-tree schemata are independent, i.e. they refer to (syntactically) disjoint regions of the search space (as opposed to GA schemata, which may not be independent even when they are different). Consider the smallest order for which more than two different rooted-tree schemata exist and compete in the population. One rooted-tree schema can overwhelm the competition and will be further specialized downwards (i.e top-down). Independent rooted-tree schemata descendents of the winning schema, having higher order, start to compete amongst themselves. This is a process of *specialization*. All schemata of a given order can be overthrown by a competing schema at a lower order, which appeared by a random genetic change closer to the root of the tree. This can be viewed as a process of *gener-*

alization. The process of specialization will refine this competing schema top-down. Specialization and generalization alternate. However, when one looks at the genealogic tree of a solution one sees a process of top-down refinement, nicely illustrated by *structure-trees* in [12].

The top-down refinement hypothesis is consistent with the observation that crossover acts as a "macromutation." In other words, simply using a mutation operator that replaces a randomly chosen subtree with a new randomly created subtree results in at least similar performance to GP with crossover [1].

Although the top-down and bottom-up views are radically different, structure trees show that they mutually contribute to GP solution refinement. Rooted fragments establish the context for solution refinement in GP. Top-down refinement builds on whatever parts of the solutions are effective at the current stage in fitness computation. Refinement is due to mutation and crossover operations, which can bring to bear random code or pieces of code from other successful programs. The overall process of solution refinement is top-down. A related idea informally states that "two types of building blocks are used to construct solutions" [10, page 284]. These would correspond to rooted-tree fragments and subtree fragments.

Other work can be aligned with the present hypothesis of top-down and bottom-up evolution duality. [6] introduces the idea of solution refinement by "case splitting." [5] discusses the phenomena of specialization and generalization. The idea is further exploited in the design of GP with code duplication by means of architecture altering operations, which enable smoother top-down refinement. [10] argues for solution refinement in two major stages as a basis for their analysis of the process, using one-point crossover. In this vein, while top-down refinement operates at all times during evolution, bottom-up is least effective in the end. [7] presents evidence for the idea of (effective) genome compression. In my view, the compression bias is due to two factors. First, GP search can not acquire solutions bottom-up from some point in time onwards. Second, parents that are subject to top-down refinements and also minimize the ratio of effective code versus total code are likely to have their offspring survive in the population. An explicit top-down refinement strategy is employed in ADATE [8], a radically different but successful program induction technique. ADATE essentially applies iterative deepening exhaustive search in the space of program constructs: it refines solutions in a top-down manner by design.

In conclusion I advocate that GP employs a dual manner solution acquisition, by combining top-down and bottom-up refinement. Bottom-up refinement exploits the functional nature of the GP representation, while top-down refinement is the result of alternative specialization and generalization of partial solutions.

References

[1] Peter J. Angeline. Subtree crossover: Building block engine or macromutation? In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 9–17, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[2] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic programming: An Introduction*. Morgan Kaufmann and dpunkt, 1998.

[3] D. Goldberg and Una-May O'Reilly. Where Does the Good Stuff Go, and Why? How contextual semantics influences program structure in simple genetic programming. In Wolfgang Banzhaf, Riccardo Poli, Marc Schoenauer, and Terence C. Fogarty, editors, *Proceedings of the First European Workshop on Genetic Programming*, volume 1391 of *LNCS*, Paris, 1998. Springer-Verlag.

[4] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[5] John R. Koza. Architecture-altering operations for evolving the architecture of a multi-part program in genetic programming. Computer Science Department STAN-CS-TR-94-1528, Stanford University, 1994.

[6] John R. Koza. *Genetic Programming II*. MIT Press, 1994.

[7] Peter Nordin and Wolfgang Banzhaf. Complexity compression and evolution. *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA95)*, pages 310–317, 1995.

[8] Roland Olsson. Inductive functional programming using incremental program transformation. *Artificial Intelligence*, 74:55–81, 1995.

[9] Una-May O'Reilly and D. Goldberg. How fitness structure affects subsolution acquisition in genetic programming. In John R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. L. Riolo, editors, *Proceedings of the Third Annual Genetic Programming Conference*, Madison, WI, USA, 1998. Morgan Kaufmann.

[10] Ricardo Poli and William B. Langdon. A new schema theory for GP with one-point crossover and point mutation. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 278–285, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[11] Justinian P. Rosca. Genetic programming exploratory power and the discovery of functions. In John Robert McDonnell, Robert G. Reynolds, and David B. Fogel, editors, *Evolutionary Programming IV Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 719–736, San Diego, CA, USA, 1995. MIT Press.

[12] Justinian P. Rosca. *Hierarchical Learning with Procedural Abstraction Mechanisms*. PhD thesis, University of Rochester, Rochester, NY 14627, February 1997.

[13] Justinian P. Rosca and Dana H. Ballard. Rooted-tree schemata in genetic programming. In L. Spector, W.B. Langdon, U. O'Reilly, and P. Angeline, editors, *Advances in Genetic Programming III*, chapter 11. MIT Press, Cambridge, MA, USA, 1999.