Sampling of Unique Structures and Behaviours in Genetic Programming

Steven Gustafson, Edmund K. Burke, and Graham Kendall

School of Computer Science & IT
University of Nottingham, UK
{ smg | ekb | gxk }@cs.nott.ac.uk

Abstract. This paper examines the sampling of unique structures and behaviours in genetic programming. A novel description of behaviour is used to better understand the solutions visited during genetic programming search. Results provide new insight about deception that can be used to improve the algorithm and demonstrate the capability of genetic programming to sample different large tree structures during the evolutionary process.

1 Introduction

Genetic programming searches for solutions to a given objective using program-like representations. However, the task of evolving both a solution's structure and content is complex and difficult to understand [1–3]. Our previous research has focused on understanding the relationship between diversity and search, particularly on the kind and level of diversity that encourages good performance [4, 5]. Recently, we showed how increased diversity negatively and positively effects performance on several problems [6]. A metaphor of hill-climbing search helped explain the results, where deception appeared to be a cause of poor performance. To further understand diversity and search, particularly with respect to deception introduced by the problem and representation, we are interested in the type of structures and behaviours that genetic programming samples during the evolutionary process.

Research into code growth and operator biases can be used to understand the type of structures sampled by genetic programming. Subtree crossover and the representation predispose solutions toward code growth and bloat [7–13]. While programs continue to grow, they tend to grow toward deeper and less-bushier structures. Also, the space of tree shapes visited during genetic programming search has been studied [7, 11, 14, 15], showing that there are types of shapes that are more easily sampled than others. If a problem's solution is not within the more easily sampled structures, the problem will be difficult for genetic programming [16].

With respect to the growth of solutions, the subtree crossover operator is shown to be a more "local" operator, where the upper-portion of trees become fixed and variations mainly occur near the leaves [17–20]. Diversity research also

demonstrates the effects of the convergence of structures [4, 5, 21]. However, it is also the content of trees (functions and terminals) that provide a solution to a given problem. By using stochastic sampling techniques, the proportion of solutions of increasing size was shown not to increase beyond a certain threshold [14, 22]. That is, the space of increasingly larger solutions did not yield a higher proportion of solutions.

As the fitness function is typically a very coarse description of behaviour, it is more difficult to understand the type of solutions sampled by genetic programming. However, comparisons between genetic programming and hill-climbing methods using similar representations and operators can be helpful [22–25]. While genetic programming performed better and worse on various problem domains, comparisons emphasised the domains in which more hill-climbing or explorative search is beneficial. These results were particularly useful in explaining the effects of increased genetic diversity [6]. In any case, much of the solutions' *behaviour* remains hidden behind the fitness function value.

To better understand the sampling of structures and behaviours during search, this paper examines the number of unique structures (genotypes without node content) and behaviours (an enriched definition of fitness) sampled. The structure aspect of the following study provides additional views of the search process, while the coarseness of fitness function values is addressed with problem-specific behaviour descriptions that reflect fitness but elucidate the behaviour of the solutions better.

2 Problems and Methods

We use three problem domains that are frequently used to understand genetic programming: The Artificial Ant problem on the Sante Fe Trail, the Even-5-Parity problem and a regression problem using the Binomial-3 function [1]. Each domain causes genetic programming to behave differently and thus experiments spanning all three domains provide a good basis for understanding. The Ant problem attempts to pick up 89 food pellets on a grid with the functions {if-food-ahead, prog2n} and the terminals {left, right, move}. The Parity problem attempts to classify all 2^5 combinations of 5-bit length strings of {1,0} with the functions {and, or, nand} and five boolean terminals. The Binomial-3 regression problem attempts to approximate the function $f(x) = (1 + x)^3$ using the terminals x, ephemeral random constants in the range of $[-10, 10]$, and the functions $\{+, -, \times, p/\}$, where division is protected and returns 1.0 if the denominator is extremely small. All problems are minimisation of errors; the number of missed food pellets in the ant problem, the mean squared error in regression of the Binomial-3 function and the number of misclassified bit-strings in the parity problem.

In an evolutionary algorithm, a scalar value is typically used to define a solution's behaviour (although multiobjective methods may use a vector). This value, defined by a fitness function, must be able to distinguish different degrees of solution quality. This coarseness often leads to deception, as in the Artificial

Ant problem [22], or fails to identify solutions that are relatively good in the current population but extremely poor to continue a search with, as in the case of very small solutions in regression problems. Thus, measuring the sampling of behaviours during a run using only the fitness value may not be as informative as one would like, and so we define problem specific definitions of behaviour as follows:

- *Ant*: we label each food item uniquely and create a vector representing the order in which the food is collected,
- *Parity*: a vector of integer values, where the size is the total classified correctly and the contents indicate which of the 2^5 test cases were correctly classified,
- *Regression*: a vector of integer values represents the angles between test points (from the horizontal) taken by a solution.

These definitions capture more information than their typical scalar value fitness would, but are still a reduction of the complete behaviour of a solution. The ant behaviour represents the unique sequence of food collection, where the largest sequence is 89 and the smallest is 0. The parity behaviour describes which specific instances are correctly classified. The definition of behaviour for the regression domain describes the change in angle from the horizontal between each function point in the candidate solution. By casting these angles as integers, when two successive angles in the vector are identical, only the first is kept. Thus, the size of the vector alludes to the complexity of the solution (the number of "bends" in the graph of that function), but not directly to the fitness value as the target function is not considered.

In this paper, the tracking of sampled structures is performed by considering the binary tree structures regardless of tree content. We count the number of unique structures of each size that are sampled during the run of the algorithm. Note that there are many unique tree structures of equal size and depth.

A canonical genetic programming system is run for 51 generations, using a generational algorithm with a population size of 500. Initial tree creation is carried out using ramped half-n-half with tree sizes between depths 2 and 4. Subtree crossover, with internal node selection set at 90% probability and maximum depth of 10, is used for recombination – no mutation or duplication is used. There are 30 random runs collected for each problem domain.

The results for each problem domain are depicted in two graphs showing the average number of unique structures sampled and unique behaviours sampled of a given size. Each line represents the cumulative total of unique structures (or behaviours) in each generation during a run. Each successive ten generations are highlighted with symbols. Thus, the space between two lines represents the number of new unique structures (or behaviours) of a given size that were sampled in a generation. The larger the space, the more effort genetic programming spends on searching unique structures (or behaviours) of that size.

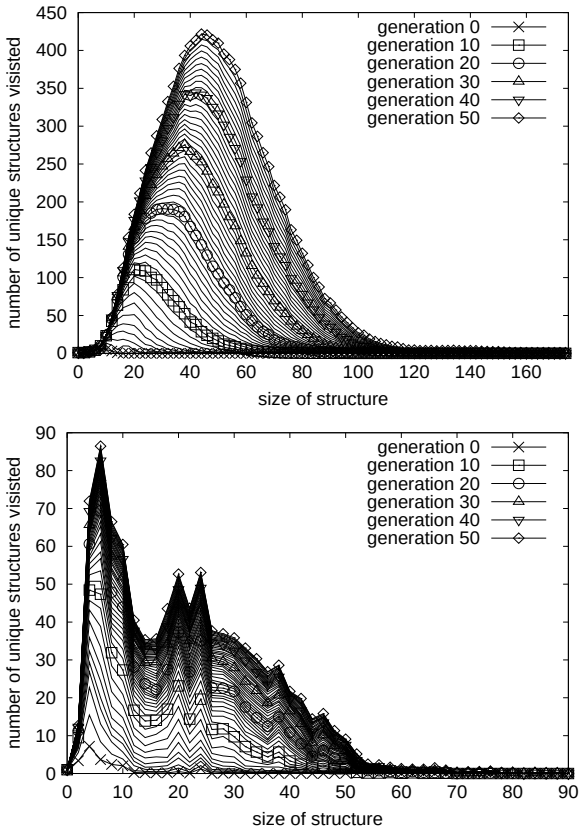

Fig. 1. Ant results, cumulative sampling of unique structures and behaviours.

3 Results

Figure 1 shows the structure and behavioural sampling for the ant problem. There is a distinct trend of the highest number of unique structures sampled toward sizes of 45. The bottom graph in Figure 1 shows the sampled behaviours of each size for the ant problem. Note that the number of unique behaviours sampled of all sizes in each generation is greatly reduced after approximately 10 generations. Also, after two unusual peaks at behaviours of size 20 and 24, the number of unique sampled behaviours greatly decreases for behaviours of large size (which represent more "fit" solutions).

The sampling of structures for the parity problem is shown in the top graph of Figure 2. This problem samples fewer unique structures but at larger sizes. The number of unique behaviours sampled in the parity problem are shown in the bottom graph of Figure 2. A behaviour has a maximum length of 32, which represents all 2^5 correct classifications. However, there are $\binom{32}{k}$ possible unique behaviours for a length of k. For the expected random strategy classification of

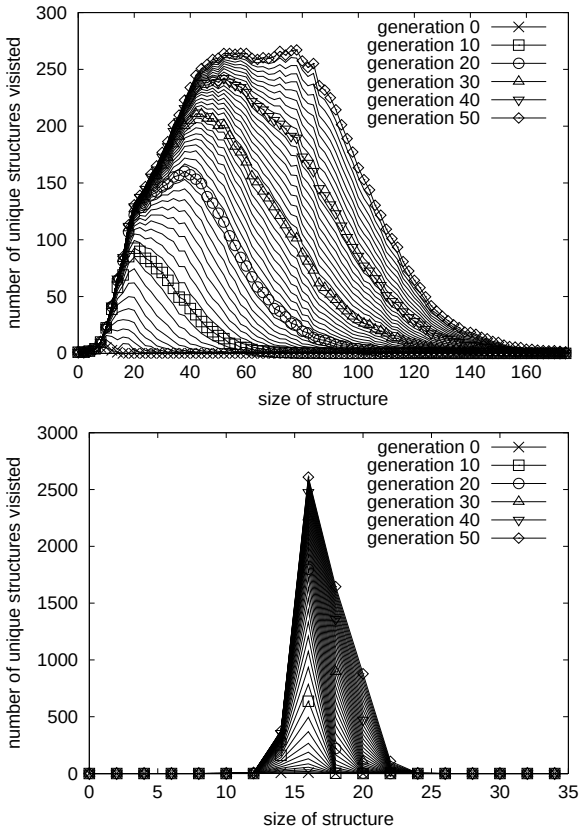

Fig. 2. Parity results, cumulative sampling of unique structures and behaviours.

size 16, nearly 2500 unique behaviours are sampled over the course of a run. Genetic programming spends a large amount of effort searching neutral behaviours equivalent to a random strategy, with slight peaks at fitness 18 and 20. Symmetry in the parity bit-string instances probably rewards the solving of an additional instance with another symmetrical instance also solved correctly, explaining why fitness is concentrated on the random (16) strategy initially, followed by one instance additionally solved (17+1=18) and then another (19+1=20).

The regression problem's sampling of structures is shown in the top graph of Figure 3. A distinct trend is seen toward sampling unique structures of sizes near 40. Fewer unique structures of larger sizes are sampled. The number of unique behaviours sampled in the regression problem, depicted in the bottom graph of Figure 3, shows a strong attraction toward behaviours of size 12. All generations during the run sample unique behaviours of this size. As behaviour does not directly reflect the fitness, these behaviours may or may not have neutral fitness. However, the Binomial-3 fitness function contains 50 equidistant x values that

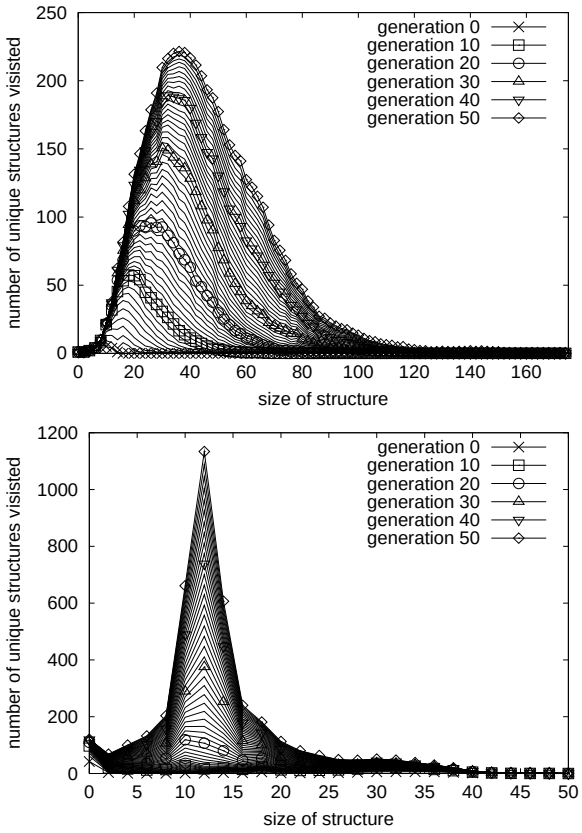

Fig. 3. Regression results, cumulative sampling of unique structures and behaviours.

generate the target y values for testing an individual. The angle gradient between successive y values is nearly always greater than 1. Thus, for our behaviour measure, if a behaviour is to represent the function ideally, it will need close to 50 angle changes between points.

The 95% confidence bars for each of the average distributions from Figures 1 to 3 are shown in Figure 4. From left to right, the ant, parity and regression structures are shown on the top row, and behaviours on the bottom row. The sampling between the 30 runs is fairly uniform with the greatest variations occurring near the peaks in the regression problem.

4 Discussion

How does one search for computer programs, and how do we know if one program is better than the other? The intuitive broad sampling of a complex solution space by a population in genetic programming is one reason why the

algorithm may be considered useful. Early work by Cramer [26] and Koza [27] laid the foundations for this procedural representation, where the use of single-value scalars as fitness combined and complex representations were already in use in other evolutionary algorithm domains. While it may have been straightforward to apply these methods to program evolution for genetic programming, the conflicts arising in this representation are well documented [1, 2, 16, 28]. The results presented here, based on the sampling of unique tree structures and unique behaviours, further describe genetic programming's ability to sample a complex solution space.

The behaviour definition used here enriched the description of the sampling of solutions by genetic programming. In the ant problem, the geographic distribution of food pellets on the ant trail may allow for many ant behaviours that are able to collect 20 to 24 pellets, but unable to easily collect more. In this problem, the standard definition of fitness creates deception [22] that has already been shown to negatively impair fitness improvement. However, the more gradual decrease in the ability to sample different behaviours of higher fitness explains why genetic programming often outperforms hill-climbing methods that hill-climb with poor solutions. The peaks of behaviours sampled at size 20 and 24 may represent local optima that trap search. The behaviour distribution for the parity problem is particularly interesting as it shows the ability of genetic programming to sample may different near-random type behaviours. Sampling such high numbers of different behaviours is promising but a likely cause of deception and negative contextual shifts of subtrees, explaining why hill-climbing and elitist strategies are more effective on this problem [23, 25].

While the regression problem used a behaviour that did not directly reflect its fitness value, the behaviour did reflect the complexity of solutions. The prefer-

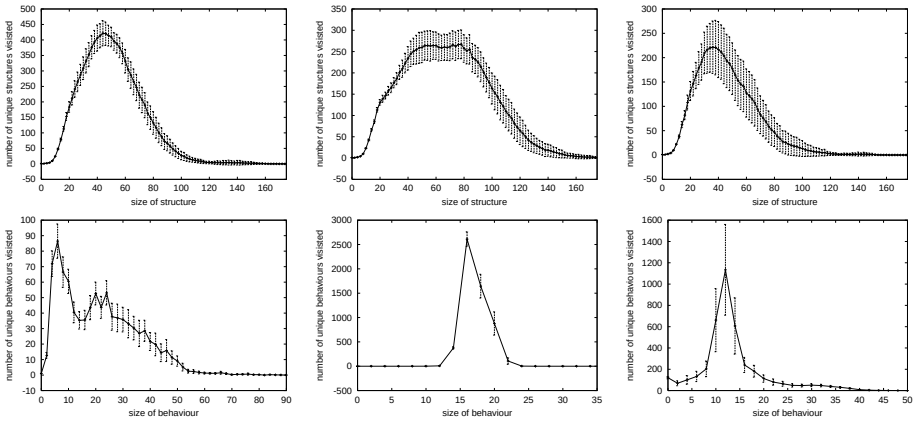

Fig. 4. 95% confidence bars for the average cumulative structure and behaviour sampling distributions. From left to right is the ant, parity and regression problems with structure on top and behaviour below.

ence toward sampling behaviours of a small size and complexity could be caused by neutrality in the fitness values or the result of a biased structure sampling. In the latter case, the inability to increase structure size is hypothesized to be positively correlated with the inability to increase the complexity of solutions. This hypothesis seems plausible for the regression domain, considering that subtree crossover typically functions near the leaves (terminals).

With respect to the sampling of unique structures of various sizes, all problems showed that while genetic programming is predisposed to code growth (and bloat), unique tree structures of large sizes are sampled less. That is, a fewer number of big tree structures are sampled. The different structures of the same size that are sampled more frequently would appear to be attractive to genetic programming for one reason or another. The depth limit imposed here is one possible cause.

These results also suggest several possibilities to improve performance. The sizes of behaviours that were sampled in high numbers may represent areas of deception in the fitness space. This deception may be reduced by pressuring the search away from areas of neutral fitness. Previous research showing a positive correlation between high fitness-based diversity with good fitness could be the result of avoided deception leading to better fitness. Thus, when a population contains many different but equal in fitness behaviours, we may wish to move the search toward a less deceptive region of the search space. Leading the ant to follow a specific food trail reduced deception and improved fitness [22]. For the regression domain, using a component of complexity in the fitness function may also improve the performance of genetic programming. Similarly, the parity fitness function could be amended to reflect which instances a solution solves correctly to reduce deception. Of course, exploration (global search) and exploitation (local search) ability will be affected when such methods are employed to reduce deception.

5 Conclusions

This paper examined the sampling of unique tree structures and unique behaviours in genetic programming. An enriched definition of behaviour provided more information about solutions than typical fitness functions. As the genetic programming algorithm requires the search of structure and content, it is important to understand issues such as deception and the effort the algorithm spends on searching different types of behaviours and structures. The behaviour sampling results showed sampling trends that help explain previous diversity research and suggests new ways to improve search. The structure sampling results showed that while bloat and code growth occur, fewer different unique structures of these large sizes are sampled. Problems which require specific structures at these large sizes are likely to be more difficult for genetic programming.

Acknowledgments: The authors would like to thank the anonymous reviewers, Natalio Krasnogor and Anikó Ekárt.

References

1. J.M. Daida, R.R. Bertram, S.A. Stanhope, J.C. Khoo, S.A. Chaudhary, O.A. Chaudhri, and J.A. Polito II. What makes a problem GP-hard? analysis of a tunably difficult problem in genetic programming. *Genetic Programming and Evolvable Machines*, 2(2):165–191, June 2001.
2. U.-M. O'Reilly. The impact of external dependency in genetic programming primitives. In *Proceedings of the 1998 IEEE World Congress on Computational Intelligence*, pages 306–311, Anchorage, Alaska, USA, 5-9 May 1998. IEEE Press.
3. J. Hu, K. Seo, S. Li, Z. Fan, R.C. Rosenberg, and E.D. Goodman. Structure fitness sharing (SFS) for evolutionary design by genetic programming. In W.B. Langdon et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 780–787, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
4. E. Burke, S. Gustafson, and G. Kendall. A survey and analysis of diversity measures in genetic programming. In W. B. Langdon et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 716–723, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
5. E. Burke, S. Gustafson, G. Kendall, and N. Krasnogor. Advanced population diversity measures in genetic programming. In J.J. Merelo Guervós et al., editors, *Parallel Problem Solving from Nature*, volume 2439 of *LNCS*, pages 341–350, Granada, Spain, September 2002. Springer.
6. E.K. Burke, S. Gustafson, G. Kendall, and N. Krasnogor. Is increasing diversity in genetic programming beneficial? an analysis of the effects on fitness. In B. McKay et al., editors, *Congress on Evolutionary Computation*, pages 1398–1405, Canberra, Australia, December 2003. IEEE Press.
7. T. Soule and J.A. Foster. Code size and depth flows in genetic programming. In J.R. Koza et al., editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 313–320, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.
8. W.B. Langdon, T. Soule, R. Poli, and J.A. Foster. The evolution of size and shape. In Lee Spector et al., editors, *Advances in Genetic Programming 3*, chapter 8, pages 163–190. MIT Press, Cambridge, MA, USA, June 1999.
9. W.B. Langdon. The evolution of size in variable length representations. In *1998 IEEE International Conference on Evolutionary Computation*, pages 633–638, Anchorage, Alaska, USA, 5-9 May 1998. IEEE Press.
10. W. Banzhaf and W. B. Langdon. Some considerations on the reason for bloat. *Genetic Programming and Evolvable Machines*, 3(1):81–91, March 2002.
11. W.B. Langdon. Quadratic bloat in genetic programming. In D. Whitley et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 451–458, Las Vegas, Nevada, USA, 10-12 July 2000. Morgan Kaufmann.
12. T. Soule and R.B. Heckendorn. An analysis of the causes of code growth in genetic programming. *Genetic Programming and Evolvable Machines*, 3(3):283–309, Sept. 2002.
13. S. Luke. Modification point depth and genome growth in genetic programming. *Evolutionary Computation*, 11(1):67–106, 2003.

14. W.B. Langdon. Scaling of program fitness spaces. *Evolutionary Computation*, 7(4):399–428, 1999.
15. J.M. Daida. Limits to expression in genetic programming: Lattice-aggregate modeling. In D.B. Fogel et al., editors, *Congress on Evolutionary Computation*, pages 273–278, Honolulu, USA, 2002. IEEE Press.
16. J.M. Daida, H. Li, R. Tang, and A.M. Hilss. What makes a problem GP-hard? validating a hypothesis of structural causes. In E. Cantú-Paz et al., editors, *Proceedings of the Genetic and Evolutionary Computation*, volume 2724 of *LNCS*, pages 1665–1677, Chicago, 12-16 July 2003. Springer-Verlag.
17. J. Rosca and D.H. Ballard. Causality in genetic programming. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 256–263, Pittsburgh, PA, USA, 15-19 1995. Morgan Kaufmann.
18. C. Igel and K. Chellapilla. Investigating the influence of depth and degree of genotypic change on fitness in genetic programming. In W. Banzhaf et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1061–1068, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.
19. R. Poli and W.B. Langdon. On the search properties of different crossover operators in genetic programming. In J.R. Koza et al., editors, *Proceedings of the Third Annual Genetic Programming Conference*, pages 293–301, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.
20. P. D'haeseleer and J. Bluming. Effects of locality in individual and population evolution. In K.E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 8, pages 177–198. MIT Press, 1994.
21. N.F. McPhee and N.J. Hopper. Analysis of genetic diversity through population history. In W. Banzhaf et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1112–1120, Florida, USA, 1999. Morgan Kaufmann.
22. W.B. Langdon and R. Poli. *Foundations of Genetic Programming*. Springer-Verlag, 2002.
23. U.-M. O'Reilly and F. Oppacher. Program search with a hierarchical variable length representation: Genetic programming, simulated annealing and hill climbing. In Y. Davidor et al., editors, *Parallel Problem Solving from Nature*, number 866 in LNCS, pages 397–406, Jerusalem, 9-14 October 1994. Springer-Verlag.
24. U.-M. O'Reilly and F. Oppacher. A comparative analysis of GP. In P.J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 2, pages 23–44. MIT Press, Cambridge, MA, USA, 1996.
25. A. Juels and M. Wattenberg. Stochastic hillclimbing as a baseline method for evaluating genetic algorithms. Technical Report Technical Report CSD-94-834. Computers Science Department, University of California at Berkeley, USA, 1995.
26. N.L. Cramer. A representation for the adaptive generation of simple sequential programs. In J.J. Grefenstette, editor, *Proceedings of an International Conference on Genetic Algorithms and the Applications*, pages 183–187, Carnegie-Mellon University, Pittsburgh, PA, USA, 24-26 July 1985.
27. J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.
28. U.-M. O'Reilly and D.E. Goldberg. How fitness structure affects subsolution acquisition in genetic programming. In J.R. Koza et al., editors, *Proceedings of the Third Annual Genetic Programming Conference*, pages 269–277, Madison, WI, USA, 22-25 July 1998. Morgan Kaufmann.