Characterizing the Dynamics of Symmetry Breaking in Genetic Programming

Jason M. Daida

The University of Michigan
2455 Hayward Avenue
Ann Arbor, Michigan USA 48109-2143

daida@umich.edu

ABSTRACT

This paper introduces a metric that measures symmetry in tree graphs, which allows for a statistical characterization of GP solutions by their architectural "shapes." A case study is given that applies this metric to 80.4 million trees to identify trends in GP runs. Results provide a first quantitative look at the dynamics of symmetry breaking.

Categories and Subject Descriptors

I.2.2 [**Artificial Intelligence**]: Automatic Programming – *program synthesis*; D.2.11 [**Software Engineering**]: Software Architectures – *patterns*; F.m [**Theory of Computation**]: Miscellaneous.

General Terms

Algorithms, Measurement, Experimentation, Theory, Languages

Keywords

Symmetry breaking, design patterns, graphics techniques, tree, data structures, computational geometry, analysis methods.

1. INTRODUCTION

A quantitative understanding has been emerging about the dynamics that occur as genetic programming derives a solution. For example, analyses of how GP assembles solutions from building blocks have been initially described in [24], with subsequent analyses in [2, 33, 34, 36]. Of building blocks, those that include the root node of a tree have been studied in detail for a variety of case studies [18, 29, 35]. Steps toward quantifying the dynamics of how content changes within a population over time (i.e., over the course of a GP trial) have also been studied in [5, 14, 32].

Of such investigations, the quantitative characterization of changes to the architectures of its solutions has received a fair amount of attention. For example, Koza [25] measured and described the emergence of solution architectures to suit a particular problem. Related to the concept of architecture is a solution's shape, of which a substantial amount of work has been done concerning the size and depth of solution trees [13, 25-27, 38]. Langdon [28] made detailed observations concerning the dynamics of changes to a solution's size and depth, as well as characterized the mean growth trends of solutions.

Unfortunately, size and depth can only go so far in describing the changes to architectures associated with GP solutions. Subsequently, this paper describes the derivation of a metric that can characterize the symmetry breaking associated with architecture that may occur over the course of a GP run.

In particular, Section 2 reviews work in computer science that discusses patterns and symmetry in software architectures, which is relevant but is not commonly used in the analysis of GP dynamics. Section 3 describes qualitative observations of structural patterns that occur during a GP run. Section 4 introduces a statistical metric that can characterize the structural changes that were observed. Sections 5 and 6 then cover the application of this metric to a case study that strongly indicates the existence of phases and transition regions involving the evolution of structure within a GP trial. Section 7 concludes.

2. ARCHITECTURES AND PATTERNS

Software *architecture* can be defined as the "conceptual structure and logical organization" of computer programs (Oxford American). The concept of software architecture is mainstream and appropriate for describing large complex programs, such as those created under object-oriented programming. By itself, the term does not convey any additional insight towards an analysis of GP dynamics, particularly since GP solutions are typically orders of magnitude smaller and less complex than the kinds of programs that a developer team would produce.

However by the late-1980s, the object-oriented programming community was looking for broader methodologies that could assist in the design of large, complex programs. One of the results of that search was a co-opting of design methodologies from architecture (as in buildings and urban planning). The particular set of methodologies that were adopted is now associated with the term *design patterns*, which is based on work initiated in the 1960s by architect Charles Alexander (see [3, 7, 8]), and popularized in the computer science community by [17]. A design pattern is "a general repeatable solution to a commonly-occurring problem in software design. A design pattern isn't a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations. [41]"

GECCO 2006, July 8–12, 2006, Seattle, Washington, USA.

Of interest to those in the genetic programming community are the "other" GPs that have become associated with design patterns: *generic* programming and *generative* programming. (See [1, 4].) *Generic* programming is a style of programming that uses generic (polymorphic) datatypes and is supported by languages like C++. *Generative* programming is a style of programming that combines automated code generation (typically using generic-style code) and human code generation. (See [41]). These methods are common and mainstream in large-scale software development (e.g., Visual Studio support of .NET).

Design patterns have been introduced into the GP communities on a number of occasions, although more in the context of extending GP than in analyzing it. (e.g., [16, 30]). There are also works that strongly complement many of the principles of design patterns, but that have been developed more or less independently from the work in object-oriented design (e.g., [20, 21]). That there may be many areas of complementary development in design patterns might not be surprising, since a big picture view of pattern design does share many (independent) commonalities with the big picture view of GP as a design-engine (c.f., [11]).

Of particular relevance to this paper are the attempts in pattern design to formalize the concept of *pattern* [9, 10]. The particular formalism that Coplien has chosen to quantify patterns is largely a geometric one—symmetry. "Good" software architectures have some amount of symmetry breaking; too much symmetry is typically reflective of poor architectures.

The idea of symmetry breaking as a desirable characteristic also has some precedence in the GP community as well. One of the GEC principals, John Holland, discusses the necessity of symmetry breaking in [19] as one of the essential attributes of a complex adaptive system. Streeter, Keane, and Koza also mention it in their works (such as [39]).

Neither in design patterns nor in GP are the concepts of symmetry and symmetry breaking formalized to the degree that a quantitative analysis could be conducted. Unlike GP, there has been a growing and significant body of work in design patterns that capture best practices of human developers. The case for the significance of symmetry breaking to understanding software architectures should be understood in that light.

What GP has to offer is a system that can take random programs with presumably poor architectures and transform them into a solution that works. True, the solutions that GP creates are not of the size or complexity of programs that are considered at the level of design patterns. However, any formalism that purports to analyze symmetry breaking for software architectures should be able to show symmetry breaking in the evolution of simple programs (like those produced by GP).

3. OBSERVATIONS (SINGLE TRIAL)

In spite of not having a formal means of measuring symmetry in either design patterns or GP, there have been recent developments that have indicated that it might be possible to do so. Specifically, the method [15] for visualizing whole populations of trees in GP does suggest such a possibility.

Figure 1 shows the structural evolution of a GP population for a single trial for an example problem. This figure is a reprint of an example that appeared in [15] (i.e., Figure 14 in that citation).

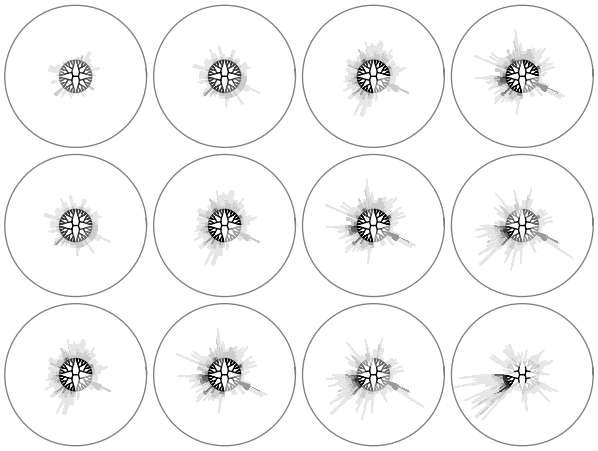

Figure 1. An example of how population structure changes over time. Shown are 12 samples from the first 23 generations of a run.

In this figure, each circular graphic represents a summary of population structure for one generation. Each summary can be thought as a composite view of many individual trees that overlay each other. Each tree (binary in this case) is stretched out on a circular lattice, where the center corresponds to the root node of a tree. Consequently, if many trees occupy the same position in this lattice, the composite view of all of them is dark. Likewise, if only a few trees occupy the same position in this lattice, the composite view of all of them is light gray.

The 12 circular graphics correspond to 12 time samples of a GP run. This particular run was sampled at every-other generation, starting from generation 1, which appears in the upper left corner. (Generation 0 represents the initial population.) The sequence progresses from left to right for each row starting from the top. The first row of graphs corresponds to samples at generation 1, 3, 5, and 7; the last row, to samples at generations 17, 19, 21, and 23.

There are at least two observations that one can make:

- The overall structure of a random population corresponds to a graphic that is radially symmetric. Intuitively, this would make sense for a ramped half-and-half method of creating an initial population. Although the structure of an initial population *individual* might not be symmetrical, the structure of an *aggregate* of randomly generated individuals turns out to be symmetrical.
- As time progresses, population structure transitions from a radially symmetric one to an asymmetrical one. The difference in population structure between generation 1 (upper-left corner) and generation 23 (lower-right corner) is pronounced. It is also in keeping with what one would expect if symmetry breaking happens in GP.

At least in this case, the visual representation of symmetry breaking is clear and intuitive. It is insufficient as proof of symmetry breaking.

Somehow, there needs to be a way to summarize the amount of symmetry that the graphics convey, so that a quantitative comparison can be made not just from time sample to time sample, but also from run to run. The difficulty, as [15] shows, is that GP can evolve structural asymmetries that look nothing like each other from run to run.

Subsequently, the following section describes the derivation of metrics that would allow for such comparisons.

4. MEASURING SYMMETRY

Although Figure 1 and other examples in [15] show a transition from symmetry to asymmetry during a GP run, measuring this transition has required the development of new metrics. As it turns out, much of the work in the measurement of shape and symmetry in graphs is not well suited for this application.

4.1 Previous Work

Part of the difficulty in identifying a metric for symmetry has to do with the degree of abstraction needed to do what is essentially a comparison of shapes. Unfortunately, as Veltkamp states [40], "There is no universal definition of shape." For example, given the graph structures shown in Figures 1 and 2, shape can be defined by a hull that encloses all the leaves of a tree graph. A hull can alternately be defined by enclosing those lattice points that have a specified cumulative density. Furthermore, one is not restricted just to hulls. One can count the number of dendrites away from the center to define a shape. Nuances that involve dendrites would include defining what exactly is a dendrite that contributes to an overall shape, since technically all of the structures in those graphs are dendrites.

Veltkamp surveys a broad variety of methods used in the comparison of shapes with graphs [40]. Although none specifically apply to the problem of tracking symmetry changes, the work discusses the nuances and challenges involved in comparing shapes.

A sampling of current work that is relevant would include [6, 22, 23, 31, 37]. In general, such methods are specific to a particular application (e.g., object tracking in scenes). Methods such as those by [6] look promising, but the class of trees considered (i.e., unrooted) overly complicate their use for this work's application.

4.2 Derivation of Metrics

This paper proposes a metric that is analogous to computing for a center of mass given a particular graph (like those shown in Figure 1). In this analogy, each node in a graph is treated as a point mass. The amount of mass associated with a point is a function of the cumulative distribution of the number of trees in a population that have a vertex. To clarify what is meant by this analogy, it would help to review how a graph of population structure is constructed according to [15].

A graph of population structure is a superposition of all the graphs that correspond to individuals in a population. It is presumed (both in this paper and in [15]) that an individual can be described by a plane binary tree. An arbitrary binary tree A can be mapped to a circular lattice L_C by showing that a set of labels k corresponding to A is a subset of L_C. To do so, one can traverse A in preorder and label each node visited in the following manner:

- The root node of tree A is designated as $k = 1$ and at depth level 0.
- The root node of the left subtree is labeled $2l$, where l is the label of that node's parent.
- The root node of the right subtree is labeled $2l+1$, where l is the label of that node's parent.

A node label maps to a position in a circular lattice L_C by the following two polar coordinate equations for radius and angle, respectively: $\rho(k)$ and $\phi(k)$.

The mapping of node label k to a ring radius r is specified as $\rho(k)$: $k \rightarrow r$, where $k \in \Im^+$, $r \in \Im^+$, and

$$\rho(k) = \begin{cases} 0, & k = 1. \\ \lfloor \log_2 k \rfloor & k > 1. \end{cases} \quad (1)$$

The mapping of node label k to an angular position θ is specified as $\phi(k)$: $k \rightarrow \theta$, such that $k \in \Im^+$, $r \in \Re$, $\theta \in \Re$, and

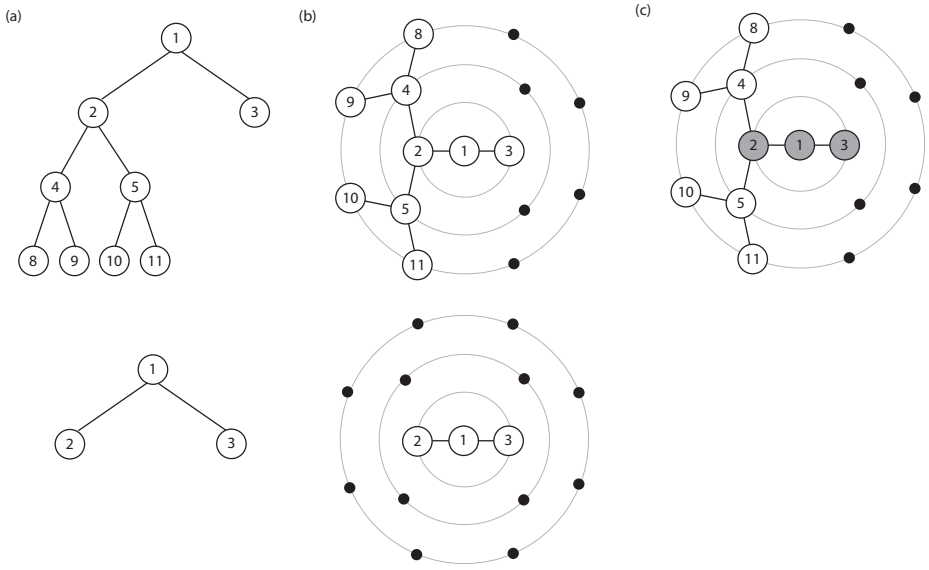

Figure 2. Example of constructing a population graph to compute for $r_{symmetry}$. (a) Two individuals with labeled nodes. (b) Same two individuals placed on a circular lattice. (c) Population graph from which $r_{symmetry}$ is computed.

$$\phi(k) = \begin{cases} 0, & k = 1. \\ \pi\left(\frac{1}{2} + \frac{1}{2^{\rho(k)}} + \frac{k \bmod 2^{\rho(k)}}{2^{\rho(k)-1}}\right), & k > 1. \end{cases} \quad (2)$$

The superposition of graphs of individual trees can be described by computing the cumulative distribution on L_C for an entire population, i.e.,

$$\mathbf{L}_P = \sum_{\forall A \in P} \mathbf{L}_A, \quad (3)$$

where A is a tree in a population P, $L_P \supset L_C$ and $\mathbf{L}_i$ is a vector corresponding to L_i such that

$$\mathbf{L}_i \equiv \sum_{\forall a \in L_i} \mathbf{i}_a. \quad (4)$$

Note that $\mathbf{i}_a$ specifies a unit component vector and that a is a label in L_i.

Given a graph that is constructed in this way, one can compute a quantity that is analogous to the center of mass if we treat each node in $\mathbf{L}_P$ as a point mass. In particular, the x- and y-components of the centroid analog are defined as

$$\bar{x} \equiv \frac{1}{M} \sum_{\forall j \in P} m_j r_j \cos\theta_j, \quad (5)$$

$$\bar{y} \equiv \frac{1}{M} \sum_{\forall i \in P} m_j r_j \sin\theta_j, \quad (6)$$

where r_i and θ_i are the polar coordinates that correspond to a node i in $\mathbf{L}_P$; m_i denotes the cumulative distribution of $\mathbf{L}_P$ at that particular node; and

$$M \equiv \sum_{\forall i \in P} m_j. \quad (7)$$

Given Equations 6 and 7, a metric that measures for symmetry would correspond to the radial component that is associated with those components. In other words,

$$r_{symmetry} \equiv \sqrt{\bar{x}^2 + \bar{y}^2}. \quad (8)$$

Using $r_{symmetry}$ as a measure of symmetry, values that are closer to zero would indicate a population structure that is more radially symmetric. (Note: This is not a *perfect* measure of symmetry, since it is possible to have asymmetric shapes that have a null value for $r_{symmetry}$. That being said, $r_{symmetry}$ is intended for use as a statistical measure, and the random growth of trees makes unlikely that such asymmetric shapes are common.)

Of interest, however, is symmetry breaking, which implies a quantity that varies as a function over time. Furthermore, it is also recognized that more asymmetry is not in and of itself an indicator of better performance. For example, asymmetry in the latter part of a run may be driven by code growth that has little to do with performance. Symmetry breaking would constitute the transition from symmetry to asymmetry, which can be captured by the derivative of $r_{symmetry}$ as a function of time. Since "time" in a GP system is discrete (either in steady-state or in generational modes), this derivative can be approximated by the following difference equation:

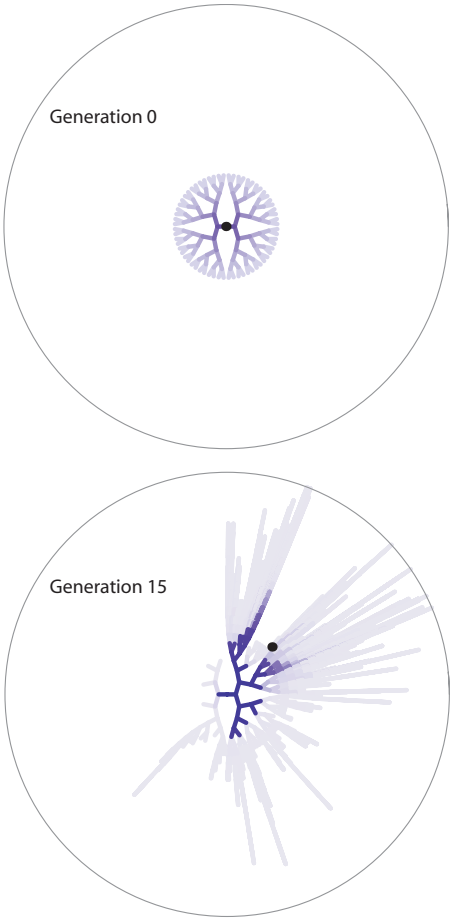

Figure 3. Example of a centroid calculation for two different generations. Centroids are depicted as large black dots. Each population graph shows the structure for 500 individuals, where darker lines indicate more individuals that had the same structure.

$$\Delta r_{symmetry} \equiv \sqrt{(\bar{x}_t)^2 + (\bar{y}_t)^2} - \sqrt{(\bar{x}_{t-1})^2 + (\bar{y}_{t-1})^2}, \quad (9)$$

where t denotes a time during a GP run. Consequently, if this were a generational system, time t would denote a generation.

4.3 Examples

This section offers three different examples of working with the Equations 1 – 7.

The first example gives an example of computing $r_{symmetry}$ for a population of two individuals, which is depicted in Figure 2a. Each node in a tree is labeled using the conventions for determining k at the beginning of Section 4.2. Each tree is then

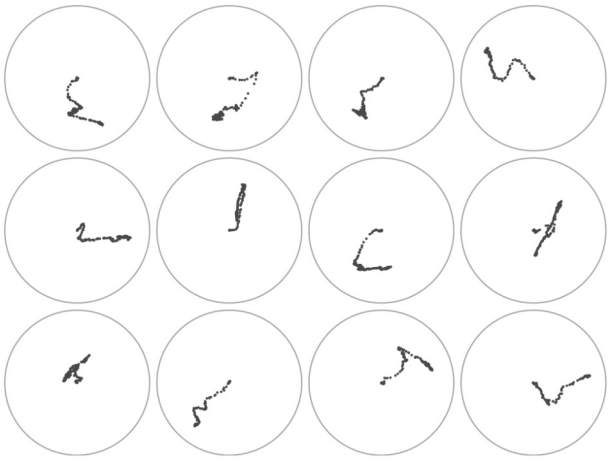

Figure 4. Path of centroid for 12 different runs. As in Figures 1 and 3, a gray circle that corresponds to depth 26 is shown for reference. Each plot shows 200 dots, with each dot depicting the location of the centroid for a given generation.

registered to a circular lattice L_C (as shown in Figure 2b). The population graph, which is shown in Figure 2c, is the superposition of the graphs shown in Figure 2b. In this case, the dark-gray nodes in Figure 2c each have a value of 2, while the white nodes have a value of 1. Consequently, the *x*-component of $r_{symmetry}$ computes to

$$\bar{x} \equiv \frac{1}{12}\left(0+(2\cdot 1)\cos 0+(2\cdot 1)\cos\pi+(1\cdot 2)\cos\frac{3\pi}{4}+(1\cdot 2)\cos\frac{5\pi}{4}+(1\cdot 3)\cos\frac{5\pi}{8}+(1\cdot 3)\cos\frac{7\pi}{8}+(1\cdot 3)\cos\frac{9\pi}{8}+(1\cdot 3)\cos\frac{11\pi}{8}\right),$$

which is approximately equal to -0.88. Likewise, the *y*-component calculates to 0. Therefore for this example, $r_{symmetry} = 0.88$.

The second example, which is shown in Figure 3, illustrates the location of the centroid analog for two different time samples for a given run in GP. The analog is overlain on its corresponding population graph and is indicated by a large black dot. At generation 0, the centroid analog started off in the center of its corresponding graph, which is not surprising because the graph is radially symmetric. As time progressed, the centroid drifted farther away from the center, which corresponds to the asymmetry of the population graph for generation 15.

The third example, which is shown in Figure 4, illustrates the location of the centroid analog for 12 different runs. For each of these runs, the population graph is removed and only the location of the centroid is indicated with a dot. Each plot in Figure 5 shows the path that the centroid analog took for 200 generations.

5. CASE STUDY

The utility of a metric like $r_{symmetry}$ lies in its ability to condense significant amounts of data so that large-scale trends can be identified. The following case study is an extended example of how $r_{symmetry}$ would apply to a computational experiment to identify trends in symmetry breaking.

5.1 Procedure

A well-documented, tunably-difficult test problem was used (i.e., *binomial-3*). The problem has been designed as a probe for understanding GP dynamics and is representative of the kinds of problems found in data modeling. Consequently, the procedure for this case study is identical to that used in [14] to facilitate comparison in future work.

In brief, the problem is an instance taken from symbolic regression and involves solving for the function $f(x) = 1 + 3x + 3x^2 + x^3$. Fitness cases are 50 equidistant points generated from $f(x)$ over the interval [-1, 0). The function set is {+, –, ×, ÷}, which corresponds to arithmetic operators of addition, subtraction, multiplication, and protected division. The terminal set is {**x**, **R**}, where **x** is the symbolic variable and **R** is the set of random constants that are distributed uniformly over the interval $[-\alpha, \alpha]$. The tuning parameter is α, which is a real number that controls problem difficulty. The *binomial-3* can be tuned from a relatively easy problem to a difficult one by adjusting the range over which these random constants occur. In general, values of α that are farther from unity result in settings that increase the difficulty for GP to solve this problem.

Table 1 lists the parameter settings considered in this paper. Most of the GP parameters were similar to those mentioned in [24], Chapter 7.

Four different experimental configurations were used, given two different selection methods and two different difficulty settings. These settings were chosen because they bracket the conditions under which GP finds this problem either "hard" or "easy." Although the difference in settings seems fairly innocuous, the difference in the likelihood that GP would identify a successful solution was chosen to be unambiguous (i.e., GP would likely be able to identify successful solutions twice as many times under "easy" conditions than under "hard" ones). There were 200 runs taken per configuration for a total of 800 runs.

Table 1. Parameter settings

Parameter	Setting
Selection	Tournament *m*=7 or Proportionate
Population Size *M*	500
Initialization Method	Ramped Half-and-Half
Initialization Depths	2–6 Levels
Max Generations *G*	200
Maximum Depth	26
Internal Node Bias	90% internal, 10% terminals
Termination Criteria	Run reaches *G*
Binomial-3 α	1 or 1000
Number of Runs	200

5.2 Results

Figure 5 shows the first set of results as a six-variable visualization for all 800 runs. It summarizes the measurement of symmetry for approximately 80.4 million trees.

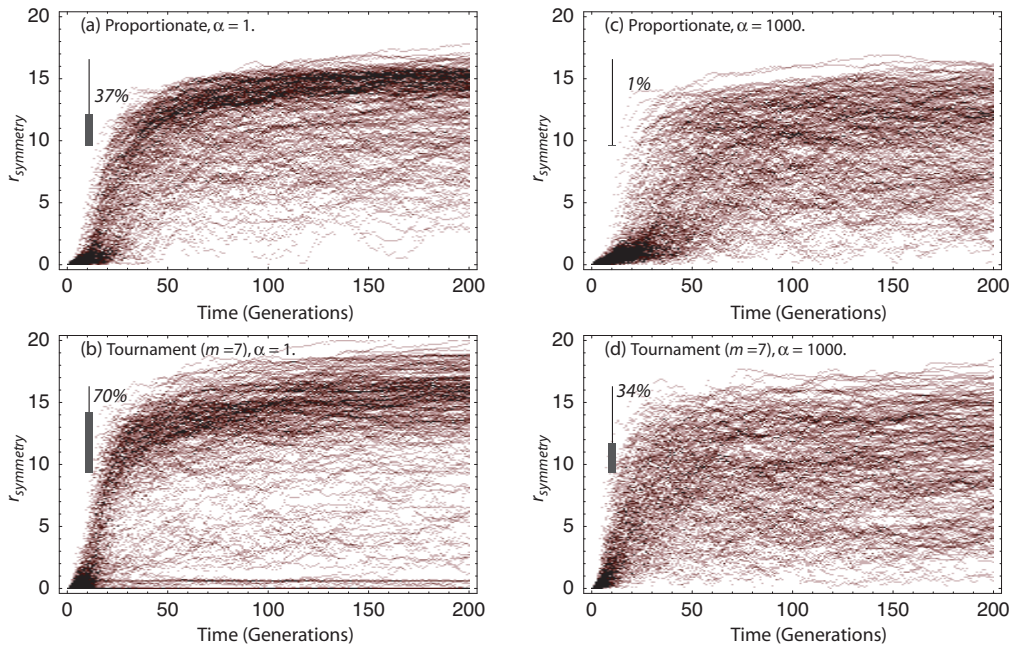

Figure 5. Density plots of $r_{symmetry}$ v. time for two different problem-difficulty settings and two different selection methods. Each density plot shows the results for 200 trials, where each trial was run for 200 generations . Darker tones correspond to more runs that had a particular measurement of $r_{symmetry}$.

Each density plot in Figure 5 shows four variables (i.e., t, $r_{symmetry}$, cumulative distribution, and measured problem difficulty). The x-axis corresponds to time t (in generations), while the y-axis corresponds to $r_{symmetry}$. Tone in the density plot is correlated to cumulative distribution: the darker the tone, the greater the number of runs that have had that particular value of $r_{symmetry}$ at that particular time. Measured success-rate is represented as a thermometer graphic: higher values on the thermometer mean that GP solved the problem more frequently. For example, a thermometer value of 100% means that GP found a successful solution in all of its runs.

The remaining two variables—selection method and difficulty setting α—were portrayed by arranging the four density plots as elements of a two dimensional matrix. Each density plot subsequently corresponds to a variation of one of these two remaining variables. In particular, the top row corresponds to proportionate selection; the bottom row, tournament selection. The left column corresponds to $\alpha = 1$ ("easy"); the right column, $\alpha = 1000$ ("hard").

Figure 6 shows the next set of results as a five-variable visualization for all 800 runs. It represents a reduction of the data depicted in the previous visualization.

Each scatter plot in Figure 6 shows three variables: t, abs($\Delta r_{symmetry}$), and measured problem difficulty. The x-axis corresponds to time t (in generations), while the y-axis corresponds to abs($\Delta r_{symmetry}$). As in the previous figure, measured success rate is represented as a thermometer graphic.

Also as in the previous figure, the remaining two variables—selection method and difficulty setting α—were portrayed by arranging the four scatter plots as elements of a two dimensional matrix. The arrangement parallels that given in Figure 5.

Figure 7 shows the next set of results as a four-variable visualization. It represents a further reduction of the data depicted in the previous visualization.

Each plot in Figure 7 shows three variables: t, abs($\Delta r_{symmetry}$), and the selection method. The x-axis corresponds to time t (in generations), while the y-axis corresponds to abs($\Delta r_{symmetry}$). Each curve denotes the upper-quartile envelope for abs($\Delta r_{symmetry}$) as a function of time. Each curve is derived by identifying a boundary where at least 75% of all measured points for abs($\Delta r_{symmetry}$) exist at each generation and then by fitting those identified upper-quartile boundaries with a nonlinear equation. The form of the fitted equation is given as

$$\left|\Delta r_{symmetry}\right|_{fit} = a_1 \exp\left(a_2 \left(\ln a_3 t\right)^2\right) + \sum_{n=1}^{4} b_n t^n . \qquad (10)$$

Each plot in Figure 7 corresponds to a difficulty setting α, where the left plot corresponds to $\alpha = 1$ ("easy"); the right plot, $\alpha = 1000$ ("hard"). The dashed lines in either plot correspond to tournament selection; the solid lines, proportionate selection.

5.3 Discussion

For a number of years, symmetry breaking was suspected of occurring in systems like GP. This modest case study has demonstrated that symmetry breaking occurs not just as isolated stories (as Figures 1, 4 and 5 show), but as a statistically measurable and quantifiable trend that applies to ensembles of trials.

If symmetry breaking occurs in a run, these results suggest that it appears as a transient near that run's beginning. At generation 0, GP started with a more or less symmetrically distributed population, which corresponds to $r_{symmetry}$ having a near-zero value

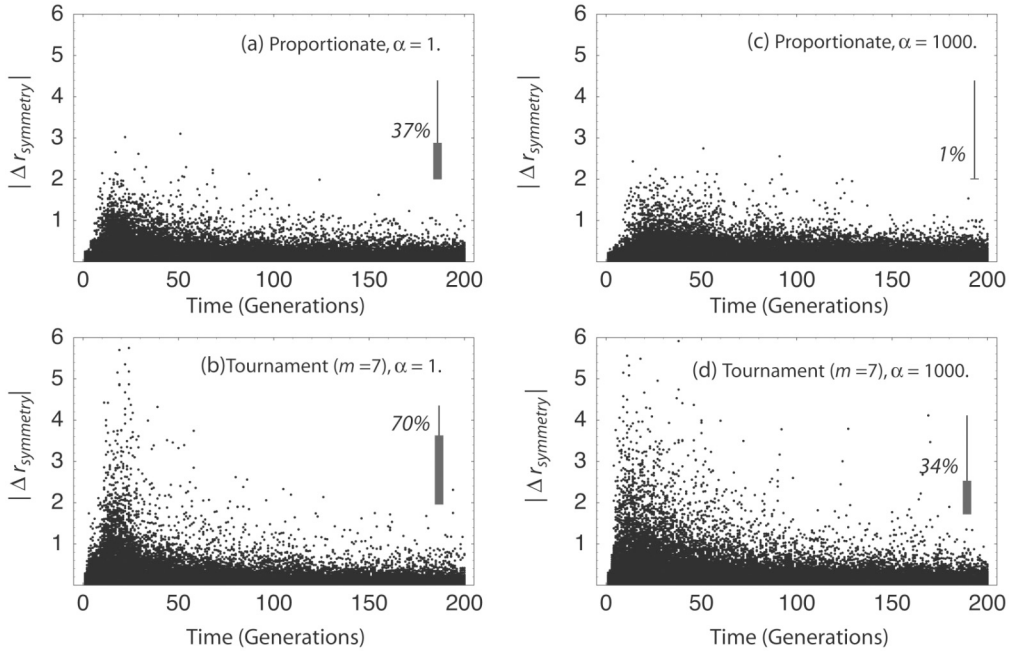

Figure 6. Scatter plots of Δ| $r_{symmetry}$ | v. time for two different problem-difficulty settings and two different selection methods. Note that Δ| $r_{symmetry}$ | corresponds to the magnitude of the rate of change in symmetry.

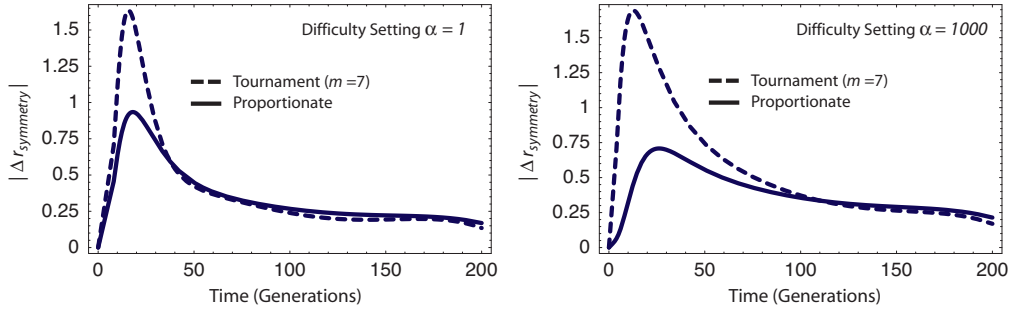

Figure 7. Curve fits that describe the outer envelope of the scatter plots of Δ| $r_{symmetry}$ | v. time for two different problem-difficulty settings and two different selection methods. These plots summarize the data shown in the previous figure.

in Figure 6 not just for a single trial, but for all 800 trials involved in this case study. The transition, which was marked by a rapid rise in $r_{symmetry}$, indicated that a population went from symmetric to asymmetric in a relatively short time (between 0 and 50 generations). After 50 generations, the rate of change in $r_{symmetry}$ slowed considerably. It was presumed that by that point, the depth limit associated with the *binomial-3* problem also had the effect of serving to limit $r_{symmetry}$. This overall trend was robust, in spite of varying settings of problem difficulty associated with this tunable problem.

That this case study did feature several bracketing configurations with a tunably difficult problem does raise additional questions. For example, the plots in Figure 7 show noticeably different nuances in the rate of change associated with $r_{symmetry}$. Of particular interest are various correspondences of the shape of the rate change envelopes with the configurations given. These correspondences raise questions such as: How different are the causes that drive the dynamics under proportionate selection and tournament selection? Is the width of a peak an indicator of likelihood of GP obtaining a solution? Are there different taxonomies of dynamic behavior that are subsumed under these curves? How transferable are these results to other problems? Could this metric be leveraged to derive increasingly complex solutions?

It is beyond the scope of this paper to address these questions. However, in problems where program semantics do not play a role in determining a solution, it is known that the overall structure of a population remains relatively symmetrical (e.g., [12]). It is expected then, that symmetry breaking — as defined by $r_{symmetry}$ — is not likely to occur. Nevertheless, even that expected negative result raises still further questions, such as: What is the transition in semantics needed to result in symmetry breaking? Is there a minimum set of semantics that could bring about symmetry breaking?

Addressing these and other questions is left to future work.

6. CONCLUSIONS

This paper identified a metric that has been shown to characterize the dynamics of symmetry breaking in GP. This metric, called $r_{symmetry}$, is based on a method of graph construction that was

originally designed as a method for visualizing populations of tree structures in GP.

This metric allowed for a comparison of aggregates of tree shapes according to degrees of symmetry, which was not possible before in the field. In particular, the work shown in the paper allowed for a statistical comparison in shape among some 80.4 million trees. As far as a visualization method goes, the reduction of a population graph to a single metric allowed for a visualization of tree shapes that was almost three orders of magnitude more than was previously possible.

Finally, the results in this paper have shown that symmetry breaking occurs in GP as a statistically measurable and quantifiable trend that applies to ensembles of trials. This result is significant, given that symmetry breaking has been identified in the (human) developer community as a distinguishing characteristic of "good" design in code crafted by humans. The results given here are the first quantifiable, statistically meaningful result of symmetry breaking in coding by either human or automatic code generating systems.

7. ACKNOWLEDGMENTS

Several current and former members from my research group contributed to this paper: S. Long, A. Hilss, M. Hodges, D. Ward, M. Byom, M. Samples, J. Thomas, K. Jham, E. Chen. Gratitude is extended to the reviewers. I thank S. Daida and I. Kristo for their continued support.

8. REFERENCES

[1] Alexandrescu, A., *Modern C++ Design: Generic Programming and Design Patterns Applied*. 2001, Boston: Addison–Wesley. 323 pp.

[2] Altenberg, L., The Schema Theorem and Price's Theorem, in *FOGA 3*, 1994, Morgan Kaufmann: San Francisco. p. 23–49.

[3] Appleton, B., Accessed 14 January 2006. http://www.cmcrossroads.com/bradapp/docs/

[4] Budinsky, F., et al., Automatic Code Generation from Design Patterns. *IBM Systems Journal*, 1996. 35(2): p. 151–171.

[5] Burke, E., S. Gustafson, and G. Kendall, Diversity in Genetic Programming. *IEEE TEC*, 2004. 8(1): p. 47–62.

[6] Chin, K.-W. and H.-C. Yen, The Symmetry Number Problem for Trees. *Information Processing Letters*, 2001. 79(2): p. 73–79.

[7] Coplien, J.O., Software Design Patterns: Common Questions and Answers, in *The Patterns Handbook*, 1998, Cambridge University Press: New York. p. 311–320.

[8] Coplien, J.O., Reevaluating the Architectural Metaphor: Towards Piecemeal Growth. *IEEE Software*, 1999. 16(5): p. 40–44.

[9] Coplien, J.O. and L. Zhao, Symmetry and Symmetry Breaking in Software Patterns, in *GCSE,* 2000: Erfurt, Germany. p. 37–56.

[10] Coplien, J.O. and L. Zhao, Symmetry Breaking in Software Patterns, in GCSE 2000. Revised 2001, Springer-Verlag: Berlin. p. 37–56.

[11] Coplien, J.O., Patterns of Engineering. *IEEE Potentials*, 2004. 23(2): p. 4–8.

[12] Crane, E. and N.F. McPhee, Effect of Size and Depth Limits on Tree-Based GP, in *GPTP III*, 2006, Springer: New York.

[13] Daida, J.M. and A.M. Hilss, Identifying Structural Mechanisms in Standard GP, in *GECCO*, E. Cantú-Paz, et al., 2003, Springer-Verlag: Berlin. p. 1639–1651.

[14] Daida, J.M., Towards Identifying Populations that Increase the Likelihood of Success in Genetic Programming, in *GECCO,* 2005.

[15] Daida, J.M., et al., Visualizing Tree Structures in GP. *GPEM*, 2005. 6: p. 79–110.

[16] Eggermont, J., A.E. Eiben, and J.I. van Hemert, Adapting the Fitness Function in GP for Data Mining, in *EuroGP*. 1998, Springer-Verlag: Berlin. p. 193–202.

[17] Gamma, E., et al., *Design Patterns: Elements of Reusable Object-Oriented Software*. 1995, Reading: Addison-Wesley Professional.

[18] Hall, J.M. and T. Soule, Does GP Inherently Adopt Structured Design Techniques? in *GPTP II*, U.-M. O'Reilly, et al., 2004, Kluwer Academic: Boston.

[19] Holland, J., Hidden Order: How Adaptation Builds Complexity. 1995, Reading: Addison-Wesley.

[20] Hornby, G.S., H. Lipson, and J.B. Pollack, Generative Representations for the Automated Design of Modular Physical Robots. *IEEE TRA*, 2003. 19(4): p. 703–719.

[21] Jacob, C., Genetic L-System Programming, in *PPSN III*, 1994, Springer-Verlag: Berlin. p. 334–343.

[22] Klein, P., et al., A tree-edit-distance algorithm for comparing simple, closed shapes, in *ACM-SIAM Symp Discrete Alg, 2000*, SIAM: Philadelphia. p. 696–704.

[23] Klein, P., T.B. Sebastian, and B.B. Kimia, Shape Matching Using Edit-Distance, in *Symp Discrete Alg*, 2001, SIAM: Philadelphia. p. 781–790.

[24] Koza, J.R., *GP*, 1992, Cambridge: The MIT Press.

[25] Koza, J.R., *GP II*. 1994, Cambridge: The MIT Press.

[26] Koza, J.R., Two Ways of Discovering the Size and Shape of a Computer Program to Solve a Problem, in *ICGA*, 1995, Morgan Kaufmann: San Francisco. p. 287–294.

[27] Langdon, W.B., et al., The Evolution of Size and Shape, in *AGP* 3, L. Spector, et al., 1999, The MIT Press: Cambridge. p. 163–190.

[28] Langdon, W.B., Quadratic Bloat in Genetic Programming, in *GECCO*, 2000, Morgan Kaufmann: San Francisco. p. 451–458.

[29] Langdon, W.B. and R. Poli, *Foundations of GP*. 2002, Berlin: Springer-Verlag.

[30] Lenaerts, T. and B. Manderick, Building a GP Framework, in *EuroGP*, 1998, Springer-Verlag: Berlin. p. 196–208.

[31] Liu, T.-L. and D. Geiger, Approximate tree matching and shape similarity, in *IEEE ICCV*, 1999, IEEE Computer Society: Los Alamitos. p. 456–462.

[32] McPhee, N.F. and N.J. Hopper, Analysis of Genetic Diversity through Population History, in *GECCO*, 1999, Morgan Kaufmann: San Francisco. p. 1112 – 1120.

[33] Poli, R., W.B. Langdon, and U.-M. O'Reilly, Analysis of Schema Variance and Short Term Extinction Likelihoods, in *GP*, 1998, Morgan Kaufmann: San Francisco. p. 284–292.

[34] Poli, R., General Schema Theory for Genetic Programming with Subtree-Swapping Crossover, in Genetic Programming: Proceedings of EuroGP 2001, April 18–20, 2001, Milan, J.F. Miller, et al., 2001, Springer-Verlag: Berlin. p. 143–159.

[35] Rosca, J.P., Analysis of Complexity Drift in Genetic Programming, in Genetic Programming 1997: Proceedings of the Second Annual Conference, July 13-16, 1997, Stanford University, J.R. Koza, et al., 1997, Morgan Kaufmann Publishers: San Francisco. p. 286–94.

[36] Sastry, K., U.-M. O'Reilly, and D.E. Goldberg, Population Sizing for GP Based on Decision Making, in *GPTP II*, 2004, Kluwer Academic: Boston. p. 49–65.

[37] Sebastian, T.B. and B.B. Kimia, Curves vs. Skeletons in Object Recognition. *Signal Processing*, 2005. 85(2): p. 247–263.

[38] Soule, T., J.A. Foster, and J. Dickinson, Code Growth in GP, in *GP*, 1996, The MIT Press: Cambridge. p. 215 – 223.

[39] Streeter, M.J., M.A. Keane, and J.R. Koza, Routine Duplication of Post-2000 Patented Inventions by Means of GP, in *EuroGP*, 2002, Springer-Verlag: Berlin. p. 27–36.

[40] Veltkamp, R.C., Shape matching: similarity measures and algorithms, in *Int Conf Shape Model and App*, 2001, IEEE Computer Society: Los Alamitos. p. 188–197.

[41] Wales, J., Design Pattern (Computer Science), Generative Programming, Generic Programming. *Wikipedia*, 2001, accessed 14 January 2006.