Genetic Programming and Evolvable Machines, 6, 79–110, 2005

Visualizing Tree Structures in Genetic Programming

JASON M. DAIDA, ADAM M. HILSS, DAVID J. WARD AND STEPHEN L. LONG
Center for the Study of Complex Systems and Department of Atmospheric, Oceanic and Space Sciences, The University of Michigan, 2455 Hayward Avenue, Ann Arbor, MI 48109-2143, USA

Submitted October 15, 2003; Revised March 6, 2004

Abstract. This paper presents methods to visualize the structure of trees that occur in genetic programming. These methods allow for the inspection of structure of entire trees even though several thousands of nodes may be involved. The methods also scale to allow for the inspection of structure for entire populations and for complete trials even though millions of nodes may be involved. Examples are given that demonstrate how this new way of "seeing" can afford a potentially rich way of understanding dynamics that underpin genetic programming. The examples indicate further studies that might be enabled by visualizing structure at these scales.

1. Introduction

"Structure always affects function." — S. Strogatz in ([62], p. 268)
In saying this, Strogatz was commenting on the nature of complex networks. To those who study networks, the structure of interconnections can have a pronounced effect on function. True, he was thinking along the lines of complex networks like an electrical power grid or the Internet. However, he could just as well have been referring to the kinds of solution outcomes that happen in genetic programming (i.e., GP).

For example, some of the early theoretical work in GP has alluded to the consequences of structure in determining solution outcomes. Rosca and Ballard [52, 53] hypothesized that solution outcomes are determined, in part, by rooted-tree schema. They argued that during a run, GP identifies these structures first and subsequently builds upon these structures to form solutions.

Other researchers have independently supported Rosca and Ballard's theory, which highlighted the importance of root structure. This support has come largely in the form of empirical studies that featured selected benchmark problems, such as [19, 44]. Of these two, McPhee and Hoppers's work made the strongest empirical case that root structures play a key role in determining solution outcomes for GP in general.

Structure, of course, has played a significant role in other studies in GP as well. For example, structure was a key determinant of fitness in one of the earliest adaptations of the Royal Road problem [45] in genetic algorithms to GP [51]. Work by Goldberg and O'Reilly [20, 47] resulted in the development of the ORDER and MAJORITY fitness problems in which the structures of solution outcomes were driven by the problem at hand. Structure has played a key role in the analysis by Soule, Foster et al. [57–59], who have looked at the evolution of shape in GP outcomes. Soule and Foster's work on the evolution of tree shapes has since been carried forward and extended by Langdon and others [38, 40, 41].

Structure plays a key role, too, in current theories on GP. Poli and Langdon's work on a GP schema theory [40, 49, 50] presumes schemata that are structurally based (i.e., has a certain size and depth). Theories concerning bloating have also been associated with structure (i.e., tree shape). For example, it has been primarily Langdon's contention that bloating occurs partly as a result of a random walk on a "landscape" of tree-shapes (e.g., [37, 39]).

Unfortunately, in spite of this longstanding interest in the role of structure, the direct visualization of such structures has *not* progressed far. Nearly all visualizations of structure have been "for illustrative purposes only," as was first done by Koza [35]. There has not been a viable means of inspecting structures for typical trees that have been discussed in GP, let alone a viable means of inspecting structures for an entire population. Consequently, our investigations have focused on the visualization of tree structures and other topologies of interconnections between nodes. Our first paper on this subject appeared in [11].

The purpose of this paper, then, is twofold. The first purpose is to amplify [11] and to offer additional methods, visualizations, and data in support of that work. The second purpose is to point out new areas of investigation that are suggested by these visualizations—that this new way of "seeing" affords a potentially rich way of understanding the dynamics that underpin genetic programming.

This paper is organized as follows. Section 2 covers the background for this paper, which includes the previous work and mathematical context. Section 3 describes our method for visualizing the structure for an entire tree and provides examples of these visualizations. Section 4 extends the methods described for individual trees and scales them to encompass populations. Section 5 concludes. We also provide two appendices: the first comments on demonstration software; the second gives an expanded historical context to the work described in this paper.

2. Background

The background to this paper considers several topics that are typically not considered in GP. Part of the reason for this is because visualization tends to be an interdisciplinary pursuit. Another part of the reason is because our treatment of structure is unusual in the field. Consequently, this section covers six independent topics. We suggest jumping to Section 2.4 for those who would want minimal background.

Section 2.1 discusses our assumptions and draws distinctions in how we approach structure. Section 2.2 lists our criteria of what we were looking for from a visualization method. Section 2.3 summarizes key graphic-arts principles, which would otherwise go without saying in the scientific visualization literature, but are not a given in GP. Section 2.4 describes previous work. Section 2.5 describes the mathematical context to this paper, because we have used mathematics that has not been used in GP. Section 2.6 summarizes this paper's case study, which provides the material with which to visualize.

2.1. Assumptions

In genetic and evolutionary computation, the issue of trees would typically fall under the category of *problem representation* (e.g., [1, 2]). One of the basic considerations in genetic and evolutionary computation is the matter of choosing a representation that is suited for

solving a particular problem [15]. The manner in which one represents an appropriate solution—say, a vector of real values—in turn drives the type of operators, the type of objective function, even the type of genetic or evolutionary algorithm to be used [17]. Representations, and their corresponding evolutionary algorithms, are graded according to their worth to solving a particular problem. There is no single representation or evolutionary algorithm that is universally superior to the rest when it comes to optimization (i.e., No-Free-Lunch theorem [68]).

However, *information structure* is distinct from *problem representation*. When one does an analysis in problem representation, one implicitly assumes that what counts is information from a problem's domain, as opposed to the structure that carries that information. This view makes sense when considering many algorithms in genetic and evolutionary computation. In such cases, the information structure—e.g., a `float`, `char`, or an array—remains static for the duration of that algorithm's processing. What changes is content. For all practical purposes, structure can be (correctly) factored out of an analysis.

There are fields in computer science and in mathematics, however, where a static information structure is not a given. Of interest, instead, are the consequences of information structures that are variable and dynamic. Trees are one such structure. Consequently, when one does an analysis of trees as information structure, it is common to treat trees as mathematical entities *apart* from the information such trees would carry. This treatment effectively renders information structure as a level of abstraction that is distinct from that of problem representation.[1]

The distinction between *trees-as-problem-representation* and *trees-as-information-structure* has consequences when considering the previous work and methods for visualization. The matter of *trees-as-problem-representation* would infer a visualization method that works only for specific problems (i.e., tree-like) and specific kinds of GP (i.e., standard, as opposed to linear GP [46]). However, the matter of *trees-as-information-structure* implies a much broader scope. As Knuth points out in ([34], p. 312), "any hierarchical classification scheme leads to a tree structure." One needs to consider, then, other methods to visualize trees that look nothing like trees (as in Figure 1). One also needs to consider that the visualization of trees extends to types of GP that are "linear," as well as "tree-based."

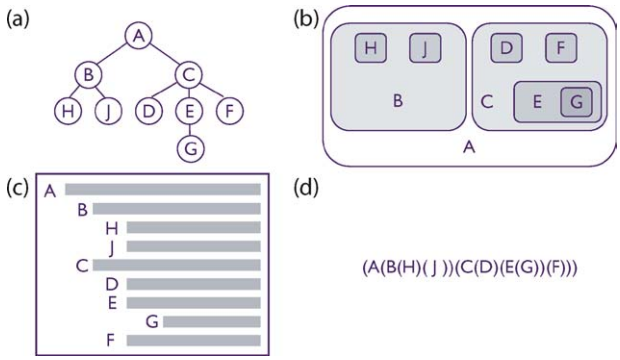

Figure 1. Visualizing hierarchical information. This figure shows four different views of the same information. (a) Tree structure, (b) Corresponding nested sets, (c) Indented text, (d) Nested parentheses. Based on Figure 20 by Knuth [34].

Since this paper considers *trees-as-information-structure*, we expanded our scope of previous work to include relevant means of displaying hierarchical information. We subsequently included much of this information in Section 2.3 and Appendix B.

2.2. Criteria

Several criteria were considered in the design of our visualizations. They were as follows:

- The visualization should be able to depict the structure of a solution from genetic programming in its entirety. This has the implication that the method be able to routinely display trees up to and beyond, say, depth 25.
- The visualization should be organized on an absolute coordinate system. In the broader view of what is needed to understand the dynamics of genetic programming, the analysis of one tree is not sufficient, since GP is stochastic. An absolute coordinate system would allow for both a visual and a quantitative comparison of structure between trees. It would also allow for a statistical analysis of an ensemble of trees.
- The method used to visualize one tree should be scalable to at least a population level. It would be helpful for the visualization to be able to summarize the structural trends of an entire population.
- The method used to visualize one population should be scalable to examine the dynamics of a complete run (sometimes called a *trial* in other works) in GP.
- The method should conform to established principles in the broad field of scientific visualization.

2.3. Visualization principles

The visualization principles that were employed in this work were those described in Tufte's influential work in data graphics [63–65]. He is known for a particular, minimalist style of visualization that we use in the construction of quantitative graphics. This particular style is not commonly evoked in the GP community. For that reason, it is worthwhile quoting several of his design principles from [64], pp. 105, 121:

- *Above all else show the data.*
- *Maximize the data-ink ratio.* In other words, let a data graphic represent more data and less "ink." A bar chart that shows five quantities has a low data-ink ratio. A scatter plot that depicts a trend for thousands of samples probably has a high data-ink ratio.
- *Erase non-data-ink.* In other words, remove graphical elements that contribute little to nothing to the information that is being portrayed. For example, the graphical elements that turn two-dimensional charts into quasi-three-dimensional ones are not recommended.
- *Erase redundant data-ink.* In other words, remove graphical elements that represent the same data that other graphical elements already do. For example, the line around colored bars in histograms can probably be removed without a loss in presented information.
- *Forgo chartjunk, including moiré vibration, the grid, and the duck.* In other words, remove graphical elements that call attention to themselves and not to the data that is being displayed. For example, much of the grid in two-dimensional charts can probably be

removed, as well as the extraneous artwork that accompanies many news-style data graphics.

The value of deferring to Tufte is that one can use his principles as the basis for arguing the merits of a visualization method. We note that a rigorous analysis of any visualization method would include human-subject testing. Visualization is a perceptual activity and there are standard psychological methods to qualify or disqualify methods according to "taste." An analysis according to Tufte's principles can broadly anticipate the outcomes of a rigorous analysis with human-subject testing. We did not have the resources for human-subject testing.

2.4. Previous works

The prevailing convention for visualizing trees in GP is largely the convention used by Koza in [35], where tree structures are typically depicted as in Figure 1(a). Contemporary variations of this convention appear in [28].

Referring to Figure 1(a), we note that the prevailing convention for drawing trees starts with the root node at the top and grows downward with the terminal nodes at the bottom. *Right*, *left*, *top*, and *bottom* are subsequently denoted as the reader's right, the reader's left, top going to the top of a page, and bottom going to the bottom of a page. *Depth*, as opposed to *height*, is also typically used. This convention follows the standard practice in computer science ([34], p. 311). Stanley explicitly states this convention ([60], p. 295) that for typical trees, lines from a parent node to child nodes are drawn at equal angles that are symmetric with respect to a vertical axis. This convention works if the total number of nodes remains small. For example, Knuth depicts two small trees that still consume a page of illustration (for one, the number of nodes is 63; for the other, the number of nodes is 61).[2]

For many problems, drawing a large tree is not needed, since a small tree would suffice as a representative sample. For GP, however, we claim that it is instructive to visualize trees of a few thousand nodes. Our reasoning is that such visualizations would show broader patterns of solution structure that are not readily seen with smaller samples or with structural metrics (like *number of nodes* or *depth level*). Unfortunately for these larger trees, there is little by way of a drawing convention. Consequently, Section 3 introduces a method for drawing large trees.

Our methods are distinguished from current convention in the following manner:

- Our work represents an increase in the ability to examine tree structures by several orders of magnitude. Before this work, most trees that were displayed in GP were representative (small samples that consist of tens of nodes). The methods of this paper allow for the examination of whole GP trees (thousands to tens of thousands of nodes), as well as entire populations (thousands to hundreds of thousands of nodes). There are simple extensions described in this paper that allow for the examination of entire trials (hundreds of thousands to millions of nodes).
- Our work represents the first use of a circular coordinate system to visualize and compare whole trees in the GP community. It represents a first in the GP community to visualize structures of entire populations. It represents a novel use of an absolute coordinate system

that allows for both quantitative analysis and visualization. (Absolute coordinate systems have been used before, although in different contexts and purposes. See [44, 48].)

We note that as pointed out by Figure 1, trees do not have to be depicted as tree graphs or diagrams. One compact rendition of trees is a consequence of a method that was originally developed for image processing and analysis [32]. The idea was to encode two-dimensional shape information with quadtrees, since efficient algorithms existed for manipulating trees (see [25, 33, 54, 55]).

2.5. Mathematical preliminaries

Our treatment of trees—which would be common in an analysis of information structure—differs from typical treatments of trees in GP. GP uses conventions that tend to focus on node content, which makes it difficult to incorporate mathematics like enumerative combinatorics, which could enhance the field. For example, it has often been sufficient in GP to specify the structure of a tree largely by considering simple metrics (like *depth*, *number of nodes*, *number of terminals*). Unfortunately, what is typically defined as a *tree* in GP does not have the precision or rigor necessary to articulate classes or types of tree structures that occur when one does a structural analysis. In contrast, the treatment of trees in this paper presupposes the use of conventions that focus on structure.

The specific conventions and terms that are used in this paper are those of Knuth [34] and Stanley [60, 61]. Although it is possible to recast our visualization techniques without having to resort to this level of formalism, we believe it appropriate since terms that are common in GP and in enumerative combinatorics—like the term *binary tree*—are not equivalent in meaning between those respective fields.

A *tree* T is formally defined as a finite set of nodes such that:

(a) There is one node that is designated as the *root* of the tree, and
(b) The remaining nodes (excluding the root) are partitioned into $m \geq 0$ disjoint, non-empty sets $T_1, \ldots, T_m$, each of which is a tree. The trees $T_1, \ldots, T_m$ are called *subtrees* of the root.

The number of subtrees associated with a node is called the *degree* of that node. As in GP, a node of degree zero is called a *terminal node* or *leaf*. Each root is considered as the *parent* of the roots of its subtrees; the roots of these subtrees are said to be the *children* of their parent and *siblings* to each other.

A *plane tree* or *ordered tree* is defined similarly to a *tree*, except that (b) becomes

(b′) The remaining nodes (excluding the root) are placed into an ordered partition $(T_1, \ldots, T_m)$ of $m \geq 0$ disjoint, non-empty sets $T_1, \ldots, T_m$, each of which is a plane tree.

If only the relative orientation of nodes is considered (i.e., the tree's structure, assuming that content associated with each vertex does not matter) and not their ordering (i.e., the same tree, except that the content associated with each vertex does matter), a tree is said to be *oriented*.

Let $m \geq 2$ in the definition of a tree. An m-ary tree is defined similarly to a *tree*, except that (a) and (b) become

(a″) Either T is empty or there is one node that is designated as the *root* of T, and
(b″) The remaining nodes (excluding the root) are put into an ordered partition of exactly m disjoint (possibly empty) sets $T_1,\ldots,T_m$, each of which is an m-ary tree.

An m-ary tree is said to be *complete* if every node that is not a leaf has m children. In deference to GP (and not Knuth), we use the term *full* to refer to a complete m-ary tree that has m^d terminal nodes, where d is the depth of that tree and the root node is considered at depth 0. A *binary* tree is a 2-ary tree.

Note that an m-ary tree is *not* a special case of the formal definition of a tree. Consequently, a binary tree is not assumed to be an instance of a *tree* T. By definition, however, a complete binary tree is such an instance.

The aforementioned definitions describe various classes of trees and various levels of similarities between trees with a precision that is not readily available with the typical treatment of trees in GP. For example, the term *complete* or *plane binary tree* refers to a tree in GP that consists of arity-2 internal nodes. (The term *binary tree* refers to a tree in GP that could consist of *both* arity-1 and arity-2 internal nodes. However, there is structural specificity that is implicit in that term that is not translatable in GP.) Furthermore, such a tree is assumed to be labeled and to have an ordering of nodes—all of which are assumed for binary tree in GP. What becomes awkward in GP is when one wishes to group trees into a class that consist of the same structure, but not necessarily trees with identical node content. In enumerative combinatorics, such a class would simply be considered as an instance of an *oriented* tree.

This paper derives results for oriented complete binary trees by considering a remapping of node labels to a lattice. A *lattice* is a poset for which every pair of elements has a least upper bound and greatest lower bound. A *poset* P, or *partially ordered set* P, is a set that satisfies the following three axioms:

1. For all $x \in P$, $x \leq x$. (*reflexivity*)
2. If $x \leq y$ and $y \leq x$, then $x = y$. (*antisymmetry*)
3. If $x \leq y$ and $y \leq z$, then $x \leq z$. (*transitivity*)

An *upper bound* of two elements x and y in a poset P is an element z in P such that $z \geq x$ and $z \geq y$. A *least upper bound* (or *supremum*) for these two elements is an element z in P such that every upper bound w of x and y satisfies $w \geq z$. Likewise, a *lower bound* of two elements x and y in a poset P is an element z in P such that $z \leq x$ and $z \leq y$. A *greatest lower bound* (or *infimum*) for these two elements is an element z in P such that every lower bound w of x and y satisfies $w \leq z$.

2.6. This paper's case study

Throughout the remainder of this paper, we draw visualized examples from a data modeling (i.e., symbolic regression) problem that has been documented elsewhere in the literature (e.g., [5, 9]).

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 was $\{+, -, \times, \div\}$, which corresponds to arithmetic operators of addition, subtraction, multiplication, and protected division (which returns unity when the denominator is zero). The terminal set was $\{\mathtt{X}, \mathrm{R}\}$, where X is the symbolic variable and R is the set of ephemeral random constants that are distributed uniformly over the interval $[-\alpha, \alpha]$. The parameter α is used for tuning. In general, values of α that approach unity correspond to easier problems; values of α that become large (e.g., $\alpha = 1000$) correspond to harder problems. Most of the GP parameters were similar to those mentioned in Chapter 7 in [35]. Unless otherwise noted: population size = 500; crossover rate = 0.9; replication rate = 0.1; population initialization with ramped half-and-half; initialization depth of 2–6 levels; and tournament selection ($n_{\text{tournment}} = 7$). Other parameter values were maximum generations = 200 and maximum tree depth = 26. The replacement scheme was generational.

From the standpoint of typical metrics that quantify aspects of tree structure (i.e., *number of nodes* and *depth level*), the resulting GP binary trees have been unremarkable [14]. However, the problem is a tunably difficult one and can tune from "very easy" (nearly all runs result in a "perfect" solution) to "very difficult" (for all practical purposes, no runs result in a "perfect" solution). Significant differences in problem behavior also occur as a function of selection type (i.e., tournament selection v. fitness-proportionate selection).

Although previous analyses have explained why such tunability may come about, the visualizations shown here suggest that problem difficulty is correlated with solution structure. For this paper, we do not address why this is so; see [7, 8, 10, 13] for more on that question. Instead, we use this implied correlation as an opportunity to point out that the visualization of structure in and of itself affords a potentially rich way of understanding the dynamics that underpin genetic programming.

For the sake of comparison in this paper, all visualization examples were taken from runs in which a "perfect" solution was identified. This differs from our previous work (i.e., [8, 11]) in which representative samples were shown. "Representative" is not the same as "perfect." For example, "perfect" solutions to a difficult problem are anything but representative of a typical run. Indeed, for fitness-proportionate selection and a "very difficult" setting, "perfect" solutions have appeared in about 1 run out of every 100.

3. Visualizing tree structure

3.1. Visualization method

Our method for visualizing trees involves mapping m-ary (2-ary for this paper) to a circular grid. For binary trees, this grid can be derived by starting with a full binary tree, which we designate as C. The nodes of C are labeled by a positive integer k ($k \in \Im^+$) in the following way:

1. The root node is designated as $k = 1$ and at depth level 0.
2. The leftmost node for each depth level n is designated as $k = 2^n$.
3. Nodes at depth level 1 are labeled from left to right as $k = 2$ and $k = 3$, respectively.

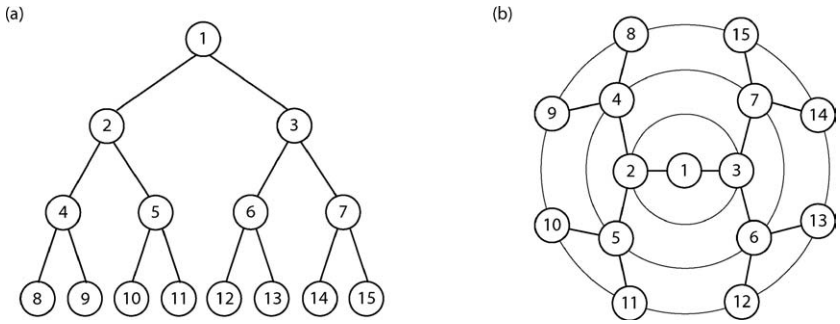

Figure 2. Mapping a full binary tree to a circular grid. (a) Full binary tree of depth 3. (b) Corresponding circular grid.

4. Nodes at each subsequent depth level n are labeled from left to right using the following sequence: $\{2^n, 2^n + 1, 2^n + 2, \ldots 2^{n+1} - 1\}$.

The nodes of this labeled full binary tree can be mapped to a circular grid in polar coordinates (r, θ) by first assigning depth levels to a set of concentric rings. The exact mapping of depth levels to ring radii is arbitrary, so long as depth level 0 remains at the center and increasing depths are assigned to rings of increasing radius. For convenience, we let depth level n be mapped to a ring of radius n. Figure 2(a) gives an example of a full binary tree of depth three that has been labeled in this manner.

Figure 2(b) also shows a full binary tree of depth three, except that this tree is mapped to a circular grid. This visual mapping is arguably straightforward and intuitive. What perhaps is not so intuitive is that each of these node labels, an integer, uniquely determines a position in this grid: i.e., given a node label k, it is possible to map k onto (r, θ) and vice versa. Consequently, a set of positive integers can represent the structure of an entire full tree, which turns out to be a useful transformation (e.g., see [10]). What is subsequently needed, at least for the purposes of visualization, is a means to map k onto (r, θ). The next two equations—$\rho(k)$ and $\phi(k)$—specify this mapping.

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

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

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

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

For the moment, we designate lattice L_C of plane binary trees to be defined by the collection of labeled nodes (vertices) of C in polar coordinate space. Consequently, Figure 2(b) is an example of a full binary tree of depth level three that has been mapped to L_C.

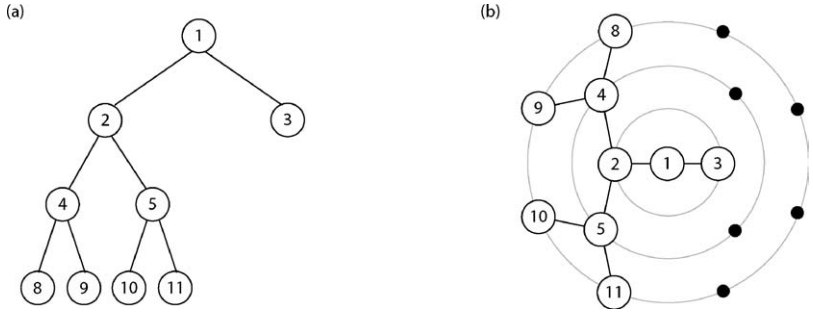

Figure 3. Mapping an arbitrary plane binary tree to a circular grid. (a) Arbitrary plane binary tree of depth 3. (b) Corresponding circular grid.

We can show that an arbitrary plane binary tree A (or for that matter, an arbitrary binary tree) can be mapped to L_C by showing that a set of labels corresponding to A is a subset of L_C. To do so, we traverse A in preorder[3] and label each node visited in the following manner:

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

We designate this set of labels for A as L_A. Figure 3 depicts an example of an arbitrary plane binary tree A that is mapped to L_C. If A is a full plane binary tree, it can be readily shown that this preorder labeling does result in an identical labeling that corresponds to C.

If one forgoes the information content in each node and treats each node as a vertex, one can use the described visualization method to examine the gross structure of (plane binary) trees. The compact representation allows for the depiction of structures containing several thousands of vertices.

Appendix A describes the software implementation of this method.

3.2. Visualization examples

Figure 4 shows an example of a full binary tree of depth level 10 (2047 nodes) and an example of a plane binary tree of depth level 26 (541 nodes) that result from GP.

In comparison to a full binary tree, the GP plane tree that is shown in Figure 4 is sparse and is characterized by a few radial spikes in which many of the nodes are found. There are significant areas of the lattice that are not populated. It is asymmetric. As it turns out, though, this gross pattern is representative of the population from which it was taken.

Figure 5 shows this individual tree structure in the context of 47 other individuals. (It is located in the uppermost left corner.) All of these trees have been arbitrarily taken from the same population of 500 at generation 200. Figures 6–8 show other sets of 48 individuals out of a population of 500, except that these individuals were generated with a slightly different

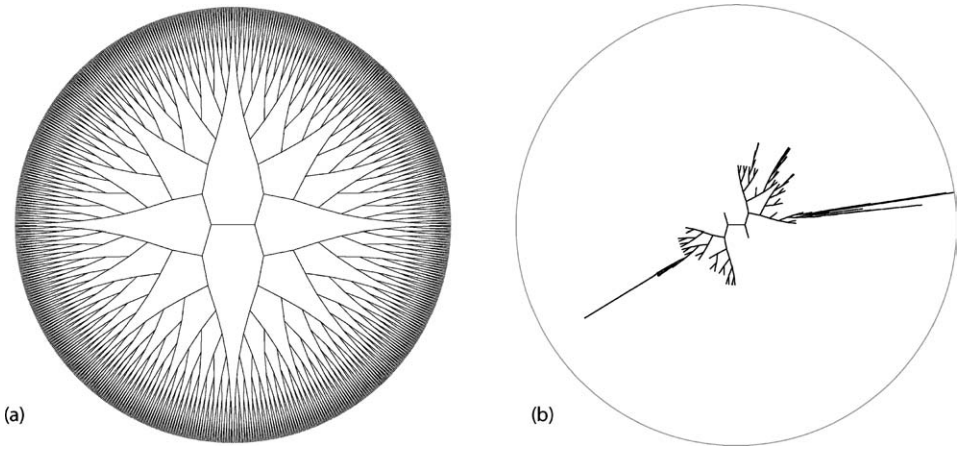

Figure 4. Two examples of plane binary trees. (a) Full binary tree, depth 10. (b) Structure of a GP generated solution (using arity-2 functions), depth 26. The gray circle around (b) represents depth 26 and is shown for reference.

set of GP parameters. As in Figure 5, all individuals that are displayed in Figures 6–8 have been chosen arbitrarily out of the same population at generation 200. Although many of the plane binary tree structures look similar (even identical) in each of the Figures 5–8, they are, in fact, distinct individuals. Close inspection shows minor structural differences between even the most similar-looking individuals.

Figures 5–8 raise a number of questions. What drives the shape of these structures? Are these figures representative of GP as a whole? Are there limits to the kinds of structures that can be generated in GP? How do structures evolve over time? How are structures correlated to problem difficulty? Why is there not more by way of structural diversity? Does the type of selection influence the kinds of structures that are generated? How does content correlate to structure? Which metrics best characterize these structural features? How do structures vary between trials? Clearly, there is a high degree of structural correlation in these small samples. Structure also seems to be correlated with problem difficulty, whereby more difficult problems seem to exhibit more kinds of shapes. These patterns suggest possibilities for using these visualizations, even though node content is not explicitly shown. We leave the pursuit of understanding and possibly using these patterns for future work.

3.3. Visualization alternative: Image encoding

Although our method for visualizing individual trees results in a graphic that is compact and concise, it is not the only method that can do so. Section 2.4 mentioned the use of image encoding. This section subsequently elaborates on that alternative and further elaborates on why we have chosen *not* to use that alternative for the visualization of individual trees.

Figure 9 demonstrates this alternative method as applied to a binary tree. Unlike our method, which focuses on the *linkages* between nodes, the alternative method focuses primarily on visualizing the *terminals* of a tree.

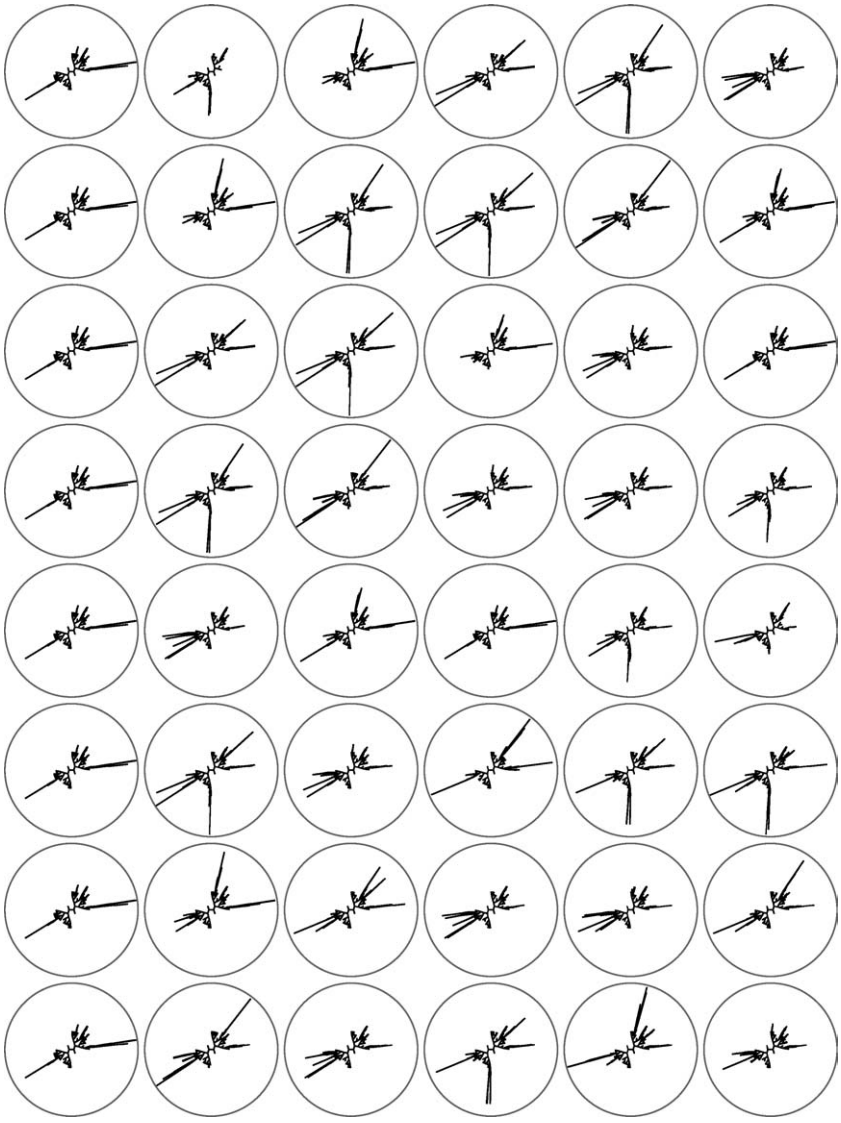

Figure 5. Sample of a population (very difficult, $\alpha = 1000$, fitness-proportionate selection). These are 48 representatives out of a population of 500. The individual that is shown in Figure 4(b) is the first from the top-left corner. These results are for a symbolic regression problem (binomial-3) using fitness-proportionate selection. As a reference, a gray circle around each individual represents depth level 26.

The process starts with the root node and a square box. (The root node is assumed to be at depth $= 0$.) If the root node is not a terminal, the box is halved. If the root of the left subtree is an internal node, the left side of the box is halved. If the root of the right subtree is an internal node, the right side of the box is halved. The process is repeated as nodes are traversed in preorder.

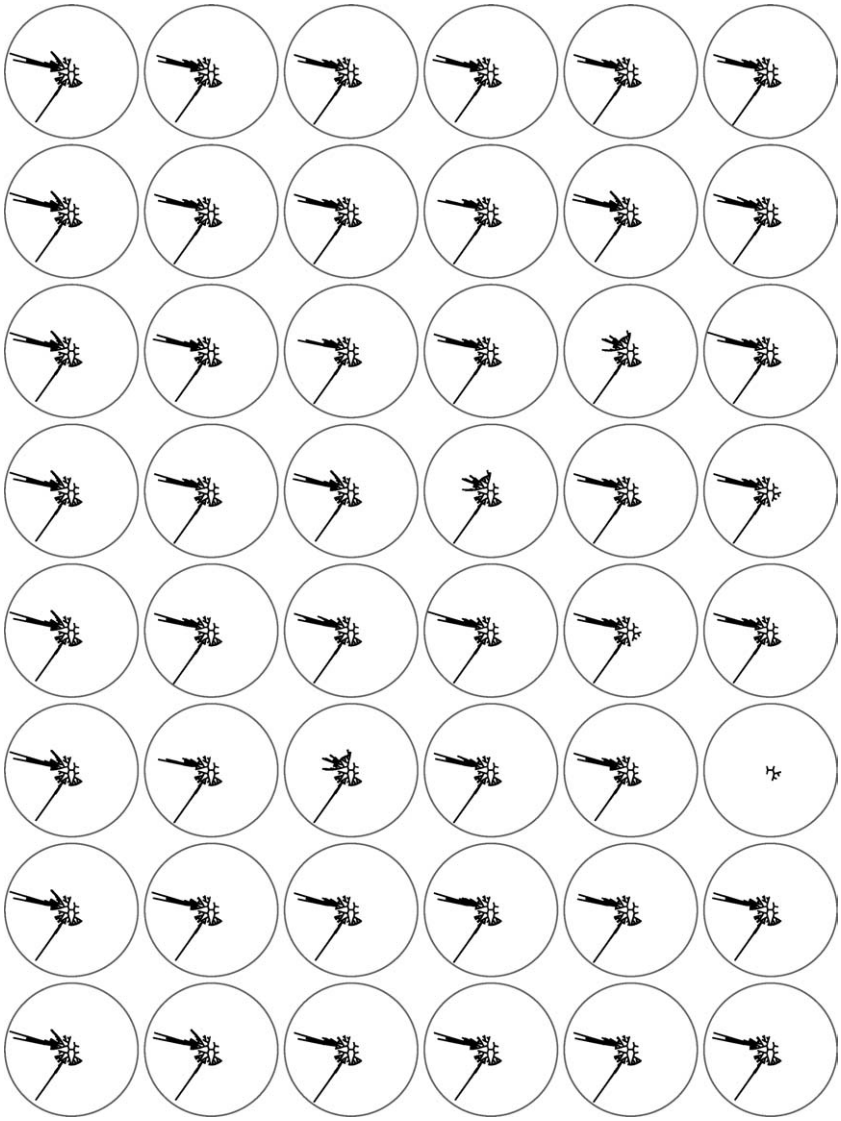

Figure 6. Sample of a population (moderate, $\alpha = 1000$, tournament selection). These are 48 representatives out of a population of 500. This problem was about 34 times easier to solve than the problem corresponding to Figure 5. The *only* difference that accounts for this change was the use of tournament ($n_{\text{tournament}} = 7$), instead of fitness-proportionate selection. As a reference, a gray circle around each individual represents depth level 26.

Figure 10 shows the identical data that was given for Figure 5, except they were visualized as encoded images.

We discontinued further use of the alternative on the basis of our design criteria (Section 2.2) and Tufte's visualization principles (Section 2.3).

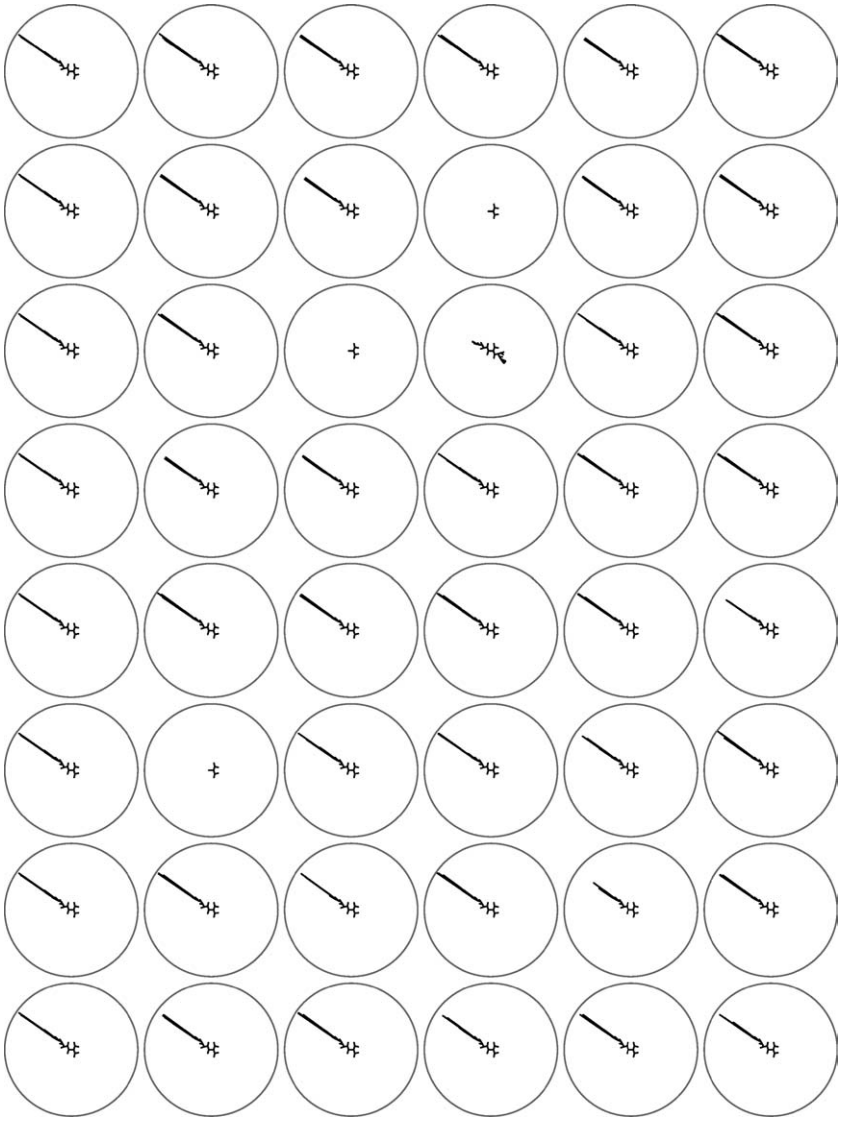

Figure 7. Sample of a population (moderate, $\alpha = 1$, fitness-proportionate selection). These are 48 representatives out of a population of 500. This problem was about 37 times easier to solve than the problem corresponding to Figure 5. Although the selection method remained the same (fitness proportionate), parameter α was tuned to a much easier setting. As a reference, a gray circle around each individual represents depth level 26.

Although the alternative meets our design criteria, there are difficulties concerning the use of a coordinate system. In our method, the coordinate system is a circular grid. We would argue that the circular grid also allows for a rapid, visual retrieval of depth information, since depth is directly correlated to radius from the center. In contrast, the alternative uses a rectangular grid. Both rectangular coordinates are needed to obtain depth information—depth is not a separable quantity for the alternative.

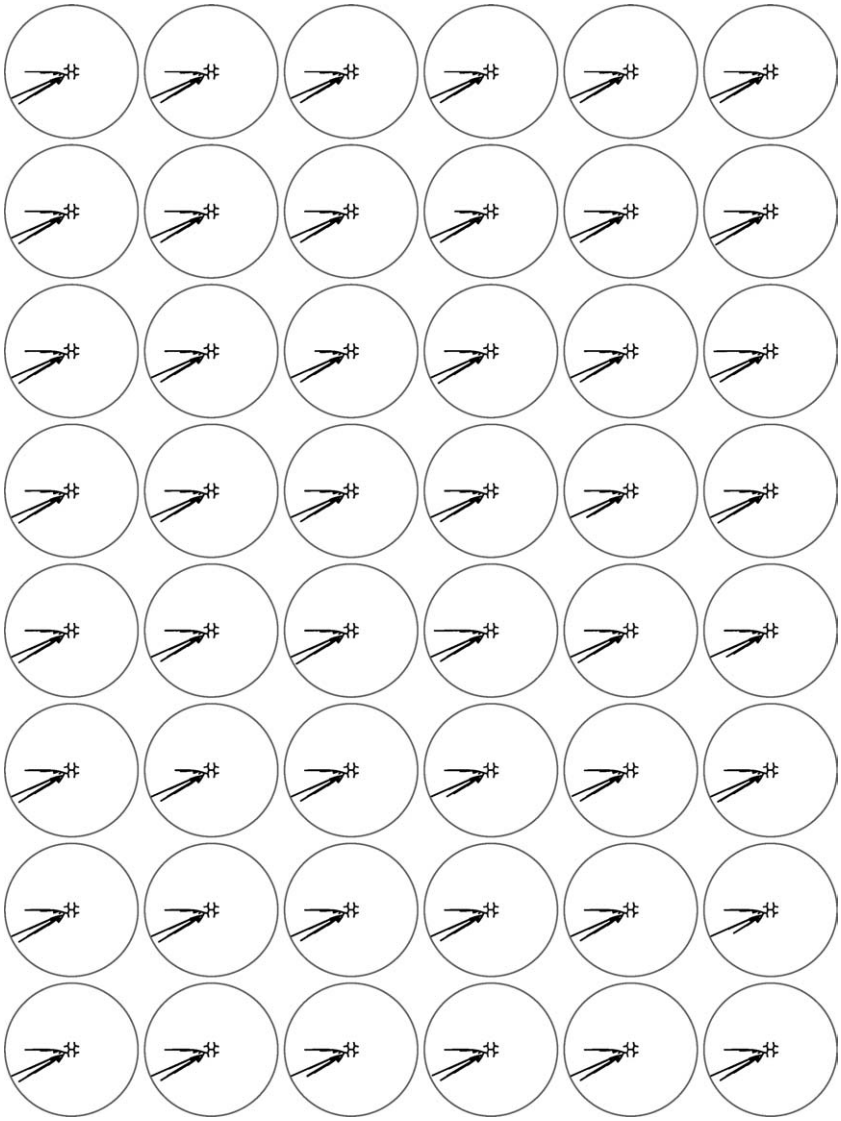

Figure 8. Sample of a population (easy, $\alpha = 1$, tournament selection). These are 48 representatives out of a population of 500. This problem was about 70 times easier to solve than the problem corresponding to Figure 5. Again, tournament selection ($n_{\text{tournament}} = 7$) was largely responsible for the performance improvement over the similar problem shown in Figure 7. As a reference, a gray circle around each individual represents depth level 26.

The alternative does not pass two of the five visualization principles. In particular, Tufte's first principle (i.e., "Above all else show the data") is violated by the alternative, as it is difficult to pick out the patterns of similar and different trees from the competing grid of horizontal and vertical lines. Tufte's fifth principle (i.e., "Forgo chartjunk") is also violated, because moiré vibration occurs between each square that corresponds to a tree.

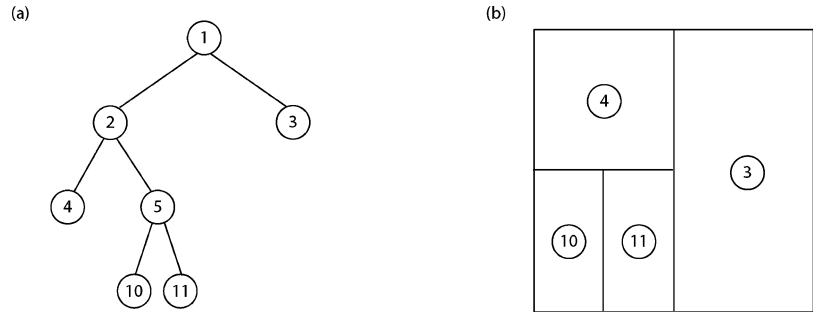

Figure 9. Mapping an arbitrary plane binary tree to a square image. (a) Arbitrary plane binary tree of depth 3. (b) Corresponding image.

4. Visualizing structure in a population

4.1. Visualization method

Although only 9.6% of their respective populations have been depicted in Figures 5–8, the high degree of structural correlation depicted in those figures suggests that the rest of their populations are also structurally correlated. We can test this possibility by starting with a cumulative distribution on L_C for an entire population, i.e.,

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

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

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

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

In other words, the (un-normalized) cumulative distribution of a population P can be treated as a sum of the vectors corresponding to the trees in that population. The vector that corresponds to each tree A is defined as a sum of unit vectors, where each unit vector corresponds to an occupied grid-point in L_C. Consequently, the tree that spans the population is described by L_P.

For example, suppose a population P consists of four binary trees that have the labels $\{1, 2, 3, 6, 7, 14, 15\}$, $\{1, 2, 3\}$, $\{1, 2, 3, 4, 5\}$, and $\{1\}$. The corresponding un-normalized cumulative distribution for this population would be $\mathbf{L}_P = 4\mathbf{i}_1 + 3\mathbf{i}_2 + 3\mathbf{i}_3 + \mathbf{i}_4 + \mathbf{i}_5 + \mathbf{i}_6 + \mathbf{i}_7 + \mathbf{i}_{14} + \mathbf{i}_{15}$, with $L_P = \{1, 2, 3, 4, 5, 6, 7, 14, 15\}$.

Visualization of a population can subsequently be done in three steps:

1. Compute $\mathbf{L}_P$.
2. Normalize each component of $\mathbf{L}_P$ by the number of individuals in population P.

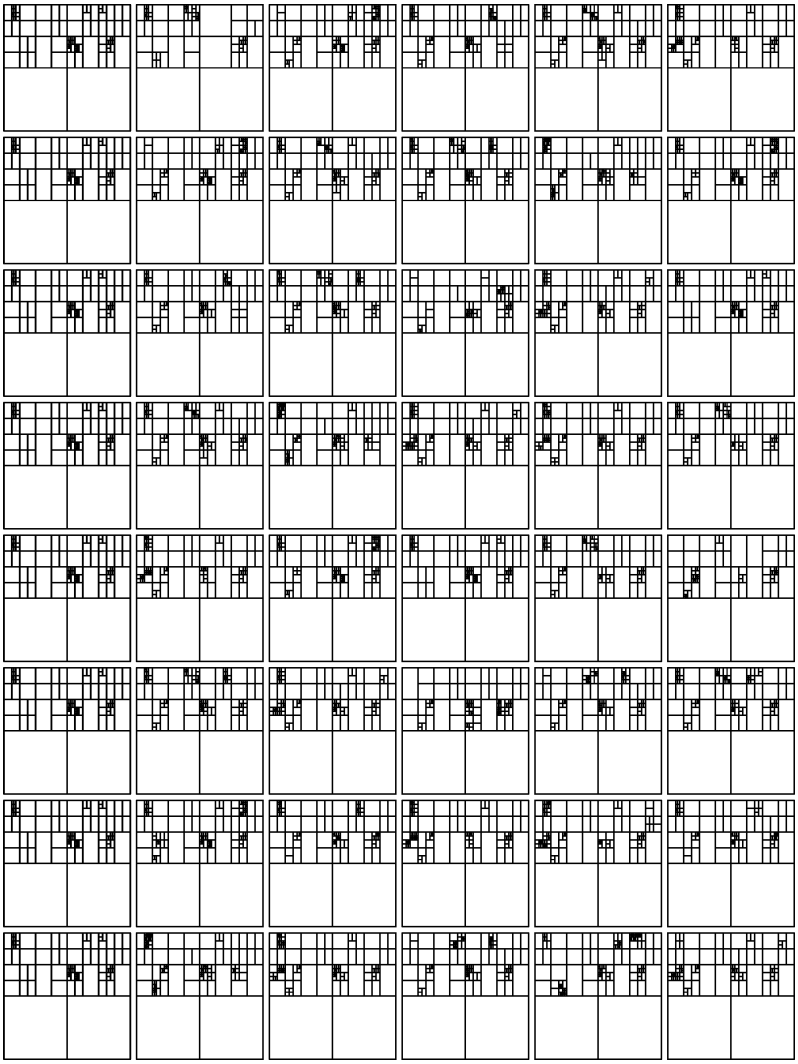

Figure 10. Image encoded sample of a population (very difficult, $\alpha = 1000$, fitness-proportionate selection). The data that is shown is identical to that given in Figure 5. The differences between individual trees, however, are not well defined with this method of visualization.

3. Construct the tree that corresponds to L_P, except that the grayscale value of each line from parent to child is proportionate to the magnitude of the component vector corresponding to that child. (Assume that higher magnitudes map to darker grays).

The resulting visualization is tantamount to overlaying all of the trees in a population. Structures that are used most often show up darker; the least used structures, lighter.

4.2. Visualization examples

Figure 11 shows the population summaries that correspond to the samples shown in Figures 5–8. Each graph summarizes the structure of 500 trees, which represents an entire population. For comparison, all populations were taken from generation 200. The more common a structure is in a population, the darker it appears in a graphic. Figure 11(a) corresponds to the population of which Figure 5 is a subset; Figure 11(b), the population of which Figure 6 is a subset; Figure 11(c), the population of Figure 7; and Figure 11(d), the population of Figure 8.

While individual trees provide insight into the degree of structural similarity within a population, they are but a hint. Even though our method of visualizing single trees enables one to see orders-of-magnitude more structures than has been possible, it is still not enough for understanding structural trends at a population level.

For those reasons, we would argue that the visualizations of Figure 11 do help in seeing some of these larger trends. In particular, we can observe the following:

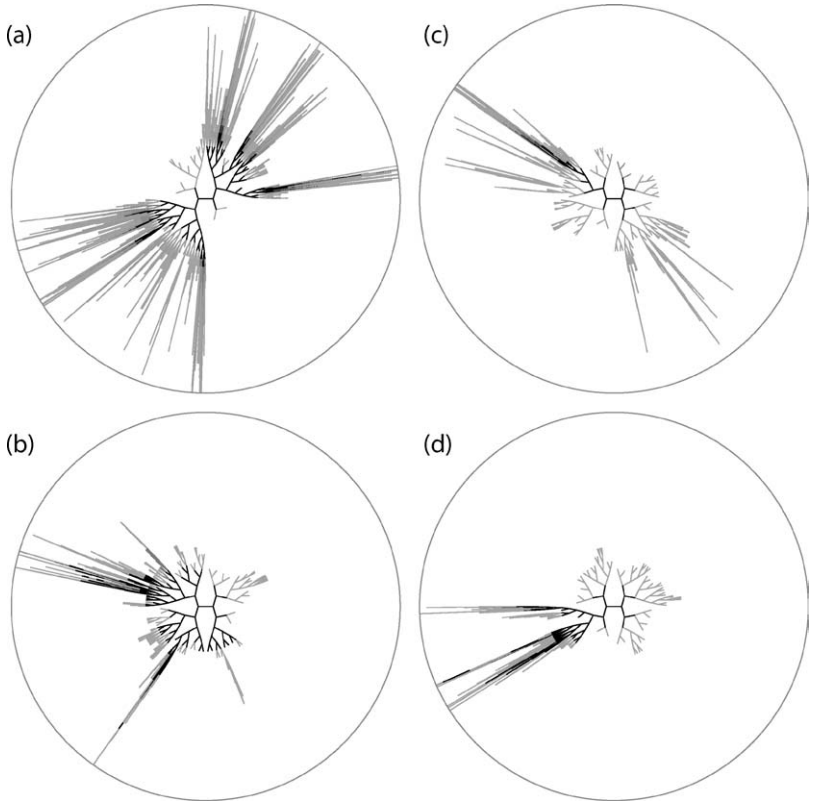

Figure 11. Visualization of populations. Each graph visualizes a population of 500 individuals. All population summaries are taken from generation 200. (a) Very difficult, $\alpha = 1000$, fitness-proportionate selection. (b) Moderate, $\alpha = 1000$, tournament selection. (c) Moderate, $\alpha = 1$, fitness-proportionate selection. (d) Easy, $\alpha = 1$, tournament selection.

- The darkest regions of each graph occur in the center and radiate outward.[4] (Darker regions represent locations of higher structural correlation.) This indicates that structures near the root are highly correlated within a population. This observation is consistent with theory and observations about the rapid convergence of root structure within GP (e.g., [44, 52, 53]).
- The lightest regions tend to occur near the edges of each graph. This means that the structures proximal to the terminals are the least likely to be found from individual to individual within a population. That they tend to be the lightest parts of each graph in Figure 11 is also consistent with theory and observations (e.g., [19]).
- Graphs that correspond to easier problems seem to exhibit less structural diversity than graphs that correspond to harder problems.

Of these three observations, the last one requires some effort in interpretation. It is somewhat arguable to judge by visual inspection alone whether a population is more or less structurally diverse than another. Fortunately, the visualization method for populations requires the computation of a cumulative distribution. We can use the information from Equations 3 and 4 to plot cumulative distribution by rank (i.e., from grid points that are the most used to grid points that are the least used). The result of this is shown in Figure 12.

Figure 12 shows the cumulative rankings that correspond to the population summaries shown in Figure 11. Each graph represents a frequency versus rank plot for a population of 500 individuals. Lattice coordinates that are used most often in a population have higher frequencies. Lattice coordinates are arranged on the x-axis according to frequency, from highest to lowest frequencies. The highest possible frequency of a lattice coordinate to be used is equal to the number of individuals in a population. For example, all trees use lattice

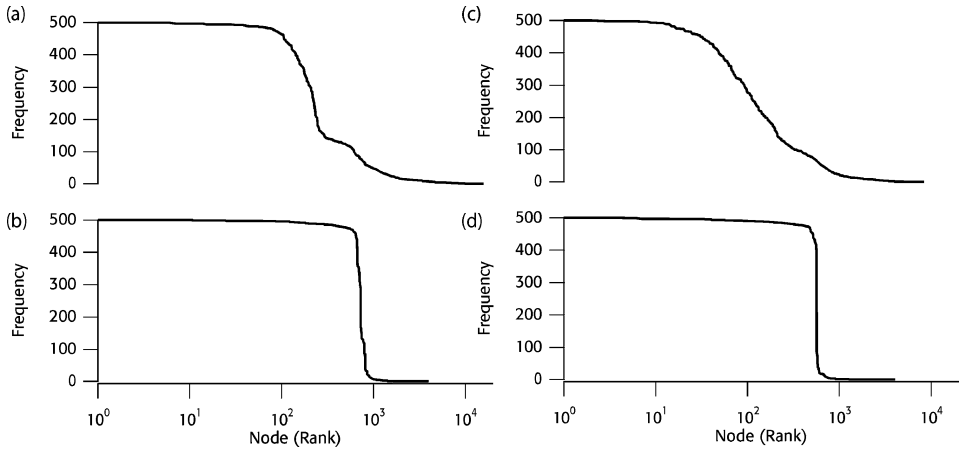

Figure 12. Cumulative rankings. Each graph represents a frequency versus rank plot for a population of 500 individuals. Lattice coordinates that are used most often in a population have higher frequencies. Lattice coordinates are arranged on the x-axis according to frequency, from highest to lowest frequencies. The specific conditions that correspond to each plot are as follows: (a) Very difficult, $\alpha = 1000$, fitness-proportionate selection. (b) Moderate, $\alpha = 1000$, tournament selection. (c) Moderate, $\alpha = 1$, fitness-proportionate selection. (d) Easy, $\alpha = 1$, tournament selection.

coordinate 1; it would have a frequency of 500. The lowest possible frequency is 0, which means that not one individual in that population uses that particular lattice coordinate. Figure 12(a) corresponds to the population summary of Figure 11(a); Figure 12(b), the population summary of Figure 11(b); Figure 12(c), the summary of Figure 11(c); and Figure 12(d), the summary of Figure 11(d).

We can observe the following from these plots:

- There are distinct differences in the distribution of rankings between selection methods. In Figure 12, the top row corresponds to fitness-proportionate selection; the bottom row, tournament selection. The plot for tournament selection shows that grid points are either occupied by most of the population or by nearly none at all—which is what one would expect for a structurally similar population. In comparison, the fitness-proportionate case is heavy-tailed and indicates that many more structures are being preserved. This finding is consistent with theoretical work in tournament selection, which argues that diversity is lost under tournament selection (e.g, [3, 4]).
- The change in distribution ranking is correlated to a change in problem difficulty. In Section 3.2, we noted that it *seemed* as if the trees were structurally less diverse with decreasing problem difficulty. Figure 12 provides quantitative evidence for that observation. Specifically, when all GP parameters are fixed with the exception of selection method, tournament selection is correlated to significantly decreased problem difficulty. Tournament selection, in turn, is correlated to an increase in structural similarity.
- Selection and structural similarity is not the only mechanism that accounts for problem difficulty. Even though Figures 12(b) and (c) are roughly equivalent in problem difficulty, their comparable distributions are distinct. In [8], we explore possible reasons for this behavior.

Figures 11 and 12 pose a number of new questions. What drives the distribution of these structures? Are these figures representative of the kinds of solutions that GP derives? Are there other statistical methods that can be used for analysis? Is there a causal link between structure and problem difficulty? Does the type of selection influence the statistics of structures that are generated? How does content correlate to structural statistics? If diversity is "good" and more diversity is "better," why is the opposite true for these examples? We leave the consideration of these questions for future investigations.

4.3. Extension to the visualization of a GP run

In [63, 64], Tufte describes the technique of small multiples to describe changes over time. We can use this technique to extend a visualization of a population to a visualization of a run.

Figures 13 and 14 show visualizations of the first 23 generations of each of the four runs that have been featured in this paper. Each run has been sampled to show a population summary for every other generation. Increasing time moves from left to right, starting from the upper left corner of a set, then progresses to the next row. Consequently for each set, the summary from generation 1 is shown in the upper left corner; the summary from generation

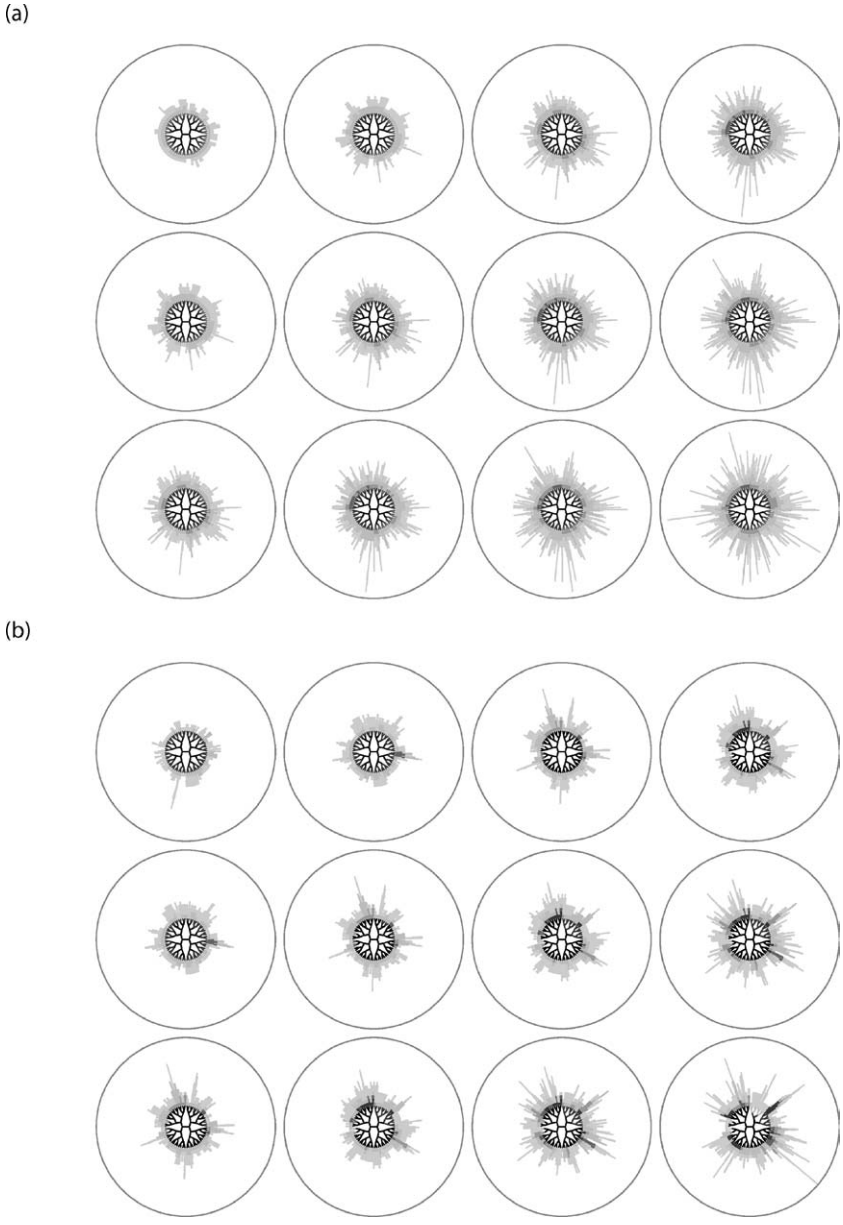

Figure 13. Visualization of the first 23 generations ($\alpha = 1000$). Each graphic represents a population summary for one generation. A run has been sampled to show a population summary for every other generation. For each set, the summary from generation 1 is shown in the upper left corner; the summary from generation 23, in the lower right corner. As a reference, a gray circle around each summary represents depth level 26. (a) Very difficult, fitness-proportionate selection. (b) Moderate, tournament selection.

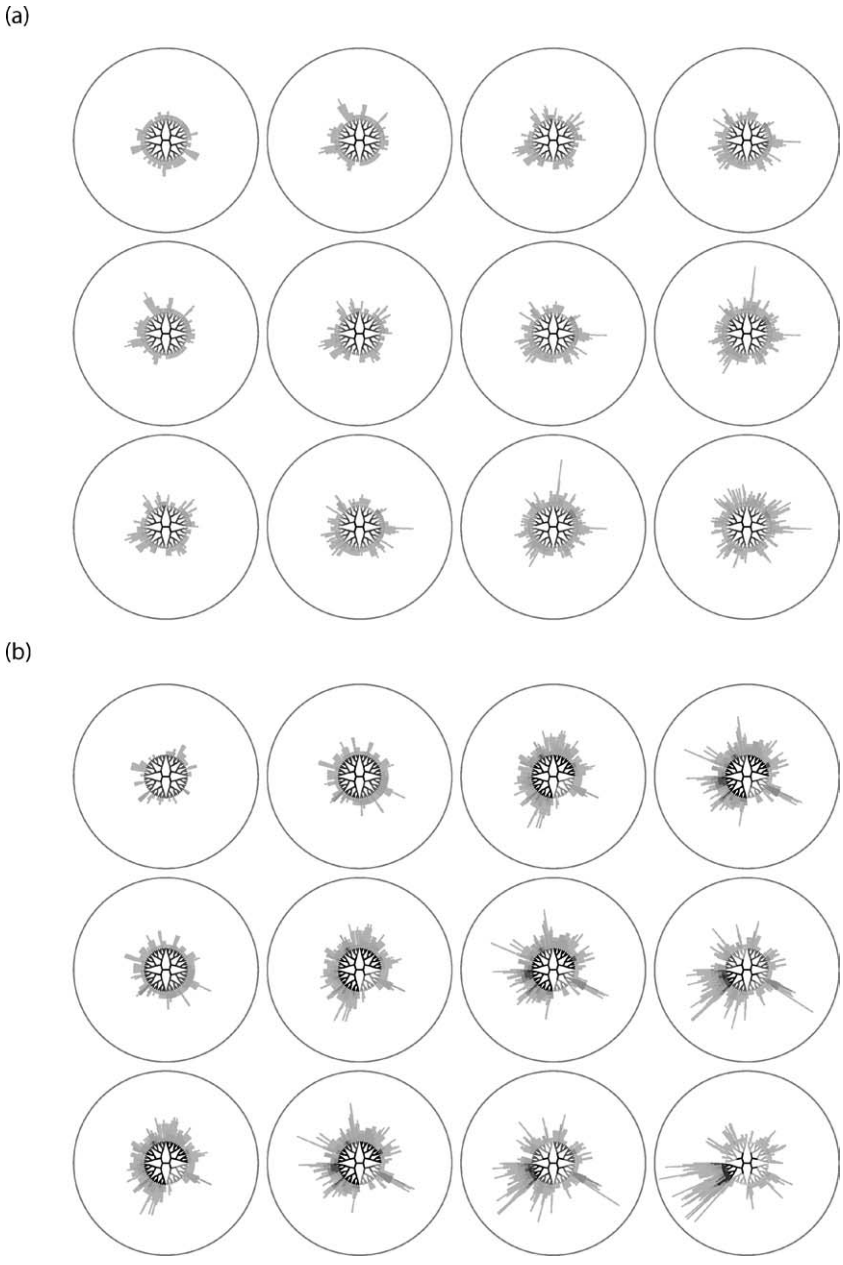

Figure 14. Visualization of the first 23 generations ($\alpha = 1$). Each graphic represents a population summary for one generation. A run has been sampled to show a population summary for every other generation. For each set, the summary from generation 1 is shown in the upper left corner; the summary from generation 23, in the lower right corner. As a reference, a gray circle around each summary represents depth level 26. (a) Moderate, fitness-proportionate selection. (b) Easy, tournament selection.

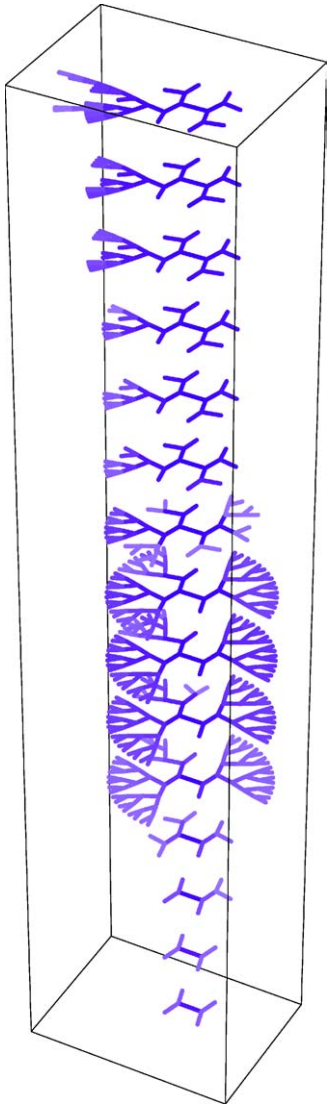

Figure 15. Visualization of the first 15 generations (easy, $\alpha = 1$, tournament selection). Each horizontal plane corresponds to a generation. Generation 1 is situated in the lowest horizontal plane. Increasing time goes upward. For simplification, we omitted lattice points in which less than half a population occupied.

23, in the lower right corner. As a reference, a gray circle around each summary represents depth level 26.

Figure 15 shows an alternative display using small multiples. Instead of a two-dimensional representation, time is used to supply a third dimension. In this instance, increasing time moves upward. Each horizontal plane of (r, θ) corresponds to a population summary for one generation. The run that corresponds to Figure 15 was sampled at each of the first 15 generations.

Again these figures pose a number of questions. What are the dynamics that shape the evolution of structure over time? How is structural symmetry of a population correlated to time? Are these time series representative of GP as a whole? How are structural dynamics correlated to problem difficulty? What is happening to structural diversity? Does the type of selection influence the kinds of structural dynamics that are generated? How does content correlate to structural dynamics? Which metrics best characterize these structural features? Again, the questions that can be raised by these visualizations are more than what can be adequately answered by this paper. We leave consideration of them to future work.

5. Conclusions and future work

"Structure always affects function."

5.1. Conclusions

This paper's opening quotation was strongly stated and independent of GP. Nevertheless, work done in the research community does suggest that structure plays a significant role in understanding how GP works. This paper has reviewed a number of investigations that concern the structure of solution outcomes under GP. In spite of these efforts in understanding the role of structure in GP, there has been a deficit in visualization methods that can depict these structures at the scales that would be of interest to the GP community.

Consequently, this paper has described two methods to visualize the structure of trees that occur in GP. The first method allows for the inspection of structure of entire trees. The second method allows for the visual inspection of structure at the level of an entire population. Both methods are amenable for further quantitative analysis. There are extensions that were described in this paper that allow for the examination of entire trials. Taken together, these methods represent an increase in the ability to examine tree structures by several orders of magnitude (i.e., an increase from tens of nodes to millions of nodes).

For completeness, we have considered a third method that met our design criteria but was ultimately not used, because it did not pass established principles for the display of quantitative information.

Examples were given from a typical class of problems in GP. We used these examples as a case study that demonstrates how these visualization algorithms can be used. For example, a typical journal-length paper may show several trees consisting of tens of nodes each. In this paper alone, we have shown visualizations corresponding to 33,742 trees (on the order of 10 million nodes), nearly all of which have not been published before. Given this gain in ability to see what is happening in GP, we have presented structures that have not been seen in any detail before. We have further indicated how such visualizations raise new questions, with the hope that these can help in the development of new investigations or to complement existing investigations in GP.

One does not have to look far to find new questions to address. If our case study was any indication, even "old" problems like tournament selection versus fitness-proportionate selection yielded results that surprise and contradict. For example, those results indicated that fitness-proportionate selection did maintain a more structurally diverse population than tournament selection—an expected result. However, more diversity was *negatively*

correlated with better problem-solving ability—a contradiction concerning diversity. While this paper did not provide much explanation for this, it does point to the possibilities of investigations that could happen when one is able to evaluate and visualize structure.

5.2. Future work

On one hand, this paper presented a small idea in visualizing a tree on a circular grid. We enjoyed many conversations with our colleagues after the conference version of this paper was presented. Many expressed that they had thought of something like this but decided not to pursue it further. The hesitation is understandable, if only because the effort in graphing just *one* tree can be nontrivial.

On the other hand, the big idea behind this paper is that structure is in and of itself a worthwhile pursuit. Unlike many other algorithms in genetic and evolutionary computation, GP uses a variable-length structure that changes during the course of a run. For many, this would represent just implementation detail. Nevertheless, the implications of having a variable-length structure may be significant, given the amount and degree of patterning that has been indicated here. The effort that is needed to graph *just* the structure of a tree may be worthwhile after all.

What has been presented in this paper represents only a few of the reasons to visualize structures. The reviewers suggested several more applications. For example, one reviewer held that our visual analysis could be used in real-time to monitor trends during a run. Another suggestion was to use our methods to figure out why something is not working, which in that reviewer's opinion was the *strongest* reason to visualize. Other suggestions included looking at other problems like Goldberg and O'Reilly's ORDER and MAJORITY, McPhee's work on diversity, or Langdon's work on bloat. Still other suggestions included expanding the visualization method to more than just binary trees or incorporating an examination of content with visualizations of structure. There are undoubtedly still other avenues of future work—much of which is left for others to try.

Appendix A: Software implementation

The visualization techniques used in this paper are not a part of any commercial packages of which we are aware. That being said, the techniques are relatively simple and can be coded in any language that allows for the display and manipulation of graphics primitives. Open GL, visual extensions to Python (e.g., VBL), *Matlab*, and *Mathematica* come to mind. One of my students (i.e., M. Samples) managed to hack GNU Plot to display trees. S. Gustafson showed me examples of his implementation based on [7], which was impressive since our description of our visualization method was minimal in that paper. Apparently, one of the referees to [11] implemented our techniques and managed to verify some of our results during that paper's review period.

There are several caveats, though, in implementing our visualization techniques.

- The techniques have only been fully described for complete oriented binary trees. We are aware of several workarounds that allow for the display of incomplete m-ary trees. Some of these workarounds are simple and allow for the visualization of trees without

having to compute for lattice coordinates. Other workarounds require some effort, since there are properties of an m-ary lattice that are not apparent in this work.

- Deep trees mean that large integers are needed for lattice coordinates. For example, a depth n tree may require the specification of coordinates up to $2^{n+1} - 1$. Subsequently, for most machines and software, this implies a maximum depth of 31. It is partly for this reason that we chose to do our original implementation in *Mathematica*, since this software allows for integers of arbitrary precision. It is also for this reason that some of our non-*Mathematica* software for visualization incorporates the GNU Multiple Precision Arithmetic Library [21].
- The visualization of populations can be memory intensive. We made no mention of the storage requirements that our visualizations use, if only because data visualization represents only an aspect of our investigation into GP dynamics. Our interests have meant that we use visualization as a post-processing exploration tool for terabyte-sized data sets. We note that the size of our data sets are driven by the amount of detail that is captured for study, only some of which is germane to the visualization techniques that are given in this paper. We would be interested in hearing from others about the tweaks, concessions, and modifications that are needed for using our visualization techniques in real-time.

A version of our code [12] is located at Wolfram's *MathSource* site (http://library.wolfram.com/infocenter/MathSource/5163). It includes documentation, our code, and a number of examples. The code assumes that trees have been converted into lattice coordinates. Unfortunately, we did *not* include a parser that converts trees into lattice coordinates. Parsers tend to be specific to particular GP kernels, specific problems, and idiosyncratic formatting conventions. Our own parser was coded in `lex` and `yacc` and was wrapped in PERL. We included a set of sample output from our parser, so that one could see the particular format the *Mathematica* code uses for inputting lattice coordinates. The code is complete enough so that researchers can visualize their own trees.

The use of our code does require *Mathematica* if one wants to visualize one's own trees. For those who would like to just view the notebook, Wolfram makes available a free reader (i.e., http://www.wolfram.com/products/mathreader). Modifications to the code do require knowledge of *Mathematica* programming. If one is not familiar with this language, we recommend [22, 66] as resources. We also recommend Jacob's book [28], which includes many other *Mathematica* examples that are specific to genetic and evolutionary computation.

Appendix B: Historical antecedents

The visualization of trees as we have described has a history that extends well over a century. It encompasses different sciences and different branches of mathematics. Different names are ascribed to these trees. However, in spite of this history, the techniques that have been described in this paper seem to be new for trees in general and not just in GP. The purpose for this appendix, then, is to summarize these historical antecedents and to place our work in this expanded context.

Inverted, rooted trees, like those in Figure 1(a), belong to the class of information graphics called *tree* (or *dendrite*) *diagrams*. Closely associated with such diagrams are *dendrograms* (also called *linkage trees*), which presume that some sort of clustering has been imposed

on nodes. (See [23]). *Phylogenetic trees*, which appear in biology (e.g., [43]), are a form of dendrogram that specifically address the genetic lineages of organisms. Both tree diagrams and dendrograms can appear in *circular* form, in which nodes of these graphs are arranged along concentric circles. They can also appear in *radial* form, in which nodes are arranged along "spokes" that emanate from a common node (e.g., [42], p. 8). Dendrograms can be fairly large and can have several hundred nodes. (Apparently for that reason, Jacob [28] adapts dendrograms for use in GP.) However, large dendrograms are rare since they typically require a fair amount of real estate.

Trees are not just diagrams, though. In graph theory, trees can be treated as mathematical entities that have specific properties. This paper's method bears some resemblance to several different types of tree graphs: a *star graph*, a *caterpillar graph*, a *Cayley tree*, and a *Steiner tree*. Each of these has a specific topology that can allow for a radial organization of nodes. Of these graphs, Steiner trees have received a significant amount of attention in engineering and computer science (see [26, 27]).

In some sense, trees do not even have to be "trees" to be considered germane to this paper. For example, this paper's methods have historical antecedents[5] that pre-date the mathematical formulation of trees (i.e., [31]). As it turns out, some small Steiner trees look like a few of the visualizations prepared by this paper's method. Proper attribution, however, does not go to Steiner. The term *Steiner trees* is traceable to [6], which mistakenly credited a nineteenth century Jakob Steiner with the mathematical entities that now bear his name. Proper attribution for this paper's antecedents goes at least to Gauss, who wrote a letter describing a problem that would lead to such a tree on March 21, 1836, (see [56]). It could be argued, too, that proper attribution could also go to Fermat (c. 1646), who proposed the problem that led to a solution by Torricelli. Torricelli's solution also looks like one of the visualizations that have been prepared by this paper's methods.

Part of the reason why Gauss's and Fermat's works pre-date Kirchoff's formal work on trees is because these works were considered as investigations of networks. Both networks and trees can be displayed using similar methods, as mentioned in [67]. Coincidently, one such mapping of a network bears a close resemblance to our methods for visualizing a complete 3-ary tree (see [67], pp. 479*ff*.) Methods for undirected graphs/networks also include ways to minimize "total energy" (e.g., [18, 29]), which have been useful in the examination of scale-free networks. Other methods for visualizing networks can be found in [30].

Our methods are distinguished from this body of historical work in the following manner:

- Both dendrograms and circular trees can and have been used to organize nodes in both circular and radial directions. Our visualizations can be considered a kind of circular tree, except that an absolute coordinate system for node occupancy is implied. Our coordinate system represents a contribution that allows for both a quantitative and visual comparison between trees.
- Star graphs, caterpillar graphs, and Cayley trees also allow for the organization of nodes in both circular and radial directions. Furthermore, all have precise, mathematical formulations that allow for the creation of an absolute coordinate system. However, star graphs and caterpillar graphs are restrictive about what nodes can be interconnected; many simple trees in GP would be difficult to map using these graphs. We use Cayley trees as a basis for our coordinate system. However, we further specify a function that

bijectively maps positive integer sequences to absolute coordinates on a Cayley tree. We also specify a circular coordinate system for the purpose of visual display.

- Steiner trees are more coincidental than applicable to our work. Steiner trees are solutions to optimization problems in networks. A few of these solutions for smaller networks look like visualizations for small trees. Larger Steiner trees increasingly diverge from any resemblance to the kinds of visualizations that are possible with our methods. This is not surprising since Steiner trees and our visualization methods represent solutions to distinct problems.
- Network layout and visualization are possible candidates for later consideration. However, they are not discussed in this paper, in part because methods of displaying trees can be considered a subset of those methods that can be used to display networks. Other methods—such as total energy minimization—were not considered because of a need for an absolute coordinate system for nodes.

Finally, there are other visualization methods (see [24]) that have not been considered in this paper that we believe should merit further consideration by those in GP. In particular, the developments in information visualization are relevant, some of which bear a close resemblance to our visualization of individual trees. This would include methods such as focus+context (i.e., fish-eye) [36], which involves visualizing trees on a hyperbolic surface instead of a circular grid. Much of the work in information visualization has concentrated on single, albeit large, hierarchies. Our methods for visualizing populations and trials should be compatible with these other methods in information visualization and should subsequently allow for hybrid methods that are rich in potential.

Acknowledgments

We thank J. Foster and E. Cantú-Paz for their kind invitation to this issue of GPEM. We also thank the following individuals and organizations for their help: CSCS U-M/Santa Fe Institute Fall Workshops, L.M. Sander, S. Stanhope, J. Polito 2, P. Litvak, S. Yalcin. D. Maclean, W. Worzel, M. Samples, S. Larson, M. Hodges, J. Kriesel, and UMACERS teams *Magic*, *Meta-Edge*, *Royal*, *Niihau*, and *Binomial-3*. C. Jacob's encouragement at GECCO '03 was much appreciated. We thank the reviewers of both this article and its shorter GECCO '03 counterpart. The first author also acknowledges I. Kristo and S. Daida.

Notes

1. In Chapter 2 of [34], Knuth spends the first half discussing fundamental information structures (like trees). Beginning with Section 2.4, Knuth then builds upon these fundamental structures by relating how these structures would go about representing information.
2. Even so, Knuth adjusted convention slightly by laying each tree on its side.
3. Preorder traversal is defined recursively and is described in [34], p. 319. There are three steps that are taken: visit the root; traverse the left subtree; traverse the right subtree.
4. In some cases, there may appear isolated dark regions that occur within light regions. Such regions are typically artifacts, which occur because visible lines can be made only so thin and subsequently some unintentional occlusion happens.
5. We acknowledge [16] for tracing the history concerning Steiner, Gauss, Fermat, and Torricelli.

References

1. P. J. Angeline, "Parse trees," in Handbook of Evolutionary Computation, T. Bäck, D. B. Fogel, and Z. Michalewicz (Eds.), Institute of Physics Publishing: Bristol, 1997, pp. C1.6:1–C1.6:3.
2. W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and Its Applications, Morgan Kaufmann Publishers: San Francisco, 1998.
3. T. Blickle, "Tournament selection," in Handbook of Evolutionary Computation, T. Bäck, D. B. Fogel, and Z. Michalewicz (Eds.), Institute of Physics Publishing: Bristol, 1997, pp. C2.3:1–C2.3:4.
4. T. Blickle and L. Thiele, "A mathematical analysis of tournament selection," in ICGA95: Proceedings of the Sixth International Conference on Genetic Algorithms, July 15–19, Pittsburgh, L. J. Eshelman (Ed.), Morgan Kaufmann Publishers: San Francisco, 1995, pp. 9–16.
5. O. A. Chaudhri, J. M. Daida, J. C. Khoo, W. S. Richardson, R. B. Harrison, and W. J. Sloat, "Characterizing a tunably difficult problem in genetic programming," in GECCO 2000: Proceedings of the Genetic and Evolutionary Computation Conference, July 10–12, 2000, Las Vegas, L. D. Whitley, D. E. Goldberg, E. Cantú-Paz, L. Spector, I. Parmee, and H.-G. Beyer (Eds.), Morgan Kaufmann Publishers: San Francisco, 2000, pp. 395–402.
6. R. Courant, What is Mathematics? An Elementary Approach to Ideas and Methods. Oxford University Press: London, 1941.
7. J. M. Daida, "Limits to expression in genetic programming: Lattice-aggregate modeling," in The 2002 IEEE World Congress on Computational Intelligence: Proceedings of the 2002 Congress on Evolutionary Computation, May 12–17, Honolulu, Hawaii. 2002, IEEE: Piscataway, 2002, pp. 273–278.
8. J. M. Daida, "What makes a problem GP-Hard? A look at how structure affects content," in Theory and Applications in Genetic Programming, R. L. Riolo and W. Worzel (Eds.), Kluwer Academic Publishers: Dordrecht, 2003, pp. 99–118.
9. J. M. Daida, R. B. Bertram, J. A. Polito 2, and S. A. Stanhope, "Analysis of single-node (building) blocks in genetic programming," in Advances in Genetic Programming 3, L. Spector, W. B. Langdon, U.-M. O'Reilly, and P. J. Angeline (Eds.), The MIT Press: Cambridge, 1999, pp. 217–241.
10. J. M. Daida and A. Hilss, "Identifying structural mechanisms in standard genetic programming," in Genetic and Evolutionary Computation—GECCO 2003: Genetic and Evolutionary Computation Conference, Chicago, IL, USA, July 2003, E. Cantú-Paz, J. A. Foster, K. Deb, L. D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. A. Dowsland, and N. J. J. Miller (Eds.), Springer-Verlag: Berlin, 2003, pp. 1639–1651.
11. J. M. Daida, A. Hilss, D. J. Ward, and S. Long, "Visualizing tree structures in genetic programming," in Genetic and Evolutionary Computation—GECCO 2003: Genetic and Evolutionary Computation Conference, Chicago, IL, USA, July 2003, E. Cantú-Paz, J. A. Foster, K. Deb, L. D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. A. Dowsland, and N. J. J. Miller (Eds.), Springer-Verlag: Berlin, 2003, pp. 1652–1664.
12. J. M. Daida, A. M. Hilss, D. J. Ward, and S. L. Long, Visualization of Tree Structures: Examples from Genetic Programming and A Model of Tree Growth. The University of Michigan: Ann Arbor, 2004.
13. J. M. Daida, H. Li, R. Tang, and A. Hilss, "What makes a problem GP-hard? validating a hypothesis of structural causes," in Genetic and Evolutionary Computation—GECCO 2003: Genetic and Evolutionary Computation Conference, Chicago, IL, USA, July 2003, E. Cantú-Paz, J. A. Foster, K. Deb, L. D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. A. Dowsland, and N. J. J. Miller (Eds.), Springer-Verlag: Berlin, 2003, pp. 1665–1677.
14. J. M. Daida, J. A. Polito 2, S. A. Stanhope, R. R. Bertram, J. C. Khoo, S. A. Chaudhary, and O. Chaudhri, "What makes a problem GP-hard? Analysis of a tunably difficult problem in genetic programming," Genetic Programming and Evolvable Machines, vol. 2, no. 2, pp. 165–191, 2001.
15. K. Deb, "Introduction [to Issues in Representations]," in Handbook of Evolutionary Computation, T. Bäck, D. B. Fogel, and Z. Michalewicz (Eds.), Institute of Physics Publishing: Bristol, 1997, pp. C1.1:1–C1.1:4.
16. D.-Z. Du, P. M. Pardalos, and W. Wu, Mathematical Theory of Optimization. 273 pp. ed. Nonconvex Optimization and Its Applications, Kluwer Academic Publishers: Dordrecht, 2001.

17. D. B. Fogel and P. J. Angeline, "Guidelines for a suitable encoding," in Handbook of Evolutionary Computation, T. Bäck, D. B. Fogel, and Z. Michalewicz (Eds.), Institute of Physics Publishing: Bristol, 1997, pp. C1.7:1–C1.7:2.
18. T. M. J. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," Software, Practice and Experience, vol. 21, pp. 1129–1164, 1991.
19. C. Gathercole and P. Ross, "An adverse interaction between crossover and restricted tree depth in genetic programming," in Genetic Programming 1996: Proceedings of the First Annual Conference: July 28–31, 1996, Stanford University, J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo (Eds.), The MIT Press: Cambridge, 1996, pp. 291–296.
20. D. E. Goldberg and U.-M. O'Reilly, "Where does the good stuff go, and why?," in Proceedings of the First European Conference on Genetic Programming, Paris, France, W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty (Eds.), Springer-Verlag: Berlin, 1998, pp. 16–36.
21. T. Granlund, GNU MP: The GNU Multiple Precision Library, The Free Software Foundation, Inc.: Boston, 2002.
22. J. Gray, Mastering Mathematica: Programming Methods and Applications. 2nd editors, Academic Press: San Diego, 1998, p. 629.
23. R. L. Harris, Information Graphics: A Comprehensive Illustrated Reference: Visual Tools for Analyzing, Managing, and Communicating, Oxford University Press: New York, 1999.
24. I. Herman, G. Melançon, and M. S. Marshall, "Graph visualization and navigation in information visualization: A survey," IEEE Transactions on Visualization and Computer Graphics, vol. 6, no. 1, pp. 24–43, 2000.
25. G. M. Hunter and K. Steiglitz, "Operations on images using quad trees," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-1, no. 2: pp. 145–153, 1979.
26. F. Hwang, D. Richards, and P. Winter, The Steiner Tree Problem, North-Holland: Amsterdam, Netherlands, 1992.
27. A. O. Ivanov and A. A. Tuzhilin, Minimal Networks: The Steiner Problem and Its Generalizations, CRC Press: Boca Raton, FL, 1994.
28. C. Jacob, Illustrating Evolutionary Computation with Mathematica, Morgan Kaufmann Publishers: San Francisco, 2001, p. 578.
29. T. Kamada and S. Kawai, "An algorithm for drawing general undirected graphs," Information Processing Letters, vol. 31, pp. 7–15, 1989.
30. P. R. Keller and M. M. Keller, Visual Cues: Practical Data Visualization, IEEE Press: Piscataway, NJ, 1993.
31. G. Kirchoff, Annalen der Physik und Chemie, vol. 72, pp. 497–508, 1847.
32. A. Klinger, "Patterns and search statistics," in Optimizing Methods in Statistics, J. S. Rustagi (Ed.), Academic: New York, 1971, pp. 303–337.
33. A. Klinger and C. R. Dyer, "Experiments on picture representation using regular decomposition," Computer Graphics and Image Processing, vol. 5, pp. 68–105, 1976.
34. D. E. Knuth, The Art of Computer Programming: Volume 1: Fundamental Algorithms. 3rd edition, Vol. 1. Reading: Addison–Wesley, 1997.
35. J. R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection. Complex Adaptive Systems, The MIT Press: Cambridge, 1992.
36. J. Lamping, R. Rao, and P. Pirolli, "Focus+context based on hyperbolic geometry for visualizaing large hierarchies," in Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems Part 1 of 2, May 7–11, 1995, Denver, CO., ACM: New York, 1995, pp. 401–408.
37. W. B. Langdon, "Quadratic bloat in genetic programming," in GECCO 2000: Proceedings of the Genetic and Evolutionary Computation Conference, July 10–12, 2000, Las Vegas, L. D. Whitley, D. E. Goldberg, E. Cantú-Paz, L. Spector, I. Parmee, and H.-G. Beyer (Eds.), Morgan Kaufmann Publishers: San Francisco, 2000, pp. 451–458.
38. W. B. Langdon, "Size fair and homologous tree crossovers for tree genetic programming," Genetic Programming and Evolvable Machines, vol. 1, nos. 1/2, pp. 95–119, 2000.
39. W. B. Langdon and R. Poli, "Fitness causes bloat," in Soft Computing in Engineering Design and Manufacturing, P. K. Chawdhry, R. Roy, and R. K. Pant (Eds.), Springer-Verlag: London, 1997, pp. 23–27.

40. W. B. Langdon and R. Poli, Foundations of Genetic Programming, Springer-Verlag: Berlin, 2002.
41. W. B. Langdon, T. Soule, R. Poli, and J. A. Foster, "The evolution of size and shape," in Advances in Genetic Programming 3, L. Spector, W. B. Langdon, U.-M. O'Reilly, and P. J. Angeline (Eds.), The MIT Press: Cambridge, 1999, pp. 163–190.
42. L. Margulis and K. V. Schwartz, Five Kingdoms: An Illustrated Guide to the Phyla of Life on Earth. Third edition, W.H. Freeman and Company, New York, 1999.
43. A. R. Mast, S. Kelso, A. J. Richards, D. J. Lang, D. M. S. Feller, and E. Conti, "Phylogenetic Relationships in *Primula* L. and Related Genera (Primulaceae) Based on Noncoding Chloroplast DNA," International Journal of Plant Science, vol. 162, no. 6, pp. 1381–1400, 2001.
44. N. F. McPhee and N. J. Hopper, "Analysis of genetic diversity through population history," in GECCO '99: Proceeding of the Genetic and Evolutionary Computation Conference, 13–17 July 1999, Orlando, W. Banzhaf, J. M. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith (Eds.), Morgan Kaufmann Publishers: San Francisco, 1999, pp. 1112–1120.
45. M. Mitchell, S. Forrest, and J. H. Holland, "The royal road for genetic algorithms: fitness landscapes and GA performance," in Proceedings of the First European Conference on Artificial Life, Toward a Practice of Autonomous Systems, F. J. Varela and P. Bourgine (Eds.), The MIT Press: Cambridge, 1992, pp. 245–254.
46. P. Nordin, Evolutionary Program Induction of Binary Machine Code and Its Applications. 1997, der Universitat Dortmund am Fachereich Informatik: Dortmund.
47. U.-M. O'Reilly and D. E. Goldberg, "How fitness structure affects subsolution acquisition in genetic programming," in Genetic Programming 1998: Proceedings of the Third Annual Conference, July 22–25, 1998, University of Wisconsin, Madison, J. 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 (Eds.), Morgan Kaufmann Publishers: San Francisco, 1998, pp. 269–277.
48. R. Poli, "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, M. Tomassini, P. L. Lanzi, C. Ryan, G. B. Tettamanzi, and W. B. Langdon (Eds.), Springer-Verlag: Berlin, 2001, pp. 143–159.
49. R. Poli and W. B. Langdon, "A new schema theory for genetic programming with one-point crossover and point mutation," in Genetic Programming 1997: Proceedings of the Second Annual Conference, July 13-16, 1997, Stanford University, J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo (Eds.), Morgan Kaufmann Publishers: San Francisco, 1997, pp. 279–85.
50. R. Poli and W. B. Langdon, "Schema theory for genetic programming with one-point crossover and point mutation," Evolutionary Computation, vol. 6, no. 3, pp. 231–252, 1998.
51. W. Punch, D. Zongker and E. Goodman, "The royal tree problem, a benchmark for single and multiple population genetic programming," in Advances in Genetic Programming, P. J. Angeline and J. K.E. Kinnear (Eds.), The MIT Press: Cambridge, 1996, pp. 299–316.
52. J. P. Rosca, "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, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo (Eds.), Morgan Kaufmann Publishers: San Francisco, 1997, pp. 286–94.
53. J. P. Rosca and D. H. Ballard, "Rooted-tree schemata in genetic programming," in Advances in Genetic Programming 3, L. Spector, W. B. Langdon, U.-M. O'Reilly, and P. J. Angeline (Eds.), The MIT Press: Cambridge, 1999, pp. 243–271.
54. H. Samet, "Neighbor finding techniques for images represented by quadtrees," Computer Graphics and Image Processing, vol. 18, pp. 37–57, 1980.
55. H. Samet and M. Tamminen, "Computing geometric properties of images represented by linear quadtrees," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-7, no. 2, pp. 229–240, 1985.
56. P. Schreiber, "On the history of the so-called Steiner Weber problem." Wiss. Z. Erust-Monitz-Arndt-Univ. Greifswald, Math.-nat.wiss. Reihe, vol. 35, no. 3, 1986.
57. T. Soule and J. A. Foster, "Code size and depth flows in genetic programming," in Genetic Programming 1997: Proceedings of the Second Annual Conference, July 13-16, 1997, Stanford University, J. R. Koza, K.

Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo (Eds.), Morgan Kaufmann Publishers: San Francisco, 1997, pp. 313–320.
58. T. Soule and J. A. Foster, "Removal bias: A new cause of code growth in tree based evolutionary programming," in The 1998 IEEE International Conference on Evolutionary Computation Proceedings: IEEE World Congress on Computational Intelligence. IEEE Press: Piscataway, 1998, pp. 781–786.
59. T. Soule, J. A. Foster, and J. Dickinson, "Code growth in genetic programming," in Genetic Programming 1996: Proceedings of the First Annual Conference: July 28–31, 1996, Stanford University, J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo (Eds.), The MIT Press: Cambridge, 1996, pp. 215–223.
60. R. P. Stanley, Enumerative Combinatorics I, Cambridge Studies in Advanced Mathematics, ed. W. Fulton, D. J. H. Garling, K. Ribet and P. Walters. Vol. 1. Cambridge University Press: Cambridge: 1997.
61. R. P. Stanley, Enumerative Combinatorics II. Cambridge Studies in Advanced Mathematics, W. Fulton, D. J. H. Garling, K. Ribet, and P. Walters (Eds.), vol. 2, Cambridge University Press: Cambridge, 1999.
62. S. H. Strogatz, "Exploring complex networks," Nature, vol. 410, pp. 268–276, 2001.
63. E. R. Tufte, Envisioning Information, Graphics Press: Cheshire, CT, 1990.
64. E. R. Tufte, The Visual Display of Quantitative Information. Graphics Press: Cheshire, CT, 1983.
65. E. R. Tufte, Visual Explanations: Images and Quantities, Evidence and Narrative. Graphics Press: Cheshire, CT, 1997.
66. T. Wickham-Jones, Mathematica Graphics: Techniques and Applications, TELOS: New York, 1994.
67. S. Wolfram, A New Kind of Science, Wolfram Media, Inc: Champaign, IL, 2002.
68. D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," IEEE Transactions on Evolutionary Computation, vol. 1, no. 1, pp. 67–82, 1997.