Visualizing Genetic Programming Ancestries

Nicholas Freitag McPhee
Div. of Science and Math
Univ. of Minnesota, Morris
Morris, MN USA-56267
mcphee@morris.umn.edu

Maggie M. Casale
Div. of Science and Math
Univ. of Minnesota, Morris
Morris, MN USA-56267
casal033@morris.umn.edu

Mitchell Finzel
Div. of Science and Math
Univ. of Minnesota, Morris
Morris, MN USA-56267
finze008@morris.umn.edu

Thomas Helmuth
Computer Science Dep't
Washington and Lee Univ.
Lexington, VA USA-24450
helmutht@wlu.edu

Lee Spector
Cognitive Science
Hampshire College
Amherst, MA USA-01002
lspector@hampshire.edu

ABSTRACT

Previous work has demonstrated the utility of graph databases as a tool for collecting, analyzing, and visualizing ancestry in evolutionary computation runs. That work focused on sections of individual runs, whereas this paper illustrates the application of these ideas on the entirety of large runs (up to three hundred thousand individuals) and combinations of multiple runs. Here we use these tools to generate graphs showing *all* the ancestors of successful individuals from a variety of stack-based genetic programming runs on software synthesis problems. These graphs highlight important moments in the evolutionary process. They also allow us to compare the dynamics for successful and unsuccessful runs. As well as displaying these full ancestry graphs, we use a variety of standard techniques such as size, color, pattern, labeling, and opacity to visualize other important information such as fitness, which genetic operators were used, and the distance between parent and child genomes. While this generates an extremely rich visualization, the amount of data can also be somewhat overwhelming, so we also explore techniques for filtering these graphs that allow us to better understand the key dynamics.

1. INTRODUCTION

Reporting of results in genetic programming (GP) and evolutionary computation (EC) research is frequently limited to aggregate statistics such as mean best fitness or percentage of successful runs. Unfortunately this fails to convey the complex dynamics of such evolutionary systems and obscures or omits potentially valuable information about *why* the runs behaved as they did. While it's clearly valuable to know that Approach A is "better" in some sense (e.g., more successes) than Approach B, it's also valuable to understand *why* it succeeds more often, a question that summary statistics rarely shed any light on.

One way to move past the limitations of summary statistics is to collect ancestry information on runs, recording and analyzing parent–child relationships [3, 4, 12]. Most previous ancestry work in EC has been limited to fairly small datasets, however, in part because of the challenges of storing and working effectively and efficiently with the hundreds of thousands of ancestry relationships present in most EC runs.

Previous work [9] has shown the utility of graph databases as tools for collecting and analyzing large collections of ancestry data from GP runs, helping to identify key moments in runs, and general behavioral trends. That work, however, was focused on sections of individual runs. In this paper we illustrate the use of these tools as a means of visualizing and exploring entire ancestry trees, as well as combinations of ancestry trees. We use the Titan graph database[1] along with the Tinkerpop query tools[2] to store the parent-child relationships from genetic programming runs, and to extract the ancestry trees of specified individuals. We then visualize these subgraphs using the Graphviz `dot` graph layout tool.[3]

In the next section we will describe the test environment used to generate the data used in this paper. Section 3 describes the basic graph structure used in these visualizations, including detailed descriptions of how the rendering of edges and nodes conveys additional information about the individuals and run dynamics. The initial graphs are large enough to be rather overwhelming, so in Section 4 we describe ways to filter these large graphs into more comprehensible subgraphs. Section 5 provides examples of our ability to compare multiple runs side-by-side, illustrating similarities and differences in the run dynamics. We wrap up with some ideas for future work in Section 6 and conclusions in Section 7.

2. OUR TEST ENVIRONMENT

All the visualizations presented here are on runs using the Clojush implementation[4] of the PushGP genetic programming (GP) system [10, 11], but all of the ideas here could easily apply to almost any evolutionary computation (EC) system. The use of graph databases to capture ancestor lineages is a completely general concept and could be applied to any system that implements a notion of descent with modification. Some of the visualization specifics are tied to particular metrics (e.g., the Damerau–Levenshtein distance between linear genomes), but these could be replaced with other metrics as appropriate to the application domain (e.g., manhattan distances between genetic algorithms bitstrings, or tree edit distances in tree-based GP).

GECCO'16 Companion, July 20 - 24, 2016, Denver, CO, USA

ISBN 978-1-4503-4323-7/16/07…$15.00

DOI: http://dx.doi.org/10.1145/2908961.2931741

[1] http://thinkaurelius.github.io/titan/
[2] https://tinkerpop.apache.org/
[3] http://www.graphviz.org/
[4] https://github.com/lspector/Clojush

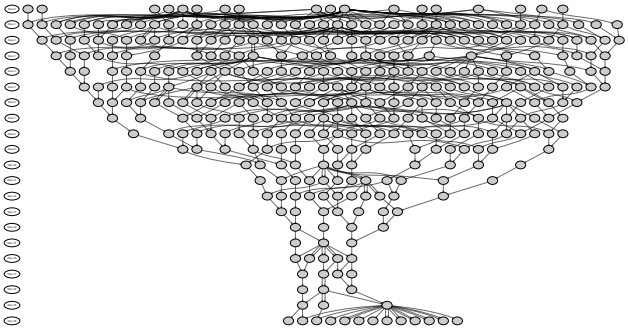

Figure 1: Basic layout of ancestry trees, showing parent-child relationships. Generations are labeled on the left-hand side and run from generation 0 (the start of the run) at the top down to the end of the run at the bottom.

One important feature of the PushGP system is the use of *lexicase selection* [7]. One of the design goals for lexicase selection was the preservation of behavior diversity. The ancestry information in the graph databases and in the tree visualizations in this paper can be used to visually and quantitatively check how well those goals are being met. One property of lexicase selection is that it enables the possibility of *hyperselection* events [5], where an individual receives an extremely high proportion of the selections in a given generation. Occasionally, for example, an individual that "discovers" an important new behavior can receive over 90% of the selections and consequently be a parent of nearly all the children in the next generation. These hyperselection events will be visually obvious in our graph visualizations below.

We have applied these visualizations to a number of different problems taken from a suite of software synthesis benchmark problems [6], but in the interest of space we will focus here on a single test problem: Replace Space With Newline. In this problem the goal is to evolve a program that takes a string as input and has two tasks: (a) it should print the input string, with all spaces replaced by newlines, and (b) it should return an integer that is the number of characters in the input string that were *not* replaced. The existence of these two distinct types of error values will be used in some of the visualizations as described in Section 3.2.

The details of the system and settings used in these runs are as described in [6]. There are, however, two parameters that are directly relevant to these visualizations: The population size was 1,000, and the runs were stopped after 300 generations if a successful individual was not found. This means that we were storing and processing considerable information on up to 300,000 individuals per run. We have also generated visualizations on runs in other settings where there were over a million individuals; these graphs are too large to meaningfully include in a paper such as this.

3. VISUALIZING ANCESTRY GRAPHS

Our primary visualization of these EC runs is via ancestry trees where individuals are represented as nodes, and parent-child relationships are represented as edges. An example of this basic structure is illustrated in Figure 1. For a successful run, where at least one individual has found a solution, we obtain the ancestry tree by first finding all successful individuals in the final generation. We go back through the generations to retrieve these individuals' parents, grandparents, etc. until we have every ancestor from start to finish. In an unsuccessful run, where no individual has found a solution to the problem, we take all of the individuals in the last generation and create a tree out of their ancestors. The visualization in Figure 1 captures the basic ancestry structure, but conveys no information about the particular individuals, or their relationships. In the remainder of this section we'll describe several techniques which greatly increase information density of the visualizations.

3.1 Edges

As stated previously, edges show the parent-child relationship in our ancestry diagrams. To increase the amount of information conveyed in these visualizations, however, we use aspects of the edges such as color, style, width, and transparency to convey more information about these runs. First, we use the color and style of an edge to indicate what operator or combination of operators was used to create a child:

- Solid & Black: Alternation, followed by Uniform Mutation
- Dashed & Black: Alternation
- Solid & Orange: Uniform Mutation
- Dashed & Orange: Uniform Closed Mutation

The width of an edge is determined by the Damerau–Levenshtein distance between a parent's and child's genome.[5] We use this as a measure of how similar a child is to its parent; the smaller this distance, the more similar their genomes. Since we want to emphasize strong parent-child relationships, we set the edge to be width of the edge to be inversely proportionate to the distance; if a child is similar to its parent, the wider the edge will be, and larger distances will result in thinner edges.

The last aspect of edges is their transparency, which we based on the number children an individual has in this graph. An incoming edge is more opaque if the individual has many children in the current ancestry tree. This helps us find the parents of individuals that have many children in the displayed tree since these edges are more opaque than others.

3.2 Nodes

Just as we adjust aspects of the edges in our diagrams to tell us more information about the runs, we adjust aspects of the nodes as well. Starting form a simple rectangular shape, we alter the width, height, and the color of the node.

The width of a node is based on the individual's number of selections. This is the number of times an individual was selected to be a parent. A node that is very wide will have an overall high number of selections, and thus be a parent of many children; not all of those children will necessary be displayed in this graph, so it's possible to have very wide nodes with few displayed children. A very wide node is also a clear indicator of *hyperselected* individuals [5] which often play very important roles in the dynamics of runs using lexicase selection. The height of a node is based on how many children an individual has this ancestry graph. Similar to the opacity of an edge, the more children a node has in this ancestry tree, the taller the node.

In the following graphs, we illustrate the use of two different coloring schemes for nodes, each of which conveys information about the errors (and thus the fitness) of the individuals represented by nodes. The first, *dual coloring*, uses color to represent success on the two distinct types of errors (see Section 2). The second, *RBM coloring*, uses restricted Boltzmann machines (RBMs) to compress the 200 error values into 24-bit RGB color values.

The *dual coloring* approach uses hue-saturation-lightness (HSL) coloring based on the "success" of a given individual. The Replace Space With Newline problem is special in the sense that

[5]The Clojush system uses linear Plush genomes that are then converted to PushGP programs. All genetic operations act on these genomes.

Figure 2: Dual Colored (left) and RBM Colored (right) versions of a successful lexicase run (run 0) ancestry tree.

there are two halves of the problem, printing and returning. The test cases that track these aspects are also split into two sets of one-hundred cases. We take advantage of this by separating the two sets of cases and assigning a color to each half, with the color of the left side of a node based on the printing errors, and the color on the right side based on the return errors. For a test case to be passing, there needs to be an error of zero. The hue of one half of a node is based on the percent of zeros, i.e., successful cases, for that half. The hue ranges from red (the worst, with no zeros) to yellow (the best, with all zeros). This coloring tells us how many test cases an individual solves, but gives no information on how far off it is on the other test cases. For this we incorporated lightness into the coloring as well. The lightness of one side of a node is based on the individual's total error on that half of the test cases, with higher total errors receiving darker shading. An example of this is in Figure 2, where the *dual coloring* graph is on the left-hand side.

One potential concern with the dual coloring approach is that we are simply counting successful test cases to determine the hue, and using the total error for the lightness. Both of these are aggregate measures that can obscure valuable details such as *which* test cases are being solved. Because lexicase selection bases selection on entire error vectors (instead of, for example, just using total error), which test cases are being solved becomes much more important than just how many, or what the total error is. To address this, we generated a second coloring that used a simple implementation[6] of restricted Boltzmann machines (RBMs) as a dimensionality reduction tool [8]. Here we trained RBM autoencoders to map 200-bit vectors to 24-bit vectors, where the inputs were binary versions of the error vectors where every non-zero value was converted to 1, and the outputs were interpreted as 24-bit RGB colors. This allowed us to see valuable relationships between individuals that were successful on similar sets of test cases.

Figure 2 shows both color schemes side-by-side on the same ancestry tree from a short, successful run. Both colorings highlight major changes in the error vectors over time, but in different ways. In the center of the dual color graph, for example, there is a large individual that is purple on the left side and green on the right, with the green indicating a major improvement on the test cases that require a returned value. It's also worth noting that the size of this individual indicates that it received a high proportion of the parent selections, and was a parent of a substantial number of the individuals in the next generation. The fact that the incoming edge is solid orange tells us that it was created through uniform mutation, suggesting that a fairly small change to the genome led to a substantial change in the behavior. Five generations later we see a large node that is yellow on the left and purple on the right. The bright yellow indicating that it is perfect on most of the printing test cases, with low total error across these same cases. The purple suggests this individual is not very successful on the integer return test cases. This individual's behavior is thus a "mirrored" version of the behavior of its purple-green ancestor from five generations earlier.

The RBM coloring on the right hand side of Figure 2 does not capture the differences between the two types of test cases, but still reflects the same major changes in its color scheme. Its coloring also shows more variation in the first half of the run, where the low brightness in the dual color graph limits the visible variations.

4. FILTERING

While the large graphs of full runs can provide an excellent "big picture" view of the run dynamics, there is *so* much information that it can be difficult to isolate specific features. We can, however, extract and visualize subgraphs that focus on specific areas or events in the runs. The left hand graph in Figure 3, for example, is the full ancestry of the successful individual from one of our runs, and contains 22,435 nodes and 35,403 edges. This full ancestry graph gives us a strong sense of the large scale dynamics of the run, while the shape of the graph as well as the sizing and coloring of nodes highlight some major events in the history of the run. There are, however, large sections of the graph composed of hundreds of very small nodes that make it very difficult to trace through and discover what might be the most important paths through that part of the genetic history.

An obvious approach to this problem is to filter the results, we have already been using one simple filtering throughout the paper. In all the successful runs that we visualize here, we are only showing individuals that are ancestors of the successful individual(s) in the final generation.[7] For example, if we visualized *every* individual in the run shown in Figure 3 the graph would contain

[6]https://github.com/echen/restricted-boltzmann-machines

[7]Here is one place where graph databases really shine. To extract this ancestry information from a relational or document-oriented database would be an expensive series of recursive queries. With a graph database system such as TitanDB and Tinkerpop, however, this becomes a simple one line query.

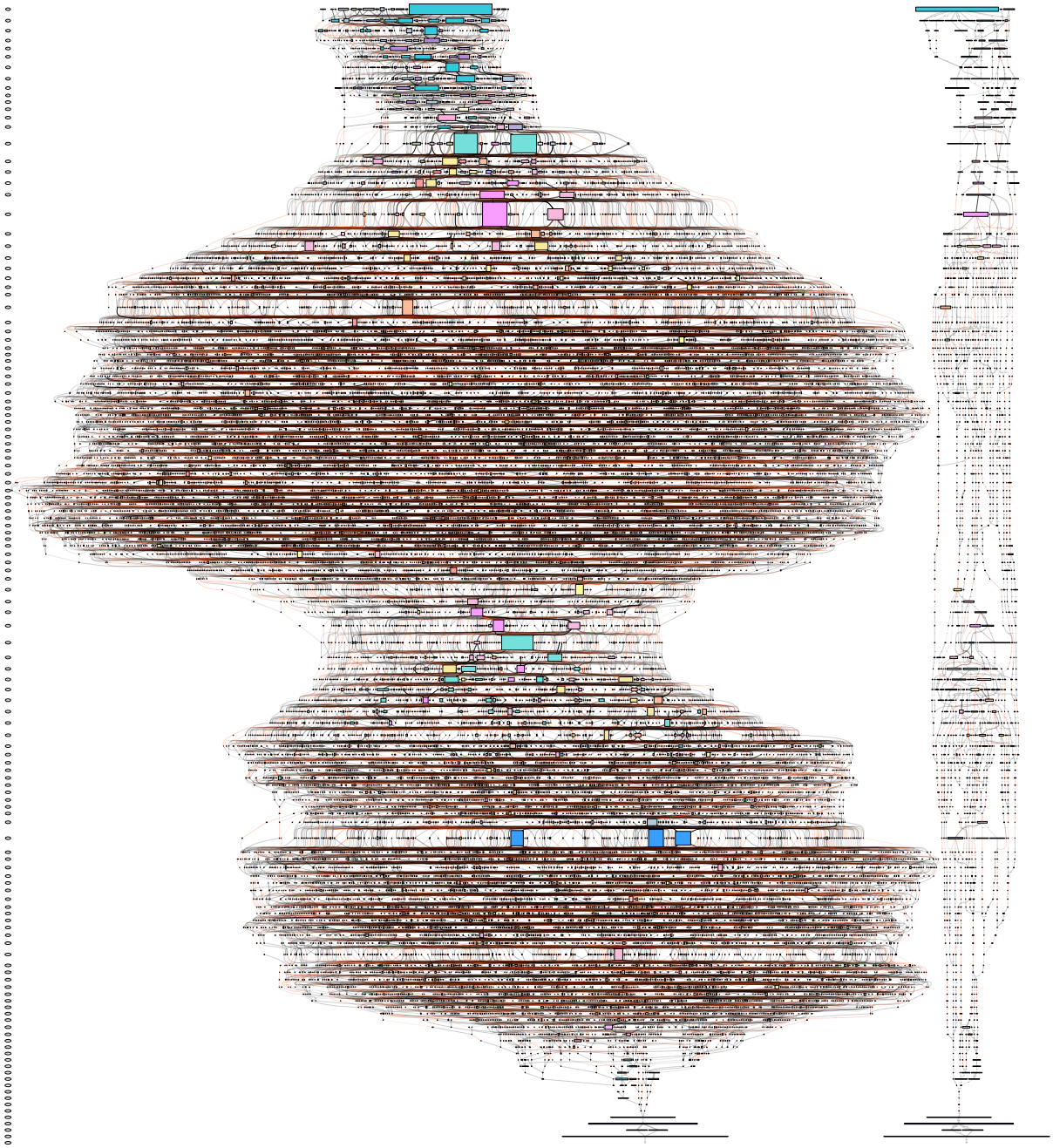

Figure 3: Unfiltered (left) and filtered (right) versions of a successful run (run 1), both with the same color map.

over six times as many elements, with 130,000 nodes and 219,541 edges. Most individuals in evolutionary systems aren't ultimately ancestors of individuals in the final population; they're evolutionary "dead ends", either because they have no children, or their descendants eventually fail to have children. All the graphs in this paper use this type of filter to substantially cut down on the number of visualized nodes, only looking at ancestors of successful individuals or, in the case of unsuccessful runs, only looking at ancestors of individuals in the final generation. This typically reduces the number of nodes in a generation (row of the graph) from the original 1,000 to 300 or less, and sometimes down to a few dozen.

As discussed above, while this filtering strongly focuses the visualization, this still can leave us with too much information if our goal is to trace key paths through the history. Here we demonstrate the use of a combination of ancestry information and genome distances between parents to substantially tighten the focus of the ancestry graphs. The key idea, which grew out of observations of our early graphs, is to use a distance metric to determine which parents are contributing "substantial" genetic material to the resulting child. This allows identification of crossovers where one of the parents contributed the preponderance of the child's genetic material. If we filter out parents that are making limited contributions to their children's genetics, then we end up recursively removing many of their ancestors as well, profoundly thinning the graph.

Unfortunately there's no easy way to guarantee that any non-zero contribution, however small, might not in fact be crucial to the behavior of the child. A parent might only contribute one instruction to its child, but that could be the vital piece that leads to a solution. Even worse, it's possible that the *presence* of an individual could have had a subtle but important impact on the dynamics of a run even if that individual never directly contributed any genetic material to the eventually successful individual. Acknowledging those complexities, we have still found it useful to filter ancestry trees, recognizing that we might need to "unfilter" some individuals if further analysis suggests that their contributions were more substantial than initially expected.

The method used here was based on the Damerau–Levenshtein distance between the parent and child genomes. Our filtering algorithm was fairly simplistic: Assume we have two parents, p and q, and a child c such that $a = \text{DL-distance}(p, c)$, $b = \text{DL-distance}(q, c)$, and $s = \text{genome-length}(c)$, and assume without loss of generality that $a \leq b$. Then filter out parent q if either $a < 0.2 \times s$ or $b \geq 2 \times a$. Thus parent q will be filtered out if parent p is particularly close to the child, or if parent q is more than twice as far away from the child as parent p. The choices of the constants 0.2 and 2 are obviously somewhat arbitrary, but appeared to work reasonably well on these datasets. There are, however, examples where a filtered parent did in fact contribute a significant number of instructions to the offspring, so one would need to be careful in not making overly broad assumptions based on an individual not being in the filtered version of a graph.

Returning to Figure 3, the right hand graph is a filtered version of the left hand graph. Both graphs use the same RBM coloring, so it is in many cases possible to pick out corresponding individuals in the two graphs. In the filtered graph, however, it's entirely possible to trace every relationship from the beginning of the run to the successful individual 130 generations later, where this is unfeasible in the unfiltered graph. The filtered graph has 1,597 vertices and 1,794 edges, roughly 20 times fewer than in the unfiltered graph, and roughly 100 times fewer than in the full graph.

Comparing the two graphs reveals several important events and phases in the run. Both graphs show an initial "settling out" phase, where most of the initial random population contributes little or nothing and there are quite a few very strong selection events as evidenced by the variety of wide nodes in the early generations. Starting around generation 20, however, the dynamics in both runs start to become more complex, with both graphs moving to more individuals in each generation and a more complex edge structure. While the unfiltered graph remains quite wide up to generation 56 or so, the filtered graph starts to thin out, leading to a fairly small set of largely linear "threads". In generation 50, for example, the unfiltered graph has 305 individuals out of the 1,000 individuals in that generation, whereas the filtered graph only has 13 individuals. While the nodes in those threads are quite small, several of the threads are sequences of individuals whose colors are similar within the thread, but different from the colors in other threads. This suggests (and additional analysis of the data in the database further supports) that at least some of these threads may represent sub-populations that are focusing on different parts of the problem.

Moving further into the run, both graphs show a significant change by generation 63, with a series of hyperselection events suggesting that there is some sort of discovery. In the unfiltered graph this expresses itself as a strong narrowing of the graph, as the hyperselection events substantially reduce the number of individuals in those generations that go on to be ancestors of the successful individual. In the filtered graph the representation is almost the opposite, where the few threads give way to a "knot" of more individuals with much less linear ancestries with more complex mixing. After a while this settles down, with the unfiltered graph widening back out and the filtered graph returning to a decreasing number of mostly linear ancestries. At the end of the run there are visual indications of major discoveries that lead to large hyperselection events that dominate the dynamics in the last few generations.

5. COMPARING RUNS

In this section we will demonstrate the ability to graph multiple runs side by side for comparison, similar to the unfiltered and filtered graphs in Figure 3. We will first compare three successful runs, and then a collection of four different runs, two of which were successful and two of which were not.

5.1 Successful Runs

Figure 4 shows the filtered graphs from three successful runs (from left to right, run 1, 99, and 6) using the same RBM color scheme for all three.[8] The shared color scheme allows us to see similar individuals (in terms of error vectors) across multiple runs. All these runs, for example, start with green nodes, suggesting those individuals have similar error vectors. It is interesting to note that runs 1 and 99 both have several very strongly selected nodes in the first 15 generations that introduce new but related colors, with both runs having both blue and pink nodes. Run 6, on the other hand, remains primarily green for quite a bit longer and does not have many highly selected individuals until around generation 40. Some of the new colors introduced in run 6 are blues that look similar to nodes in runs 1 and 99, but there are also beige nodes that appear different from the early nodes in the other graphs.

All three runs share some substantial large scale features. All have initial "settling out" periods, which eventually develop into a relatively small set of largely linear lineages. All have one or more "knots" where there are higher degrees of connectivity for several generations before returning to the more linear ancestries. The end of all three runs feature several very strong hyperselection events.

[8]The leftmost graph in Figure 4 is the same data as the rightmost (filtered) graph in Figure 3 above, with a different RBM coloring and a slightly different layout.

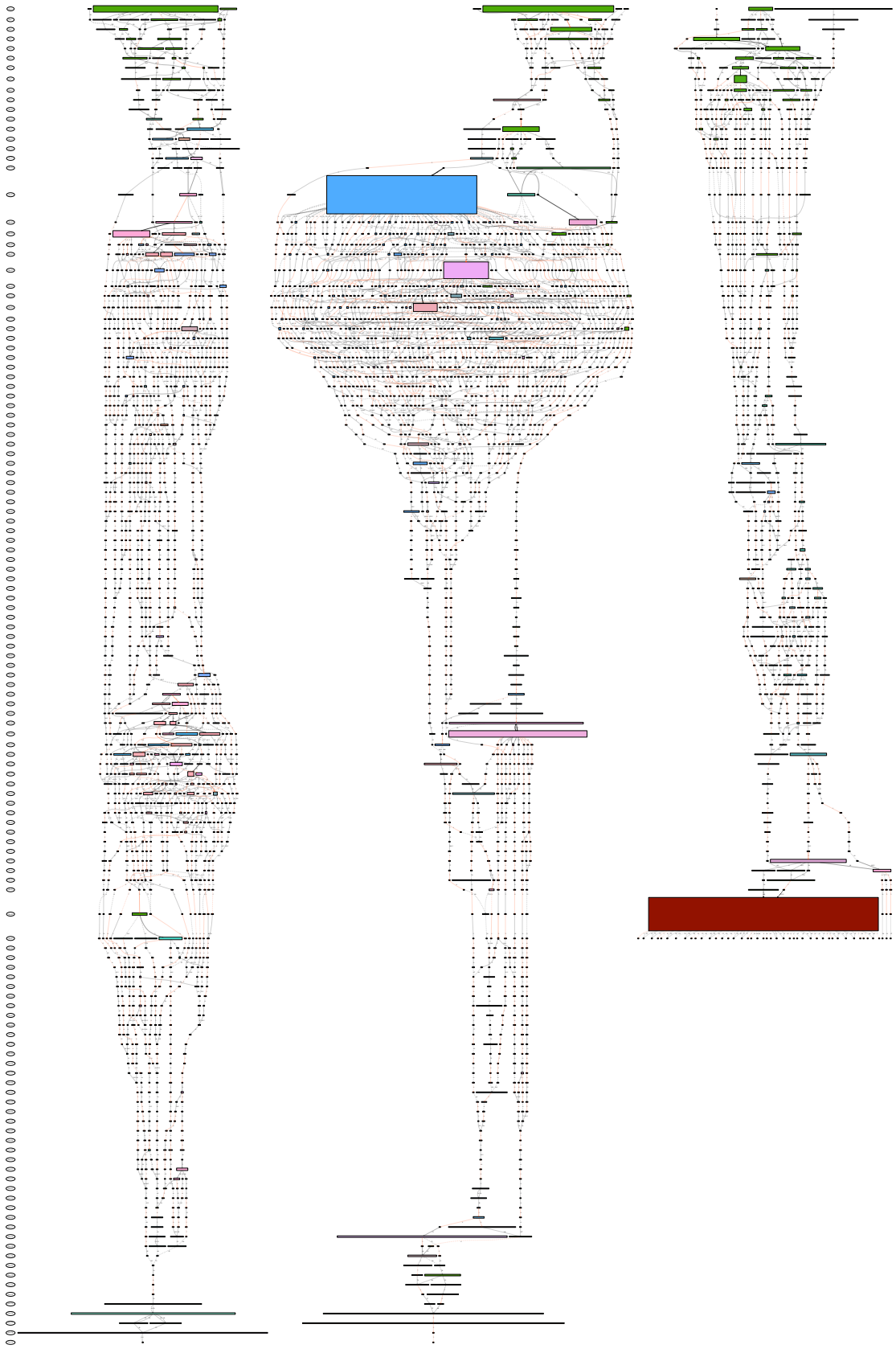

Figure 4: Filtered ancestry graphs of three successful runs (left to right, runs 1, 99, and 6), all with the same color map.

Figure 5: Full ancestry graphs of 4 different runs (left to right, runs 92, 2, 1, and 99), all with the same color map.

5.2 Successful vs. Failed Runs

Figure 5 shows the full unfiltered ancestries for four different runs (from left to right, runs 92, 2, 1, and 99). The leftmost two (runs 92 and 2) were unsuccessful, so these graphs show the ancestors of all the individuals present in the final generation.[9] Runs 1 and 99 were successful and are also visualized in Figure 4. All four graphs use the same dual color scheme, which shows that in all four cases similar individuals are discovered early that are very successful (bright yellow) on the integer return cases (right hand side of nodes) and have similar (purple) success rates on the printing cases (left hand side of nodes). The most selected individuals in the initial random populations for the two unsuccessful runs have very dark colors on the right hand sides of their nodes, indicating that the most selected initial individuals had *very* high total error on the integer return cases.

After the initial settling out, run 2 has no significant hyperselection events, and no substantial changes in the visual structure of the graph. The other three runs, however, have a variety of highly selected individuals, and associated changes in the shape of their graphs. It's possible, therefore, that run 92 might have eventually discovered a solution if given more generations, where run 2 seems much less likely to make progress.

6. FUTURE WORK

Two major issues in this work so far are the static nature of the figures and the challenges of displaying and working with such large, dense graphs. Static visualizations are not inherently bad, and are obviously important for things like print work. There is, however, only so much data that can be compressed into static visualizations such as this, and it would certainly be nice to be able to interact with the graphs dynamically, clicking on nodes or edges to access additional information that is in the database, but not directly or exactly displayed. A dynamic display such as used in the DeepTree exhibit [2] or using tools such as Gephi[10] or Google Maps [1] might also make it easier to move around in very large graphs.

A further application of these ideas would be to use these graphs to compare the dynamics of evolutionary systems using, for example, different selection mechanisms. Previous research [9] has compared the dynamics of sections of runs using tournament selection and lexicase selection; this work could be extended to compare the ancestry graphs of entire runs.

7. CONCLUSIONS

Here we have demonstrated the ability to collect and visualize ancestry data from large genetic programming runs. The resulting graphs provide valuable information about both the overall dynamics of the runs, as well as highlighting specific important events in the evolutionary process, such as instances of hyperselection and, through filtering, the existence largely independent subpopulations focusing on different test cases. Use of these sorts of visualizations have proved valuable in understanding properties of important tools such as selection operators and can help guide the development of new techniques based on a more detailed understanding of the behavior of evolutionary systems.

8. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grants No. 1129139 and 1331283. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Thanks to William Tozier for numerous suggestions and support. Thanks also to the members of the Computational Intelligence Lab at Hampshire College for ideas and feedback.

[9] In fact we have removed the last two generations from the unsuccessful graphs since those have many more individuals than the earlier generations, and their presence distorts the layout.

[10] https://gephi.org/

9. REFERENCES

[1] A. Agarwal. Embed large pictures with Google Maps Viewer, September 2012. [Online; accessed 3-April-2016].

[2] F. Block, M. S. Horn, B. C. Phillips, J. Diamond, E. M. Evans, and C. Shen. The DeepTree exhibit: Visualizing the tree of life to facilitate informal learning. *Visualization and Computer Graphics, IEEE Transactions on*, 18(12):2789–2798, 2012.

[3] B. Burlacu, M. Affenzeller, M. Kommenda, S. Winkler, and G. Kronberger. Visualization of genetic lineages and inheritance information in genetic programming. In *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*, pages 1351–1358. ACM, 2013.

[4] A. Cruz, P. Machado, F. Assunção, and A. Leitão. ELICIT: Evolutionary computation visualization. In *Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference*, pages 949–956. ACM, 2015.

[5] T. Helmuth, N. F. McPhee, and L. Spector. The impact of hyperselection on lexicase selection. In *GECCO '16: Proceedings of the 2016 Conference on Genetic and Evolutionary Computation*, July 2016.

[6] T. Helmuth and L. Spector. General program synthesis benchmark suite. In *GECCO '15: Proceedings of the 2015 Conference on Genetic and Evolutionary Computation*, July 2015.

[7] T. Helmuth, L. Spector, and J. Matheson. Solving uncompromising problems with lexicase selection. *IEEE Transactions on Evolutionary Computation*, 19(5):630–643, Oct. 2015.

[8] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[9] N. F. McPhee, D. Donatucci, and T. Helmuth. Using graph databases to explore the dynamics of genetic programming runs. In R. Riolo, B. Worzel, M. Kotanchek, and A. Kordon, editors, *Genetic Programming Theory and Practice XIII*, Genetic and Evolutionary Computation. Springer.

[10] L. Spector, J. Klein, and M. Keijzer. The Push3 execution stack and the evolution of control. In *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1689–1696, Washington DC, USA, 2005. ACM Press.

[11] L. Spector and A. Robinson. Genetic programming and autoconstructive evolution with the push programming language. *Genetic Programming and Evolvable Machines*, 3(1):7–40, Mar. 2002.

[12] L. Vaseux, F. E. Otero, T. Castle, and C. G. Johnson. Event-based graphical monitoring in the EpochX genetic programming framework. In *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*, pages 1309–1316. ACM, 2013.