Visualizing genetic programming ancestries using graph databases

Nicholas Freitag McPhee
University of Minnesota, Morris
Morris, Minnesota, USA
mcphee@morris.umn.edu

Maggie M. Casale
Design Center Inc.
St. Paul, Minnesota, USA
mcasale@designcenterideas.com

Thomas Helmuth
Washington and Lee University
Lexington, Virginia, USA
helmutht@wlu.edu

Lee Spector
Hampshire College
Amherst, Massachusetts, USA
lspector@hampshire.edu

ABSTRACT

Previous work has demonstrated the utility of graph databases as a tool for collecting and analyzing ancestry in evolutionary computation runs. That work focused on sections of individual runs, whereas this poster illustrates the application of these ideas on the entirety of large runs (up to one million 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 when using different evolutionary tools, such as different selection mechanisms or representations, as well as comparing the dynamics for successful and unsuccessful runs.

CCS CONCEPTS

•Human-centered computing → Graph drawings; Scientific visualization; •Computing methodologies → *Heuristic function construction; Genetic programming;*

KEYWORDS

visualization; genetic programming; graph database; ancestry

ACM Reference format:
Nicholas Freitag McPhee, Maggie M. Casale, Thomas Helmuth, and Lee Spector. 2017. Visualizing genetic programming ancestries using graph databases. In *Proceedings of GECCO '17, Berlin, Germany, July 15-19, 2017,* 2 pages.
DOI: http://dx.doi.org/10.1145/3071178.3071330

1 INTRODUCTION

Reporting results of genetic programming (GP) and evolutionary computation 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. Previous work [3] has demonstrated the utility of graph databases as tools for collecting and analyzing ancestry in GP runs, but was focused on sections of individual runs. In this poster we illustrate the use of these tools as a means of exploring entire ancestry graphs. We use the Titan graph database along with the Gremlin shell and the Tinkerpop query tools to store the parent-child relationships from genetic programming runs, and to extract the ancestry graphs of specified individuals. We then visualize these subgraphs using the **`dot`** graph layout tool from the Graphviz library.[1]

2 EXAMPLE: SUCCESS VS. FAILURE

Here we illustrate these ideas by extracting and plotting the ancestors of final generation individuals in both a successful (Figure 1) and an unsuccessful run (Figure 2) of the Replace Space With Newline software synthesis problem [1, 2] using lexicase selection. The successful run shows all the ancestors of that run's winners (i.e., individuals with total error 0, discovered at generation 129). The unsuccessful run shows the ancestors of all individuals from generation 300, when the run was terminated; the ancestry graph in Figure 2 is truncated at generation 200 for space reasons, but the run was unsuccessful up to generation 300.

In both figures generations run from the initial population at the top to the final generation at the bottom, one generation per row. Both of these runs used a population size of 1,000, but far fewer individuals are included in the graphs because only a small subset of the individuals in any given generation go on to be ancestors of individuals in subsequent generations. Each individual is represented as a rectangle whose width is proportional to the number of parent selections that individual received, and whose height is proportional to the number of its offspring that are included in the ancestry graph. The color of an individual is determined by its total error; 0 total error is bright green, moving through blues to bright red, which represents total error of 10,000 or greater. A directed edge in the graph indicates a parent-child relation, with the edge going from the parent down to the child. A child with only a single incoming edge is the result of a mutation operator and children with two incoming edges are the result of a recombination [1].

Both Figures 1 and 2 illustrate a common pattern in the early generations where there is a substantial number of highly selected individuals (i.e., very wide rectangles). This is presumably because most individuals in the early generations perform poorly, as evidenced by the prevalence of red and pink, with a few slightly-less-bad individuals receiving the bulk of the selections. After several generations, however, the population gains competence (as evidenced by more

[1] http://titandb.io, https://tinkerpop.apache.org/, and http://www.graphviz.org/

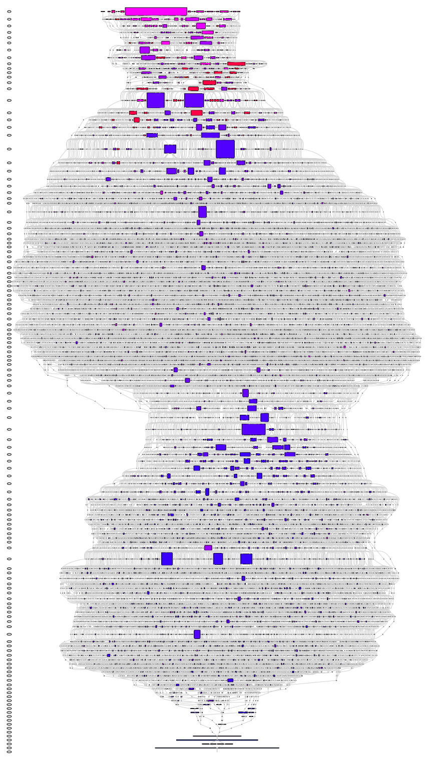

Figure 1: Ancestry graph for individual with total error 0 in successful run of the Replace Space With Newline problem.

purple and dark blue) and there are fewer instances where a single individual receives a very high proportion of the selection events, as evidenced by the large number of small rectangles.

After that initial phase, the behavior in the unsuccessful run (Figure 2) becomes static, with the width of the graph (essentially the number of parents in each generation) remaining roughly constant, with no highly selected (i.e., wide) individuals. The colors also indicate that the total error is not improving substantially over time, remaining mostly between pink and purple. The best total error in fact remains at 234 from generation 16 to 270, although there is a gradual improvement in the last 30 generations with the best error at the end of the run being 215.

In the successful run (Figure 1), however, there are clear changes in the dynamics over time. By generation 14 the best individual is able to correctly solve 124 of the 200 test cases. Then starting around generation 59 there is a "narrowing" of the graph, with several highly selected (i.e., wide) individuals dominating the ancestry for the next ten generations. This narrowing represents an important "discovery" which leads to individuals that have zero error on an additional 9% (18) of the test cases. The graph then widens out again until the rapid convergence on the solution in the final generations. The graph also highlights three highly selected individuals in generation 87 where for the first time the best individual has zero error on an additional 18 test cases.

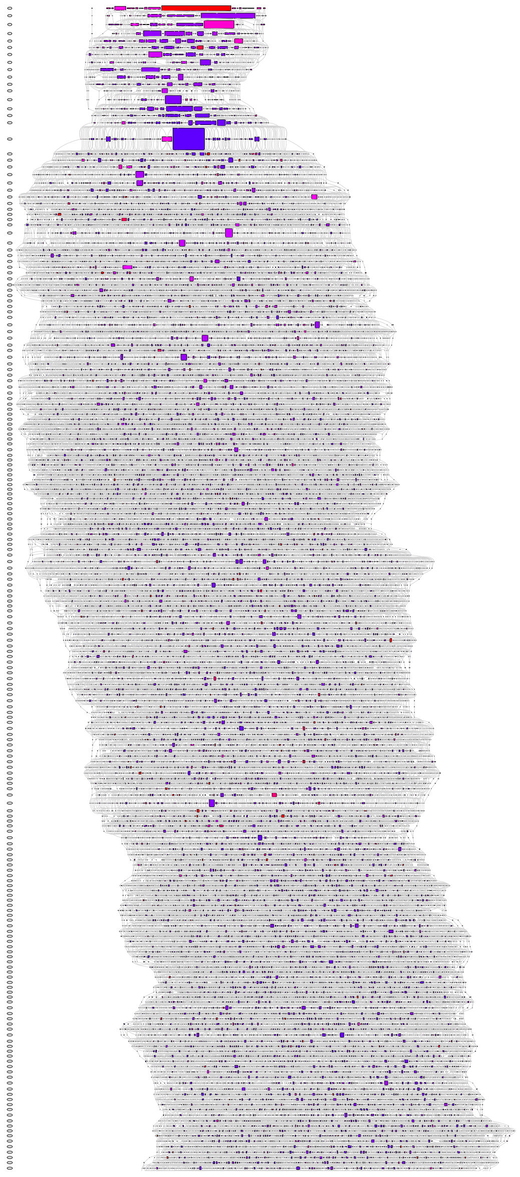

Figure 2: Ancestry graph for all of the individuals in the final generation for an unsuccessful run of the Replace Space With Newline problem. This graph is truncated at generation 200; the run went to generation 300 without success.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grants No. 1617087, 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.

REFERENCES

[1] Thomas Helmuth. 2015. *General Program Synthesis from Examples Using Genetic Programming with Parent Selection Based on Random Lexicographic Orderings of Test Cases.* Ph.D. dissertation. University of Massachusetts, Amherst. http://scholarworks.umass.edu/dissertations_2/465/

[2] Thomas Helmuth and Lee Spector. 2015. General program synthesis benchmark suite. In *GECCO '15: Proceedings of the 2015 Conference on Genetic and Evolutionary Computation* (July, 2015).

[3] Nicholas Freitag McPhee, David Donatucci, and Thomas Helmuth. 2016. Using Graph Databases to Explore the Dynamics of Genetic Programming Runs. In *Genetic Programming Theory and Practice XIII*, R. Riolo, B. Worzel, M. Kotanchek, and A. Kordon (Eds.). Springer.