Visualization of Genetic Lineages and Inheritance Information in Genetic Programming

Bogdan Burlacu
Heuristic and Evolutionary Algorithms Laboratory
Softwarepark 11, 4232
Hagenberg, Austria
bogdan.burlacu@fh-hagenberg.at

Michael Affenzeller
Heuristic and Evolutionary Algorithms Laboratory
Softwarepark 11, 4232
Hagenberg, Austria
michael.affenzeller@fh-hagenberg.at

Michael Kommenda
Heuristic and Evolutionary Algorithms Laboratory
Softwarepark 11, 4232
Hagenberg, Austria
michael.kommenda@fh-hagenberg.at

Stephan Winkler
Heuristic and Evolutionary Algorithms Laboratory
Softwarepark 11, 4232
Hagenberg, Austria
stephan.winkler@fh-hagenberg.at

Gabriel Kronberger
Heuristic and Evolutionary Algorithms Laboratory
Softwarepark 11, 4232
Hagenberg, Austria
gabriel.kronberger@fh-hagenberg.at

ABSTRACT

Many studies emphasize the importance of genetic diversity and the need for an appropriate tuning of selection pressure in genetic programming. Additional important aspects are the performance and effects of the genetic operators (crossover and mutation) on the transfer and stabilization of inherited information blocks during the run of the algorithm. In this context, different ideas about the usage of lineage and genealogical information for improving genetic programming have taken shape in the last decade.

Our work builds on those ideas by introducing an evolution tracking framework for assembling genealogical and inheritance graphs of populations. The proposed approach allows detailed investigation of phenomena related to building blocks, size evolution, ancestry and diversity. We introduce the notion of genetic fragments to represent subtrees that are affected by reproductive operators (mutation and crossover) and present a methodology for tracking such fragments using flexible similarity measures. A fragment matching algorithm was designed to work on both structural and semantic levels, allowing us to gain insight into the exploratory and exploitative behavior of the evolutionary process.

The visualization part which is the subject of this paper integrates with the framework and provides an easy way of exploring the population history. The paper focuses on a case study in which we investigate the evolution of a solution to a symbolic regression benchmark problem.

GECCO'13 Companion, July 6–10, 2013, Amsterdam, The Netherlands.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

genetic programming, genetic lineage, genealogy, visualization

Keywords

genetic programming, algorithm analysis, visualization, population genealogy

1. INTRODUCTION

Genetic programming is a nature-inspired optimization algorithm which follows the model of biological evolution. It works by optimizing a population of computer programs according to a fitness function that determines each program's ability to perform a given task. It was invented by Koza [8] in 1992 and has since gained massive popularity due to its robustness and performance even when applied to very difficult optimization problems. But despite its success, many questions regarding the dynamics of the process and the algorithmic internals remain unanswered. The analogy with natural evolution brings forward notions like genotypes, phenotypes, selection pressure, diversity and genetic operators. A well-established opinion is that the success of the algorithm depends on having enough genetic diversity in the population, so that the recombination operators can produce novel genetic variation that selection can act upon [10, 3, 9]. In practice, the creation of new genetic variation is often affected by bloat and introns; in the presence of constant selection pressure, insufficient variation will quickly lead to loss of diversity and stagnation [11, 1]. For these reasons, the preservation of genetic diversity during the run became the subject of intensive research, with results ranging from

improved crossover variants [12, 15] to algorithmic variants such as RAPGA and OSGA [1].

In this context, the usage of lineage and genealogy information for the development of novel self-adaptive genetic programming algorithms seems a logical step forward. However, it was only with the technological advances of the last decade in the field of computer engineering, that the challenge of completely recording an evolutionary run of modest dimensions became a tractable problem. A short description of previous work done in this area is provided below.

McPhee and Hopper [10] used a node numbering scheme to quantify the relative amount of genetic material (from the initial population) that is present at the end of the run. They used the most recent common ancestor (called "eve") of all the individuals in the final generation as an indication of run quality and also as a way to measure diversity by looking at the shared amount of genetic material between "eve" and the rest of the population.

Burke et al. [4] used a variant of tournament selection that grouped individuals by common genetic lineages, to encourage diversity preservation during the selection step of a genetic programming algorithm run. Using what they called "lineage selection", they allowed only one individual from each genetic lineage to take part in the tournament in order to shift the population from the "fit" to the "fit and diverse".

Also for diversity preservation, Essam and Mckay [6] used a tag system to mark individuals of common ancestry in order to prevent their crossover. Their objective was to restrict the ability of a few key individuals and their descendants from rapidly dominating the population.

Goetticher and Kaminksy [2] looked at the short-term history (previous generation) to partition individuals into fitness classes and steer selection towards the class of best 20% individuals. This method has been criticised for using a linear selection scheme, transferring individuals unchanged from one generation to the next and only using the latest ancestor.

Dong and Chen [5] used a larger ancestry (up to four ancestors) to calculate the average fitness of a given lineage before grouping individuals into fitness classes, named "best", "middle" and "worst". Additionally, as a measure for preserving diversity, they randomly retain only one individual with the same fitness, discarding the rest.

As we can see, different strategies for the selection scheme can be chosen based on genealogy information. In our opinion, using just the fitness information of an individual's ancestry, or just the root lineage of an individual, are insufficient criteria for steering the selection process. Our approach is based on the idea of recording the entire history of a genetic programming run.

The tracking framework consists of a number of components dealing with recording the information, processing it, and displaying it in a user-friendly manner. The implementation was done within HeuristicLab, a framework for heuristic and evolutionary algorithms developed by members of the Heuristic and Evolutionary Algorithms Laboratory (HEAL) at the Upper Austria University of Applied Sciences. HeuristicLab provides a complete environment with out-of-the-box support for a large variety of operators [14]. Special attention has been paid to the tree isomorphism problem which is a part of our fragment matching methodology.

In this paper, we describe in detail the visualization component of our approach, which allows the user to explore populations, genealogies and the inheritance of genetic material. We provide an example run in order to illustrate the main ideas behind our approach from a practical point of view. The paper is organized as follows: Section 2 describes the genealogy graph and the synthetic benchmark problem used as an example, while Section 3 describes the methodology for tracking inherited genetic material during the run. Finally, Section 4 is dedicated to conclusions and future work.

2. CONSTRUCTING THE GENEALOGY

Internally, the genealogical information is stored in the form of a directed graph, in which vertices represent individuals while arcs and paths in the graph represent hereditary relationships. By convention, a parent-child relationship is represented by an arc with the orientation parent→child, but for graph traversal purposes, the arc information is held by both the parent and the child.

After each iteration of the algorithm, the offspring obtained via crossover and mutation are added as vertices in the graph and connected with arcs to their parents. Each generation is represented as a horizontal layer in which vertices are ordered by decreasing fitness.

Elite individuals are also present in the graph; an elite that survives multiple generations is represented by a set of vertices (one for each generation) connected by arcs and forming a path in the graph. Technically, the directed acyclic graph that results at the end of the run is a union of all the genealogies in the population.

In the following, we apply the methodology on a symbolic regression benchmark problem. We have chosen an easier instance, namely the *Vladislavleva-5* synthetic benchmark [13] in which the objective is to find the function:

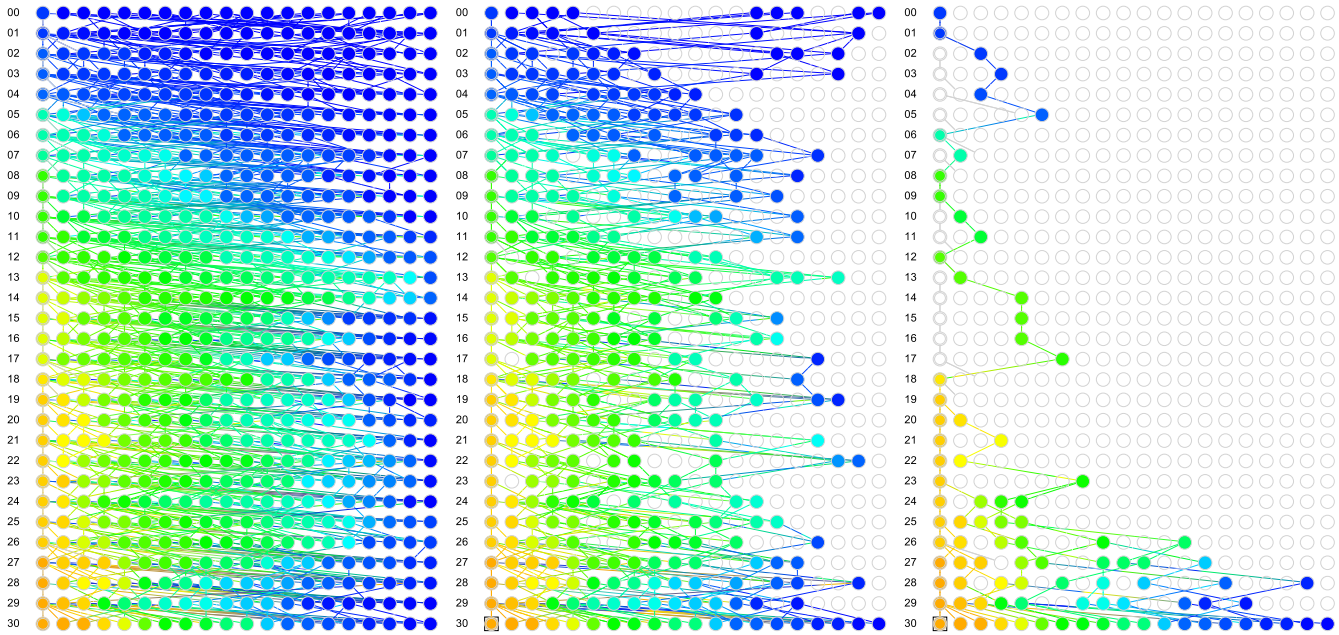
$$f(x_1, x_2, x_3) = 30 \cdot \frac{(x_1 - 1)(x_3 - 1)}{x_2^2(x_1 - 10)}$$

In the interest of producing clear graphics, the dimensions of the run were kept small and only crossover was used as a variation producing operator. The population size was set to 20 individuals and the length of the run to 30 generations. The quality of the best solution was 0.834 Pearson's R^2 on the training data (0.775 on test).

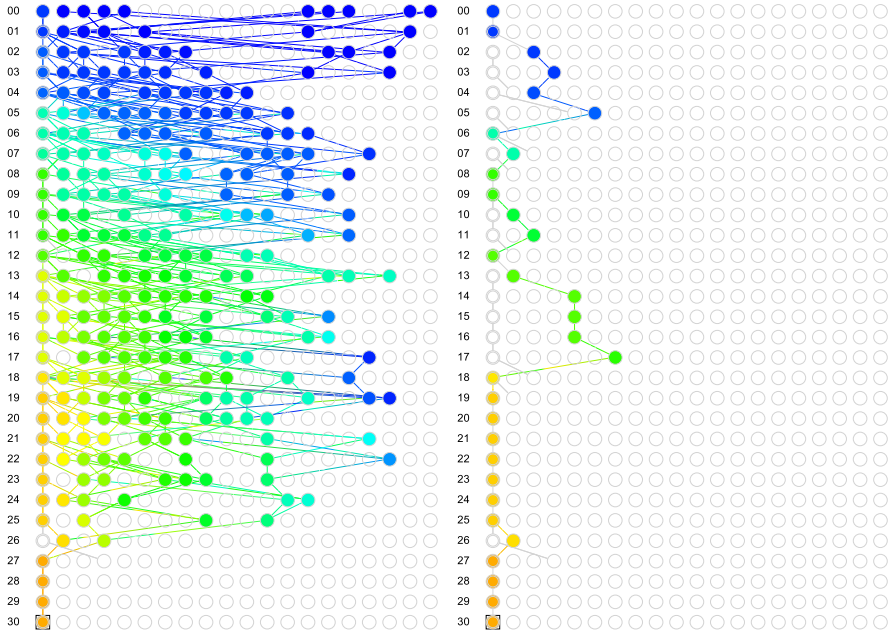
Figure 1a shows the evolution of the individual qualities for the whole population. Vertices are colored according to the quality and additionally, each graph arc is colored using a gradient stretched between the colors of the vertices it connects. The graph layers corresponding to each generation are labeled with the generation number, where 0 represents the initial population, 1 the first generation, and so on.

There are several aspects that can be observed on the graph, for example:

- The distribution of fitness values – it can be observed in the graph per generation, per lineage, or overall. For successful runs, there is a clear improvement in average fitness from one generation to the next, until the search converges. The general tendency, considering layers of vertices ordered by descending fitness, is for the fitness to follow a gradient from the top right vertex of the graph to the bottom left one (Figure 1a).
- The intervals in which the search stagnated (generations 8-11, 13-17, 19-25 and 27-30) – during these



(a) Distribution of fitness values (b) Genealogies in the last generation (c) Root lineages in the last generation



(d) Genealogy of the best individual (e) Root lineage of the best individual

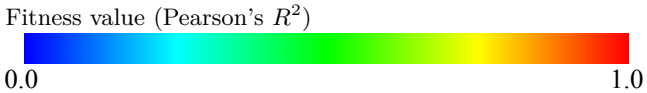


Figure 1: Distribution of fitness, genealogies and root lineages in the population graph.

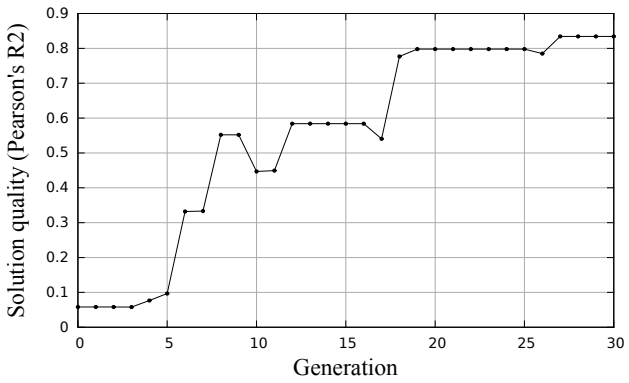


Figure 2: Evolution of qualities in the root lineage of the best individual

generations without improvement there is an increased chance that the best individual will start dominating the population, since it will get selected more often. This causes an increase of the population average fitness and a decrease in population diversity.

- The genealogies (Figure 1b) and root lineages (Figure 1c) that survive until the end of the run and the number of individuals they have in common.
- The individuals involved in the creation of the best solution (Figure 1d) and its root lineage (Figure 1e).

Empirically we notice that the root lineage of the best individual contains a significant number of elites from previous generations. This means that, with few exceptions, only the best individuals were able to create successful children. We can say that the heritability of good building blocks plays an important role in the evolutionary process. Figure 2 supports this view, showing the evolution of qualities in the root lineage of the best individual. The quality generally improves, meaning that a good parent is almost always able to produce at least one good offspring. Another interesting question is whether only good parents can produce good offspring or not – it would appear from the graph that at least one parent has to have a high fitness value, as fit individuals are more likely to contain relevant building blocks.

3. TRACKING OF GENETIC FRAGMENTS

As we have seen, each individual in the population is associated with a vertex in the genealogy graph. In this section, we detail how the genealogy graph can be used to track the propagation of genetic information. We introduce the necessary notions below:

Genetic fragment The genetic fragment (or just fragment) represents the concrete subtree that gets changed by a genetic operator in the course of producing the offspring. That is, the subtree that gets swapped from the non-root parent to the root parent by crossover, or the node or subtree that gets modified by mutation.

Node similarity criteria We define a set of conditions for deciding whether a node is similar to another node. For our approach, we use the following predicates:

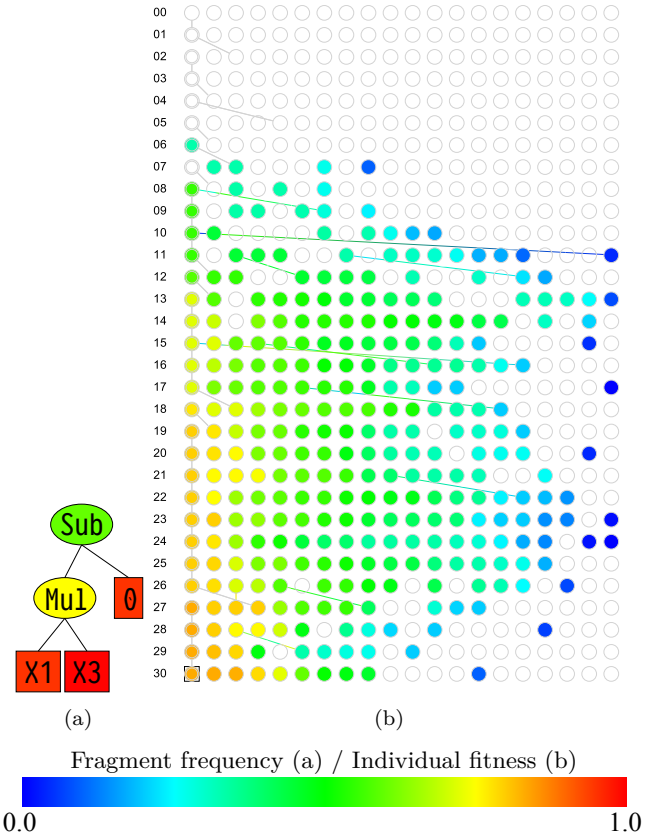


Figure 3: Individuals containing the fragment $(- (\times \texttt{X1 X3}) \texttt{0})$. The occurrence of this subtree in the root ancestor at generation 6 marks an important moment in the evolution of the population.

1. *Match constants*: returns *true* if the nodes represent the same constant value, otherwise *false*.
2. *Match variables*: returns *true* for variable nodes which represent the same variable, otherwise *false*.
3. *Match variable weights*: returns *true* for variable nodes which have the same variable weight, otherwise *false* (**note**: this condition acts independently from the one above).

Additionally, it is clear that different nodes cannot be similar (ie, constant vs variable node).

With the above similarity criteria and our tree inclusion algorithm [7], we are able to identify structurally (relaxed node matching) or semantically (strict node matching) similar subtrees. Note that similar subtrees may not be isomorphic, as one of them may contain a larger number of nodes than the other. We require isomorphism in the case of fragments that are propagated via crossover. The occurrence frequency of a given genetic fragment (or subtree) is given as the percentage of individuals in the population that contain it within their structure.

The distinction between semantic and structural similarity is important as the population may have many similar structures that only differ from each other by some constant value or some variable weight. Obviously, populations of structurally similar individuals are not particularly suited

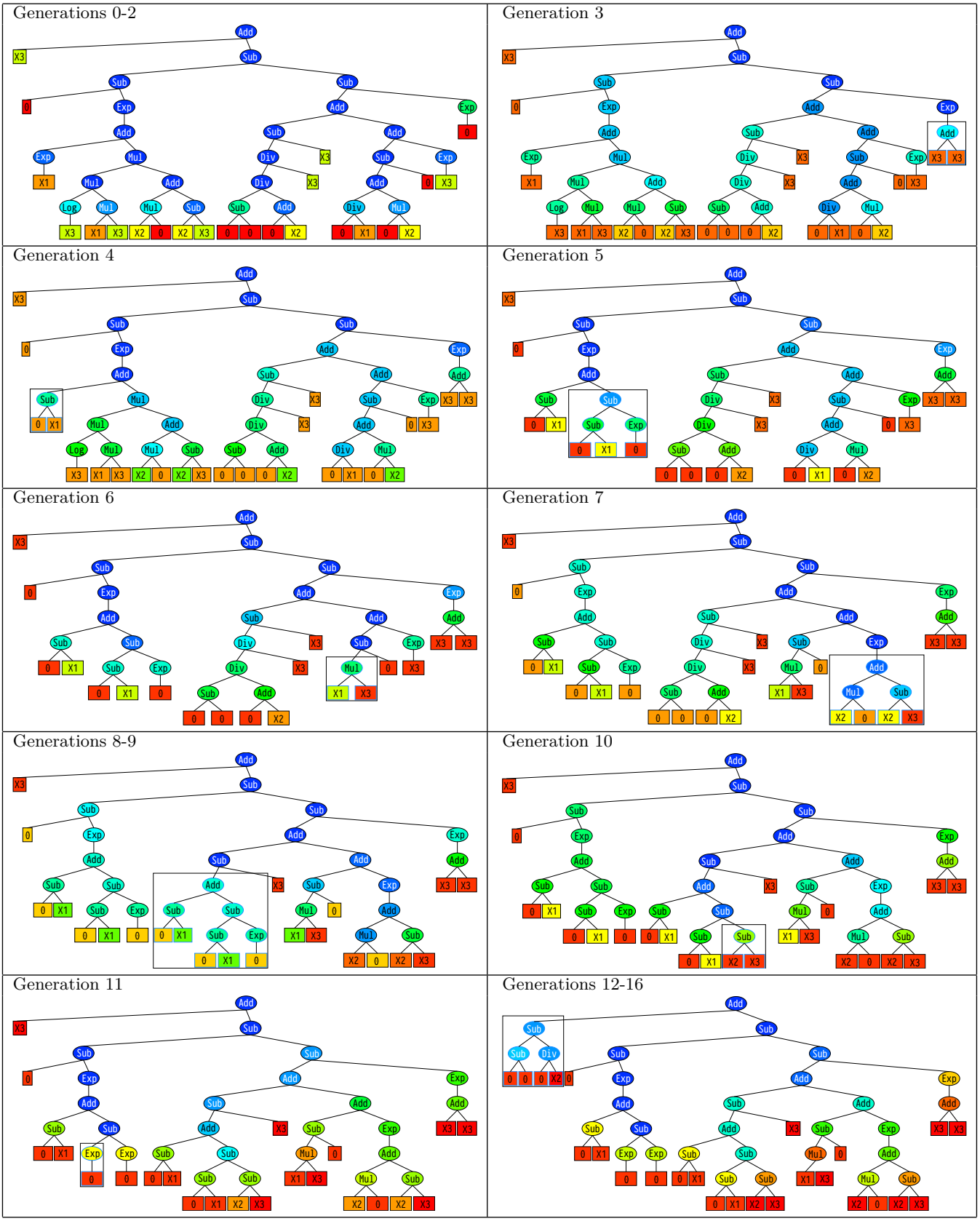


Figure 4: Root lineage of the best individual – generations 0-16

for evolution. Furthermore, if equivalent individuals (similar both structurally and semantically) occur in the population, then we have a good indication that the search has converged.

We demonstrate the tracking methodology described above on our benchmark problem. In Figure 3a, the best individual of the run is shown, while Figures 4 and 5 display its root ancestry, with subtrees received via crossover marked with rectangles. In all figures of individuals, subtrees are colored according to their occurrence frequency in the population.

With this information we can determine which crossover operations produced an improvement and which tree fragments were transferred. We can then track the origins of these fragments in the population graph to get an idea of their distribution and their impact on the evolutionary process. The following observations apply for the root lineage of the best individual, where we use the notation *root ancestor n* to denote the root ancestor of generation n (the n-th individual of the root lineage):

- The first fragment to have a major impact on the quality of the best individual was the one received by the root ancestor 6 (Figure 4). This fragment was injected into the structure of a rather large intron, (− (+ (÷ 0 X1) (× X2 0)) 0) (in prefix notation), that was changed to (− (× X1 X3) 0) (marked with a dashed rectangle in Figure 3a). Figure 3b shows the propagation of this particular subtree in the population.

- In Figure 3b we see that the subtree (− (× X1 X3) 0) appeared in generation 6 and became part of the majority of future individuals. The colored arcs in the figure mark crossover operations in which this particular subtree was exchanged. As these operations are much fewer than the actual number of individuals containing the fragment, we conclude that the selection process was mostly responsible for its propagation – the root ancestor of generation 6 was selected 8 times.

- A substantial improvement occurred when the intron (÷ (÷ (− 0 0) (+ X2 0) X3)) in root ancestor 8's structure was replaced with a subtree encoding a linear relationship between variables X1 and X3 (Figure 4, generation 8). Tracking this particular subtree we notice it was formed in Generation 5 and was contained by root ancestors 6 and 7. We see in the genealogy graph that root ancestor 8 was obtained through a crossover between root ancestor 7 and itself.

- During reproduction in generations 11 → 12, another intron was placed by crossover in the tree structure of root ancestor 12. This operation effectively removed the variable X3 from the formula, which brought a +0.135 increase in fitness.

- Crossover operations acting on root ancestors 13, 14 and 15 had a neutral effect (swapping existing fragments with identical ones), bringing no change to their respective fitness values.

- The biggest improvement in solution quality (+0.236) occurred when the fragment (− X1 X3) of root ancestor 18 was replaced with a fragment containing the variable $X1$. The quality was further improved (generations 18→19) occurred when an intron, namely the subtree (× X2 0) which evaluates to 0 was replaced with another subtree (exp 0) which evaluates to 1.

- Consequently, the root ancestor of generation 19 began to dominate the population (being the best of the run until generation 25). Figure 1c shows the root lineages of all the individuals in the last generation. We see that up to generation 19, all the lineages are identical. With regard to genetic diversity and fragment frequencies, it is clear that through the combined effects of inheritance (via crossover) and selection, the amount of shared genetic material increases during the run. In other words, the genetic diversity of the population decreases.

- Finally, in generation 27, the constant subtree (exp 0) was replaced with another subtree (− X2 X3) which, in the genetic context of root ancestor 27, had the effect of a parametric improvement to the model formula.

In addition to the above, we can also make a series of more general remarks regarding the evolutionary process. It seems that in the absence of a local search mechanism like mutation which can produce new values for the model parameters, the algorithm relies on constant subtrees (evaluating to a constant value) as a means to introduce constants in the formula. We can observe even in the final generation a number of introns in the tree structure of the best individual (for example (− (exp 0) (exp 0) or (− 0 0)) as can be observed in Figure 5, generations 27-30. With regard to population diversity, we notice the increases in frequency of several subtrees (Figures 4 and 5) and the common lineages in the population (for example in Figure 1c).

4. CONCLUSIONS AND FURTHER WORK

In this paper, we presented an evolution tracking framework which allows the user to navigate the complete history of a genetic programming run. Using a flexible similarity criteria for tree nodes, our tree inclusion algorithm is able to find similar structures in the population, thus allowing us to track down subtrees and investigate the inheritance of genetic material.

Our interactive visual representation of the genealogy graph is able to illustrate the evolution of populations over time in a compact and intuitive manner. Although – for the purposes of this paper – the example run was of reduced dimensions, our implementation is perfectly suitable for tracking large runs in which the genealogy graph varies in size between thousands and tens of thousands of vertices.

In the given example run, a population of 20 individuals was evolved over a period of 30 generations, using only crossover as a variation-producing operator.

This approach opens new possibilities for understanding the dynamics of genetic programming. It allows us to study the interplay between selection and the variation producing operators, the genotype-phenotype mapping and the evolution of diversity. The information present in the graph at any given moment can be used to aggregate various statistics regarding the genetic operators success rate, the size and average fitness of the population, or the trajectories of the best solutions in the search space. Further insights can be gained from investigating the behavior of the algorithm in terms of exploration (global search) and exploitation (local search) capabilities.

Future work may include working out a flexible tree similarity metric (for example, based on the maximum common subtree of two trees) and using it to characterize lineages

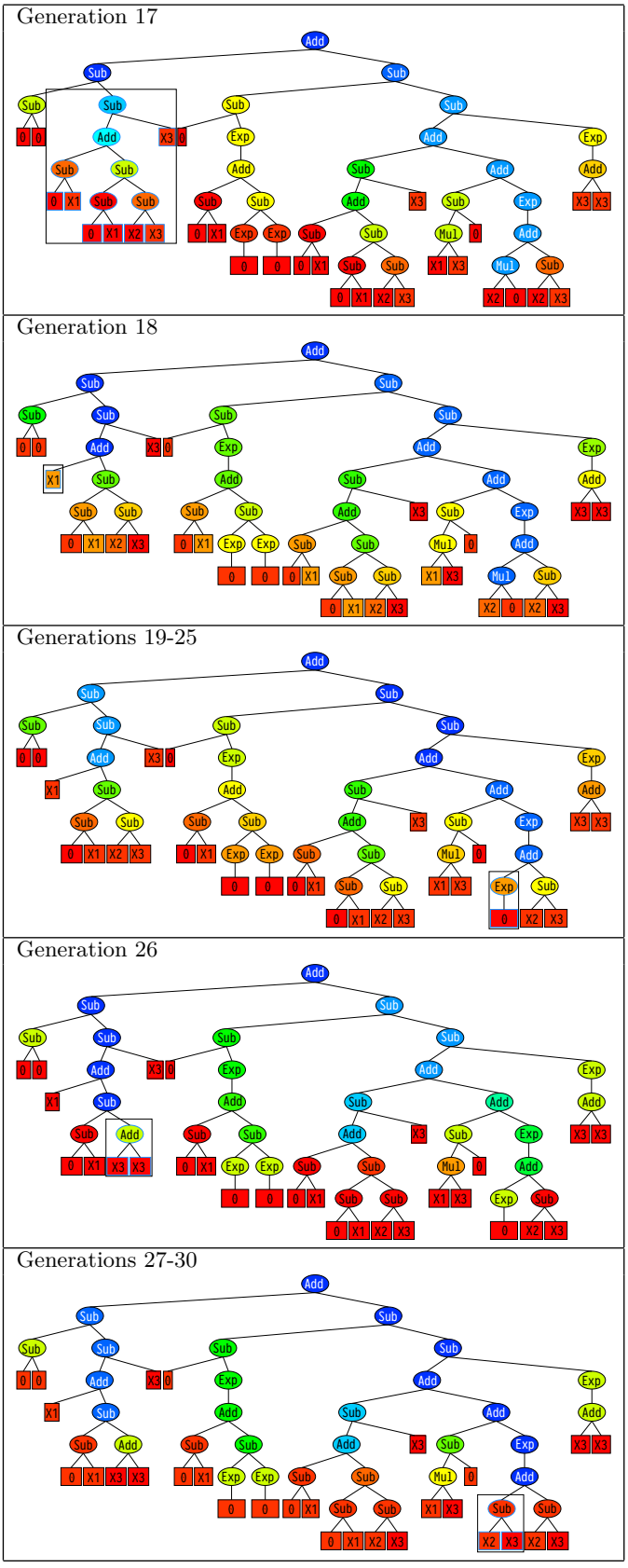


Figure 5: Root lineage of the best individual – generations 17-30

and genealogies in terms of tree distance and to quantify the amount of inherited information.

Using genealogy information combined with structural similarity pointers, next-generation algorithms may gain the ability to automatically decide which building blocks are to be preserved and which to be discarded during the run. Ideally, our algorithm will be able to retroactively steer itself towards unexplored regions of the search space, thus maintaining diversity and avoiding local optima.

5. REFERENCES

[1] M. Affenzeller, S. Winkler, S. Wagner, and A. Beham. *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications*. Numerical Insights. CRC Press, Singapore, 2009.

[2] G. D. Boetticher and K. Kaminsky. The assessment and application of lineage information in genetic programs for producing better models. In *IEEE International Conference on Information Reuse and Integration*, pages 141–146, Waikoloa Village, HI, USA, Sept. 2006. IEEE.

[3] E. Burke, S. Gustafson, and G. Kendall. A survey and analysis of diversity measures in genetic programming. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 716–723, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

[4] E. K. Burke, S. Gustafson, G. Kendall, and N. Krasnogor. Is increased diversity in genetic programming beneficial? an analysis of the effects on performance. In R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, and T. Gedeon, editors, *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, pages 1398–1405, Canberra, 8-12 Dec. 2003. IEEE Press.

[5] H.-B. Dong and J. Chen. Improved genetic programming based on lineage information. In *International Conference on Management and Service Science, MASS '09*, pages 1–5, Wuhan, China, Sept. 2009.

[6] D. Essam and R. I. Mckay. Heritage diversity in genetic programming. In *5th International Conference on Simulated Evolution and Learning*, Bexco, Busan, Korea, Oct. 2004.

[7] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18(6):341–343, June 1975.

[8] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[9] S. Luke, G. C. Balan, and L. Panait. Population implosion in genetic programming. In *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 1729–1739, Chicago, 12-16 July 2003. Springer-Verlag.

[10] N. F. McPhee and N. J. Hopper. Analysis of genetic diversity through population history. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar,

M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1112–1120, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.

[11] R. Poli, W. B. Langdon, and N. F. McPhee. *A field guide to genetic programming*. 2008. (With contributions by J. R. Koza).

[12] N. Q. Uy, N. X. Hoai, M. O'Neill, R. I. McKay, and E. Galvan-Lopez. Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12(2):91–119, June 2011.

[13] E. Vladislavleva. *Model-based Problem Solving through Symbolic Regression via Pareto Genetic Programming*. PhD thesis, Tilburg University, Tilburg, the Netherlands, Aug. 2008.

[14] S. Wagner. *Heuristic Optimization Software Systems - Modeling of Heuristic Optimization Algorithms in the HeuristicLab Software Environment*. PhD thesis, Institute for Formal Models and Verification, Johannes Kepler University, Linz, Austria, 2009.

[15] M. Zhang, X. Gao, W. Lou, and D. Qian. Investigation of brood size in GP with brood recombination crossover for object recognition. In Q. Yang and G. I. Webb, editors, *PRICAI 2006: Trends in Artificial Intelligence, Proceedings 9th Pacific Rim International Conference on Artificial Intelligence*, volume 4099 of *Lecture Notes in Computer Science*, pages 923–928, Guilin, China, Aug. 7-11 2006. Springer.