Visualizing Genetic Programming Ancestries

Nic McPhee[1], Maggie Casale[2], Finzel, Helmuth, & Spector

[1]Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA

[2]Design Center Inc.
St. Paul, Minnesota, USA

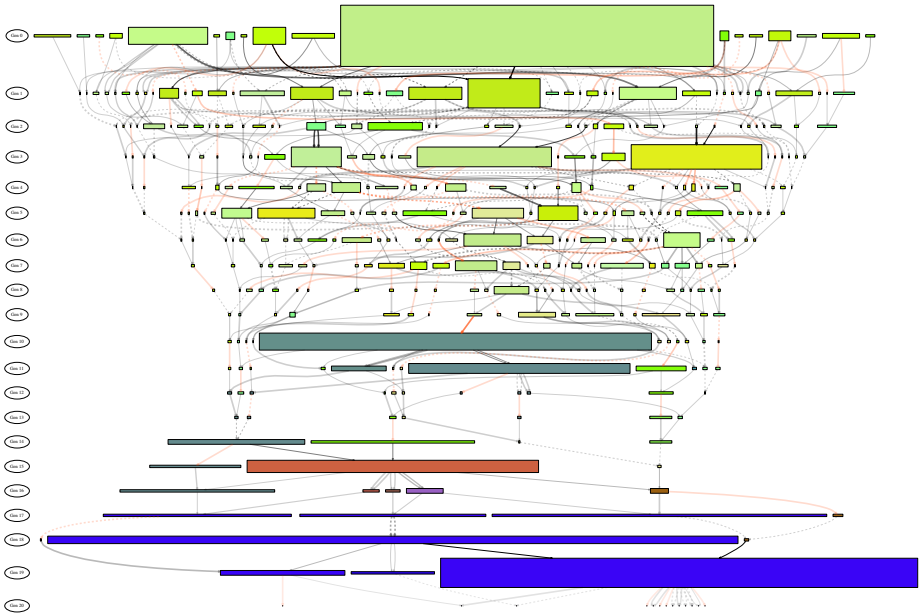
21 July 2016
VizGEC

The great waste

- Genetic programming clearly *works*
- But we rarely know *why* or *how*
- Can generate *huge* amounts of data
- We typically throw it away – & paleontologists weep!
- Silico-paleontology can help us understand and improve our tools

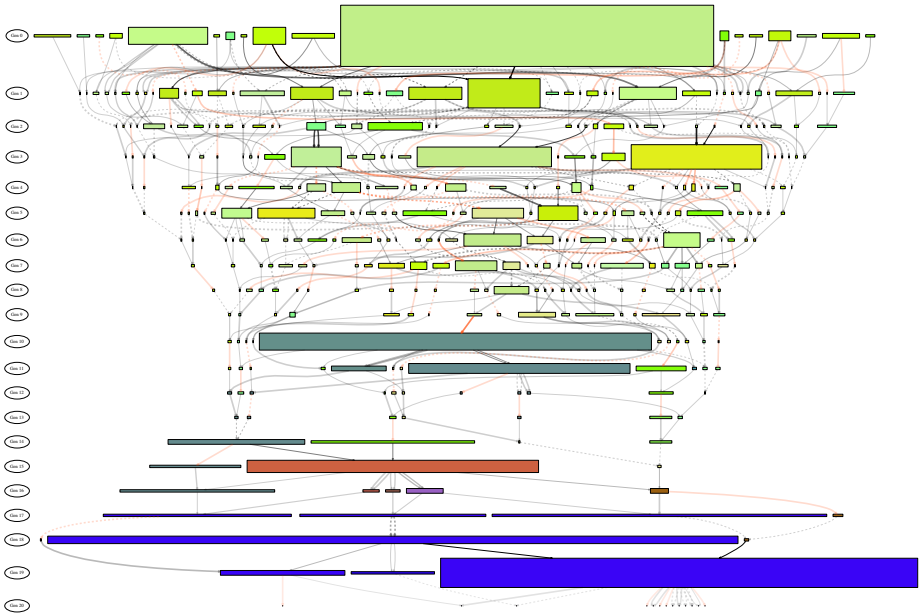
https://www.flickr.com/photos/thibaud_saintin/12361648674

We have the tools!



- Graph databases allow us to collect and analyze lineages
- Processing and *visualizing* these lineages can help us understand run dynamics

We have the tools!



These visualizations help:

- Turn complex runs into “readable” ancestry trees
- Capture and display key data in nodes and edges
- Find valuable properties and patterns in our runs

Other tools also exist

We're not alone:

- Bogdan Burlacu has done very similar work with HeuristicLab [1, 2]
- Tools like ELICIT (Cruz, et al) [3] and EpochX (Vaseux, et al) [6] also support lineage exploration

All of these are functionality provided by a particular EC system

We use:

- "Off-the-shelf" tools (graph DBs & visualization libraries)
- Could be applied to results from most any EC system

Outline

Outline

GP System

GP Tools:

- Clojush implementation of the PushGP system
- Plush (Linear Push) genomes [5]
- Lexicase selection [4]

GP Runs:

- 1,000 individuals per generation
- Up to 300 generations here,
  over 1,000 generations in other cases
  100Ks or millions of individuals

Test problem

Replace-Space-With-Newline software synthesis problem:

- Two tasks to solve:
  1. Print input string with all spaces replaced by newlines
  2. Return number of non-space characters in input
- Results in two error values per test case
- Used 100 test cases,
  So a total of 200 error values stored in vector

We have hundreds of runs of this problem which we'll use as examples throughout the talk.

Database schema

Used TitanDB graph database to store our data.

Nodes = individuals

- UUID
- Plush genome
- Error vector
- Number of children
- Number of selections

Edges = parent-child relationships

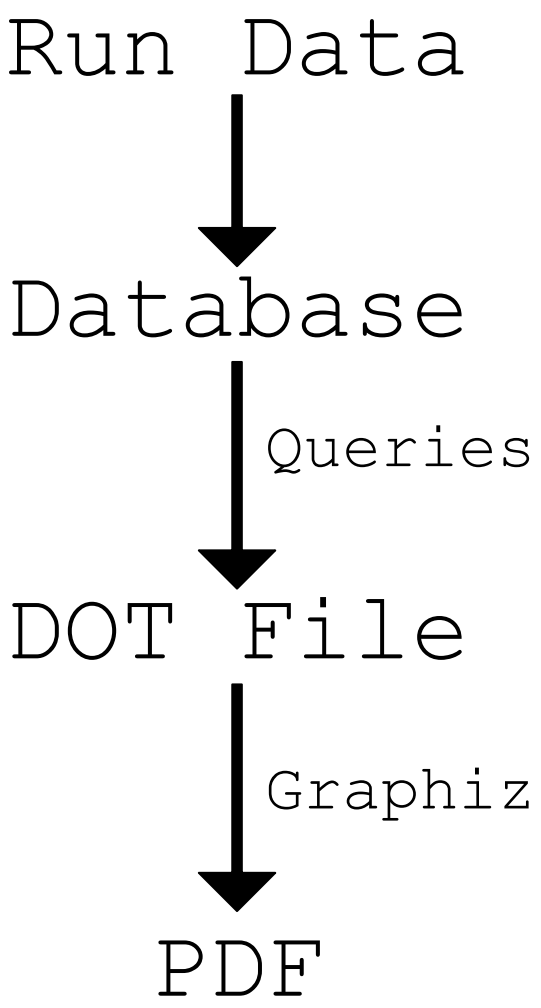
- Genetic operator
- Damerau-Levenshtein distance

Outline

Graphviz and DOT for visualization

- Load run data into database for processing & analysis
- Queries extract relevant data
- Scripts generate DOT input
- DOT generates PDF output

PDF output supports arbitrary zooming
– huge amounts of detail in paper



Genetic Operators

Edges represent parent-child relationships

Styled to indicate genetic operators used to create the child

Two classes of operators: Alternation & Mutation

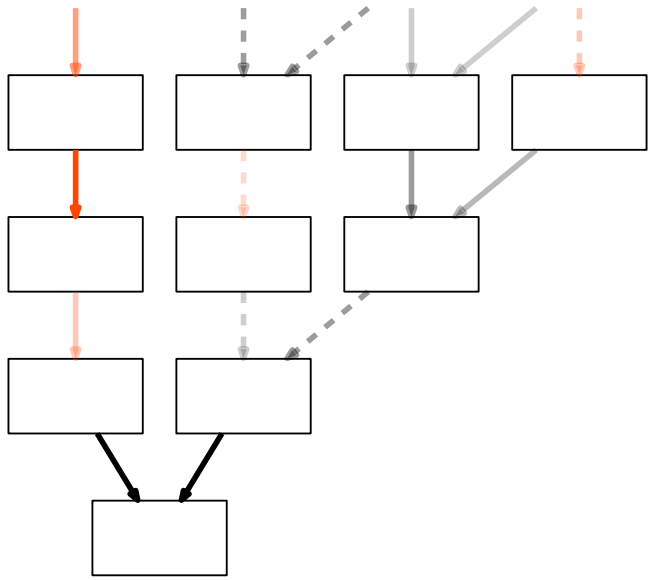
Four different operators:

- Alternation
- Uniform-Mutation
- Alternation & Uniform-Mutation
- Uniform-Close-Mutation

Edge Coloring & Style

Edge Coloring:

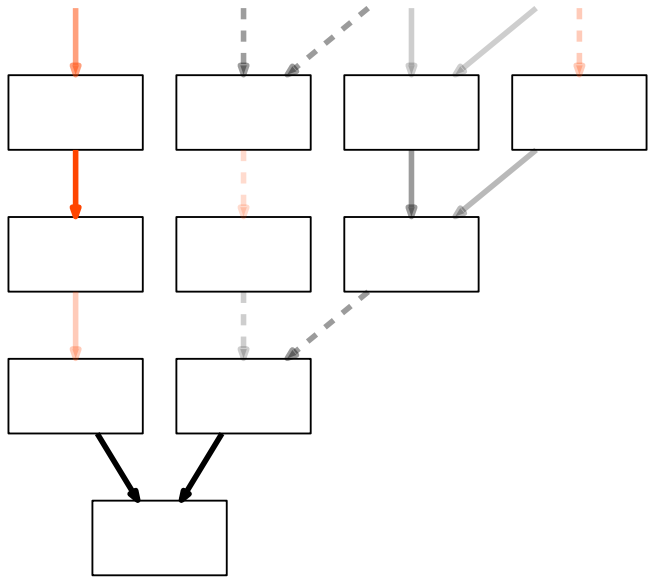
- Alternation is
  black and solid
- Uniform-Mutation is
  orange and solid
- Alternation & Uniform-Mutation
  black and dashed
- Uniform-Close-Mutation is
  orange and dashed



Edge Coloring & Style

Edge Opacity: Based on number of children of the child.

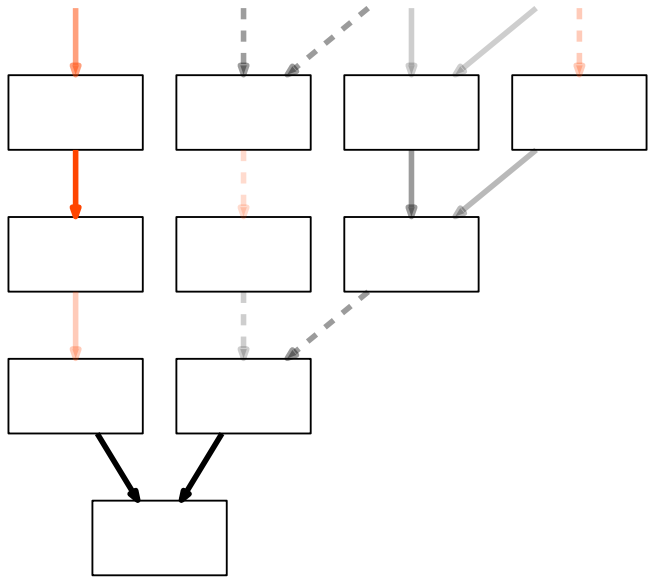
- Transparent: Few Children
- Opaque: Many Children



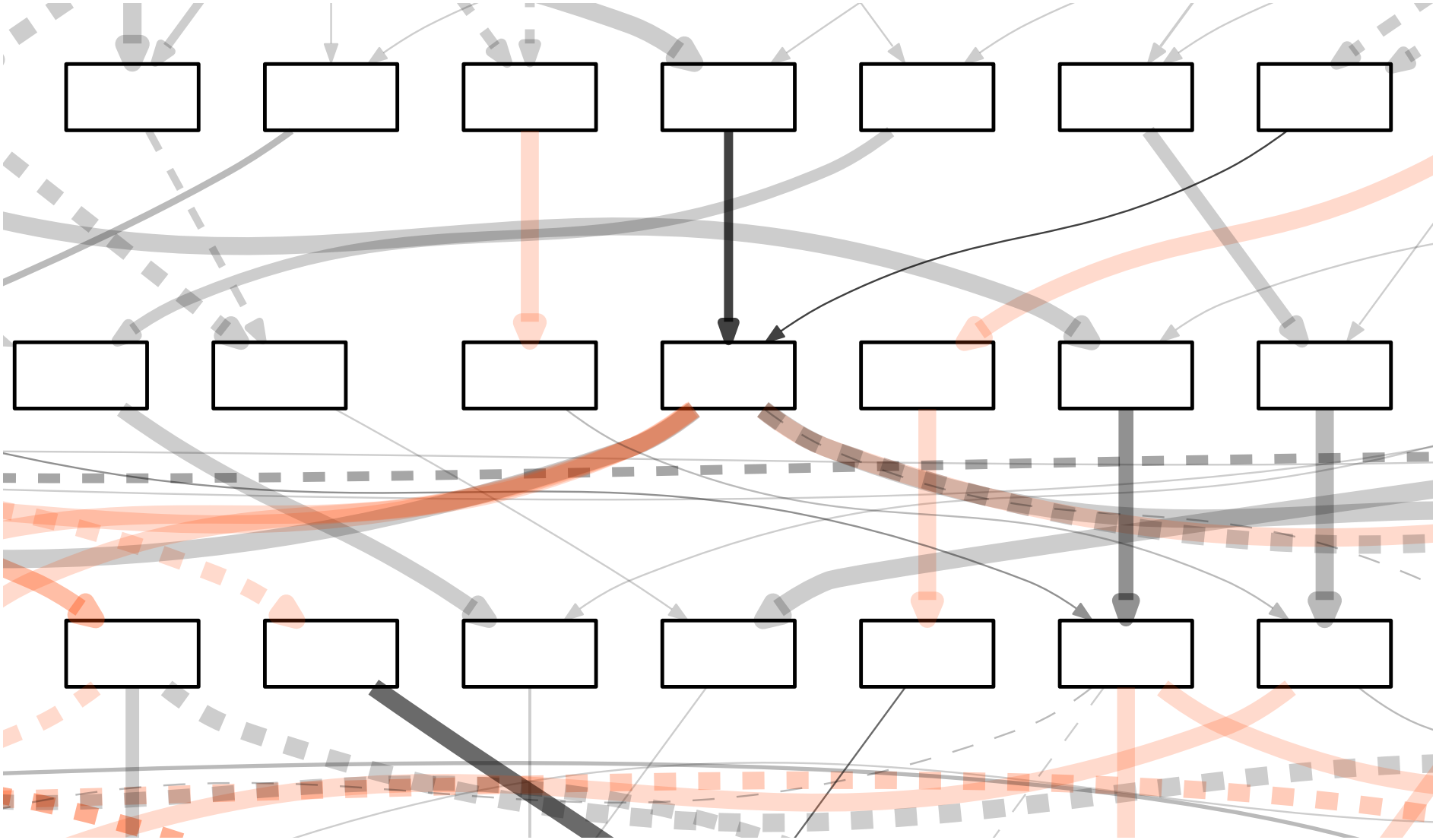
Edge Coloring & Style

Edge Thickness: Based on Damerau-Levenshtein distance, (measure of genome difference)

- Thin: Very Different Genome
  - High DL-Distance
- Wide: Very Similar Genome
  - Low DL-Distance



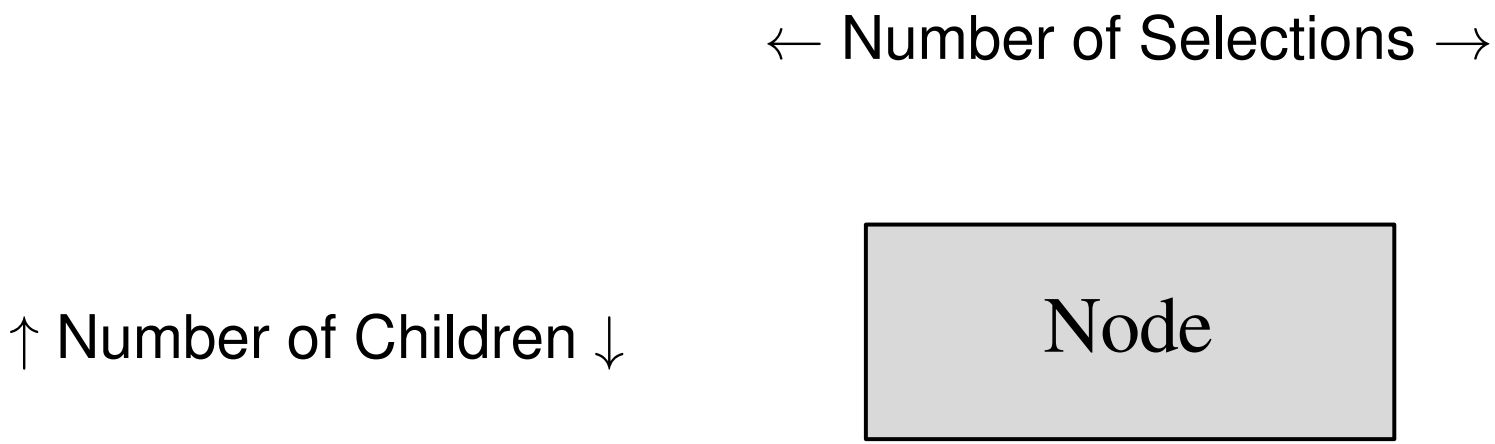
Edge Coloring & Style Example



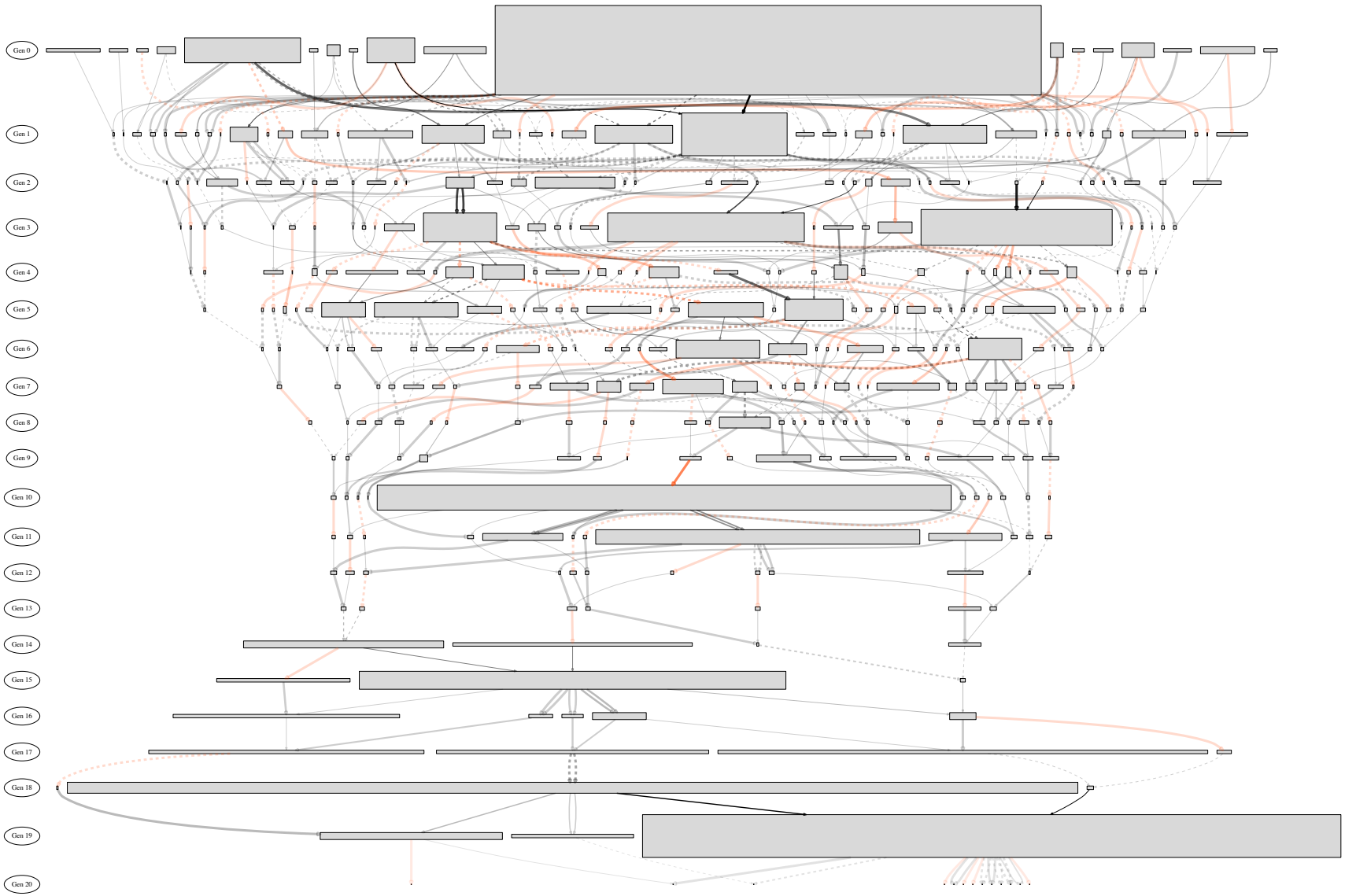
Node Basics

Nodes represent an individual or program:

- Number of selections: Width
- Number of children: Height

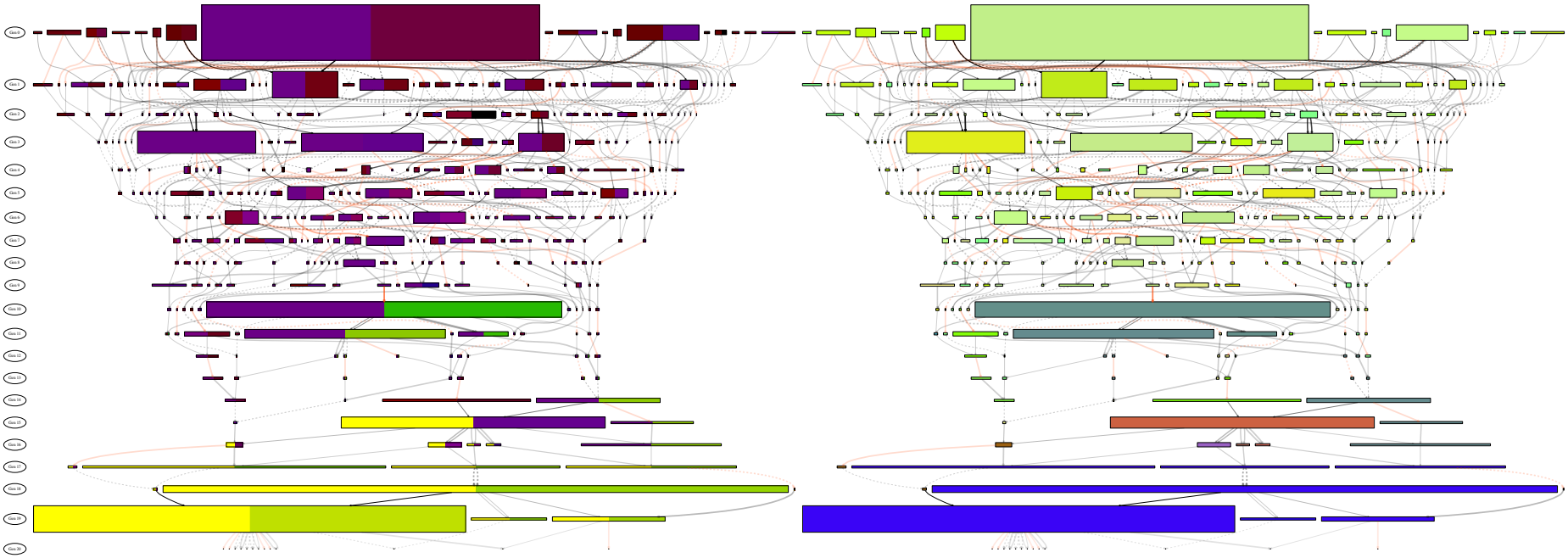


Run 0: No Node Color

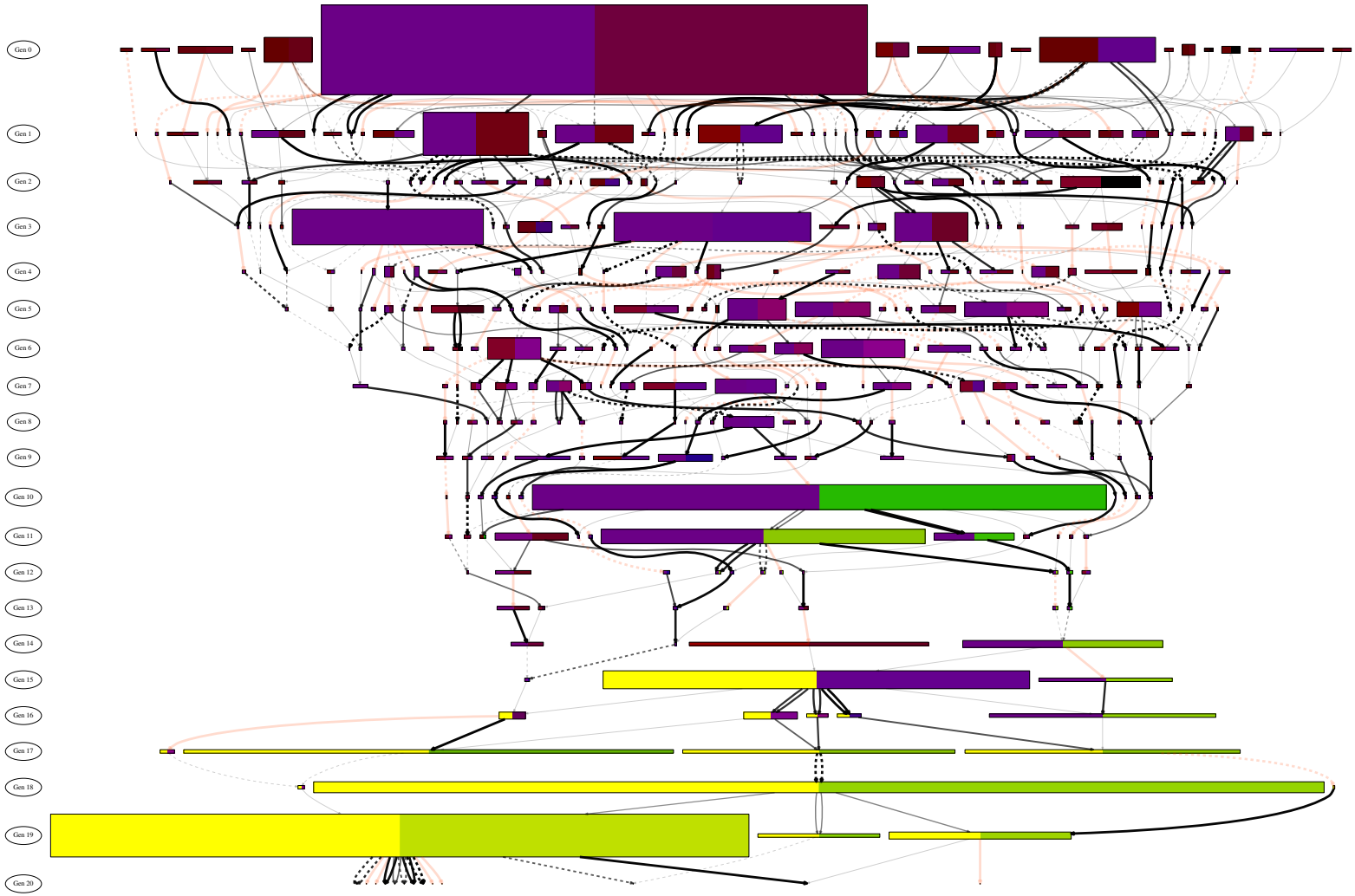


Node Coloring

There are two different techniques we use:
Error Based & RBM

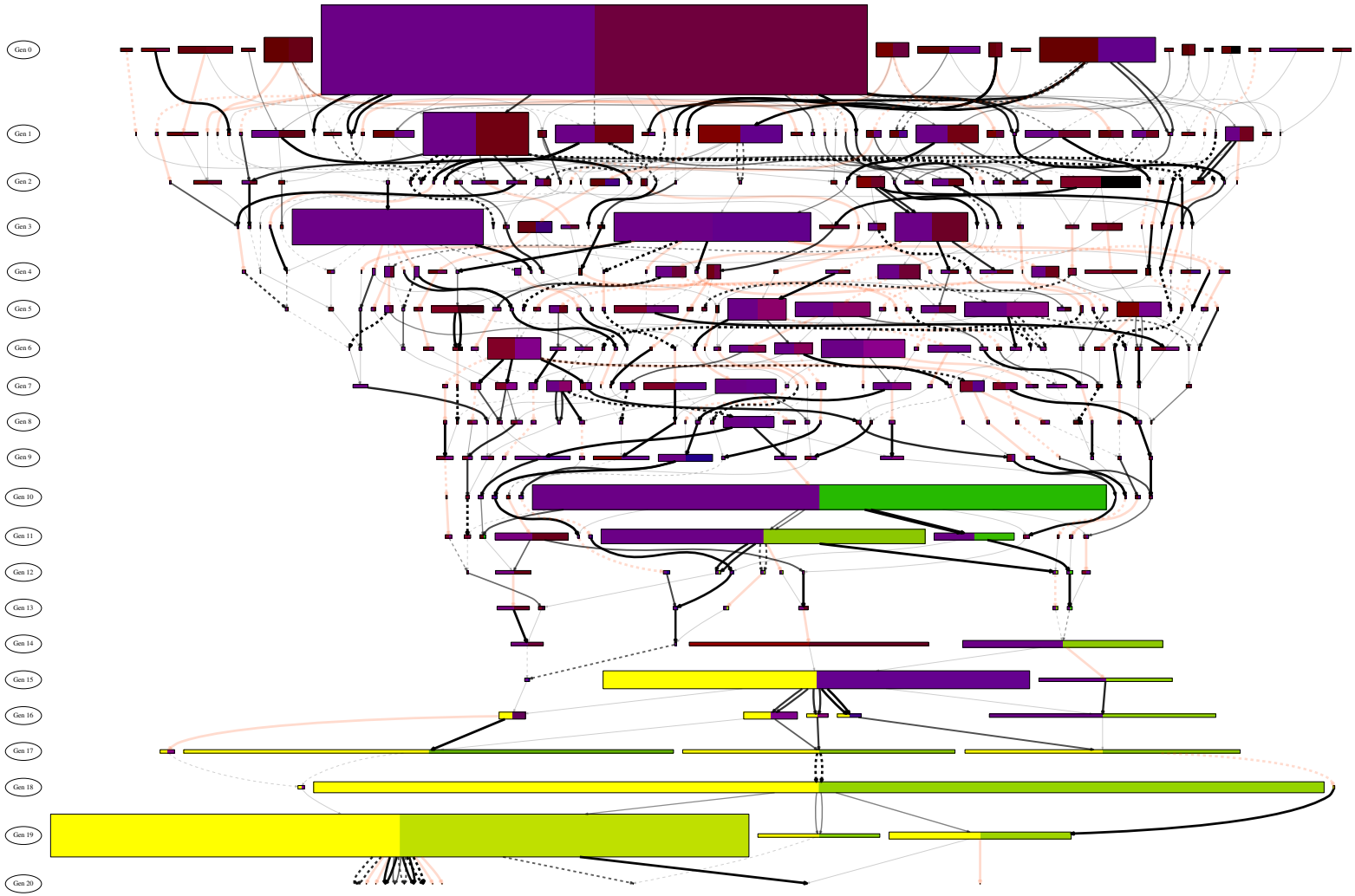


Error Based Node Coloring



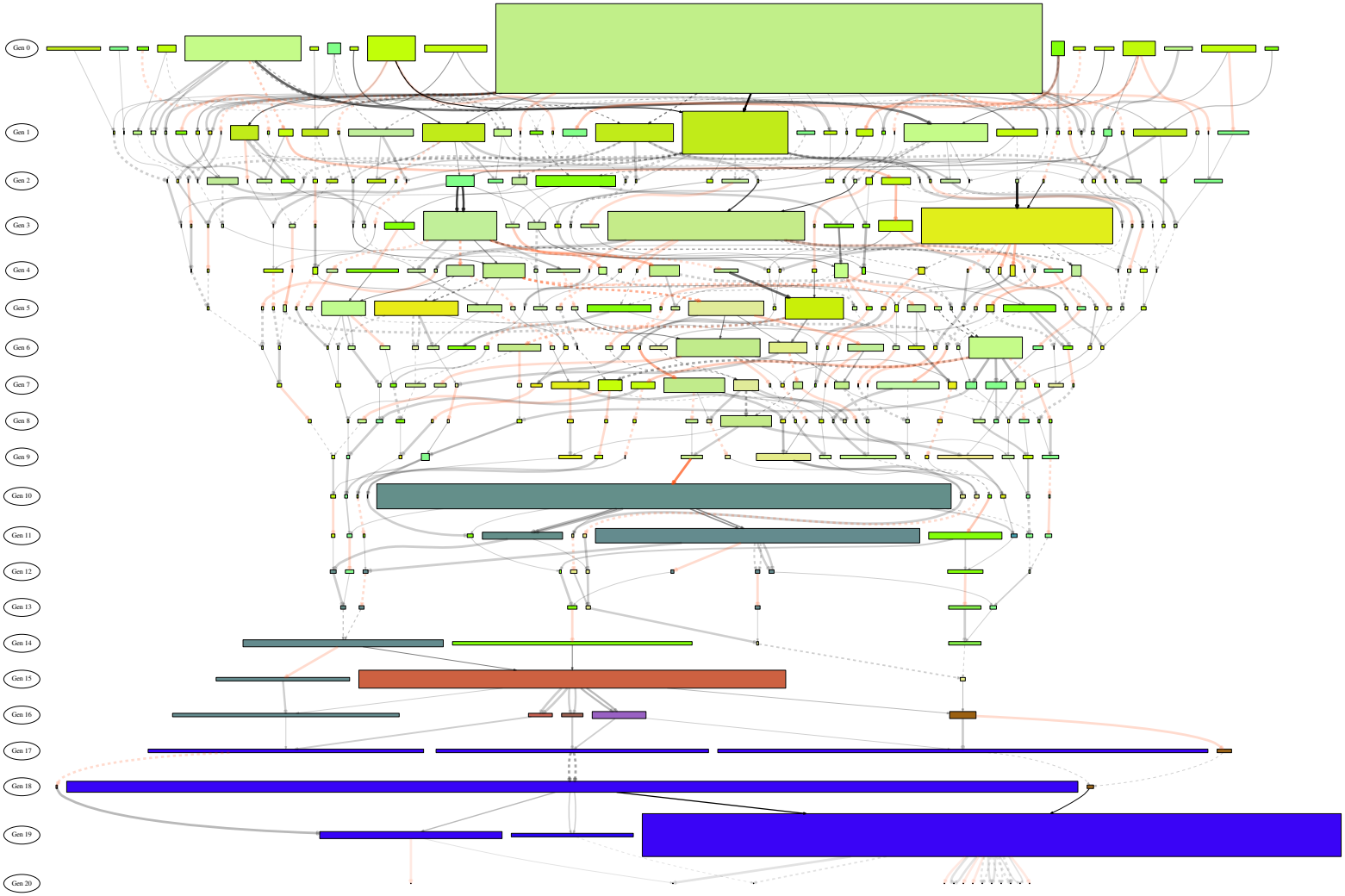
Two colors based on the percent of solved test cases for each task

Error Based Node Coloring



Additional shading (i.e., darker) for exceptionally high total errors

Restricted Boltzmann Machine Node Coloring



Dimensionality Reduction Problem: 200 test cases $\rightarrow$ 3 RGB values

Restricted Boltzmann Machine Technique

Train an RBM as a simple autoencoder: 200 inputs to 3 outputs

200 test case vector, each case is assigned 1 or 0
(1 for has error or 0 for no error)

↓

New vector made into binary string for machine input

↓

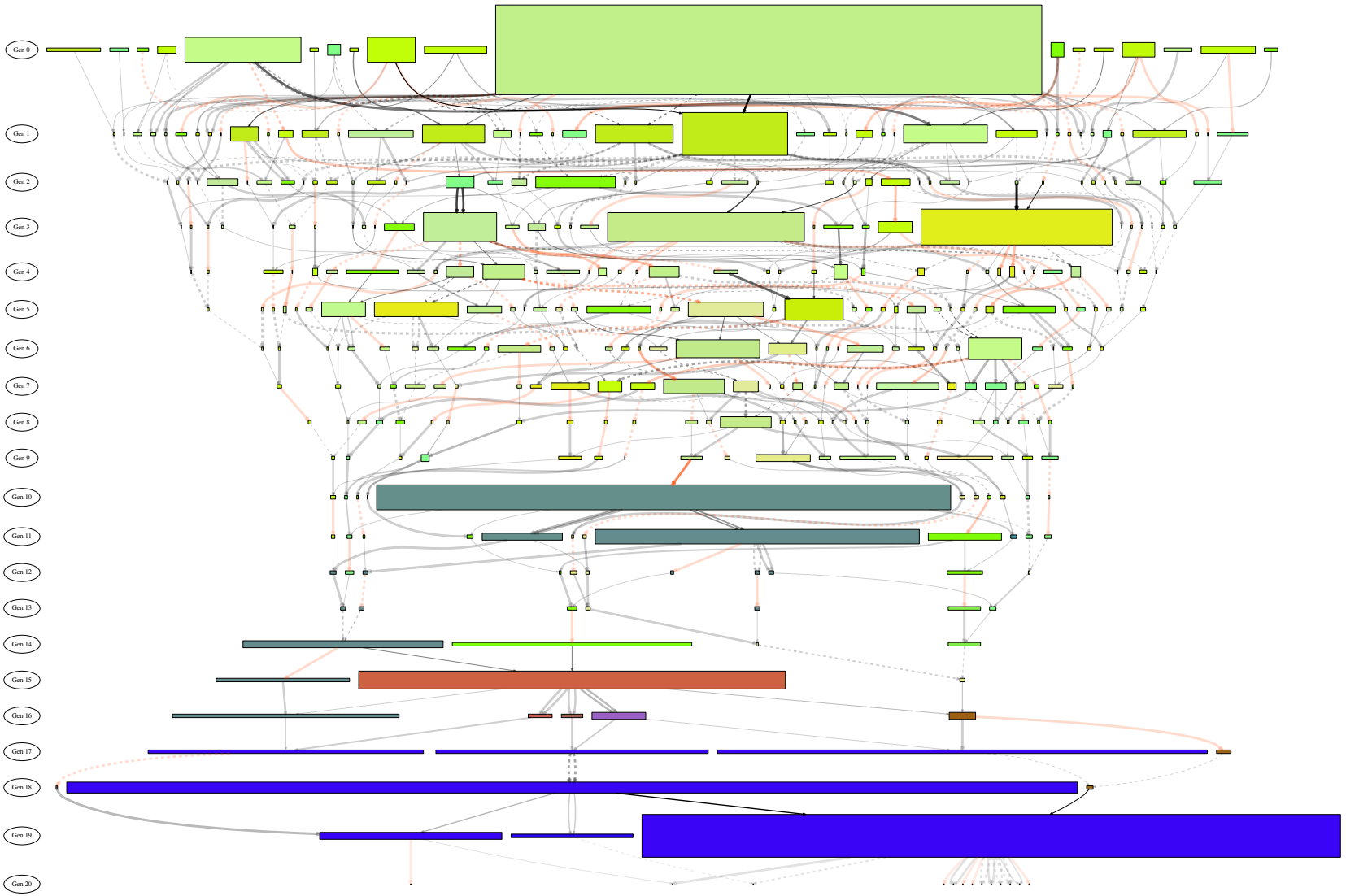
Training RBM analyzes and discovers patterns

↓

Returns RBG values

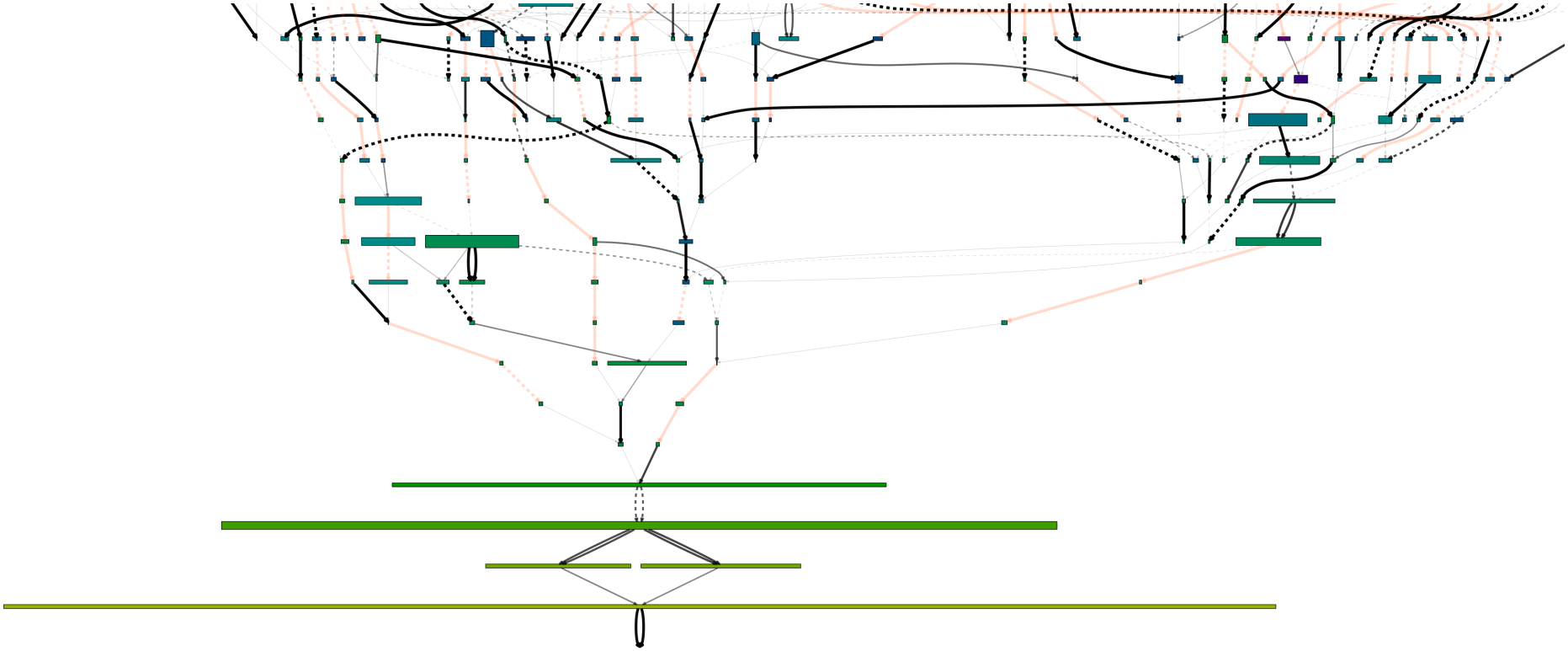
Similarly colored individuals indicate similar patterns in test cases

Restricted Boltzmann Machine Node Coloring



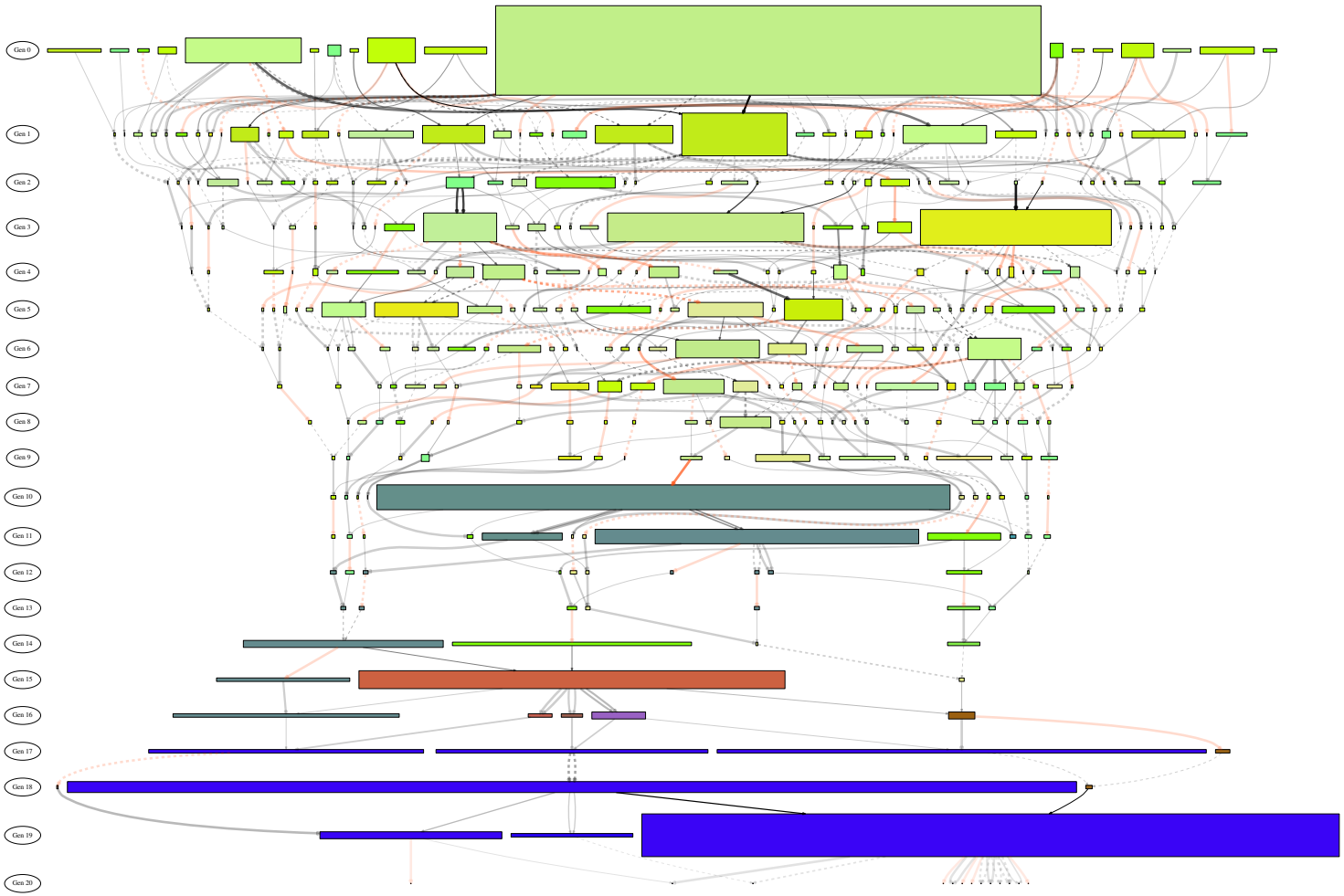
Outline

Hyperselected nodes are highly visible



In our graphs, we can easily see hyperselection as very wide nodes.

Hyperselection is very common



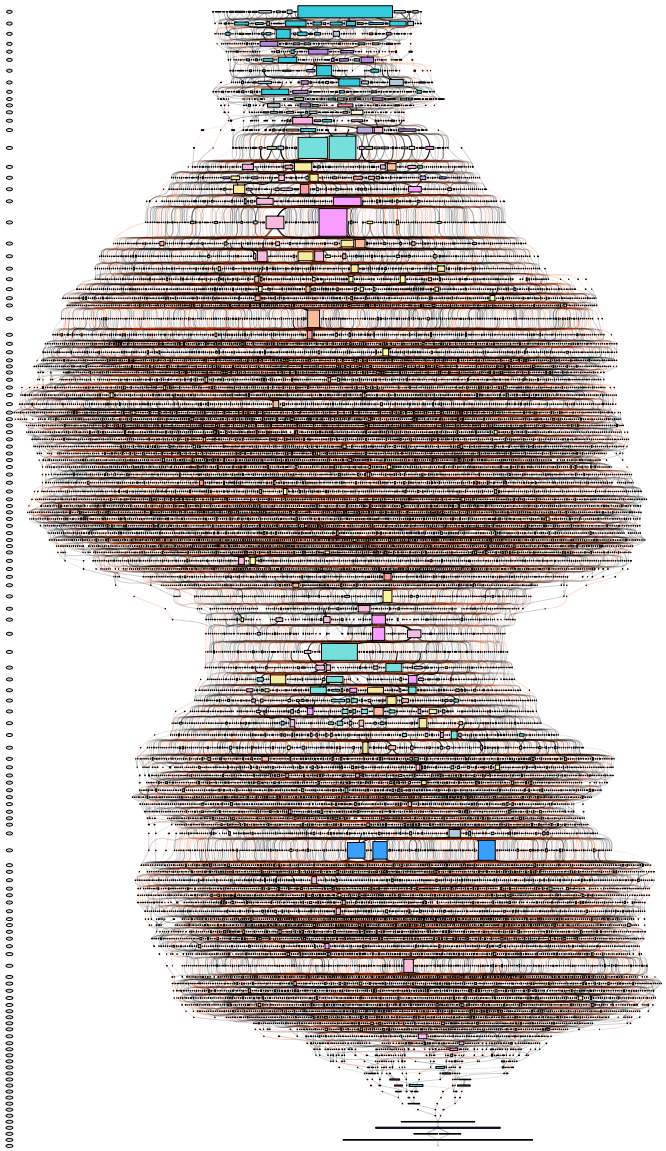
Lots of hyperselection – how important is it to the success of runs? [**?**]

Filtering ancestry trees

We only graph ancestors of “winning” individuals.

But still a lot of data!

- 22,435 individuals
- 35,403 edges
- (Avg. branching factor of 1.5)



Filtering ancestry trees

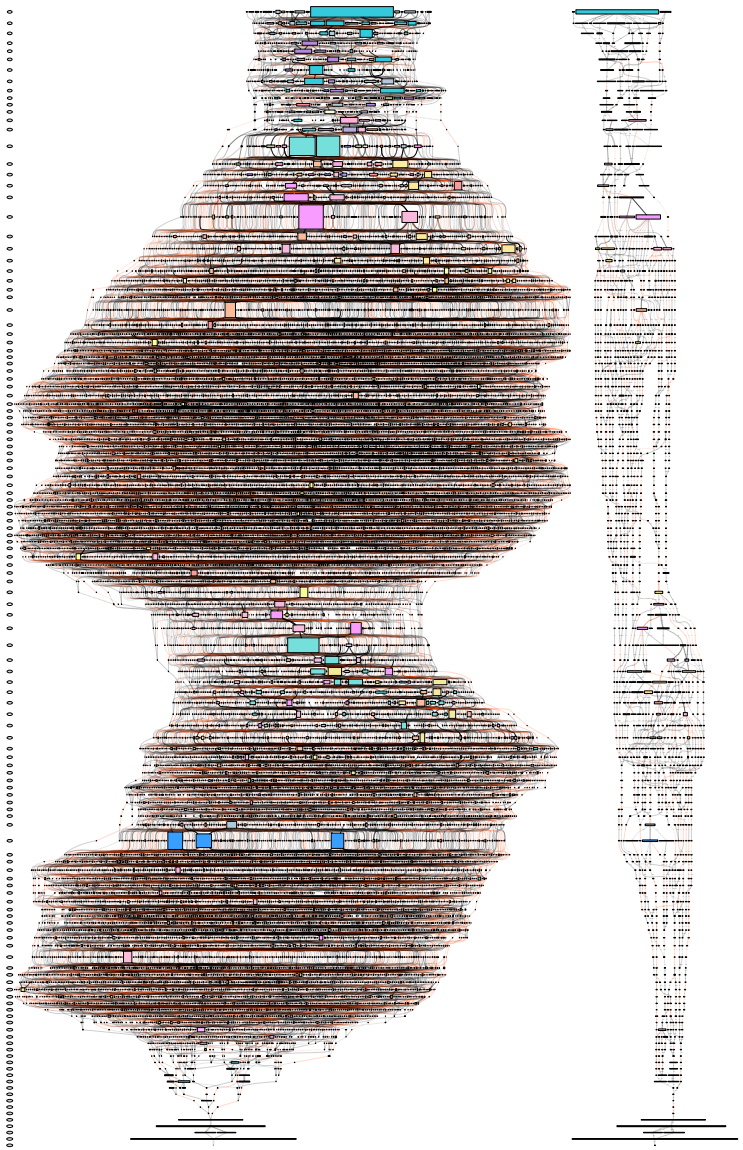
So we filter out parents who did not contribute much genetic information.

Much more “readable”:

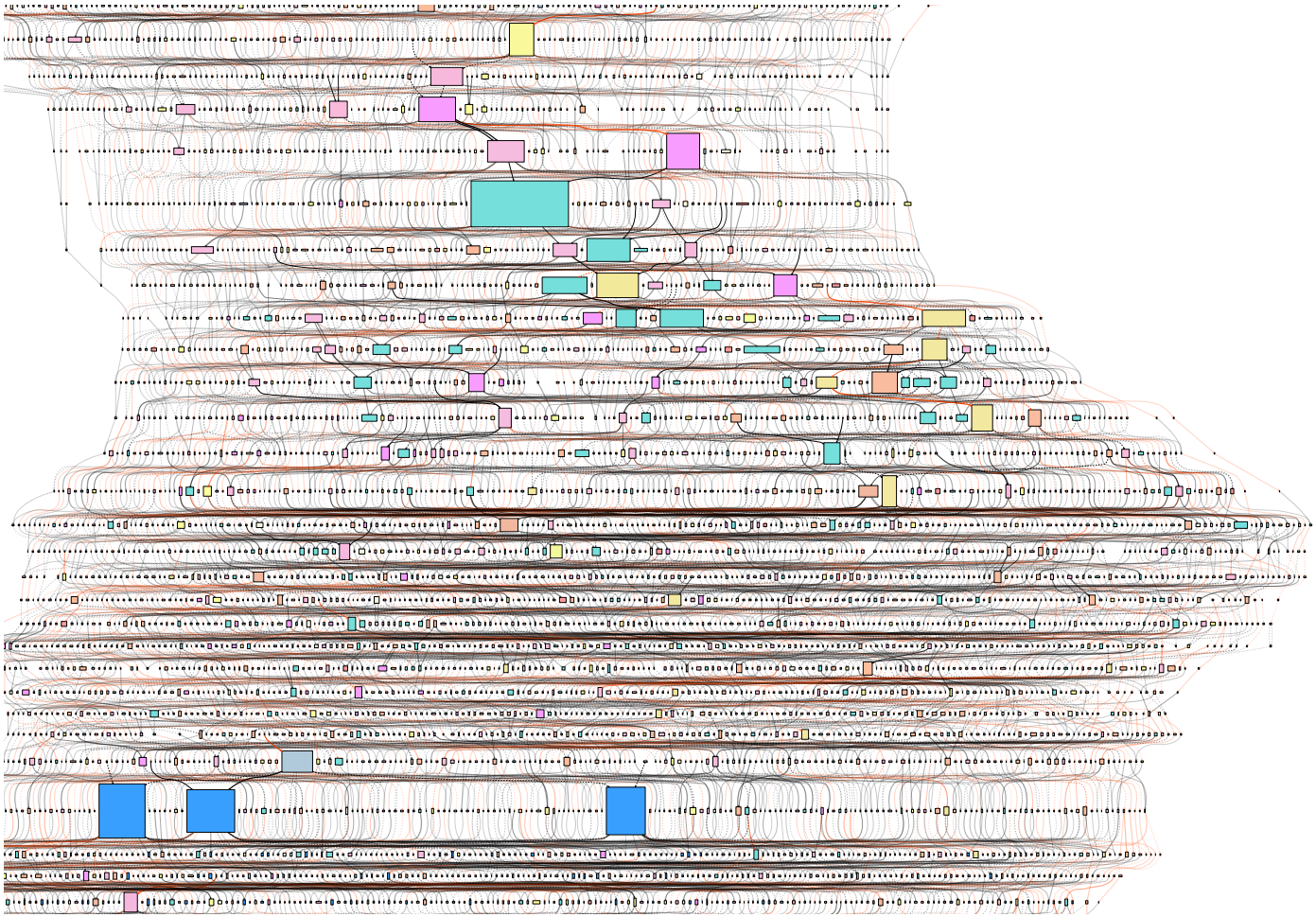
- 1,597 individuals
- 1,795 edges
- Avg. branching factor of 1.1

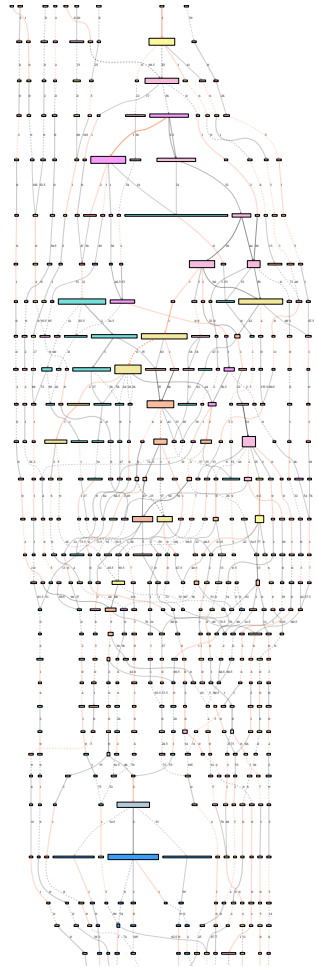
Filtering ancestry trees

- Filtering in paper is a bit of a hack
- Currently (last few weeks) working on tracking every gene (instruction) as it passes from parent to child
- Allows us to visualize *exactly* the parents that contributed genetic material



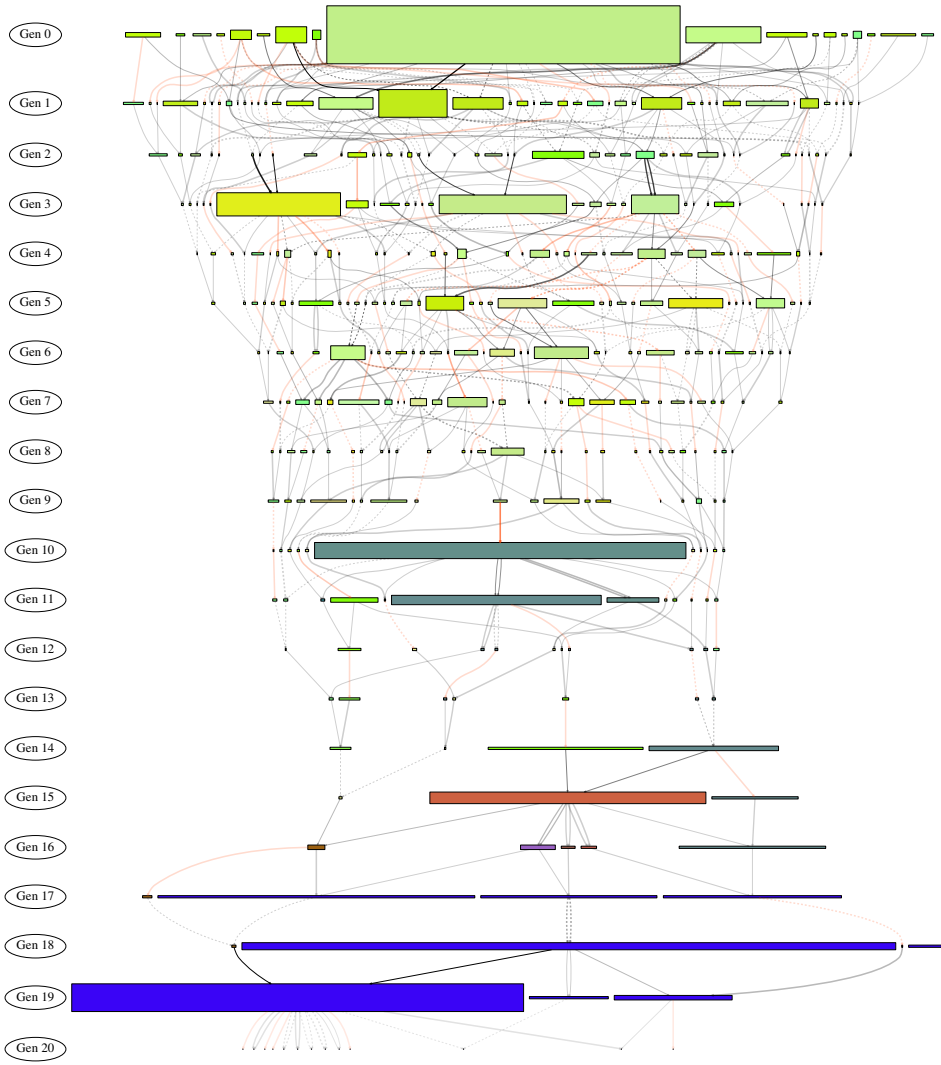
Close-up of filtering



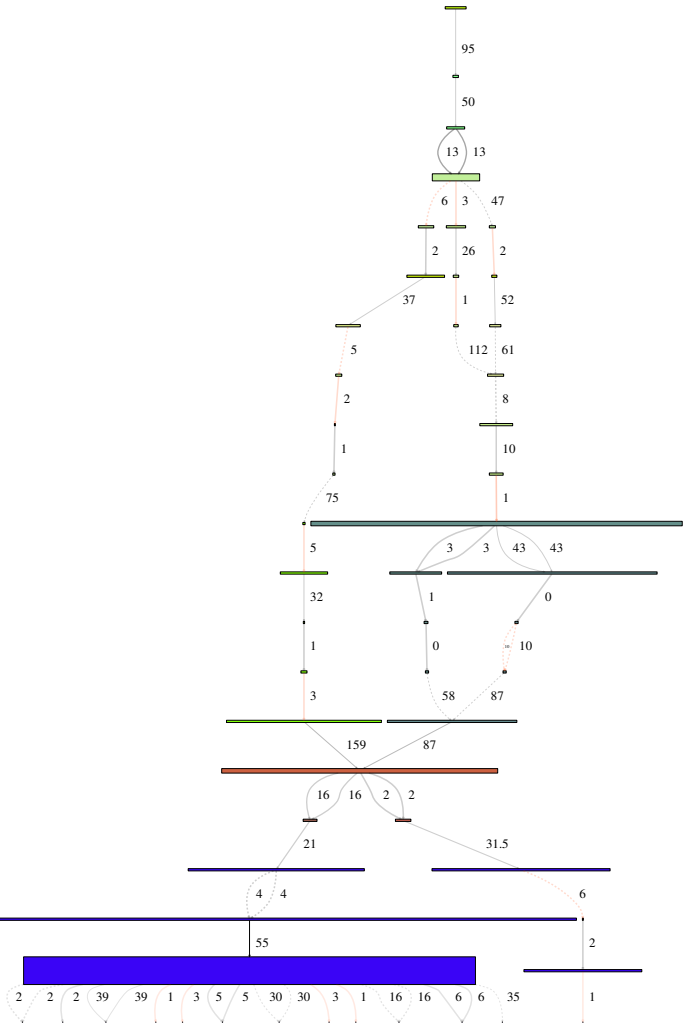


Another filtering example

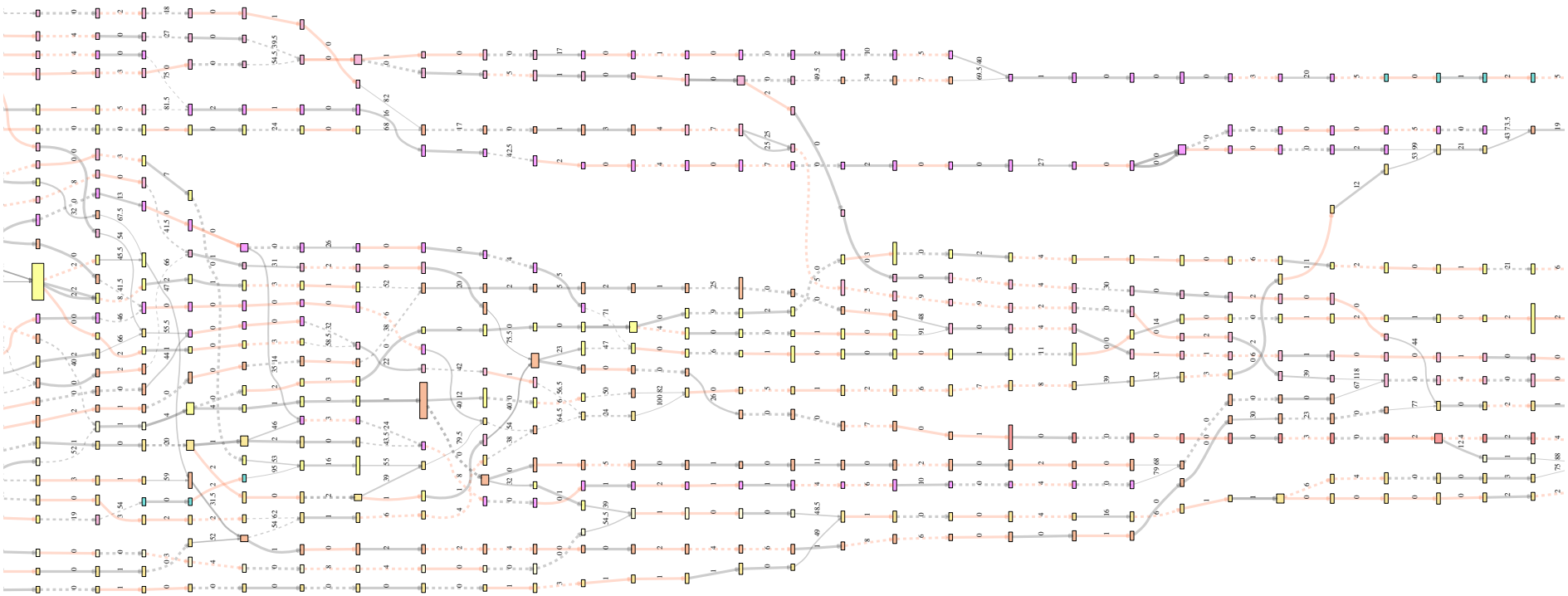
394 Individuals



54 Individuals

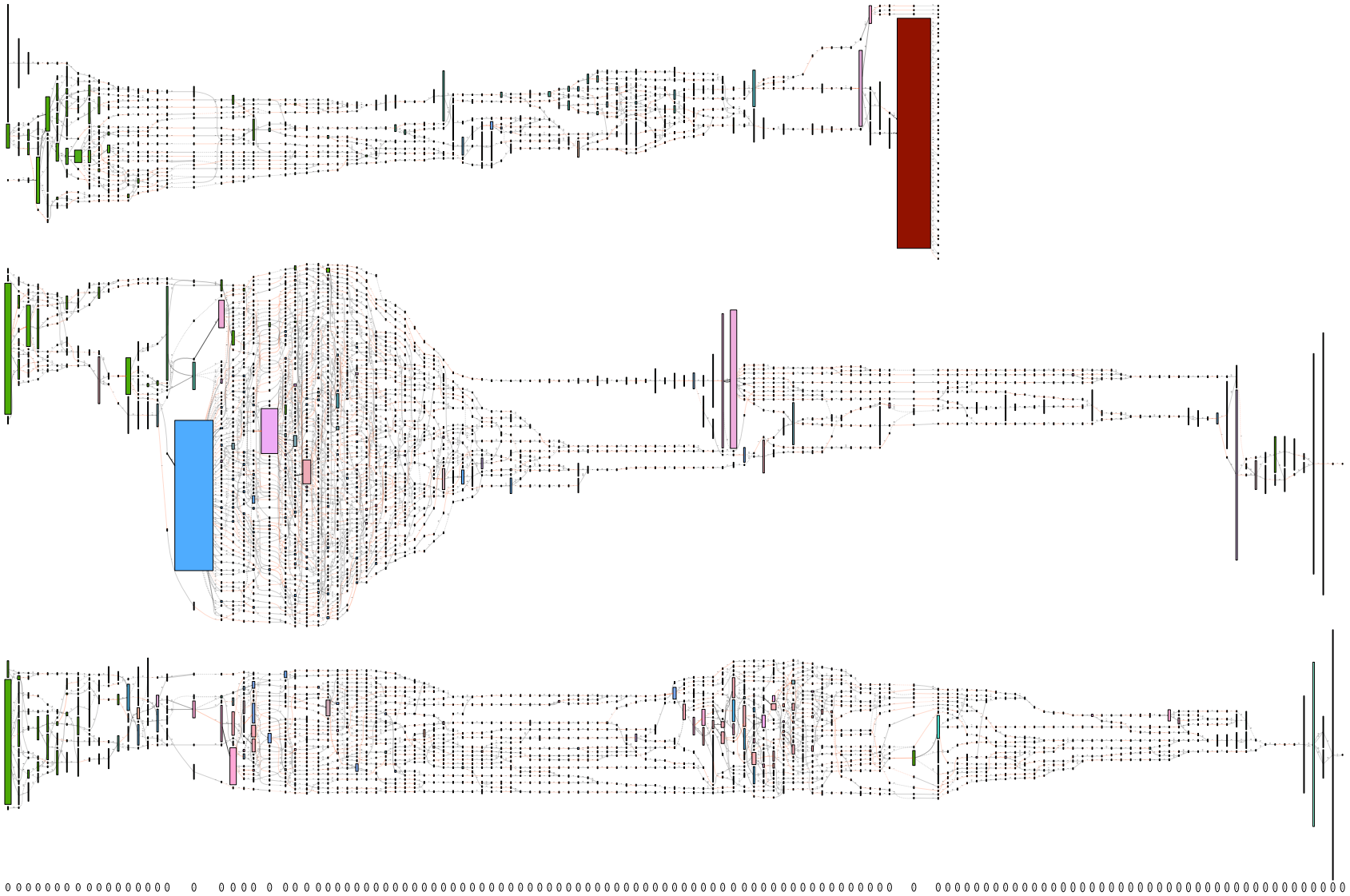


Long (nearly) linear chains



Filtering reveals long nearly linear sequences of pseudo-mutations

Ability to compare runs



Outline

Conclusions

These visualizations have:

- Helped turn complex runs into “readable” ancestry trees
- Modified aspects of both nodes and edges to capture key information from the run dynamics
- Explored ways to filter ancestry trees
- Highlighted prevalence of hyperselection in lexicase selection runs
- Highlighted long chains of pseudo-mutations
- Suggested numerous research directions

Future work

- Improve the filtering
  - Currently working to track at the level of genomes (instructions)
  - Allows more precise tracking of genetic contributions
- Enable more dynamic visualizations
  - Deep Tree exhibit [**?**]
  - Burlacu's work
- Enable more "on-line" visualization during runs

Thanks!

Thank you for your time & attention!

:-)

Special thanks to Bill Tozier, and the folks at the Computational Intelligence Lab at Hampshire College.

This material is based upon work supported by the National Science Foundation under Grants No. 1129139 and 1331283.

References I


B. Burlacu, M. Affenzeller, M. Kommenda, S. Winkler, and G. Kronberger.
Visualization of genetic lineages and inheritance information in genetic programming.
In *GECCO '13 Companion: Proceeding of the fifteenth annual conference companion on Genetic and evolutionary computation conference companion*, pages 1351–1358, Amsterdam, The Netherlands, 6-10 July 2013. ACM.


B. Burlacu, M. Affenzeller, S. Winkler, M. Kommenda, and G. Kronberger.
Methods for genealogy and building block analysis in genetic programming.

References II

In *Computational Intelligence and Efficiency in Engineering Systems*, volume 595 of *Studies in Computational Intelligence*, pages 61–74. Springer International Publishing, 2015.


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.


T. Helmuth, L. Spector, and J. Matheson.
Solving uncompromising problems with lexicase selection.
IEEE Transactions on Evolutionary Computation, 2014.

References III


T. Helmuth, L. Spector, N. F. McPhee, and S. Shanabrook.
Linear genomes for structured programs.
In *Genetic Programming Theory and Practice XIV*, Genetic and Evolutionary Computation. Springer.


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.