

Automating Algorithm Design through Genetic Programming Hyper-Heuristicss

Elsa Browning

Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA

April 15, 2017
Morris, MN

What does the title mean?

- Reducing the human component in algorithm design



[https://scratch.mit.edu/
discuss/m/topic/200574/](https://scratch.mit.edu/discuss/m/topic/200574/)

What does the title mean?

- Reducing the human component in algorithm design
 - More work at the beginning, more possibilities



[https://scratch.mit.edu/
discuss/m/topic/200574/](https://scratch.mit.edu/discuss/m/topic/200574/)

What does the title mean?

- Reducing the human component in algorithm design
 - More work at the beginning, more possibilities
 - Genetic programming hyper-heuristics as a method to the madness



[https://scratch.mit.edu/
discuss/m/topic/200574/](https://scratch.mit.edu/discuss/m/topic/200574/)

Outline

- 1 Background
- 2 Hyper-heuristics
- 3 Genetic Programming Variants
- 4 Autoconstruction
- 5 Summary

Outline

1 Background

- Evolutionary Computation
- Genetic Programming

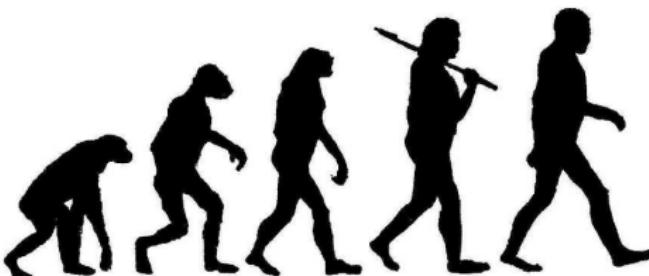
2 Hyper-heuristics

3 Genetic Programming Variants

4 Autoconstruction

5 Summary

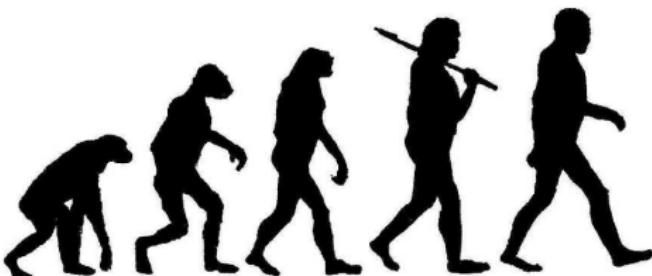
Evolutionary Computation



<https://www.spigotmc.org/attachments/evolution-jpg.137048/>

- Subfield of Artificial Intelligence

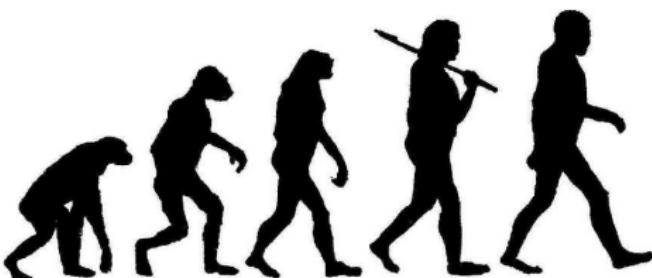
Evolutionary Computation



<https://www.spigotmc.org/attachments/evolution-jpg.137048/>

- Subfield of Artificial Intelligence
 - Algorithms based on biological evolution

Evolutionary Computation

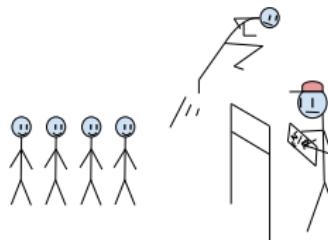
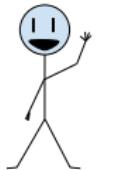


<https://www.spiqotmc.org/attachments/evolution-jpg.137048/>

- Subfield of Artificial Intelligence
 - Algorithms based on biological evolution
 - Uses lots of terminology from biology, doesn't always mean same thing as term means in biology.

Evolutionary Computation – Terminology

- **Individual** – a potential solution to a problem (or set of problems)
 - **Population** – a group of individuals
 - **Fit** – how well suited an individual is at solving a problem
 - **Fitness Test** – a set of tests to determine how fit an individual is.



Evolutionary Computation – Terminology

- **Mutation** – an insertion, deletion, or small change in the code of an individual, creating a new individual
 - **Sexual reproduction** – when two or more individuals are munged together to create a new individual

Evolutionary Computation – Terminology

- **Mutation** – an insertion, deletion, or small change in the code of an individual, creating a new individual
 - **Sexual reproduction** – when two or more individuals are munged together to create a new individual

If individual A experiences a mutation to create individual B, then:

- **Parent** – Individual A
 - **Child** – Individual B



Evolutionary Computation – Terminology

- **Generation** – a population of individuals
- **Global optima** – best solution (or solutions) possible
- **Stopping point** – time limit, or generation limit.

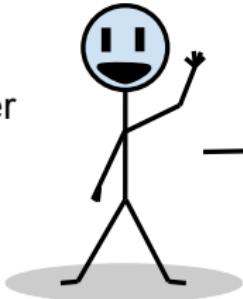
Genetic Programming

A family of algorithms in Evolutionary Computation that uses biological techniques to create programs to solve computational problems.

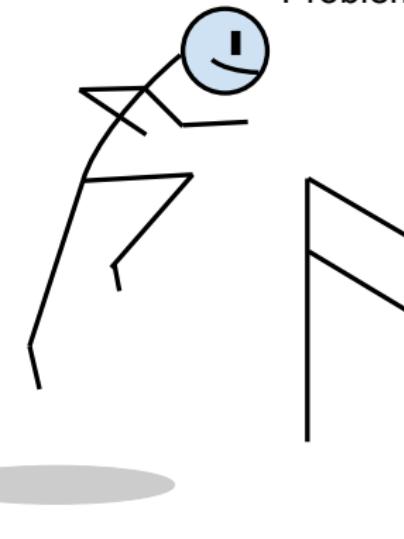
Genetic Programming

Individual/Potential Solution

Answer

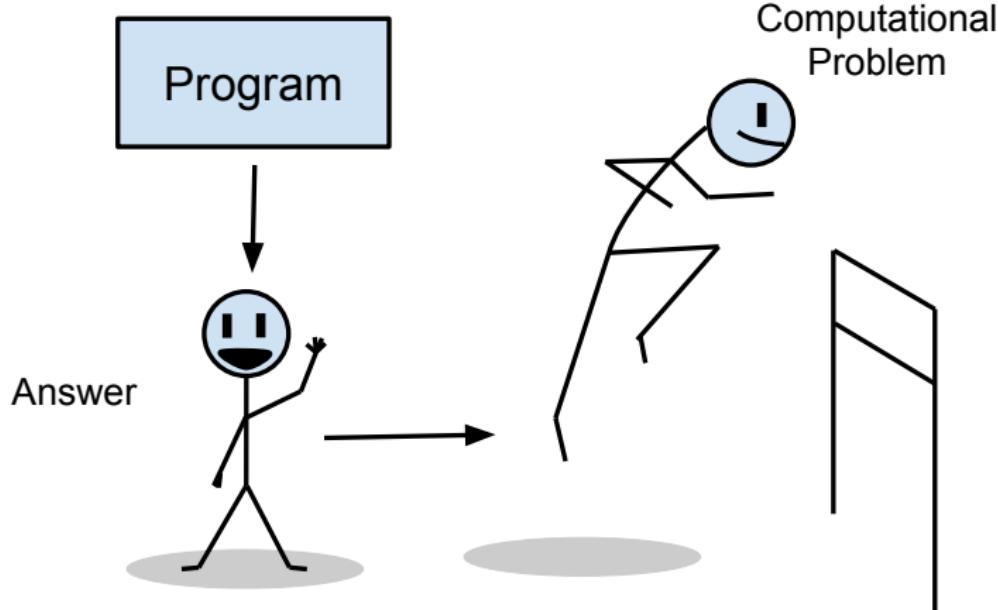


Computational Problem

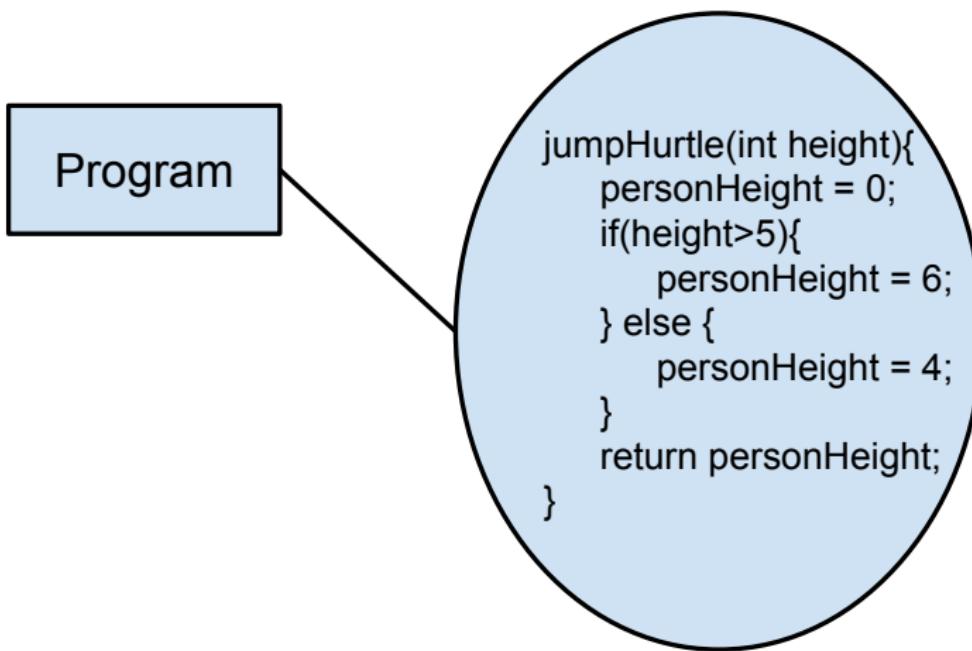


Genetic Programming

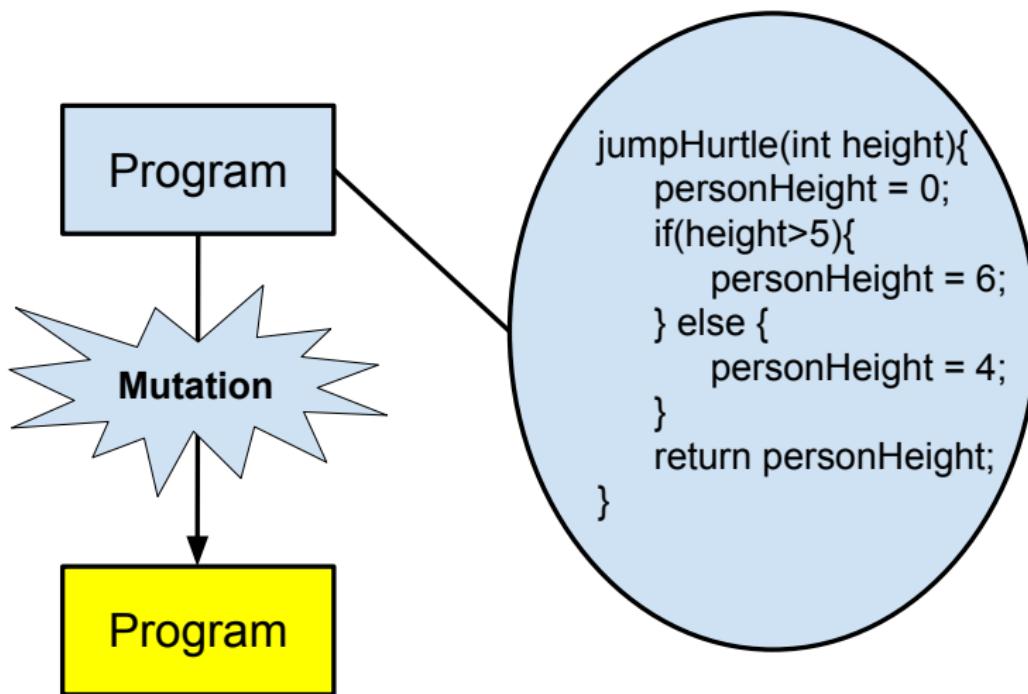
Individual/Potential Solution



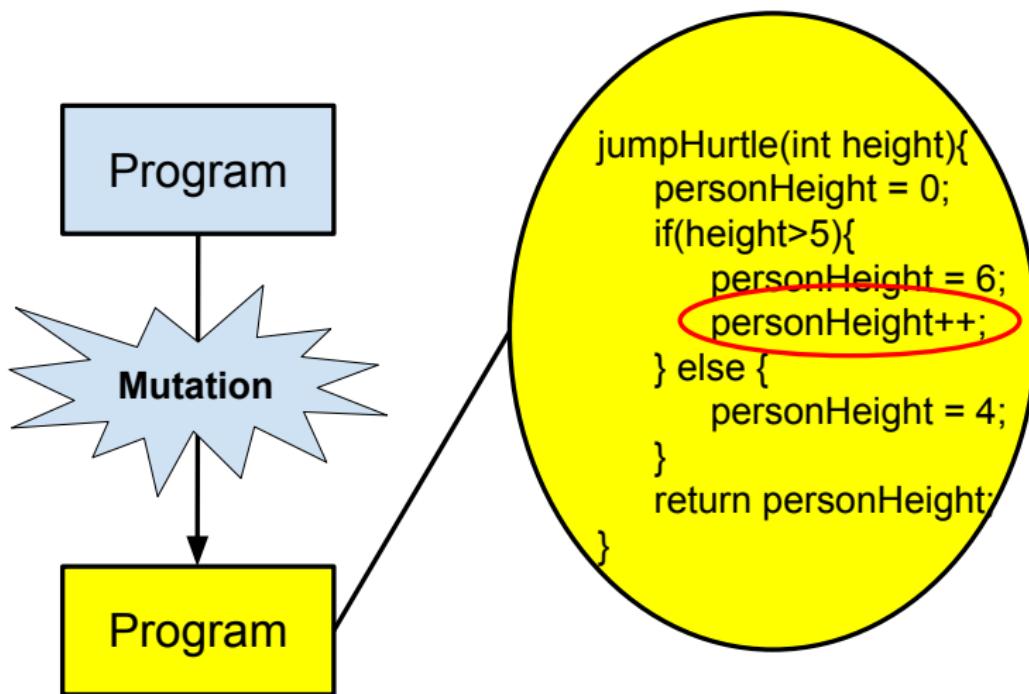
Genetic Programming



Genetic Programming



Genetic Programming



Outline

1 Background

2 Hyper-heuristics

- Heuristics
 - Hyper-heuristics

3 Genetic Programming Variants

4 Autoconstruction

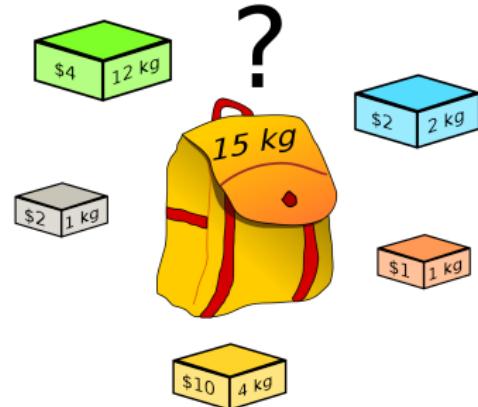
5 Summary

Heuristics

Heuristics – a function that ranks alternatives in a search algorithm at each branching step and uses that information to choose which branch to follow.

Heuristics

Heuristics – a function that ranks alternatives in a search algorithm at each branching step and uses that information to choose which branch to follow.

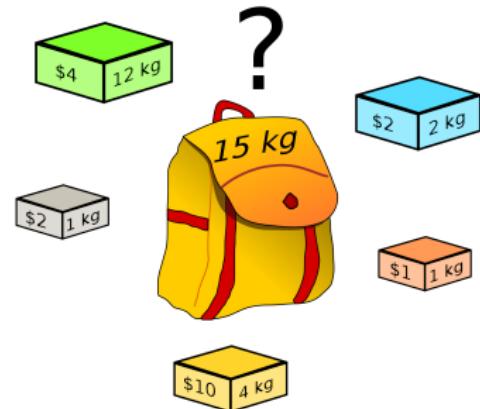


<https://upload.wikimedia.org/wikipedia/commons/thumb/f/fd/Knapsack.svg/1200px-Knapsack.svg.png>

Heuristics

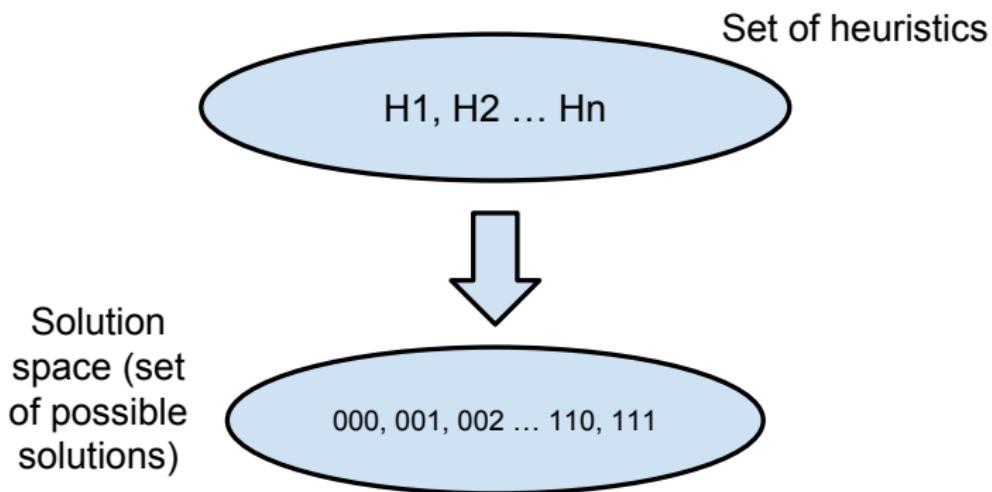
Heuristics – a function that ranks alternatives in a search algorithm at each branching step and uses that information to choose which branch to follow.

Example: “Select highest valued item and put into knapsack. If item puts knapsack overweight, select next highest value instead. Repeat until all items are gone or until the knapsack is full”



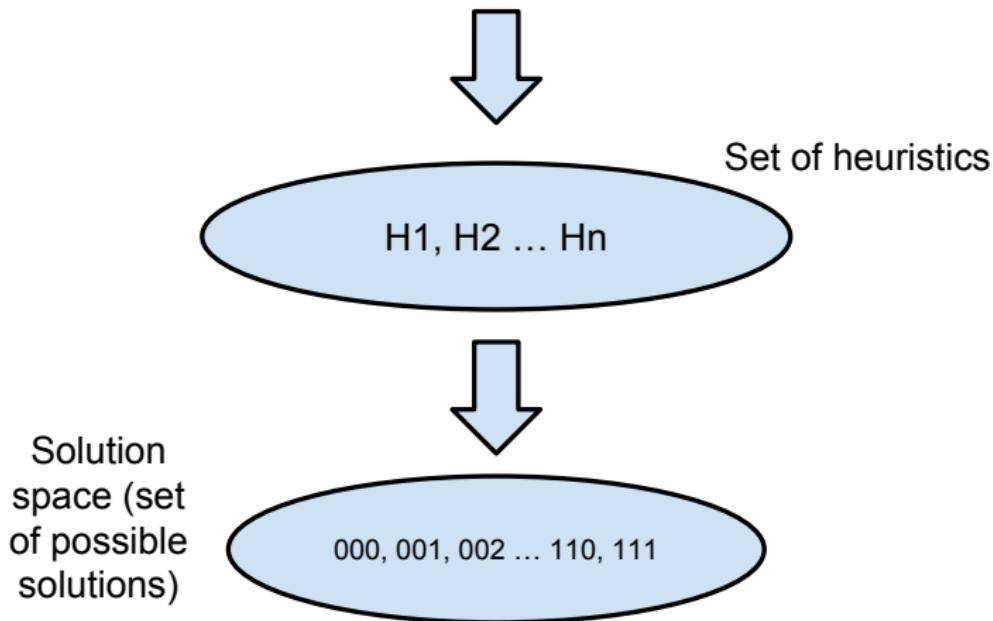
<https://upload.wikimedia.org/wikipedia/commons/thumb/f/fd/Knapsack.svg/1200px-Knapsack.svg.png>

Heuristics



Hyper-heuristics

Hyper-heuristics



Hyper-heuristics

Hyper-heuristics – heuristic search methods which seek to automate the process of selecting, generating, or adapting several simpler heuristics in order to solve computational search problems.

Hyper-heuristics

Hyper-heuristics – heuristic search methods which seek to automate the process of selecting, generating, or adapting several simpler heuristics in order to solve computational search problems.

Genetic programming hyper-heuristics – hyper-heuristics that use genetic programming for the process of selecting, generating, or adapting several simpler heuristics.

Outline

1 Background

2 Hyper-heuristics

3 Genetic Programming Variants

- What are they?
 - Why should we care?
 - Stack-based genetic programming

4 Autoconstruction

5 Summary

Genetic programming variants

GP variants – variations on the structure and setup of a genetic programming system.

Genetic programming variants

GP variants – variations on the structure and setup of a genetic programming system.

Harris et al. [1] performed an experiment to address whether or not the GP variant used affected the success of the hyper-heuristic

Genetic programming variants

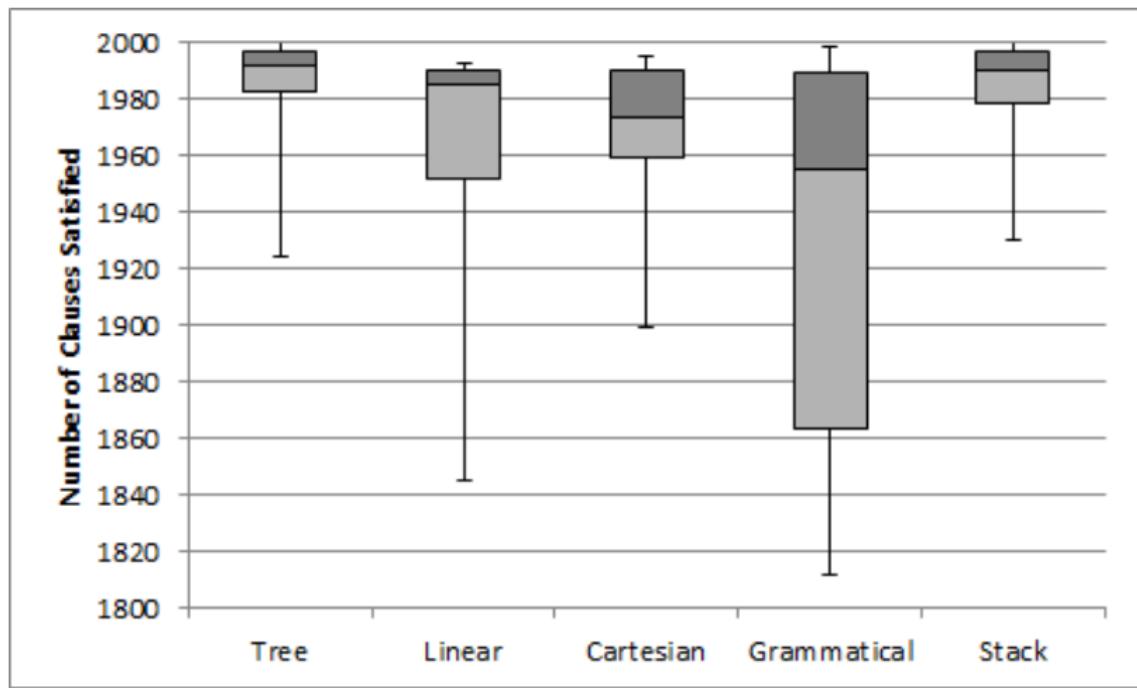
GP variants – variations on the structure and setup of a genetic programming system.

Harris et al. [1] performed an experiment to address whether or not the GP variant used affected the success of the hyper-heuristic

GP variants tested:

- Cartesian GP
- Linear GP
- Stack-based GP
- Tree-based GP
- Grammatical Evolution

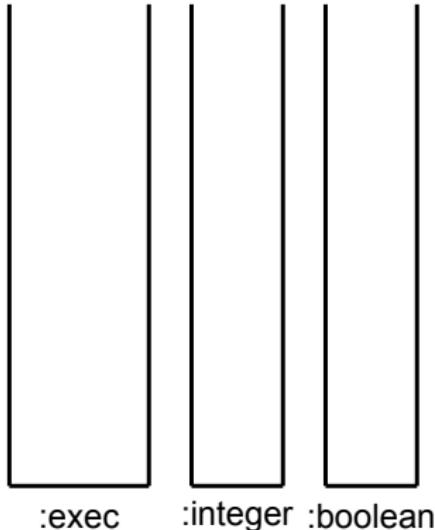
Why should we care?



Graph taken from Harris et al. [1]

Stack-based genetic programming

Data-stacks are used for managing input and output of operations.

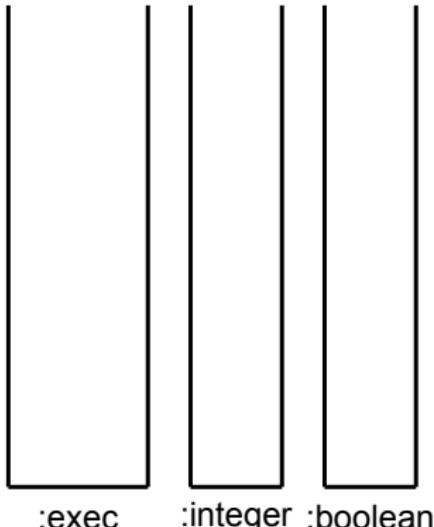


Stack-based genetic programming

Data-stacks are used for managing input and output of operations.

Programs are represented as linear sequences of literals and instructions. Below is an example of a simple Push program:

(1 integer_add 2 integer_equal)

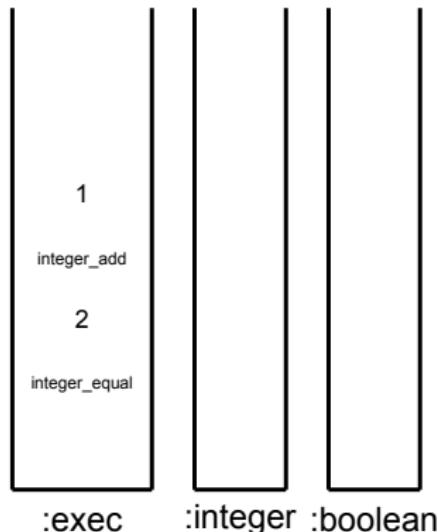


Stack-based genetic programming

Data-stacks are used for managing input and output of operations.

Programs are represented as linear sequences of literals and instructions. Below is an example of a simple Push program:

(1 integer_add 2 integer_equal)

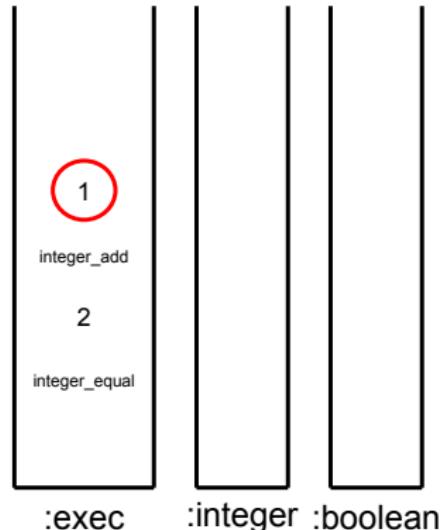


Stack-based genetic programming

Data-stacks are used for managing input and output of operations.

Programs are represented as linear sequences of literals and instructions. Below is an example of a simple Push program:

(1 integer_add 2 integer_equal)

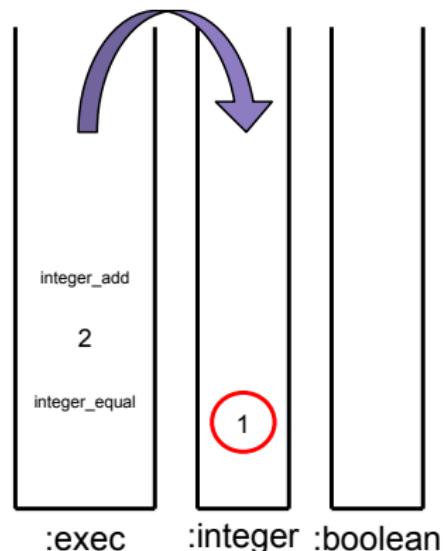


Stack-based genetic programming

Data-stacks are used for managing input and output of operations.

Programs are represented as linear sequences of literals and instructions.
Below is an example of a simple Push program:

(1 integer_add 2 integer_equal)

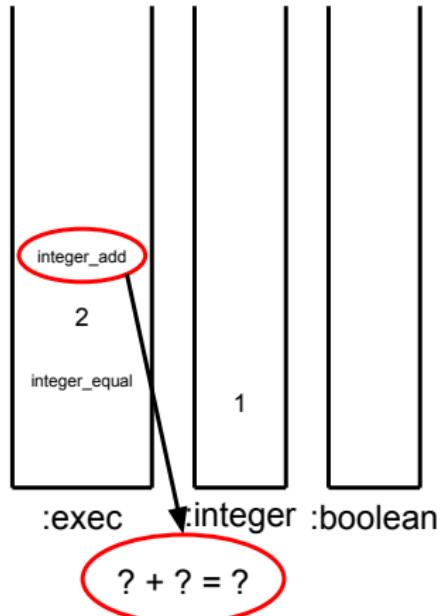


Stack-based genetic programming

Data-stacks are used for managing input and output of operations.

Programs are represented as linear sequences of literals and instructions.
Below is an example of a simple Push program:

(1 integer_add 2 integer_equal)

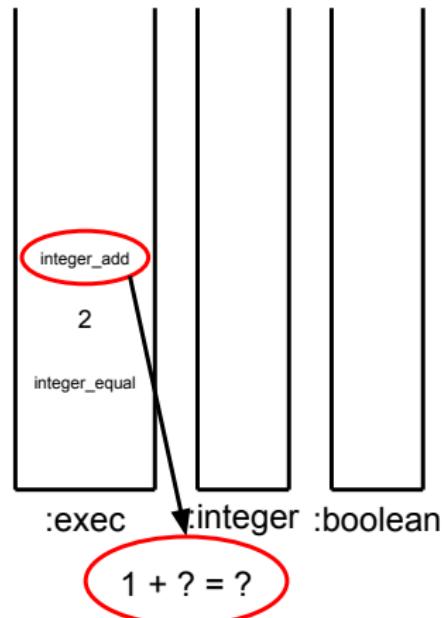


Stack-based genetic programming

Data-stacks are used for managing input and output of operations.

Programs are represented as linear sequences of literals and instructions.
Below is an example of a simple Push program:

(1 integer_add 2 integer_equal)

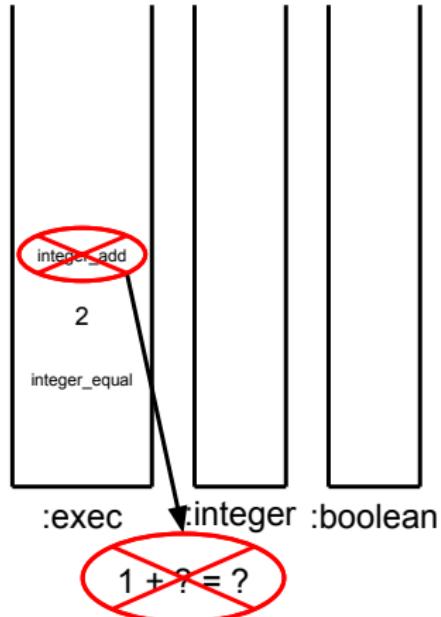


Stack-based genetic programming

Data-stacks are used for managing input and output of operations.

Programs are represented as linear sequences of literals and instructions.
Below is an example of a simple Push program:

```
(1 integer_add 2 integer_equal)
```

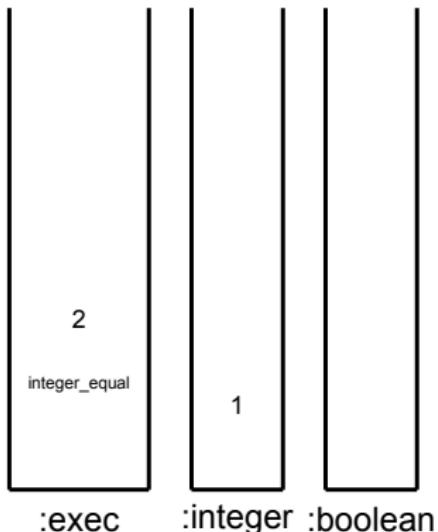


Stack-based genetic programming

Data-stacks are used for managing input and output of operations.

Programs are represented as linear sequences of literals and instructions. Below is an example of a simple Push program:

(1 integer_add 2 integer_equal)

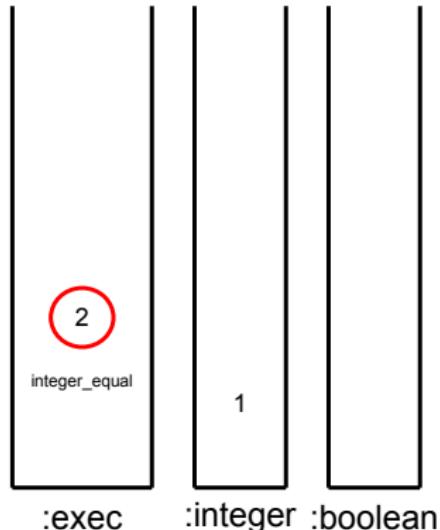


Stack-based genetic programming

Data-stacks are used for managing input and output of operations.

Programs are represented as linear sequences of literals and instructions. Below is an example of a simple Push program:

(1 integer_add 2 integer_equal)

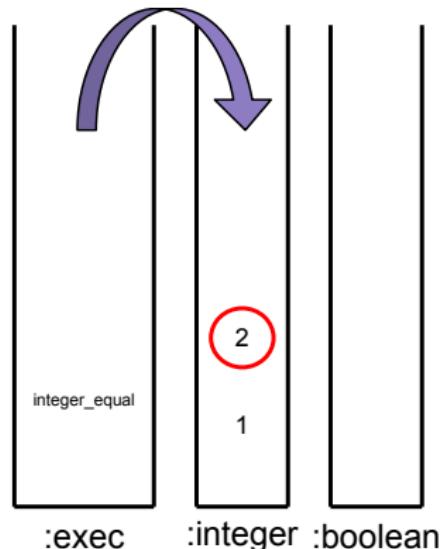


Stack-based genetic programming

Data-stacks are used for managing input and output of operations.

Programs are represented as linear sequences of literals and instructions.
Below is an example of a simple Push program:

```
(1 integer_add 2 integer_equal)
```

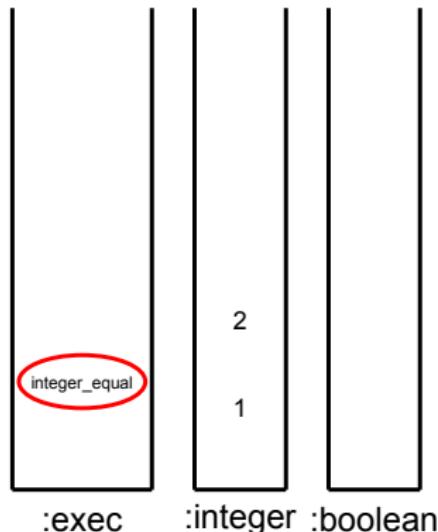


Stack-based genetic programming

Data-stacks are used for managing input and output of operations.

Programs are represented as linear sequences of literals and instructions. Below is an example of a simple Push program:

(1 integer_add 2 integer_equal)

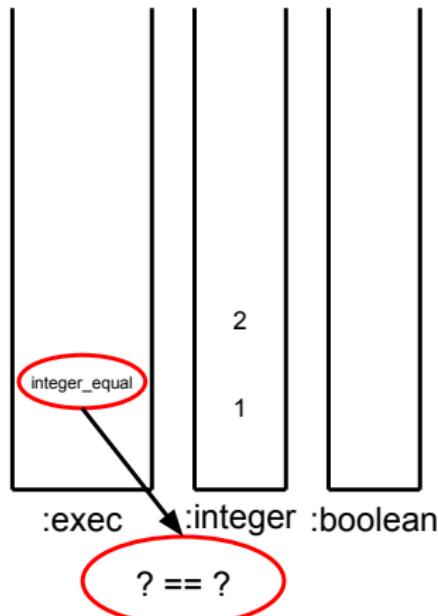


Stack-based genetic programming

Data-stacks are used for managing input and output of operations.

Programs are represented as linear sequences of literals and instructions. Below is an example of a simple Push program:

(1 integer_add 2 integer_equal)

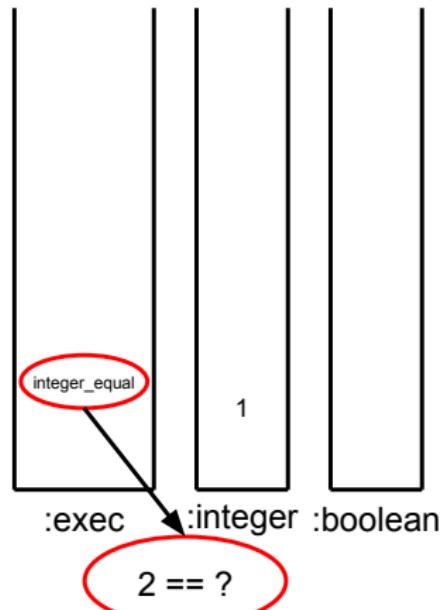


Stack-based genetic programming

Data-stacks are used for managing input and output of operations.

Programs are represented as linear sequences of literals and instructions. Below is an example of a simple Push program:

(1 integer_add 2 integer_equal)

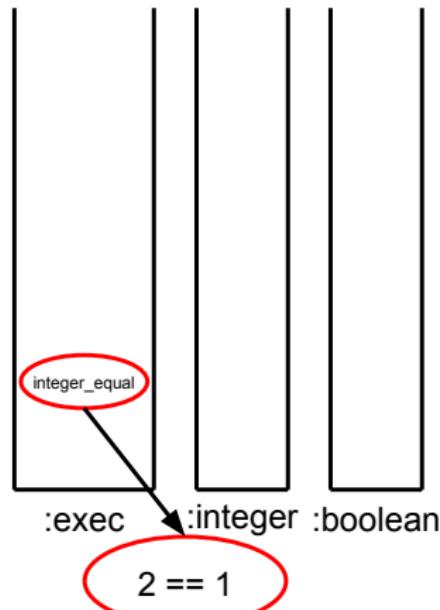


Stack-based genetic programming

Data-stacks are used for managing input and output of operations.

Programs are represented as linear sequences of literals and instructions. Below is an example of a simple Push program:

(1 integer_add 2 integer_equal)

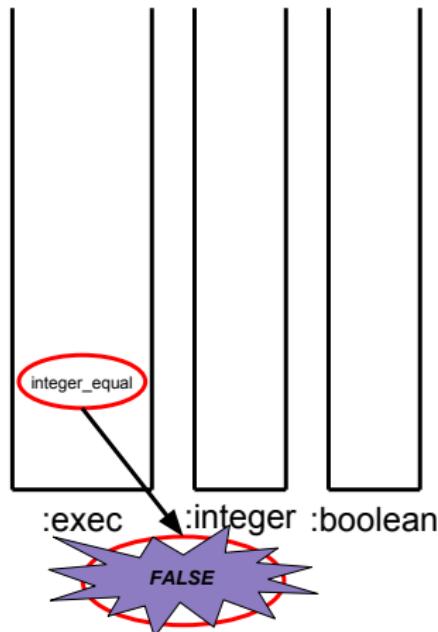


Stack-based genetic programming

Data-stacks are used for managing input and output of operations.

Programs are represented as linear sequences of literals and instructions. Below is an example of a simple Push program:

(1 integer_add 2 integer_equal)

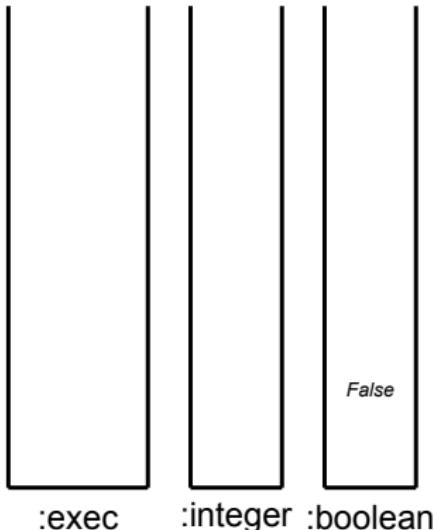


Stack-based genetic programming

Data-stacks are used for managing input and output of operations.

Programs are represented as linear sequences of literals and instructions. Below is an example of a simple Push program:

(1 integer_add 2 integer_equal)



Outline

1 Background

2 Hyper-heuristics

3 Genetic Programming Variants

4 Autoconstruction

- What is it?
 - AutoDoG
 - Results of AutoDoG

5 Summary

What is Autoconstruction?

- Autoconstruction is a type of genetic programming hyper-heuristic (GPHH)

What is Autoconstruction?

- Autoconstruction is a type of genetic programming hyper-heuristic (GPHH)
 - In most GPHH, the individual programs are evolving, but everything else is specified by the engineer; in autoconstruction, evolution is evolving as well.

What is Autoconstruction?

- Autoconstruction is a type of genetic programming hyper-heuristic (GPHH)
 - In most GPHH, the individual programs are evolving, but everything else is specified by the engineer; in autoconstruction, evolution is evolving as well.
 - Programs are responsible for evolving solutions *and* responsible for evolving their offspring.

AutoDoG

- A system designed by Spector et al. [2] that uses autoconstruction to evolve programs.

AutoDoG

- A system designed by Spector et al. [2] that uses autoconstruction to evolve programs.
 - Uses the Push programming language

AutoDog

- A system designed by Spector et al. [2] that uses autoconstruction to evolve programs.
- Uses the Push programming language
- Uses Plush, a linear genome format for Push

AutoDoG – reproduction

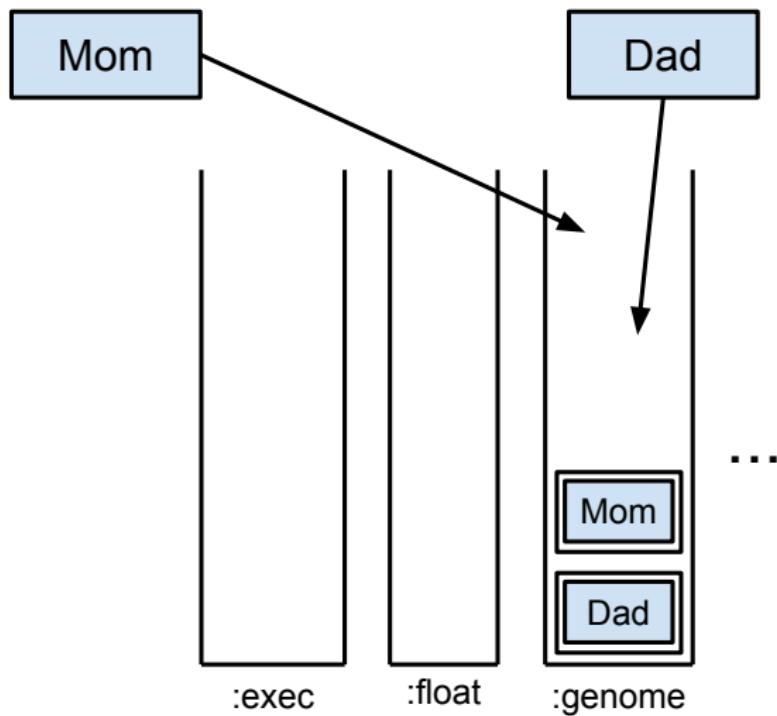
Mom

AutoDoG – reproduction

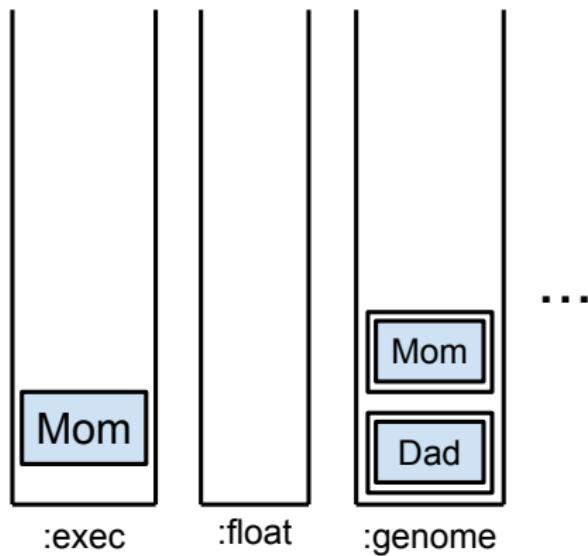
Mom

Dad

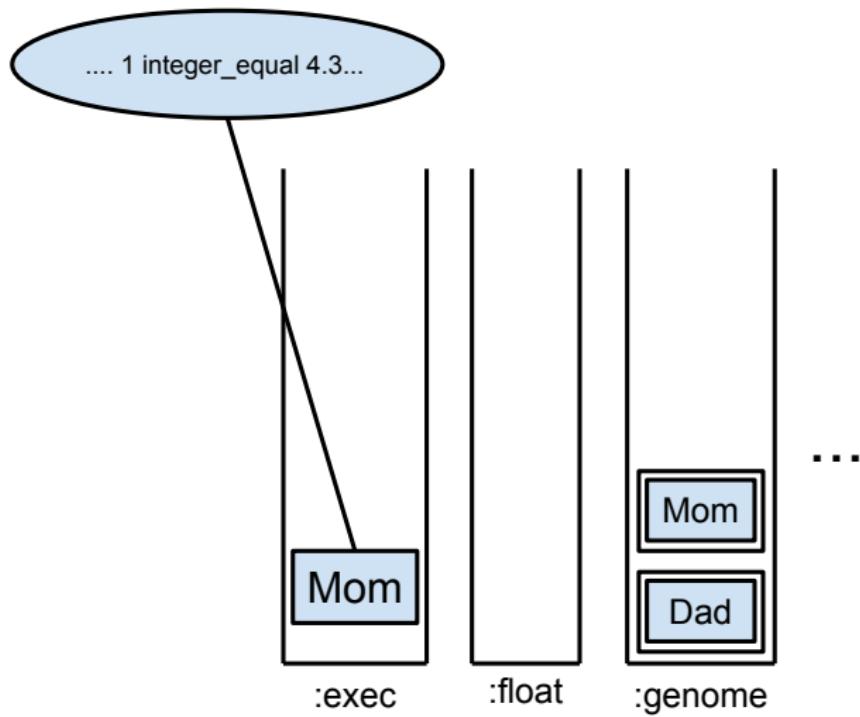
AutoDoG – reproduction



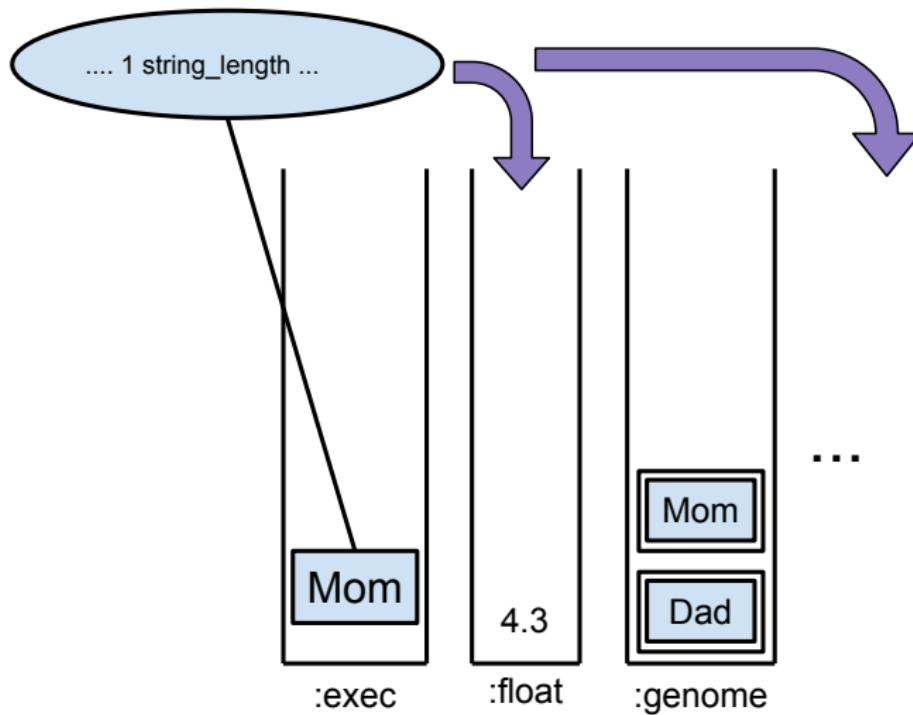
AutoDoG – reproduction



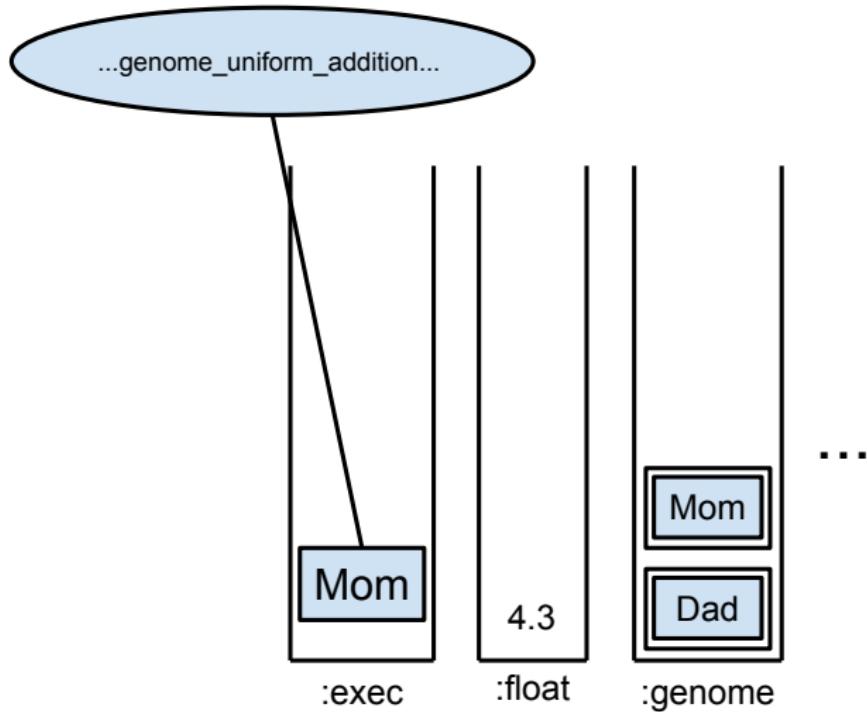
AutoDoG – reproduction



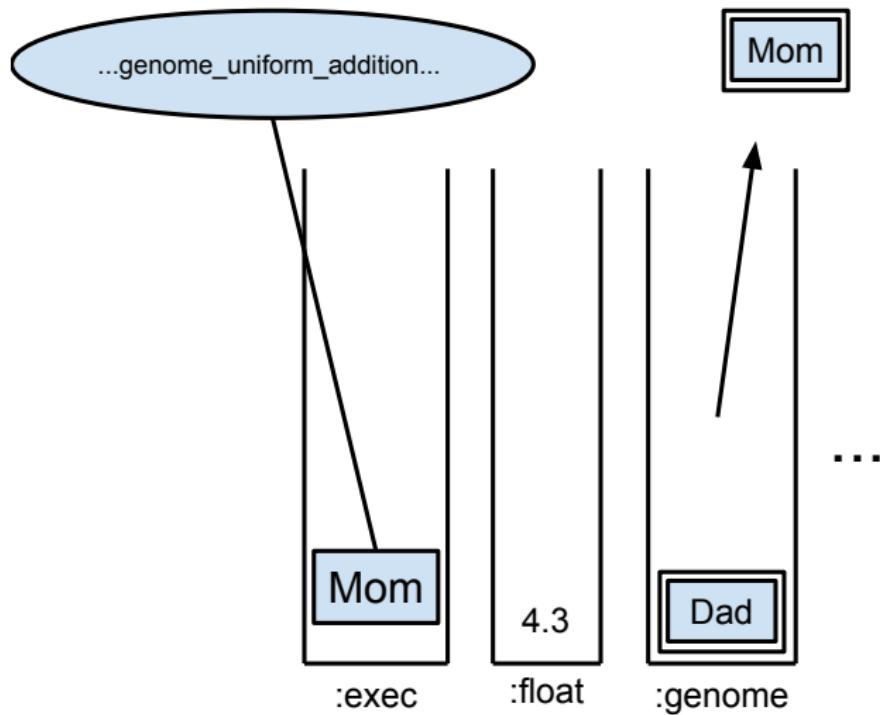
AutoDoG – reproduction



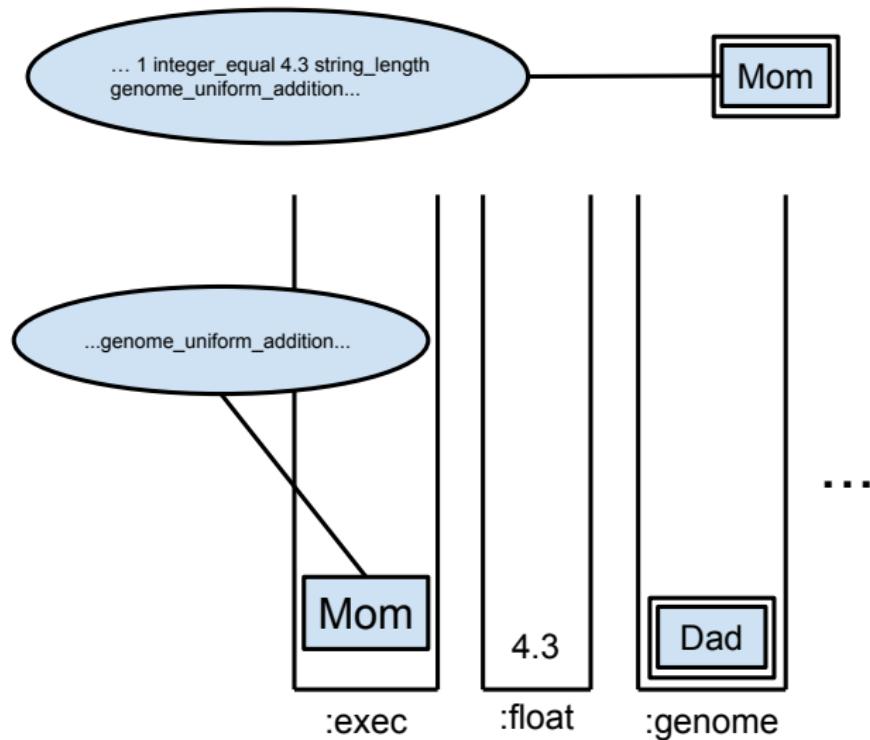
AutoDoG – reproduction



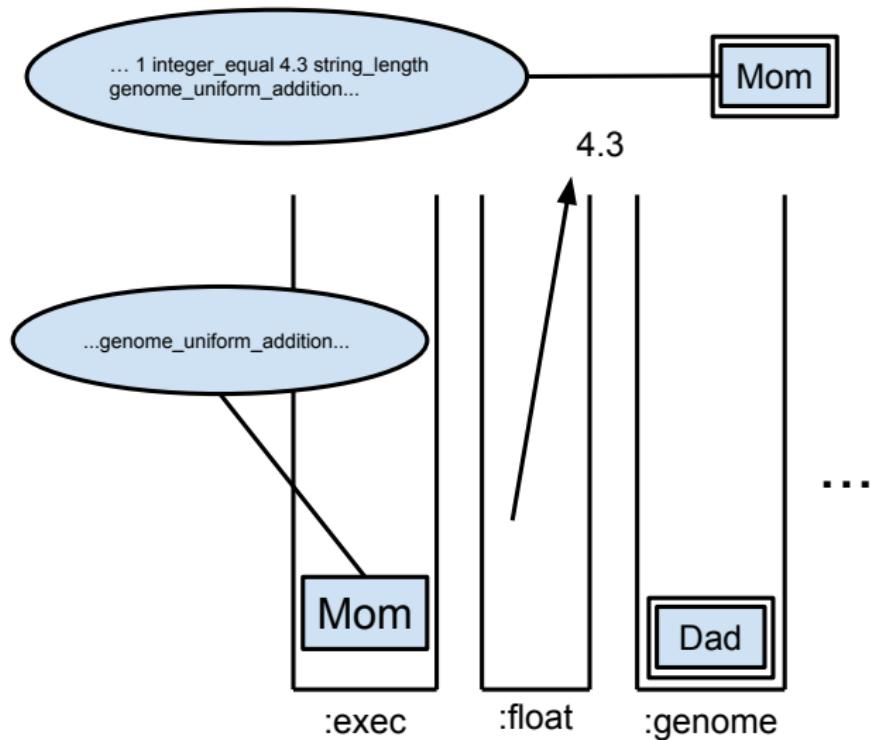
AutoDoG – reproduction



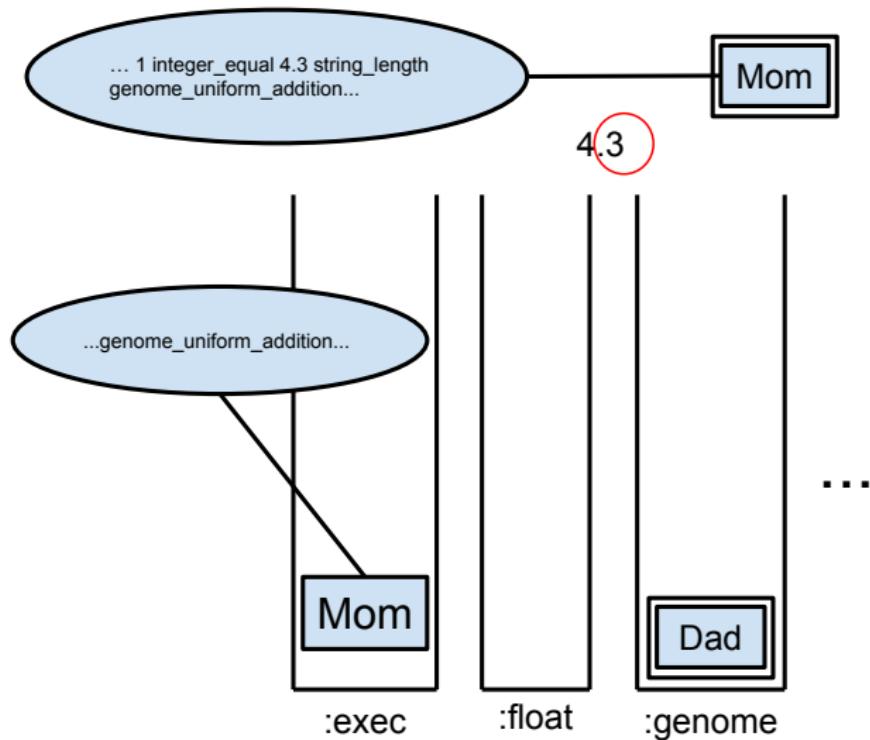
AutoDoG – reproduction



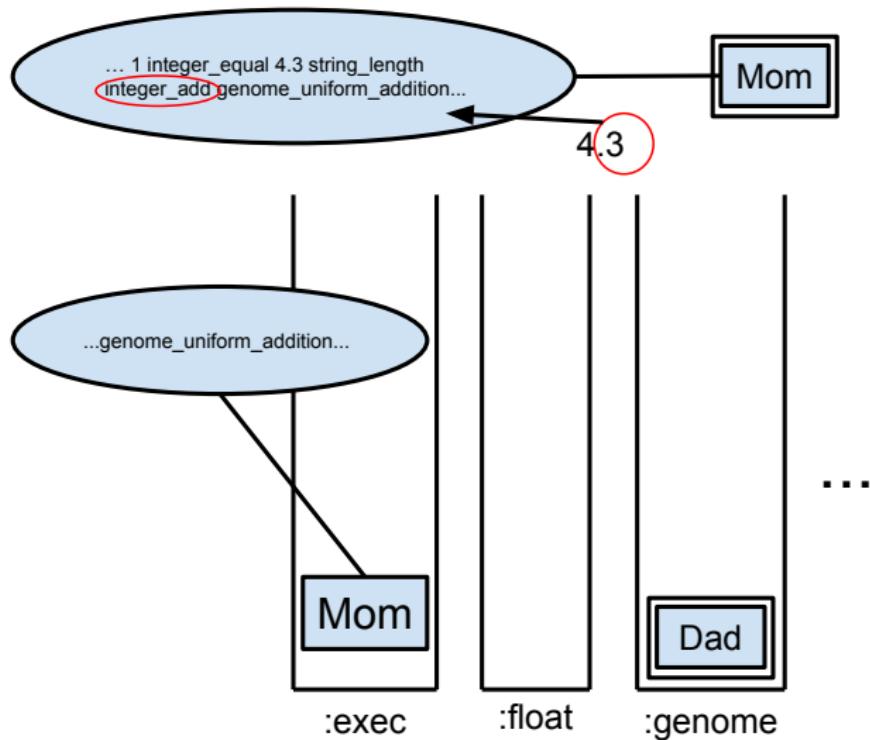
AutoDoG – reproduction



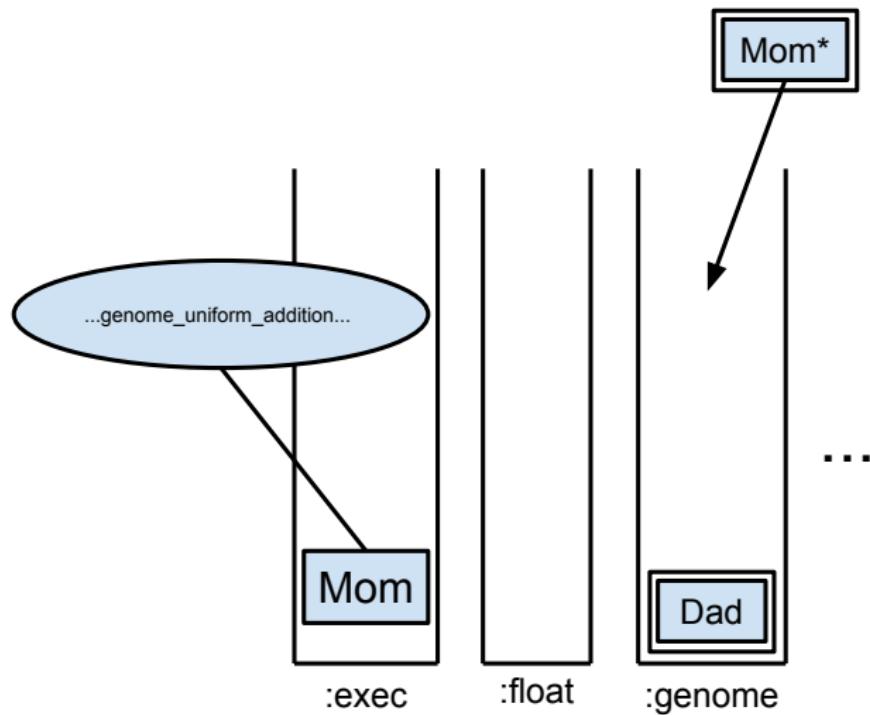
AutoDoG – reproduction



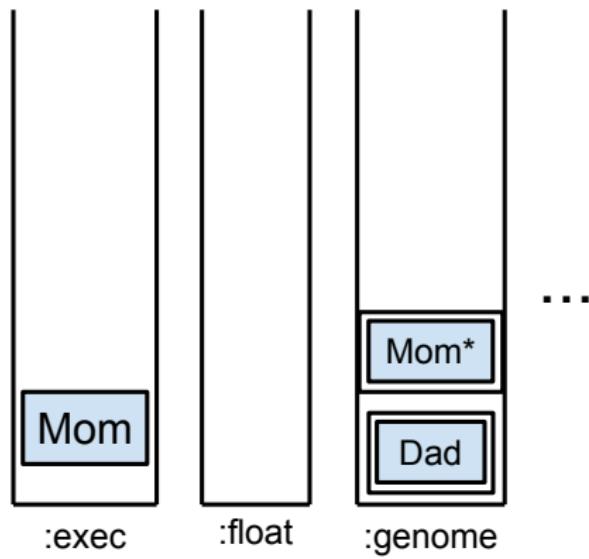
AutoDoG – reproduction



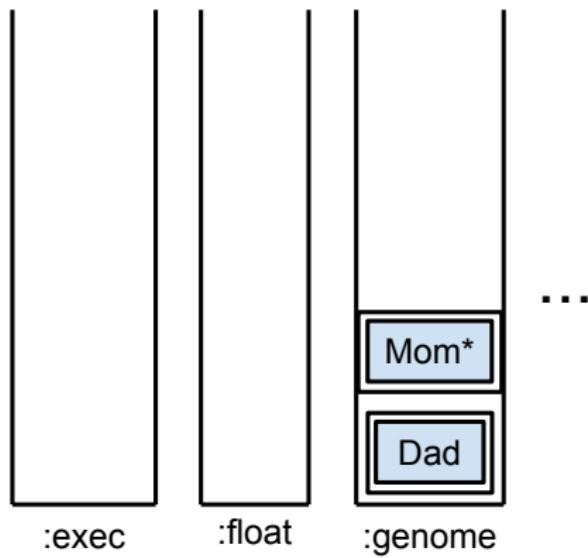
AutoDoG – reproduction



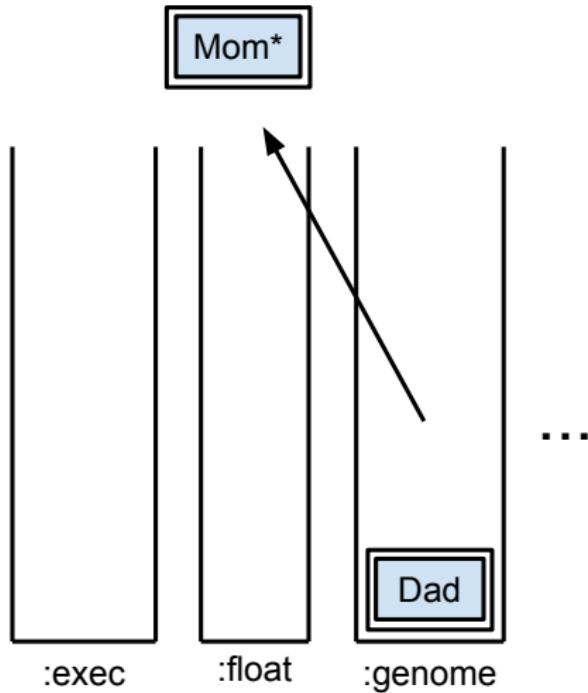
AutoDoG – reproduction



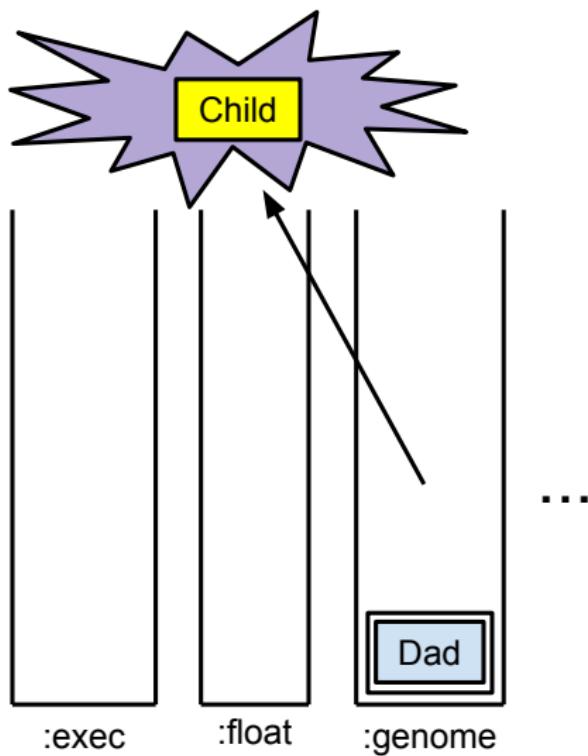
AutoDoG – reproduction



AutoDoG – reproduction



AutoDoG – reproduction



AutoDoG – reproduction

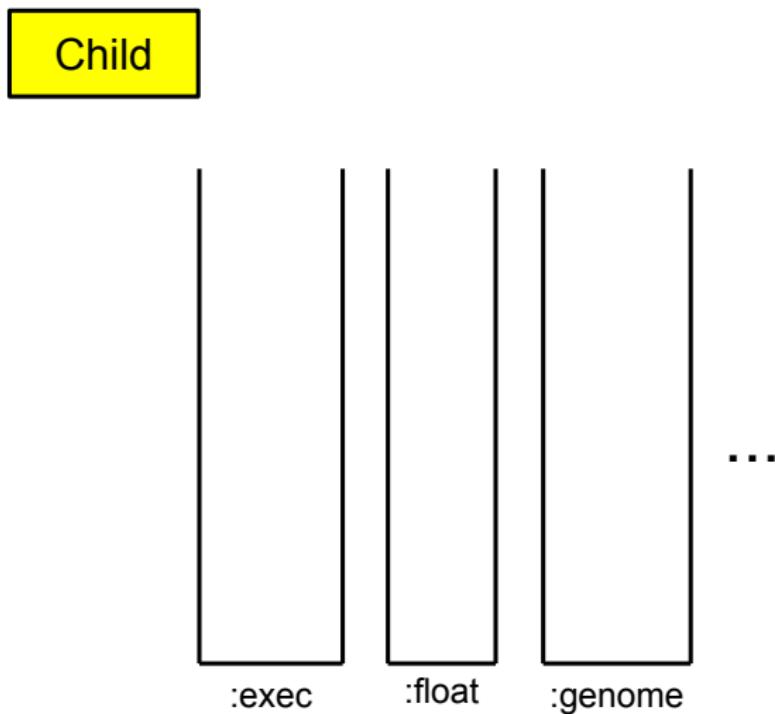
Now Child moves on to the next generation, right?

AutoDoG – reproduction

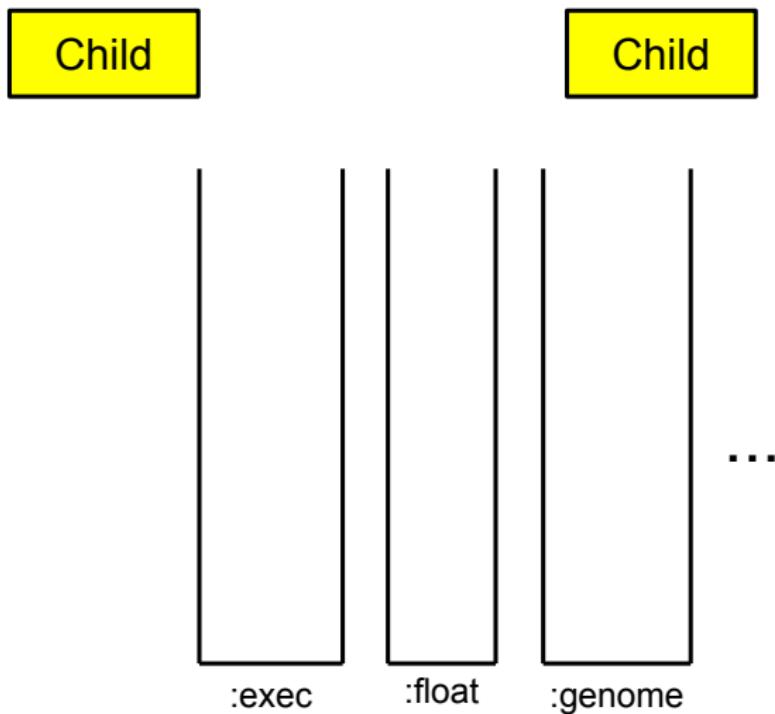
Now Child moves on to the next generation, right?

WRONG!

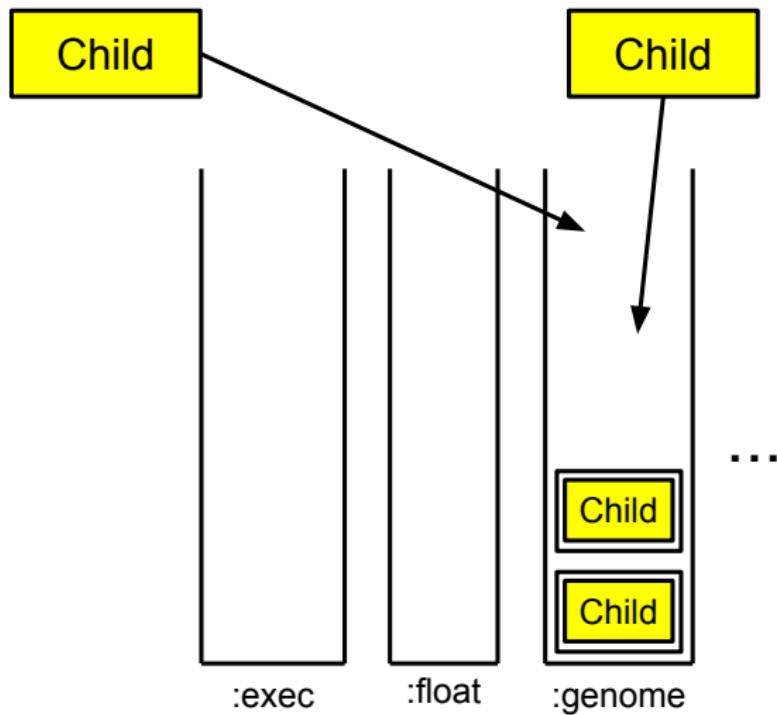
AutoDoG – diversification test



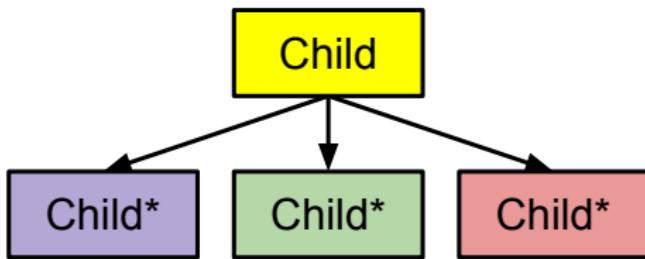
AutoDoG – diversification test



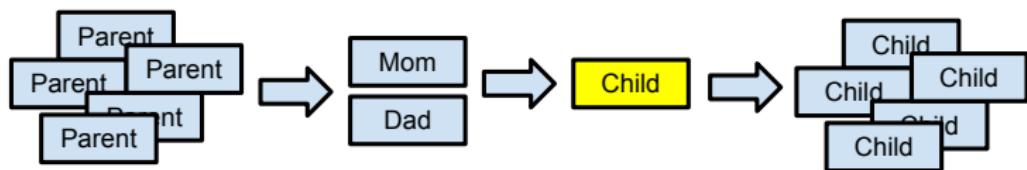
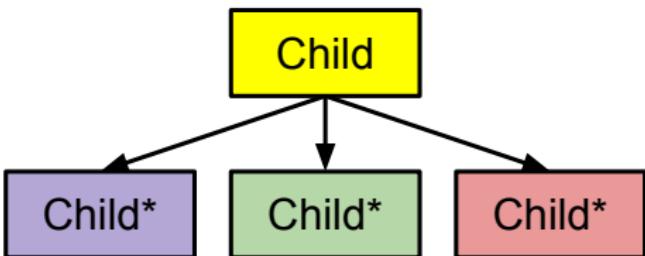
AutoDoG – diversification test



AutoDoG – diversification test



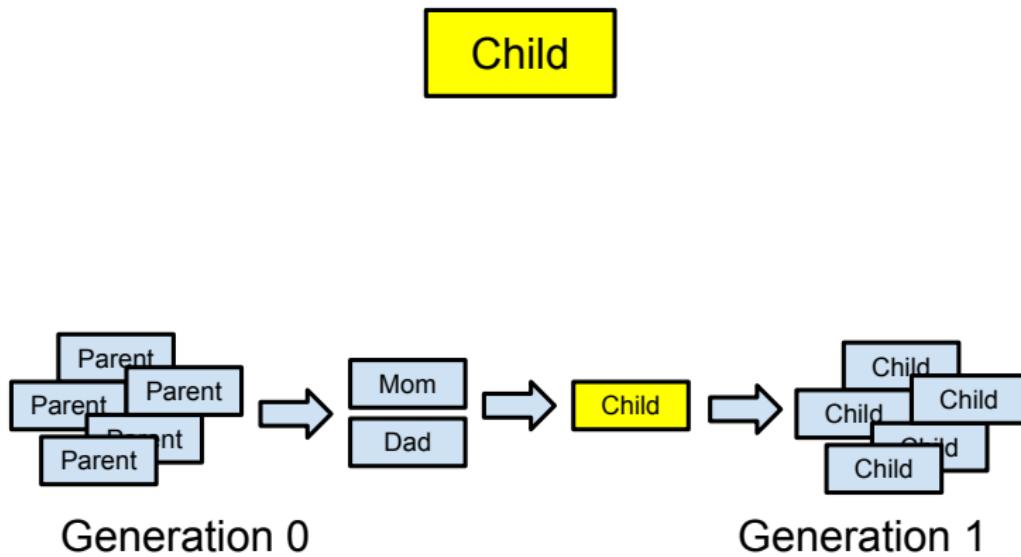
AutoDoG – diversification test



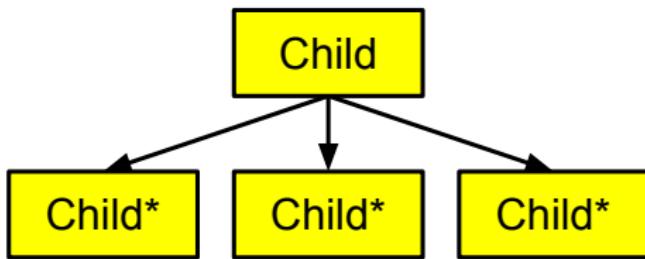
Generation 0

Generation 1

AutoDoG – diversification test



AutoDoG – diversification test

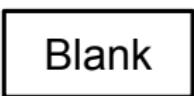


AutoDoG – diversification test

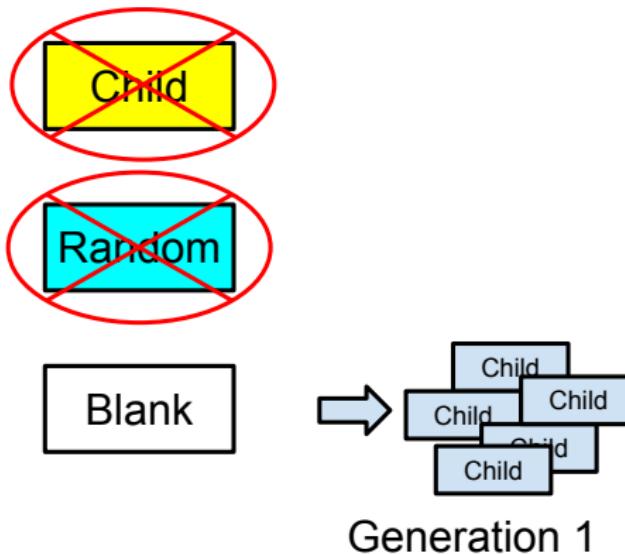
~~Child~~

Random

AutoDoG – diversification test



AutoDoG – diversification test



Results of AutoDoG

AutoDoG has solved Replace Space with New Line (RSWN).

Results of AutoDoG

AutoDoG has solved Replace Space with New Line (RSWN).

RSWN: given a string S, print S with all spaces replaced with new lines and return the integer count of all non-whitespace characters.

Results of AutoDoG

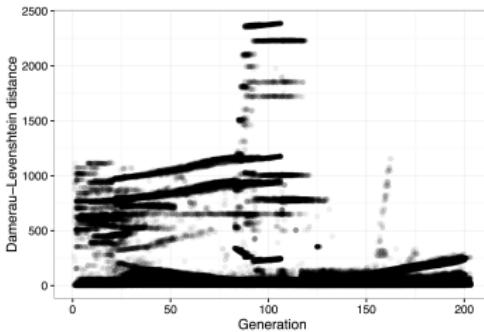
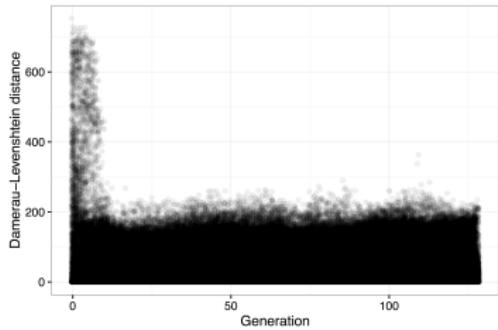
AutoDoG solves RSWN 5–10% of the time, where PushGP solves this problem 50% of the time.

Results of AutoDoG

AutoDoG solves RSWN 5–10% of the time, where PushGP solves this problem 50% of the time.

This is actually really impressive!

Results of AutoDoG



Graphs taken from Spector et al. [2]

Outline

- 1 Background
- 2 Hyper-heuristics
- 3 Genetic Programming Variants
- 4 Autoconstruction
- 5 Summary

Summary

- Genetic programming hyper-heuristics (GPHH) for heuristic/program evolution

Summary

- Genetic programming hyper-heuristics (GPHH) for heuristic/program evolution
- There are many types of GP variants, and the variant chosen may affect the success of the hyper-heuristic

Summary

- Genetic programming hyper-heuristics (GPHH) for heuristic/program evolution
 - There are many types of GP variants, and the variant chosen may affect the success of the hyper-heuristic
 - Autoconstruction is a type of GPHH

Summary

- Genetic programming hyper-heuristics (GPHH) for heuristic/program evolution
- There are many types of GP variants, and the variant chosen may affect the success of the hyper-heuristic
- Autoconstruction is a type of GPHH
- AutoDoG is a newer autoconstructive system that uses stack-based GP and has had recent success in the field of automating algorithm design

Summary

- Genetic programming hyper-heuristics (GPHH) for heuristic/program evolution
 - There are many types of GP variants, and the variant chosen may affect the success of the hyper-heuristic
 - Autoconstruction is a type of GPHH
 - AutoDoG is a newer autoconstructive system that uses stack-based GP and has had recent success in the field of automating algorithm design
 - AutoDoG/autoconstruction is special because evolution is evolving

Summary

This may drastically change the way we program in the future!

References

-  S. Harris, T. Bueter, and D. R. Tauritz.
A Comparison of Genetic Programming Variants for Hyper-Heuristics.
In Sara Silva, editors, *GECCO '15*, pages 1043–1050,
Madrid, Spain 2015.
 -  L. Spector, N. F. McPhee, T. Helmuth, M. M. Casale, and J. Oks.
Evolution Evolves with Autoconstruction.
In T. Friedrich, *et al*, editors, *GECCO '16*, pages
1349–1356, Denver, Colorado, USA 2016.

See my paper for additional references.

Acknowledgments

Special thanks to Nic McPhee and Elena Machkasova for their feedback and constructive comments.

Thanks for coming!