Bingo: A Customizable Framework for Symbolic Regression with Genetic Programming

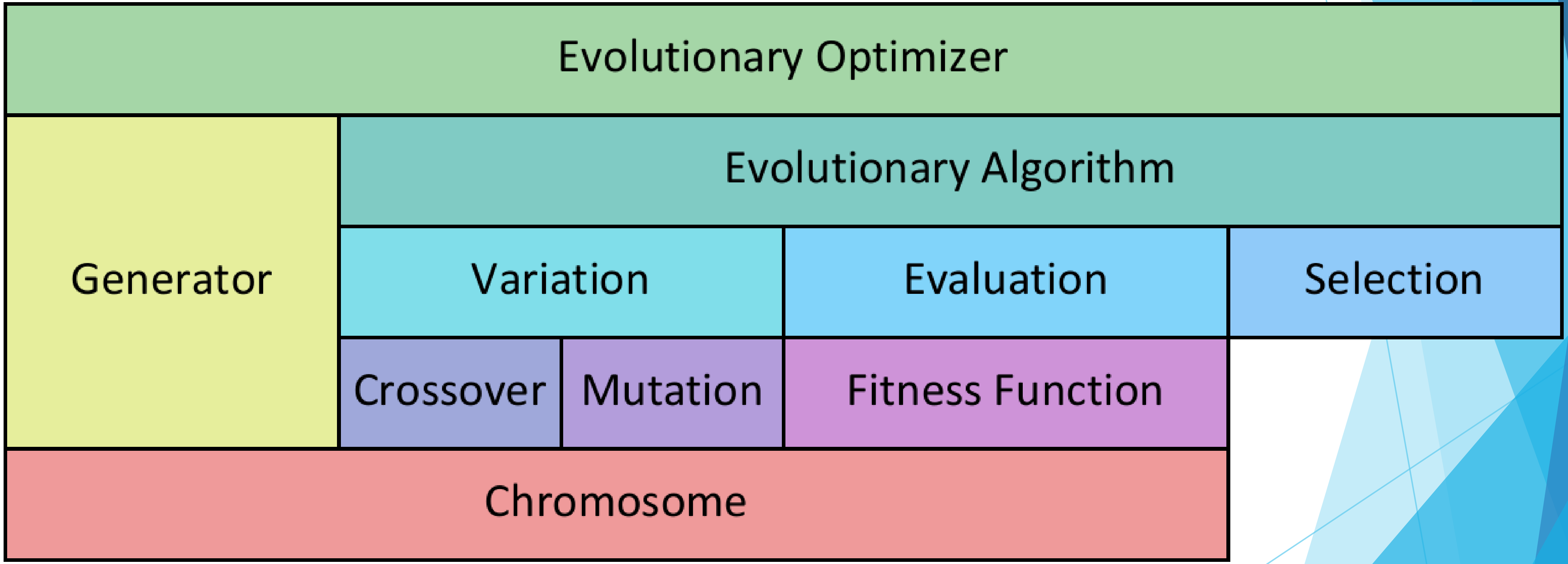
David Randall[1], Tyler Townsend[2], Jacob Hochhalter[1], Geoffrey Bomarito[3]

[1]University of Utah, [2]Microsoft (Views and opinions are personal and do not represent Microsoft), [3]NASA Langley Research Center

What is Bingo?

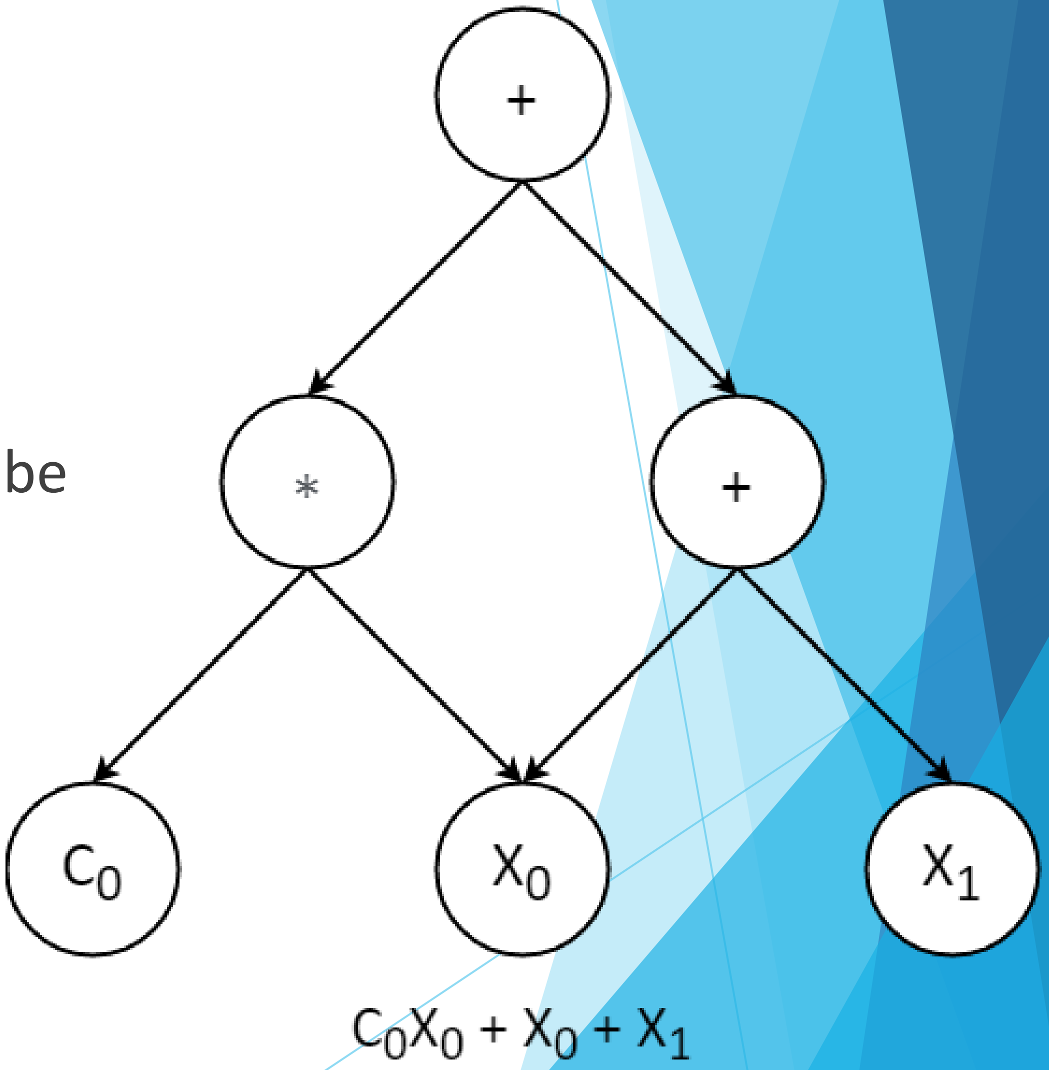
- Bingo is an open-source framework for genetic programming for symbolic regression (GPSR) developed by NASA
- Made to be
  - Modular
  - Extendible
  - Efficient
- Why use Bingo over other GPSR packages?
  - High- and low-level interfaces for ease-of-use and customizability
  - Modularity makes it easy to compare and develop components of GPSR
  - Produces simple and accurate models

Bingo’s Structure

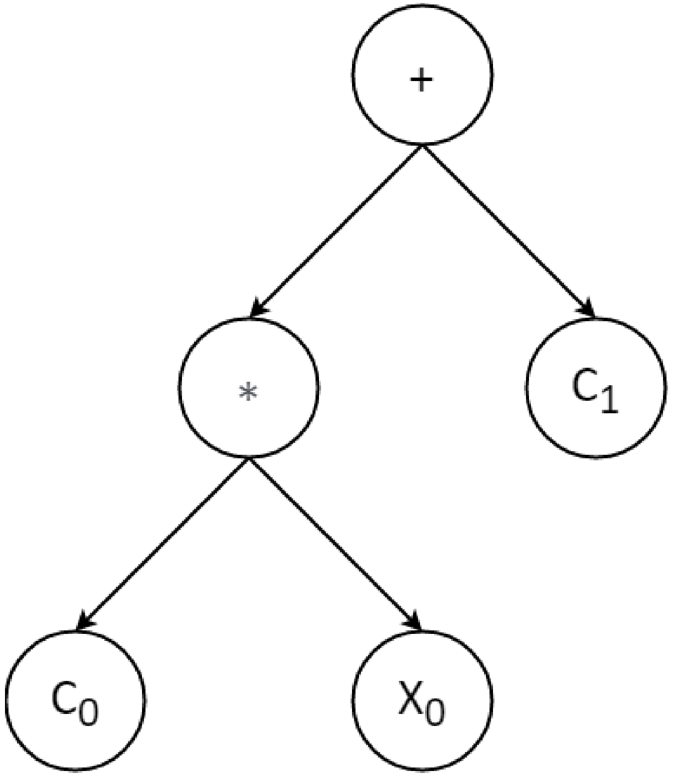


Equations/Chromosomes

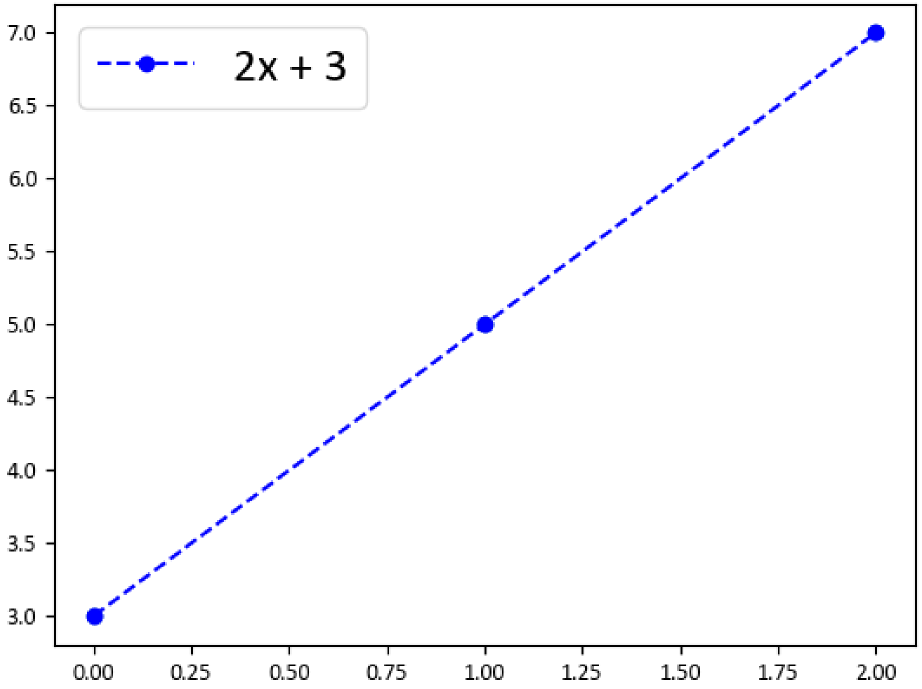
- Encoded as directed acyclic graphs (AGraphs)
  - Computational benefits over tree encodings [1]
- Have free-form constants/parameters that can be locally optimized
- Complexity
  - Measured as number of utilized nodes in the graph encoding



Local Optimization of Parameters



$C_0X_0 + C_1$



$C_0 = 2$

$C_1 = 3$

Command Arrays

- Genome of equations
- Terminal nodes and operators
- Not all commands are utilized
  - Size of command array ≠ complexity

i	node	parameter 1	parameter 2	expression
0	**constant**	**0**	**0**	C_0
1	variable	0	0	X_0
2	variable	1	1	X_1
3	*	0	1	C_0X_0
4	+	1	2	$X_0 + X_1$
5	sin	2	2	$\sin(X_1)$
6	+	3	4	$C_0X_0 + X_0 + X_1$

Command Arrays

- Genome of equations
- Terminal nodes and operators
- Not all commands are utilized
  - Size of command array ≠ complexity

i	node	parameter 1	parameter 2	expression
0	constant	0	0	C_0
1	**variable**	**0**	**0**	$\mathbf{X_0}$
2	variable	1	1	X_1
3	*	0	1	C_0X_0
4	+	1	2	$X_0 + X_1$
5	sin	2	2	$\sin(X_1)$
6	+	3	4	$C_0X_0 + X_0 + X_1$

Command Arrays

- Genome of equations
- Terminal nodes and operators
- Not all commands are utilized
  - Size of command array ≠ complexity

i	node	parameter 1	parameter 2	expression
0	constant	0	0	C_0
1	variable	0	0	X_0
2	**variable**	**1**	**1**	$\mathbf{X_1}$
3	*	0	1	C_0X_0
4	+	1	2	$X_0 + X_1$
5	sin	2	2	$\sin(X_1)$
6	+	3	4	$C_0X_0 + X_0 + X_1$

Command Arrays

- Genome of equations
- Terminal nodes and operators
- Not all commands are utilized
  - Size of command array ≠ complexity

i	node	parameter 1	parameter 2	expression
0	constant	0	0	C_0
1	variable	0	0	X_0
2	variable	1	1	X_1
3	*****	**0**	**1**	$\mathbf{C_0X_0}$
4	+	1	2	$X_0 + X_1$
5	sin	2	2	$\sin(X_1)$
6	+	3	4	$C_0X_0 + X_0 + X_1$

Command Arrays

- Genome of equations
- Terminal nodes and operators
- Not all commands are utilized
  - Size of command array ≠ complexity

i	node	parameter 1	parameter 2	expression
0	constant	0	0	C_0
1	variable	0	0	X_0
2	variable	1	1	X_1
3	*	0	1	C_0X_0
4	**+**	**1**	**2**	$\mathbf{X_0 + X_1}$
5	sin	2	2	$\sin(X_1)$
6	+	3	4	$C_0X_0 + X_0 + X_1$

Command Arrays

- Genome of equations
- Terminal nodes and operators
- Not all commands are utilized
  - Size of command array ≠ complexity

i	node	parameter 1	parameter 2	expression
0	constant	0	0	C_0
1	variable	0	0	X_0
2	variable	1	1	X_1
3	*	0	1	C_0X_0
4	+	1	2	$X_0 + X_1$
5	**sin**	**2**	**2**	**$\sin(X_1)$**
6	+	3	4	$C_0X_0 + X_0 + X_1$

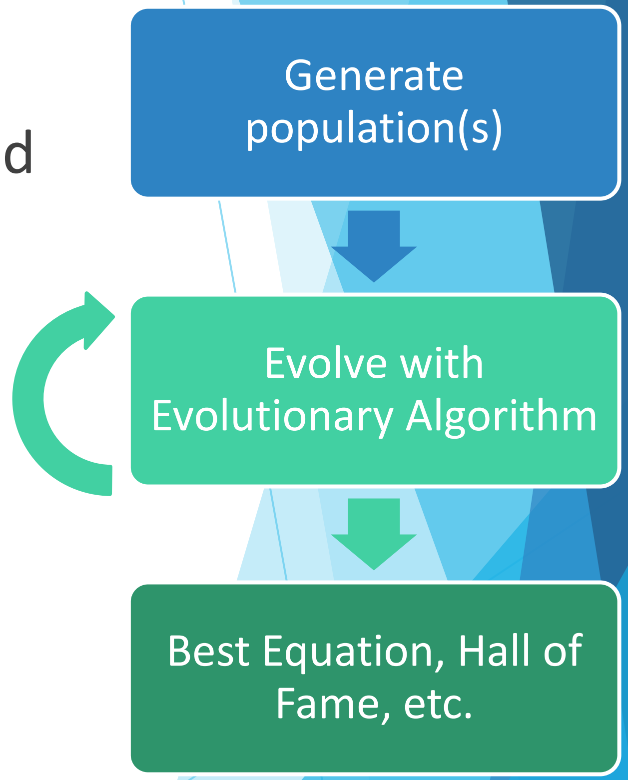
Command Arrays

- Genome of equations
- Terminal nodes and operators
- Not all commands are utilized
  - Size of command array ≠ complexity

i	node	parameter 1	parameter 2	expression
0	constant	0	0	C_0
1	variable	0	0	X_0
2	variable	1	1	X_1
3	*	0	1	C_0X_0
4	+	1	2	$X_0 + X_1$
5	sin	2	2	$\sin(X_1)$
6	**+**	**3**	**4**	$\mathbf{C_0X_0 + X_0 + X_1}$

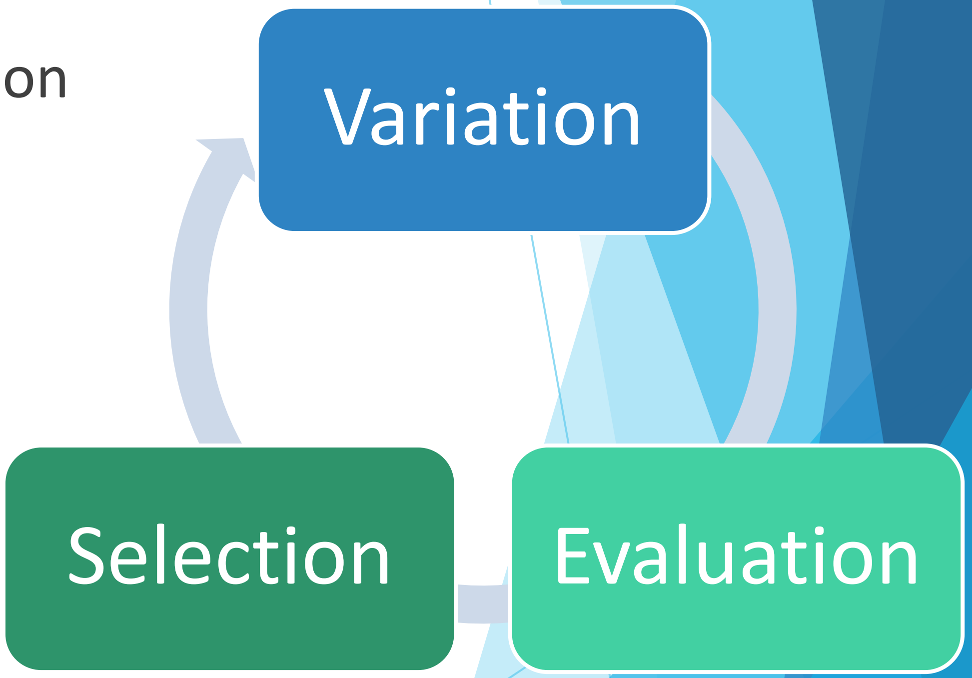
Evolutionary Optimizer

- Responsible for generating an initial population (generator) and evolving that population (evolutionary algorithm)
- Potential generator customizations
  - Initial population with all unique equations
  - Seeding of initial population with parts of previously found equations
  - Filtering of random equations
- Potential genetic programming workflow customizations
  - Archipelago
  - Coevolution



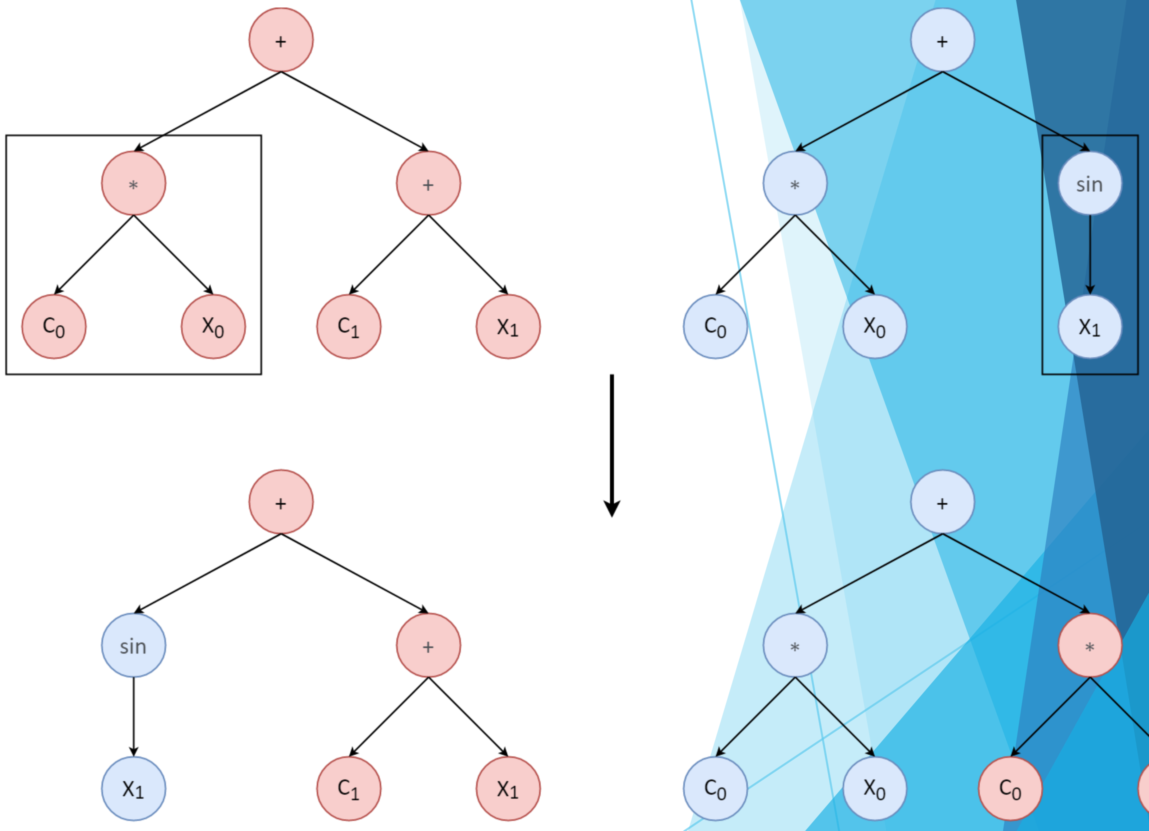
Evolutionary Algorithm

- Responsible for performing evolution on a population
- Follows a traditional GPSR workflow by default
  - Variation
  - Evaluation
  - Selection
- Can introduce other evolutionary operations
  - Simplification



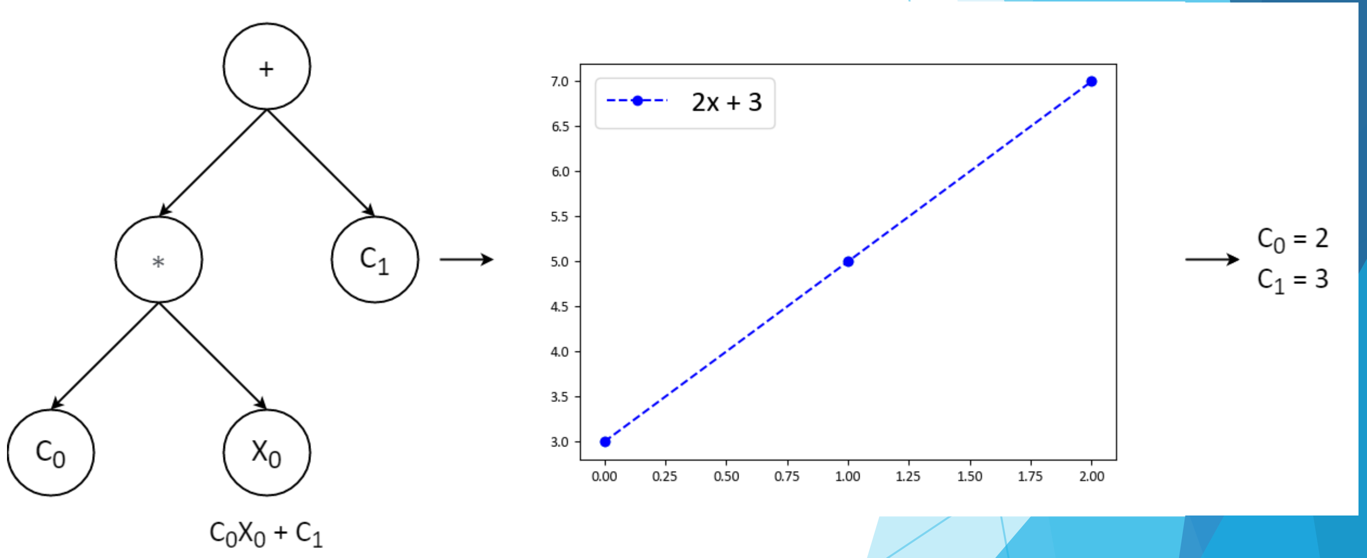
Variation

- Takes a population and varies it to form new individuals
- Two main methods
  - Crossover and mutation
  - Crossover or mutation
- Potential customizations
  - Only select variations that are more fit than their original counterparts



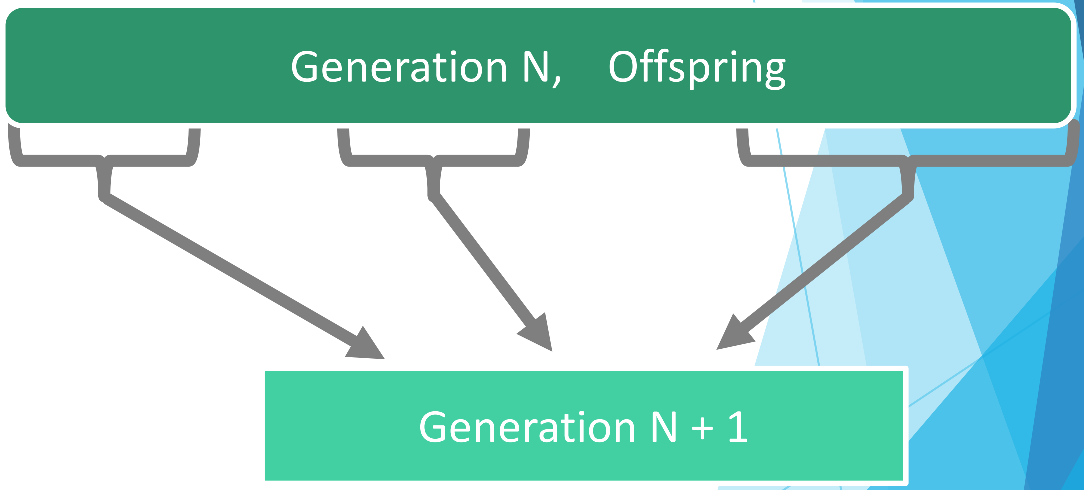
Evaluation

- Evaluates the fitness of individuals in a population
- Fitness functions
  - Explicit regression
  - Implicit regression
  - Continuous local optimization
- Potential customizations
  - Domain-specific constraints
  - Optimization with consideration of uncertainty



Selection

- Selects among original population and its variants to form next generation's population
  - Deterministic crowding
  - Age-fitness Pareto selection [2]
  - Tournament
- Potential customizations
  - Probabilistic crowding



Notable Customizations

- Fitness functions for mechanical engineering
  - Incorporation of automatic differentiation
- Tensorial GPSR
- Sequential Monte-Carlo and uncertainty quantification

$$\frac{\partial}{\partial X_0} sin(X_0^2) = cos(X_0^2) \cdot 2X_0$$

What if I don’t care about customization?

- “Out-of-the-box” scikit-learn[1] wrapper
- Easy configuration and simple interface for training
- Can use with other scikit-learn utilities
  - e.g., Cross validation classes, useful for hyperparameter tuning

```
reg = SymbolicRegressor(evolutionary_algorithm=AgeFitnessEA, ...)
reg.fit(x_train, y_train)
y_pred = reg.predict(x_test)
```

[1] This is not an endorsement by the National Aeronautics and Space Administration (NASA)

Efficiency

- Parallelism
  - Distributed memory: parallel evolution of islands
  - Shared memory: parallel evaluation of individuals
- Fitness predictors [3]
  - Subsets of data used to predict the true fitness of individuals
  - Less computational effort required for evaluation
  - Similar ideas to support vectors and batch evaluation

Efficiency (cont.)

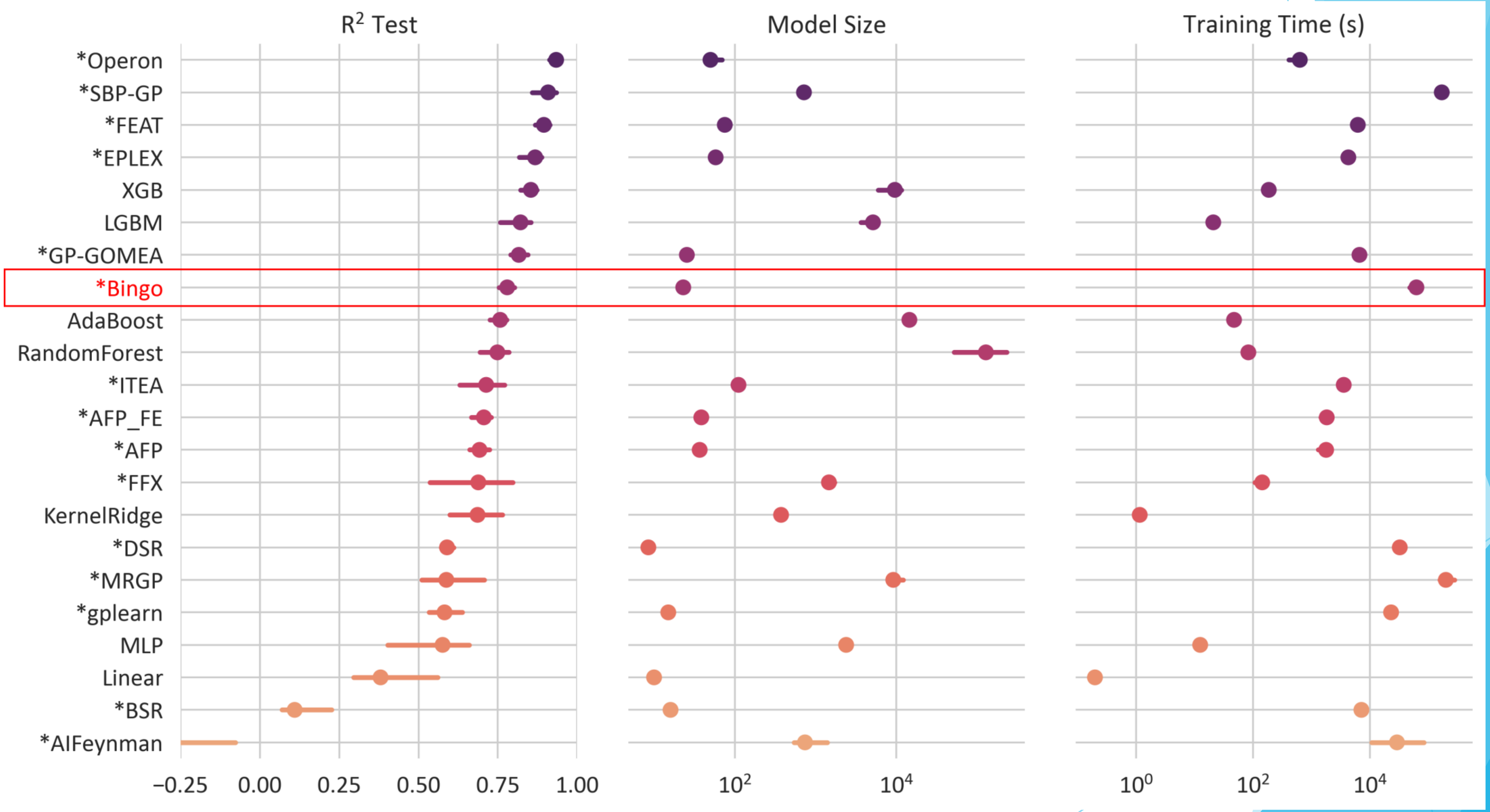
- Equation simplification
  - More complex equation -> more effort to evaluate
  - We use simplified versions of equations for evaluation
    - Doesn’t modify the original genotype
- C++ backend
  - An optional backend that can be enabled so that some components are implemented in C++

SRBench

- SRBench[2] and Penn Machine Learning Benchmarks (PMLB)[3]
  - SRBench [4]
    - Open-source benchmark for SR and ML methods
    - Used for this workshop's competition
  - PMLB [5]
    - Mix of realistic and synthetic datasets
- Black-box problems
  - General regression
- Ground-truth problems
  - Recovering an underlying equation

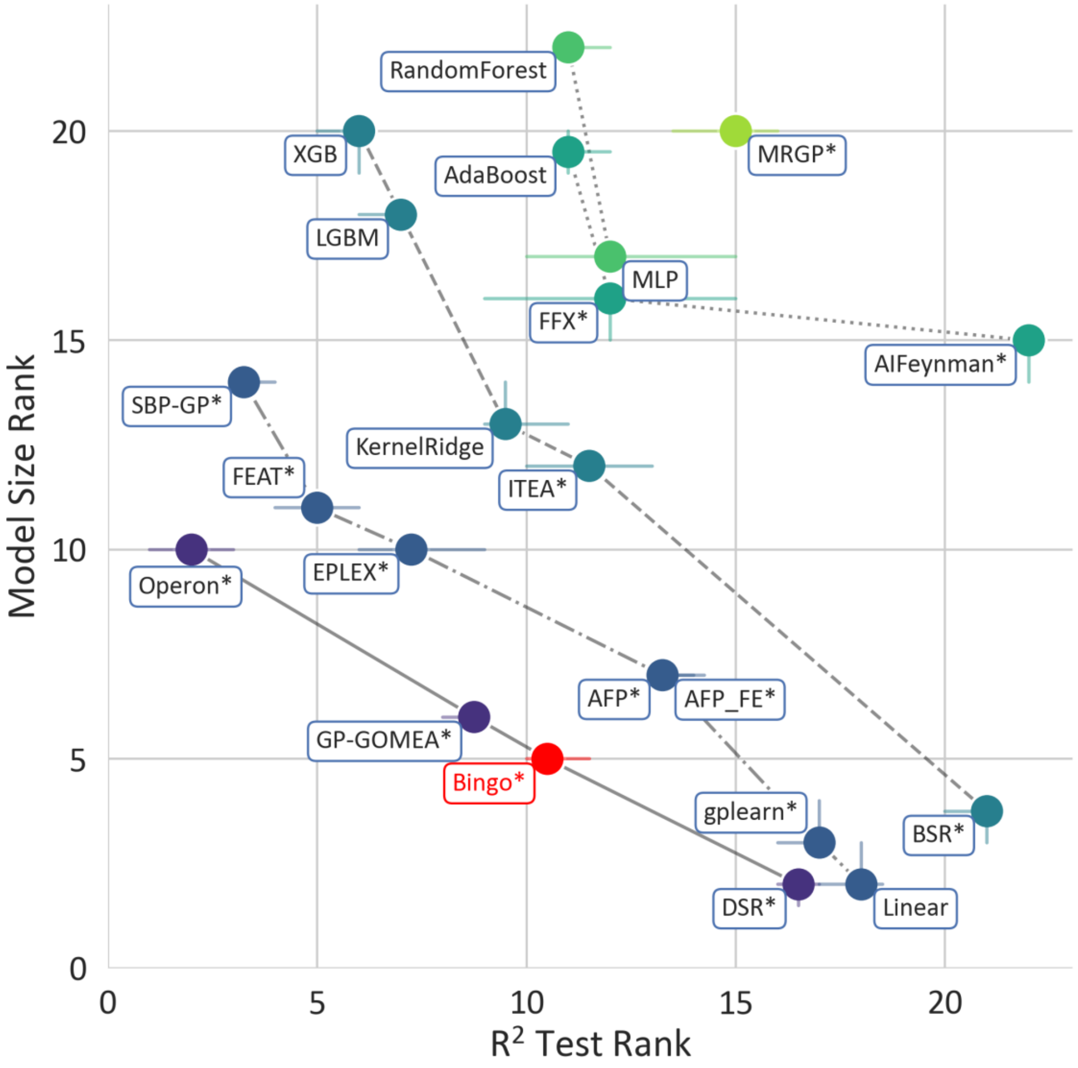
[2,3] This is not an endorsement by the National Aeronautics and Space Administration (NASA)

Results

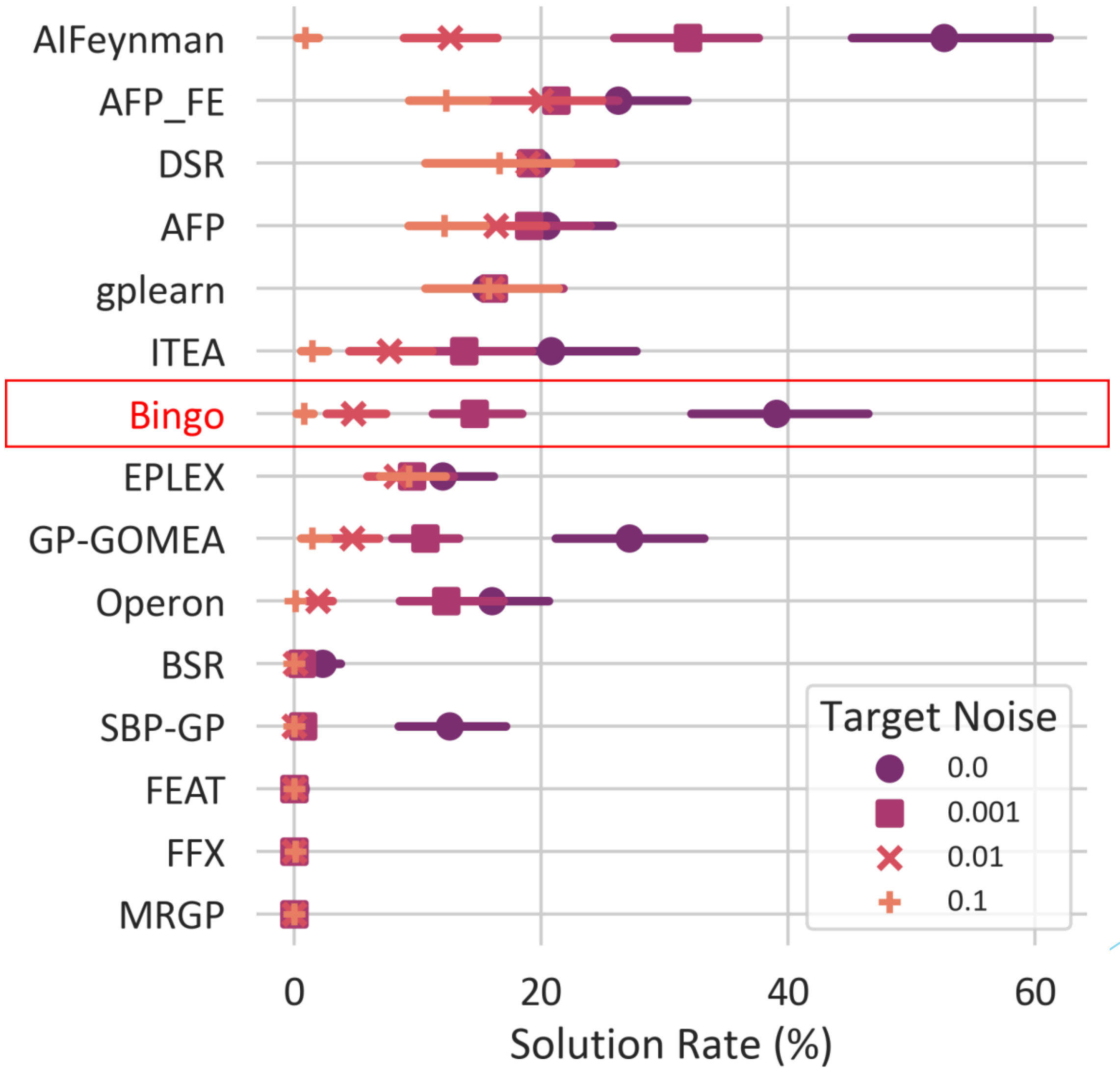


R2 Test
Model Size
Training Time (s)
*Operon
*SBP-GP
*FEAT
*EPLEX
XGB
LGBM
*GP-GOMEA
*Bingo
AdaBoost
RandomForest
*ITEA
*AFP_FE
*AFP
*FFX
KernelRidge
*DSR
*MRGP
*gplearn
MLP
Linear
*BSR
*AIFeynman
−0.25
0.00
0.25
0.50
0.75
1.00
10^2
10^4
10^0
10^2
10^4

Results (cont.)



Results (cont.)



Summary

- What is Bingo?
  - Flexible, modular framework for GPSR
  - Made to easily compare and create GPSR components
    - Sandbox
  - Can still be used for typical GPSR without modifications
    - SRBench
    - sklearn wrapper

Future Work

- Faster training times
  - More parts implemented in C++
- Built-in integration with other tools
  - SymPy[4]
  - PyTorch[5]/TensorFlow[6]
- Better performance on general regression problems
  - Revisiting and analyzing performance on SRBench

[4,5,6] This is not an endorsement by the National Aeronautics and Space Administration (NASA)

References

- [1] Michael Schmidt and Hod Lipson. 2007. Comparison of tree and graph encodings as function of problem complexity.
- [2] Michael D. Schmidt and Hod Lipson. 2010. Age-fitness pareto optimization.
- [3] Michael D. Schmidt and Hod Lipson. 2008. Coevolution of Fitness Predictors.
- [4] William La Cava, et al. 2021. Contemporary Symbolic Regression Methods and their Relative Performance.
- [5] Joseph D. Romano, et al. 2020. PMLB v1.0: an open source dataset collection for benchmarking machine learning methods.