Genetic Programming

A Tutorial Introduction

Una-May O'Reilly, Erik Hemberg
The ALFA Group: AnyScale Learning for All
CSAIL, MIT
unamay@csail.mit.edu, hembergerik@csail.mit.edu
<http://groups.csail.mit.edu/ALFA>

This work is licensed under a Creative Commons Attribution International 4.0 License.

GECCO '21 Companion, July 10–14, 2021, Lille, France © 2021 Copyright is held by the owner/author(s). ACM ISBN 978-1-4503-8351-6/21/07. <https://doi.org/10.1145/3449726.3461394>

1

Instructor: Una-May O'Reilly

- Leader: AnyScale Learning For All Group, MIT CSAIL
- Experience solving real world, complex problems requiring **AI/machine learning** where **evolutionary computation** is a core capability
- Applications include
 - Cybersecurity
 - Waveform data mining – medical applications
 - Behavioral data mining – MOOC
 - Circuits, network coding
 - Sparse matrix data mapping on parallel architectures
 - Finance
 - Flavor design
 - Wind energy
 - » Turbine layout
 - » Resource assessment
- Focus on innovation in genetic programming
 - coevolution
 - Improving its competence
 - Program synthesis

2

Instructor: Erik Hemberg

- Research Scientist: AnyScale Learning For All Group, MIT CSAIL
- Experience solving complex problems requiring **AI and machine learning** with **evolutionary computation** as a core capability, Bronze HUMIE 2018
- Applications include
 - Cybersecurity
 - Behavioral data mining – MOOC
 - Pylon design
 - Network controllers
 - Tax avoidance
- Focus on innovation and implementation in genetic programming
 - Grammatical representation
 - Coevolution
 - Estimation of Distribution

3

About You

- EA experience?
 - ES? GA? EDA? PSO? ACO? EP?
- CS experience?
- Programming? algorithms?
- Teacher?
- Native English speakers?

4

Tutorial Goals

- Introduction to GP algorithm, given some knowledge of genetic algorithms or evolutionary strategies
 - provide Black box demonstration of GP symbolic regression
- Become familiar with GP design properties and recognize them
 - ponygp in python
- You could teach it in an undergrad lecture
- Use it “out of the box”
- Set groundwork for advanced topics
 - Theory, other tutorials
 - Specialized workshops (Genetic improvement etc)
 - GP Track talks at GECCO, Proceedings of EuroGP, Genetic Programming and Evolvable Machines

5

Agenda

1. Context: Evolutionary Computation and Evolutionary Algorithms
2. GP is the genetic evolution of executable expressions
 - Black box example of GP symbolic regression
3. Nuts and Bolts Description of Algorithm Components
4. pony_gp.py demonstration from project PonyGP
5. Resources and reference material

Agenda

6

Neo-Darwinian Evolution

- Survival and thriving in the environment
- Offspring quantity - based on survival of the fittest
- Offspring variation: genetic crossover and mutation
- Population-based adaptation over generations
- Genotype-phenotype duality
- Complex and non-deterministic

Evolutionary Computation and Evolutionary Algorithms

7

EA Generation Loop

Each generation

- select
- breed
- replace

```
population = random_pop_init()
generation = 0
while needToStop == false
    generation++
    solution = bestOf(population)
    phenotypes = decoder(genotypes)
    calculateFitness(phenotypes)
    parents = select (phenotypes)
    offspring = breed(parents.genotypes)
    population = replace(parents, offspring)
    recheck(needToStop)
```

Evolutionary Computation and Evolutionary Algorithms

8

Problem Domains where EAs are Used

- Where there is need for complex solutions
 - evolution is a process that gives rise to complexity
 - a continually evolving, adapting process, potentially with changing environment from which emerges modularity, hierarchy, complex behavior and complex system relationships
- Combinatorial optimization
 - NP-complete and/or poorly scaling solutions via LP or convex optimization
 - unyielding to approximations (SQP, GEO-P)
 - eg. TSP, graph coloring, bin-packing, flows
 - for: logistics, planning, scheduling, networks, bio gene knockouts
 - Typified by discrete variables
 - Solved by Genetic Algorithm (GA)

Evolutionary Computation and Evolutionary Algorithms

8

Problem Domains where EAs are Used

- Continuous Optimization
 - non-differentiable, discontinuous, multi-modal, large scale objective functions 'black box'
 - applications: engineering, mechanical, material, physics
 - Typified by continuous variables
 - Solved by Evolutionary Strategy (ES)
- Program Search
 - program as s/w system component, design, strategy, model
 - common: system identification aka symbolic regression, modeling
 - Symbolic regression is a form of supervised machine learning
 - » GP offers some unsupervised ML techniques as well
 - Clustering
 - will show a blackbox GP example soon
 - <http://flexgp.github.io/gp-learners/sr.html>
 - <http://flexgp.github.io/gp-learners/blog.html>

Evolutionary Computation and Evolutionary Algorithms

9

EA Individual Examples

Problem	Gene	Genome	Phenotype	Fitness Function
TSP	110	sequence of cities	tour	tour length
Function optimization	3.21	variables $\mathbf{x}$ of function	$f(\mathbf{x})$	$ \text{min}-f(\mathbf{x}) $
graph k-coloring	permutation element	sequence for greedy coloring	coloring	# of colors
investment strategy	rule	agent rule set	trading strategy	portfolio change
Regress data	Executable sub-expression	Executable expression	model	Model error on training set (L1, L2)

Evolutionary Computation and Evolutionary Algorithms

10

Blackbox Example of GP Symbolic Regression

<http://flexgp.github.io/gp-learners/sr.html>
<http://flexgp.github.io/gp-learners/blog.html>

S/W by ALFA Group's FlexGP team
 Special recognition to Ignacio Arnaldo, PhD who prepared SR Learner tutorial and blog post

12

Regression

13

Regression

- Regress a relationship between a set of explanatory variables and a response variable
- Linear regression:
 - Assume linear model: $y=ax+b$
 - Optimize parameters (a,b) so data best fits model
- Logistic regression for classification
 - Maps linear model into sigmoid family

$$F(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

- Symbolic regression does NOT assume a model
 - Not parameter search
 - Model is intrinsic in GP solutions

14

FlexGP's SR Learner

- Targeted partly to be black-box for non-researchers
- sr.jar is available for download
 - Only supported for Debian linux
 - Source is on <http://flexgp.github.io>
- functionality both for performing Symbolic regression on numerical datasets and for testing the retrieved models
- Referred to as our baseline in time-aligned ALFA group publications
 - Bring Your Own Learner! A cloud-based, data-parallel commons for machine learning, Ignacio Arnaldo, Kalyan Veeramachaneni, Andrew Song, Una-May O'Reilly, IEEE Computational Intelligence Magazine, Special Issue on Computational Intelligence for Cloud Computing (Feb. 2015), Vol 10, Issue 1, pp 20-32.
 - [Multiple regression genetic programming](#), Ignacio Arnaldo, Krzysztof Krawiec, Una-May O'Reilly, GECCO '14, pp 879-888.
- Option to accelerate runs with C++ optimized execution
 - Requires gcc and g++ compilers, configuring Linux kernel parameter governing the maximum size of shared memory segments
- Option to accelerate runs with CUDA (GPU)
 - Added requirement of nvcc compiler
 - append the -cuda flag, make some extra directories...
- Easy parameter changing through a central file

15

DEMONSTRATION

- <http://flexgp.csail.mit.edu> -> LEARNERS
- <http://flexgp.github.io/gp-learners/sr.html> INSTRUCTIONS
- <http://flexgp.github.io/gp-learners/blog.html> EXAMPLE

16

Agenda

HOW DOES IT WORK UNDER THE HOOD?

WHAT IS THIS EXECUTABLE EXPRESSION?

Agenda
17

Koza's Executable Expressions

Pioneered circa 1988

- **Lisp S-Expressions**

- Composed of primitives called 'functions' and 'terminals'
- Aka operators and variables/operands

Example:

- primitives: + - * div a b c d 4
- $(*(- (+ 4 c) b) (\text{div } d a))$

In a Lisp interpreter:

1. bind a b c and d
2. Evaluate expressions

% Lisp interpreter

```
(set! a 2) -> 2
(set! b 4) -> 4
(set! c 6) -> 6
(set! d 8) -> 8
(*(- (+ 4 c) b) (div d a)) -> 12
; Rule Example
(if (= a b) c d) -> 8
; Predicate:
(> c d) -> nil
```

GP Evolves Executable Expressions

18

A Lisp GP system

A Lisp GP system is a large set of functions which are interpreted by evaluating the entry function

- Some are definitions of primitives you write!
 - » (defun protectedDivide ...)
- Rest is software logic for evolutionary algorithms

Any GP system has a set of functions that are pre-defined (by compilation or interpretation) for use as primitives also has software logic that handles

- Population initialization, iteration, selection, breeding, replacement, fitness evaluation*

GP expressions are first class objects in LISP so the GP software logic can manipulate them as data/variables as well as have the interpreter read and evaluate them

GP Evolves Executable Expressions

19

How to Evaluation an Expression

- interpreter beneath your code
 - Lisp example
- interpreter within your code
 - typical,
 - examples: SR.jar or ponygp.py
- compile then execute on your OS
 - older system in existence

20

How to Manipulate Expressions as Data

- for Crossover and Mutation we want
 - offspring can be different size and structure than parents
 - syntactic correctness
 - randomness in replication and variation
- GP solution
 - reference the parse tree
 - XO - swap subtrees between trees of parents
 - Mutation: insert, subst or delete from a parse tree (PT)
- A picture tells a 1000 words...

21

Parse Trees

Inorder: 2+3

preorder: + 2 3

Post-order: 2 3 +

Inorder: (2-3) + (a max best)

preorder: (+ (-2 3) (max a best))

Post-order: (2 3 -) (a best max) +)

- Whether parsed preorder (node, left-child, right-child) or postorder (left-child, right-child, node) or inorder (left, node, right) the expression evaluates to the same result

- (tree)GP uses an expression tree as its genotype structure

GP Evolves Executable Expressions

22

GP Tree Crossover

Nuts and Bolts GP Design

23

HVL-Mutation: substitution, deletion, insertion

Nuts and Bolts GP Design

24

GP Preparatory Steps

Assume we have a GP system with internal expression evaluator.

1. Decide upon functions and terminals
 - Terminals bind to decision variables in problem
 - Combinatorial expression space defines the search space
2. Set up the fitness function
 - Translation of problem goal to GP goal
 - Minimization of error between desired and evolved expression when executed
 - Maximization of a problem based score
 - Construct test cases for program (input examples, desired output)
3. Decide upon run parameters
 - Population size is most important
 - GP is robust to many other parameter choices
4. Determine a halt criteria and result to be returned
 - Maximum number of fitness evaluations
 - Time
 - Minimum acceptable error
 - Good enough solution (satisficing)

Nuts and Bolts GP Design

25

Top Level GP Algorithm

Begin

```

pop = random programs from a set of operators and operands
repeat
    execute each program in pop with each set of inputs
    measure each program's fitness
    repeat
        select 2 parents
        copy 2 offspring from parents
        crossover
        mutate
        add to new-pop
    until pop-size
pop = new-pop
until max-generation
or
adequate program found
    
```

Nuts and Bolts GP Design - Summary

26

Population Initialization

- Fill population with random expressions
 - Create a function set Φ and a corresponding argument-count set
 - Create an terminal set (arg-count = 0), T
 - draw from Φ with replacement and recursively enumerate its argument list by additional draws from $\Phi \cup T$.
 - Recursion ends at draw of a terminal
 - requires closure and/or typing
- maximum tree height parameter
 - At max-height-1, draw from T only
- “ramped half-half” method ensures diversity
 - equal quantities of trees of each height
 - half of height's trees are full
 - » For full tree, only draw from terminals at max-height-1

Nuts and Bolts GP Design

27

Selection in GP

- Proceeds in same manner as evolutionary algorithm
 - Same set of methods
 - Conventionally use tournament selection
 - Also see fitness proportional selection
 - Cartesian genetic programming:
 - » One parent: generate 5 children by mutation
 - » Keep best of parents and children and repeat
 - If parent fitness = child fitness, keep child

28

Determining an Expression's Fitness

- One test case:
 - Execute the expression with the problem decision variables (ie terminals) bound to some test value and with side effect values initialized
 - Designate the “result” of the expression
- Measure the error between the correct output values for the inputs and the result of the expression
 - Final output may be side effect variables, or return value of expression
 - Eg. Examine expression result and expected result for regression
 - Eg. the heuristic in a compilation, run the binary with different inputs and measure how fast they ran.
 - EG, Configure a circuit from the genome, test the circuit with an input signal and measure response vs desired response
- Usually have more than one test case but cannot enumerate them all
 - Use rational design to create incrementally more difficult test cases
 - Use class balanced data for classification

Nuts and Bolts GP Design

29

Details When Using Executable Expressions

- Closure
 - Design functions with wrappers that accept any type of argument
 - Often types will semantically clash....need to have a way of dealing with this

Practicality/Solution Feasibility

- Sufficiency
 - Make sure a correct solution can be plausibly expressed when choosing your primitive set
 - » Functions must be wisely chosen but not too complex
 - » General primitives: arithmetic, boolean, condition, iteration, assignment
 - » Problem specific primitives
 - Can you handcode a naïve solution?
 - Balance flexibility with search space size

GP Evolves Executable Expressions

30

Tree Crossover Details

• Crossover point in each parent is picked at random • Conventional practices – All nodes with equal probability – leaf nodes chosen with 0.1 probability and non-leaf with 0.9 probability • Probability of crossover – Typically 0.9 • Maximum depth of child is a run parameter – Typically ~ 15 – Can be size instead	Crossover Properties • Two identical parents rarely produce offspring that are identical to them • Tree-crossover produces great variations in offspring with respect to parents • Crossover, in addition to preserving syntax, allows expressions to vary in length and structure (sub-expression nesting)

Nuts and Bolts GP Design

32

GP Tree Mutation

- Often only crossover is used
- But crossover behaves often like macro-mutation
- Mutation can be better tuned to control the size of the change
- A few different versions

Nuts and Bolts GP Design

33

Other Sorts of Tree Mutation

- **Koza:**
 - Randomly remove a sub-tree and replace it
 - **Permute:** mix up order of args to operator
 - **Edit:** + 1 3 -> 4, and(t t) -> t
 - **Encapsulate:** name a sub-tree, make it one node and allow re-use by others (protection from crossover)
 - » Developed into advanced GP concept known as
 - Automatic module definition
 - Automatically defined functions (ADFs)
- **Make your own**
 - Could even be problem dependent (what does a subtree do? Change according to its behavior)

Nuts and Bolts GP Design

34

Top Level GP Algorithm

Nuts and Bolts GP Design - Summary

35

GP Parameters

- Population size
- Number of generations
- Max-height of trees on random initialization
 - Typically 6
- Probability of crossover
 - Higher than mutation
 - 0.9
 - Rest of offspring are copied
- Probability of mutation
 - Probabilities of addition, deletion and insertion
- Population initialization method
 - Ramped-half-half
 - All full
 - All non-full
- Selection method
 - Elitism?
- Termination criteria
- Fitness function
- what is used as “solution” of expression

Nuts and Bolts GP Design

36

GP Software Deep Dive

- flexgp.csail.mit.edu
- <http://flexgp.github.io/gp-learners/>
- **Basic:**
- https://flexgp.github.io/pony_gp/
- https://github.com/flexgp/pony_gp

37

PonyGP: Simple Symbolic Regression

- Given a set of independent decision variables and corresponding values for a dependent variable
- Want: a model that predicts the dependent variable
 - Eg: linear model with numerical coefficients
 - $Y = aX_1 + bX_2 + c(X_1X_2)$
 - Eg: non-linear model
 - $y = a x_1^2 + b x_2^3$
 - Prediction accuracy: minimum error between model prediction and actual samples
- Usually: designer provides a model and a regression (ordinary least squares, Fourier series) determines coefficients
- With genetic programming, the model (structure) and the coefficients can be learned
- Test problem:
 - $f(x) = (X_0 * X_0) + (X_1 * X_1)$
- Domain of X_0 and X_1 [-5.0,5.0]
- Choose the 4 operands (terminals)
 - $X_0, X_1, 1.0, 0$
- Choose the 4 operators (functions)
 - $+, -, *, /$ (protected)
 - protected divide: if denom==0, return numerator
- Fitness function: sum of mean squared error between y_i and expression's return values
- Prepare 121 randomized points for testing
- Out of sample training:testing ratio is 70:30, random selection of points as training or test

GP Examples
38

Agenda

Context: Evolutionary Computation and Evolutionary Algorithms

- GP is the genetic evolution of executable expressions
- Nuts and Bolts Descriptions of Algorithm Components
- Resources and reference material

Agenda
39

Reference Material

- Online Material**
- <http://geneticprogramming.com/>
- Where to search for conference and journal publications**
- Genetic Programming Bibliography
 - https://linwww.lra.uka.de/bibliography/Al/genetic_programming.html
- Digital Libraries**
- ACM digital library: <http://portal.acm.org/>
 - GECCO conferences
 - GP conferences (pre GECCO),
 - IEEE digital library: <http://www.computer.org/portal/web/csdl/home>
 - Congress on Evolutionary Computation (CEC)
 - Springer digital library: <http://www.springerlink.com/>
 - European Conference on Genetic Programming: "EuroGP"
- JOURNALS**
- Evolutionary Computation Journal (MIT Press)
 - Genetic Programming and Evolvable Machines Journal (Springer)
 - ACM Transactions on Evolutionary Learning and Optimization (ACM)
 - IEEE Transactions on Evolutionary Computation
- Software**
- <https://github.com/search?q=genetic+programming>

40

Genetic Programming Benchmarks

Genetic programming needs better benchmarks

- James McDermott, David R. White, Sean Luke, Luca Manzoni, Mauro Castelli, Leonardo Vanneschi, Wojciech Jaśkowski, Krzysztof Krawiec, Robin Harper, Kenneth De Jong, and Una-May O'Reilly.
 - In Proceedings of GECCO 2012, Philadelphia, 2012. ACM.
- Related benchmarks wiki**
 - <http://GPBenchmarks.org>
- GP Program Synthesis Benchmarks**
 - http://thelmuth.github.io/GECCO_2015_Benchmarks_Materials/
 - Thomas Helmuth, Lee Spector

41

Software Packages for Symbolic Regression

No Source code available

- **Datamodeler** - mathematica, Evolved Analytics
- **Eureqa II/ Formulize** - a software tool for detecting equations and hidden mathematical relationships in data
 - <http://creativemachines.cornell.edu/eureqa>
 - Plugins to Matlab, mathematica, Python
 - Convenient format for data presentation
 - Standalone or grid resource usage
 - Windows, Linux or Mac
 - <http://www.nutonian.com/> for cloud version
- **Discipulus™ 5 Genetic Programming Predictive Modelling**

42

Reference Material - Books

- [Genetic Programming](#), James McDermott and Una-May O'Reilly, In the Handbook of Computational Intelligence, Topic Editors: Dr. F. Neumann and Dr. K Witt, Editors in Chief Prof. Janusz Kacprzyk and Prof. Witold Pedrycz.
- **Essentials of Metaheuristics**, Sean Luke, 2010
- **Genetic Programming: From Theory to Practice**
 - 10 years of workshop proceedings, on SpringerLink, edited
- **A Field Guide to Genetic Programming**, Poli, Langdon, McPhee, 2008, Lulu and online digitally
- **Advances in Genetic Programming**
 - 3 years, each in different volume, edited
- **John R. Koza**
 - Genetic Programming: On the Programming of Computers by Means of Natural Selection, 1992 (MIT Press)
 - Genetic Programming II: Automatic Discovery of Reusable Programs, 1994 (MIT Press)
 - Genetic Programming III: Darwinian Invention and Problem Solving, 1999 with Forrest H Bennett III, David Andre, and Martin A. Keane, (Morgan Kaufmann)
 - Genetic Programming IV: Routine Human-Competitive Machine Intelligence, 2003 with Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, and Guido Lanza
- **Linear genetic programming**, Markus Brameier, Wolfgang Banzhaf, Springer (2007)
- **Genetic Programming: An Introduction**, Banzhaf, Nordin, Keller, Francone, 1997 (Morgan Kaufmann)

43