Linear Genetic Programming
In
Python Bytecode

Mark Burgess
ANU PhD student

markburgess1989@gmail.com
http://oddblog.info
Audience Poll

Put the hand up if:

A) you know what “Genetic Programming” is?
Audience Poll

Put the hand up if:

A) you know what “Genetic Programming” is?

B) have ‘done’ “Genetic Programming”? 
Audience Poll

Put the hand up if:

A) you know what “Genetic Programming” is?

B) have ‘done’ “Genetic Programming”?

C) have fiddled with Python Bytecode?
Evolutionary Paradigm
Evolutionary Paradigm
Evolutionary Paradigm

Initial Things → Evaluation
Evolutionary Paradigm

Initial Things → Evaluation → Selection
Evolutionary Paradigm

1. Initial Things
2. Evaluation
3. Variation
4. Selection
Evolutionary Paradigm

Initial Things → Evaluation → Variation → Selection
Evolutionary Paradigm

- Initial Things
- Evaluation
- Variation
- Selection
- Termination
Organisms as Programs
Organisms as Programs

0    LOAD_FAST
3    LOAD_FAST
6    COMPARE_OP
9    POP_JUMP_IF_FALSE
12   LOAD_FAST
15   LOAD_FAST
18   LOAD_FAST
21   BINARY_TRUE_DIVIDE
22   BINARY_ADD
23   RETURN_VALUE
Organisms as Programs

+++[>++++++>+++++++++++]>++<
<[-]>++++++.++++++++++.-----
-----+++.[>>>>>+<<<<<-]>>>[<<<+<<<<-]<<<<-]
<<<<<[-.>>>].----------.<<[>>>>>+<<<<<-]
]>++.<<<<<.>>>>+.+++++.>---.
Organisms as Programs

Choose/Design a Representation/Language

Type-consistency
Evaluation safety
‘Sufficiency’

Defines the Search Space
Initialisation

Initial population

Generated Algorithmically
(valid and varied)

Seeded
Evaluation

Usually according to a “Fitness Function”

Multi-facted
Comparative
Executing
Smooth
Selection

Techniques for broad select of ‘fitter’ individuals

eg. Tournament – Roulette – Deterministic

Greediness

Evaluation as needed
Variation

- Duplication
- Inversion
- Deletion
- Insertion
- Translocation

Father
Mother
Offspring

Randomly Selected Subtree
Randomly Selected Subtree
Replace Old Subtree With New Tree
Iteration and Termination
An Example – Symbolic Regression

<table>
<thead>
<tr>
<th>Table 4.1: Parameters for example genetic programming run</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective: Find program whose output matches $x^2 + x + 1$ over the range $-1 \leq x \leq +1$.</td>
</tr>
<tr>
<td>Function set: $+$, $-$, $%$ (protected division), and $\times$; all operating on floats</td>
</tr>
<tr>
<td>Terminal set: $x$, and constants chosen randomly between $-5$ and $+5$</td>
</tr>
<tr>
<td>Fitness: sum of absolute errors for $x \in {-1.0, -0.9, \ldots 0.9, 1.0}$</td>
</tr>
<tr>
<td>Selection: fitness proportionate (roulette wheel) non elitist</td>
</tr>
<tr>
<td>Initial pop: ramped half-and-half (depth 1 to 2. 50% of terminals are constants)</td>
</tr>
<tr>
<td>Parameters: population size 4, 50% subtree crossover, 25% reproduction, 25% subtree mutation, no tree size limits</td>
</tr>
<tr>
<td>Termination: Individual with fitness better than 0.1 found</td>
</tr>
</tbody>
</table>

http://www.gp-field-guide.org.uk/
An Example – Symbolic Regression
An Example – Symbolic Regression

(a) 7.7
(b) 11.0
(c) 17.98
(d) 28.7
An Example – Symbolic Regression

(a) \[ x + 1 \]
(b) \[ x \cdot x \]
(c) \[ 2 + 0 \]
(d) \[ x - 1 \]

(a) 7.7
(b) 11.0
(c) 17.98
(d) 28.7

Reproduction: a
An Example – Symbolic Regression

(a) 7.7
(b) 11.0
(c) 17.98
(d) 28.7

Reproduction: a
Mutation: c
An Example – Symbolic Regression

Reproduction: a
Mutation: c
Crossover: a, b

(a) 7.7
(b) 11.0
(c) 17.98
(d) 28.7
An Example – Symbolic Regression

(a) 7.7
(b) 11.0
(c) 17.98
(d) 28.7

Reproduction: a
Mutation: c
Crossover: a, b
DEAP: Evolving 1’s

```python
import array
import random
from deap import algorithms
from deap import base
from deap import creator
from deap import tools

creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", array.array, typecode='b', fitness=creator.FitnessMax)

toolbox = base.Toolbox()
toolbox.register("attr_bool", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_bool, 100)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
toolbox.register("evaluate", lambda x: [sum(x)])
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
toolbox.register("select", tools.selTournament, toursize=3)

pop = toolbox.population(n=300)
hof = tools.HallOfFame(1)

pop, log = algorithms.eaSimple(pop, toolbox, cxpb=0.5, mutpb=0.2, ngen=40,
                                halloffame=hof, verbose=True)

print hof[0]]
```
DEAP: Symbolic Regression

```python
import operator
import math
import random

from deap import algorithms
from deap import base
from deap import creator
from deap import tools
from deap import gp

pset = gp.PrimitiveSet("MAIN", 1)
pset.addPrimitive(operator.add, 2)
pset.addPrimitive(operator.sub, 2)
pset.addPrimitive(operator.mul, 2)
pset.addPrimitive(operator.neg, 1)
pset.addEphemeralConstant("rand101", lambda: random.randint(-1,1))
pset.renameArguments(ARG0='x')

creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", gp.PrimitiveTree, fitness=creator.FitnessMin)

toolbox = base.Toolbox()
toolbox.register("expr", gp.genHalfAndHalf, pset=pset, min=1, max=2)
toolbox.register("individual", tools.initIterate, creator.Individual, toolbox.expr)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
toolbox.register("compile", gp.compile, pset=pset)

def evalSymbReg(individual, points):
    func = toolbox.compile(expr=individual)
    sqerrors = [(func(x) - x**4 - x**3 - x**2 - x)**2 for x in points]
    return math.fsum(sqerrors) / len(points),

toolbox.register("evaluate", evalSymbReg, points=[x/10. for x in range(-10,10)])
toolbox.register("select", tools.selTournament, tournsize=1)
toolbox.register("mate", gp.cxOnePoint)
toolbox.register("expr_mut", gp.genFull, min=0, max=2)
toolbox.register("mutate", gp.mutUniform, expr=toolbox.expr_mut, pset=pset)

pop = toolbox.population(n=300)
hof = tools.HallOfFame(1)
pop, log = algorithms.eaSimple(pop, toolbox, 0.5, 0.1, 40, halloffame=hof, verbose=True)
print hof[0]
```
DEAP: Symbolic Regression

```python
def compile(expr, pset):
    args = ",".join(arg for arg in pset.arguments)
    code = "lambda {args}: {code}".format(args=args, code=str(expr))
    return eval(code, pset.context, {})
```
Python Bytecode (!)

Create a function object from a code object and a dictionary. The optional `name` string overrides the name from the code object. The optional `argdefs` tuple specifies the default argument values. The optional `closure` tuple supplies the bindings for free variables.

Create a code object. Not for the faint of heart.

**Relevant:**
- `code_string`
- `argcount`
- `nlocals`
- `stacksize`
- `constants`
- `varnames`
- `flags`

**Less Relevant:**
- `freevars`
- `cellvars`

**Unlikely Relevant:**
- `names`
- `kwonlyargcount`

**Inconsequential:**
- `filename`
- `name`
- `firstlineno`
- `lnotab`
# Flags (†)

<table>
<thead>
<tr>
<th>Flag Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CO_OPTIMIZED</td>
<td>0x0001</td>
<td>Was once an important flag</td>
</tr>
<tr>
<td>CO_NEWLOCALS</td>
<td>0x0002</td>
<td>new 'locals' dict for execution otherwise uses globals if local object unsupplied</td>
</tr>
<tr>
<td>CO_VARARGS</td>
<td>0x0004</td>
<td>For Variable args</td>
</tr>
<tr>
<td>CO_VARKEYWORDS</td>
<td>0x0008</td>
<td>For Variable kwargs</td>
</tr>
<tr>
<td>CO_NESTED</td>
<td>0x0010</td>
<td>unused</td>
</tr>
<tr>
<td>CO_NOFREE</td>
<td>0x0040</td>
<td>unused</td>
</tr>
<tr>
<td>CO_GENERATOR</td>
<td>0x0020</td>
<td>For Generators</td>
</tr>
<tr>
<td>CO_COROUTINE</td>
<td>0x0080</td>
<td>For Coroutines</td>
</tr>
<tr>
<td>CO_ITERABLE_COROUTINE</td>
<td>0x0100</td>
<td>For Coroutines</td>
</tr>
<tr>
<td>CO_GENERATOR_ALLOWED</td>
<td>0x1000</td>
<td>unused</td>
</tr>
<tr>
<td>CO_FUTURE_DIVISION</td>
<td>0x2000</td>
<td>Moot in Python3.5</td>
</tr>
<tr>
<td>CO_FUTURE_ABSOLUTE_IMPORT</td>
<td>0x4000</td>
<td>Moot in Python3.5</td>
</tr>
<tr>
<td>CO_FUTURE_WITH_STATEMENT</td>
<td>0x8000</td>
<td>Moot in Python3.5</td>
</tr>
<tr>
<td>CO_FUTURE_PRINT_FUNCTION</td>
<td>0x10000</td>
<td>Moot in Python3.5</td>
</tr>
<tr>
<td>CO_FUTURE_UNICODE_LITERALS</td>
<td>0x20000</td>
<td>Moot in Python3.5</td>
</tr>
<tr>
<td>CO_FUTURE_BARRY_AS_BDFL</td>
<td>0x40000</td>
<td>Easter Egg __(_&lt; )__/</td>
</tr>
<tr>
<td>CO_FUTURE_GENERATOR_STOP</td>
<td>0x80000</td>
<td>generator raising StopIteration</td>
</tr>
</tbody>
</table>
Easter Egg

>>> from __future__ import braces
File "<stdin>", line 1
SyntaxError: not a chance

>>> from __future__ import barry_as_FLUFL
>>> #Friendly Language Uncle For Life (FLUFL)
... #(https://www.python.org/dev/peps/pep-0401/)
... 1 <> 2
True
>>> 1 != 2
   File "<stdin>", line 1
     1 != 2
    ^
SyntaxError: invalid syntax
Example Bytecode Decomposition

```
from dis import dis, show_code

def a(x, y, z):
    return x + y * z - 4

print("\nBYTECODE:")
print([z for z in a.__code__.co_code])
print("\nDISSASSEMBLY:")
dis(a)
print("\nCODE OBJECT:")
show_code(a)
```

Documentation for opcodes in dis module documentation

See also opcodes.py

Dissassembly is elaboration of a.__code__.XYZ
Example Bytecode Reconstitution

```
BYTECODE: [124, 0, 0, 124, 1, 0, 124, 2, 0, 20, 23, 100, 1, 0, 24, 83]
DISSASSEMBLY:
  0 LOAD_FAST           0 (x)
  3 LOAD_FAST           1 (y)
  6 LOAD_FAST           2 (z)
  9 BINARY_MULTIPLY     9 (4)
 10 BINARY_ADD
 11 LOAD_CONST
 14 BINARY_SUBTRACT
 15 RETURN_VALUE

CODE OBJECT:
Name: a
Filename: 1.py
Argument count: 3
Kw-only arguments: 0
Number of locals: 3
Stack size: 3
Flags: OPTIMIZED
Constants: 0: None 1: 4
Variable names: 0: x 1: y 2: z
```

```
def a(x, y, z):
    return x + y * z - 4

from types import CodeType, FunctionType

code(argcount, kwonlyargcount, nlocals, stacksize, flags, codestring, constants, names, varnames, filename, name, firstlineno, lnotab[, freevars[, cellvars]])

c = CodeType(3, 0, 3, 3, 0,
            a.__code__.co_code,
            (None, 4), (),
            u'bilbo', u'baggins', 0, b'
        )
f = FunctionType(c, {})
```
Linear GP & python – why?

1. is fun/flexible for experimentation
2. is a reasonably well developed VM
3. has potential for distributed GP
4. faster than Tree GP
5. for kicks
Crash python, you must not.

- Stack Underflow causes seg-fault
- Division by zero / other runtime exceptions
- Strange / Random referencing
- Infinite Loops

Remedies

- Program Repairing
- Using safe operations
- Semantic preserving mutation/crossover/generation
Symbolic Regression done with Linear Genetic Programming via Python Bytecode
import random
from deap import algorithms
from deap import base
from deap import creator
from deap import tools
from opcode import opmap
from types import CodeType, FunctionType
from dis import dis
dis.show_code

operations = (
    (1, (opmap["LOAD_CONST"], 0, 0)),
    (1, (opmap["LOAD_CONST"], 1, 0)),
    (1, (opmap["LOAD_CONST"], 2, 0)),
    (0, (opmap["UNARY_INVERT"], )),
    (-1, (opmap["BINARY_MULTIPLY"], )),
    (-1, (opmap["BINARY_ADD"], )),
    (-1, (opmap["BINARY_SUBTRACT"], )),
    (-1, (opmap["BINARY_AND"], )),
    (-1, (opmap["BINARY_OR"], )),
    (-1, (opmap["BINARY_XOR"], )),
    (1, (opmap["LOAD_FAST"], 0, 0)),
)

terminate = (opmap["RETURN_VALUE"],)

negones = [o for o in operations if o[0] == -1]
zeros = [o for o in operations if o[0] == 0]
one = [o for o in operations if o[0] == 1]

max_len = 40

def fix(individual):
    if not individual.fixed:
        i = 0
        d = 0
        for l in range(max_len, len(individual)):
            individual.pop()
        while i < l:
            if (d + individual[i][0] < 1):
                individual.insert(i, random.choice(one))
                l = l + 1
            else:
                d = d + individual[i][0]
                i = i + 1
        for i in range(d, l):
            individual.append(random.choice(negones))
        individual.ephemeral = random.randint(0, 999999)
        individual.fixed = True
def compile_individual(individual):
    codebytes = []
    for d, op_code in individual:
        codebytes.append(op_code)
    codebytes.append(terminate)
    codebytes = [i for sub in codebytes for i in sub]
    codebytes = bytes(codebytes)

    co_obj = CodeType(1, 0, 1, max_len, 0, codebytes, (1, 2, individual.ephemeral), ()
    "a", u"file_name", u"code_name", 0, b")
    B = FunctionType(co_obj, [])
    return B

def evaluate(individual):
    fix(individual)
    c = compile_individual(individual)
    return sum([-c[i]-i*i+i-4)**2 for i in range(0, 10)],

def crossover(i1, i2):
    i1.fixed = False
    i2.fixed = False
    return tools.cxTwoPoint(i1, i2)

def mutate(individual, indpb):
    if random.random() < indpb:
        individual.fixed = False
        c = random.randint(0, 2)
        if c==0:
            return tools.mutShuffleIndexes(individual, 1, 1)
        elif c==1:
            a = random.randint(0, len(individual)-1)
            b = random.randint(0, len(individual)-1)
            if a > b:
                a, b = b, a
            del(individual[a:b])
            return individual,
        elif c==2:
            a = random.randint(0, len(individual)-1)
            del(individual[a:])
            i = toolbox.individual()
            [individual.append(a) for a in i]
            return individual,
    return individual,
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", type=[], fitness=creator.FitnessMax, fixed=False, ephemeral=0)

toolbox = base.Toolbox()
toolbox.register("random_operation", random.choice, operations)
toolbox.register("random_length", random.randint, 3, 8)
toolbox.register("individual", lambda c,t,n;c(t) for z in range(n()), creator.Individual, toolbox.random_operation, toolbox.random_length)

toolbox.register("population", tools.initRepeat, list, toolbox.individual)
toolbox.register("evaluate", evaluate)
toolbox.register("mate", crossover)
toolbox.register("mutate", mutate, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)

c = toolbox.population(n=300)
hof = tools.HallOfFame(1)

pop, log = algorithms.eaSimple(c, toolbox, cxpb=0.5, mutpb=0.2, ngen=80, halloffame=hof, verbose=True)

def print_code(individual):
    B = compile_individual(individual)
    print(dis(B))
    print(show_code(B))

def print_values(individual):
    B = compile_individual(individual)
    print([B[i] for i in range(0,10)])
    print([i*i-1-4 for i in range(0,10)])
    print([-(B[i]-i*1+1-4)**2 for i in range(0,10)])

print_code(hof[0])
print_values(hof[0])
TA-DA :-) 

((((a*a)|2)-a)+1)+1) = a^2-a+4
THE END