

## Genetic Programming

1

## Genetic Programming

- What is it?
  - Genetic programming (GP) is an automated method for creating a working computer program from a high-level problem statement of a problem.
  - Genetic programming starts from a high-level statement of "what needs to be done" and automatically creates a computer program to solve the problem.

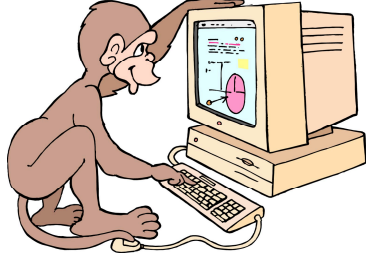
2

## Genetic Programming

- John Koza, 1992
  - "Genetic Programming: On the Programming of Computers by Means of Natural Selection"
  - [www.genetic-programming.com](http://www.genetic-programming.com)

3

## Genetic Programming



4

## Genetic Programming

- In genetic programming:
  - Problem -- involves not finding a solution, but instead creating a program that can find the best solution.
  - Phenotype (solution) is a computer program
  - Search space is the set of all possible computer programs.

5

## Genetic Programming

- In Koza's original work
  - LISP was used as the programming language
  - Parse trees were used as the genotype.
  - Straight-forward genetic mapping
    - Functional program --> parse tree.

6

## LISP and parse trees

```
(defun o()
  (setf A 1.52)
  (sqrt (* A (* A A ))))
```

Mitchell

## Genetic Programming

- GPs fit well within the EA framework
- Important distinction:
  - Genotype = tree
  - Phenotype = LISP program

8

## GPs as EAs

- To use evolutionary algorithms you must:
  - Define your problem -- must provide hints to functional building blocks
  - Define your genotype
  - Identify your phenotype
  - Define the genotype -> phenotype translation
  - Define crossover and mutation operators
  - Define fitness
  - Determine selection criteria
  - Set population parameters

9

## Defining your problem

- For programs to be evolved, must define:
  - Terminal Set
    - Leaves of the parse tree
    - Correspond to program inputs
  - Function Set
    - Interior nodes of parse tree
    - Set of functions allowable in program
    - Should be chosen with relation to problem to be solved.
    - Can have side effects.

10

## Defining your problem

- Sufficiency and Closure
  - Sufficiency
    - terminals + functions must be able to solve the problem at hand
    - Choose your terminal and function set wisely
  - Closure
    - Functions should be able to accept as argument any value returned by other functions
    - Universal return type.

11

## GPs as EAs

- To use evolutionary algorithms you must:
  - Define your problem -- must provide hints to functional building blocks
  - Define your genotype -- parse trees
  - Identify your phenotype -- LISP program
  - Define the genotype -> phenotype translation
  - Define crossover and mutation operators
  - Define fitness
  - Determine selection criteria
  - Set population parameters

12

## GPs: Crossover and Mutation

- Crossover
  - Choose random sub-tree from each parent
  - Swap them in resultant children
- Mutation
  - Replace randomly chosen sub-tree with randomly generated tree.
- Can result in increasing / decreasing the size of the genome.

13

## GPs: Crossover and Mutation

crossover

mutation

- Note that operation maintain valid individuals.

14

## GPs and crossover

15

## GPs as EAs

- To use evolutionary algorithms you must:
  - Define your problem -- must provide hints to functional building blocks
  - Define your genotype -- parse trees
  - Identify your phenotype -- LISP program
  - Define the genotype -> phenotype translation
  - Define crossover and mutation operators
  - Define fitness
  - Determine selection criteria
  - Set population parameters

16

## Fitness

- Evaluate the effectiveness of the program to solve the problem
- Two stage fitness:
  - Run resultant program
  - Compare to correct / desired results

17

## GPs as EAs

- To use evolutionary algorithms you must:
  - Define your problem -- must provide hints to functional building blocks
  - Define your genotype -- parse trees
  - Identify your phenotype -- LISP program
  - Define the genotype -> phenotype translation
  - Define crossover and mutation operators
  - Define fitness
  - Determine selection criteria
  - Set population parameters

18

## GPs as EAs

- In Koza's original work:
  - 10% of individuals will move to next generation
    - Probability based on Fitness
  - 90% formed by crossover
    - Parent chosen probabilistically based on fitness
  - Koza does not use mutation.

19

## GPs as EAs

- To use evolutionary algorithms you must:
  - Define your problem -- must provide hints to functional building blocks
  - Define your genotype -- parse trees
  - Identify your phenotype -- LISP program
  - Define the genotype -> phenotype translation
  - Define crossover and mutation operators
  - Define fitness
  - Determine selection criteria
  - Set population parameters

20

## Initializing population

- Randomly creating programs.
  - Must assure that trees match valid programs.
    - Number of function arguments must match.
  - Usually a limit is set on tree depth of generated program.
- Questions so far

21

## Trivial example

- From Mitchell
  - Program to compute P, the orbital period of a planet.
    - Parameters A = average distance from the sun.
  - Known solution:
    - $P^2 = cA^3$
    - c = constant.

22

## Trivial example

Fitness Cases:		
Planet	A	Correct Output (P)
Venus	0.72	0.61
Earth	1.00	1.00
Mars	1.52	1.84
Jupiter	5.20	11.9
Saturn	9.53	29.4
Uranus	19.1	83.5

23

## Trivial example

- Terminal set:
  - A = average distance from the sun
- Function set:
  - Typical arithmetic operations:
    - +, -, \*, /, sqrt
  - Closure -- Note that all take float and return float

24

## Trivial example

- Sample solutions

$A + (A + A)$

$A \ [(A \ A) \ (A \ A)]$

25

## Trivial example

- Fitness:

Planet	A	Correct Output (P)
Venus	0.72	0.61
Earth	1.00	1.00
Mars	1.52	1.84
Jupiter	5.20	11.9
Saturn	9.53	29.4
Uranus	19.1	83.5

26

## More interesting Example

- Block stacking problem
  - Koza 1992 as described by Mitchell
  - Given:
    - Set of blocks that can either be
      - On a stack
      - On a table.
  - Goal:
    - Find a program that will place the blocks on the stack in the "correct" order.

27

## Other GP variants

- Strongly typed GP
  - Gets around the closure idea
  - Function arguments are typed
  - Individual generation and genetic operators must respect the types.

28

## Building Block Example

- More elaborate scheme to solve this:
  - Using Strongly Typed GPs
  - Found in [Kochenderfer]
- Questions?

29

## Factoids about Genetic Programming

- From Koza's Web site
  - There are now 36 instances where genetic programming has produced a human-competitive result.
    - 15 instances where GP has created an entity that either infringes or duplicates the functionality of a previous patent
    - 6 instances where genetic programming has done the same with respect to a 21st-century invention
    - 2 instances where genetic programming has created a patentable new invention.

30

## Factoids about Genetic Programming

- GPs themselves were patented by Koza:
  - Koza, John R. Non-Linear Genetic Algorithms for Solving Problems.
    - US Patent 4,935,877. Issued June 19, 1990.
    - Australian Patent 611,350. Issued September 21, 1991.
    - Canadian Patent 1,311,561. Issued December 15, 1992.
    - German patent 3916328.8-53. Issued June 18, 1997.

31

## Genetic Programming

- In Koza's original work
  - LISP was used as the programming language
  - Parse trees were used as the genotype.
- Straight-forward genetic mapping
  - Functional program --> parse tree.
- Not the only way to do genetic programming.

32

## Genetic Programming

- In genetic programming:
  - **Problem** -- involves not finding a solution, but instead creating a program that can find the best solution.
  - **Phenotype** (solution) is a computer program
  - **Search space** is the set of all possible computer programs.
- Genotype need not be a parse tree!

33

## GPs as GAs

- GADS
  - [Paterson, Livesey]
- Phenotype = program
- Genotype = n-tuple (array).

34

## Languages as grammars

- Grammars for programming languages
  - $\langle \text{stmt} \rangle \rightarrow \dots \mid \langle \text{for-stmt} \rangle \mid \langle \text{if-stmt} \rangle \mid \dots$
  - $\langle \text{stmt} \rangle \rightarrow \{ \langle \text{stmt} \rangle \langle \text{stmt} \rangle \} \mid \epsilon$
  - $\langle \text{if-stmt} \rangle \rightarrow \text{if } ( \langle \text{expr} \rangle ) \text{ then } \langle \text{stmt} \rangle$
  - $\langle \text{for-stmt} \rangle \rightarrow \text{for } ( \langle \text{expr} \rangle ; \langle \text{expr} \rangle ; \langle \text{expr} \rangle ) \langle \text{stmt} \rangle$

35

## Phenotype Representation

- Phenotype is a program in a given language
  - Language is defined by a grammar in BNF.
  - Each production in the grammar is uniquely numbered.

lhs  $\rightarrow$  rhs  
 $\langle \text{sexpr} \rangle \rightarrow (GT \langle \text{sexpr} \rangle \langle \text{sexpr} \rangle)$

36

## Phenotype Representation

<sexp>	::=	<input>	0
<sexp>	::=	<application>	1
<input>	::=	X	2
<input>	::=	V	3
<input>	::=	-1	4
<application>	::=	(<arity1> <sexp>)	5
<application>	::=	(<arity2> <sexp> <sexp>)	6
<arity1>	::=	ABS	7
<arity2>	::=	+	8
<arity2>	::=	-	9
<arity2>	::=	*	10
<arity2>	::=	%	11
<arity2>	::=	GT	12

37

## Genotype representation

- A valid program can be described by a sequence of applications of grammar rules.
- Genotype = tuple or array representing this sequence.
  - Let's try:
    - (GT (\* -1 X) (\* V (ABS V) ) )
- Can use "standard" GA operators for tuples for crossover and mutation.

38

## Genetic Mapping

- Genetic Repairs Required:
  - Inapplicable productions.
    - Assume after a number of production applications (as defined by a genome) the generated program is:

(GT <sexpr> <numeral>)

What if the the production of the next gene does NOT have a lhs of <sexpr> or <numeral>?

39

## Genetic Mapping

- Genetic Repairs Required:
  - Inapplicable productions.
    - Genetic repair: skip the problematic gene and progress to the first gene that has an appropriate lhs.

40

## Genetic Mapping

- Genetic Repairs Required:
  - Residual non-terminals
    - What if...after all productions associated to all genes have been applied and there are still non-terminals in the program?

41

## Genetic Mapping

- Genetic Repairs Required:
  - Residual non-terminals
    - Reject this individual or
    - Have each non-terminal have a default terminal value and use this default.

42



## Fixed length vs variable length Genotype

- Can use either...
- For fixed length, stop after all non-terminals have been exhausted and replaced.
- Paper used fixed length chromosomes of length 50 and 200.

43



## Findings

- Certainly a viable approach
- Very grammar specific
  - Even different grammars for same language.
- No need to use LISP.
- Better than traditional GPs?
  - Jury is still out.

44



## Paterson Paper

- Take home messages:
  - Distinction between genotype and phenotype
  - GPs don't have to use parse trees as phenotype.
  - GPs need not only use LISP.
  - Creative genetic mapping and repair
- Questions?

45