

**GP**

## **Genetic Algorithms on Lisp Expr's**

### **What is GP?**

- **Genetic Programming**
  - Genetic algorithms applied to lisp expressions
    - Expressions can be "variable" length
  - More complex crossover, evaluation, e.g.
    - Running program determines fitness

## The GA Flowchart

- 1. Start with random population
- 2 evaluate fitness of population, stop when
  - some member meets critereon
  - or no more progress
- 3. Normalize to get relative fitness
- 4. Generate new population
- 5. go to step 2

## Population

- elements of a set or language (Phenotypes) expressed in a code or variable data structure (genotype)
- Examples
    - bitstrings of length  $n$  (default)
    - vectors of real numbers
    - graphs
    - permutations
    - arithmetic expressions (logical formula)
      - Often converted to bit-strings for religious reasons

## Fixed Width?

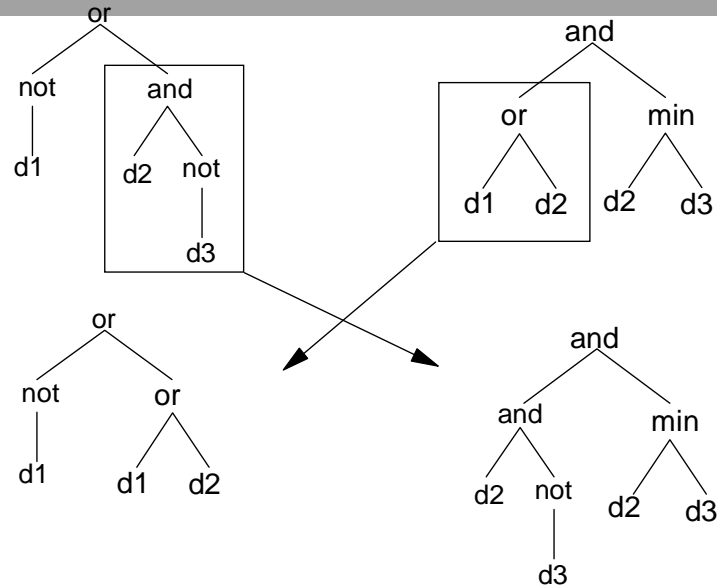
- Fixed width (binary or real) has limits
  - searching a "finite" space
  - Determined "adjacency" as bias
  - Hammer and Nail phenomenon
- What if the genotype isn't fixed width?
  - how do you do crossover?
  - How do you evaluate fitness?

## Koza's Genetic Programming

- Genotypes are formal lisp expressions with constrained set of primitives
- Fitness is done through **running program** on desired task
- Crossover is performed by tree grafting

genetic evolution of expressions  
claimed by Michael Kramer earlier

# Crossover Operator



## About crossover...

- Has to preserve "arity" of function
- 
- Q: Why should created expressions work at all?
- A: Because choice of primitives and syntax narrow space
- A: Because large populations overcome garbage

## What is needed to do a GP problem?

- Set of functions and arity
  - {+ - \* / sin cos, etc)
  - IF (X THEN ELSE)
  - IFLTE (X Y THEN ELSE)
- Set of Terminals
  - x, y, z (problem variables)
  - R (ephemeral random number)
- Fitness Function
  - How to evaluate an evolved function?

## GP Hacks

- functions which are protected from error
  - Divide which checks denominator for 0
    - (defun % (x y)(if (zerop y) 1 (/ x y)))
  - Square root which uses Absolute value
    - (defn srt (x) (sqrt (abs x)))
- Initial Function Generator
  - Which adds inductive bias
- LIMIT CUTOFF
  - Stop programs from growing (and slowing)
    - "Bloat" is a big GP problem

## Things which have been GP'ed

- Control of pendulums
- Logical problems (multiplexor)
- symbolic integration
- induction of sequences
- game learning
- classification
- Many more problems
  - Each one requires custom primitives, fitness function, parameters, etc.

## Roulette Wheel

- (defun roulette (set distribution)
- (let ((choice (random 1.0)))
- (loop for x in set as y in distribution do
- (if (< choice y) (return x))
- (setf choice (- choice y))))))

## Set up Population (a list of expressions)

- (defun make-rand-expr (nonterms terms depth)
- (if (< depth 2)(pick terms)
- (let ((n (pick nonterms)))
- (cons (car n)
- (loop for i from 1 to (cdr n) collect
- (make-rand-expr nonterms terms (random depth))))))
- 
- (defun make-init-pop (nonterms terms depth n)
- (loop for i from 1 to n collect
- (make-rand-expr nonterms terms depth)))
- 

## the functions and the terminals

- (defvar fl '((+ . 2) (\* . 2)))
- (defvar tl '(1.0 x))
- 
- This allows GP to explore space of expressions like
- (\* (+ 1.0 x)(+ x x))

## CROSSOVER (SIMPLE)

- (defun crossover (e1 e2)
- (replace-subexpr e1
- (pick-subexpr e1)(pick-subexpr e2)))
- 
- (defun pick-subexpr (expr)
- (pick (list-all-subexprs expr)))
- 
- (defun list-all-subexprs (expr)
- (if (consp expr)
- (cons expr (loop for x in (cdr expr) append (list-all-subexprs
- x))))))

## Fitness Function uses "environment" to get variables

- (defun ff-generator (x-data y-data)
  - #'(lambda (expr)
  - (loop for x in x-data as y in y-data
  - sum (abs (- y (eval expr))))))
  - 
  - (setf x1 '(1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0))
  - (setf y1 (loop for x in x1 collect (+ (\* 2 x x) 1)))
  - 
  - (setf (symbol-function 'ff1) (ff-generator x1 y1))
  - 
  - now (ff1 '(+ x x))
- BUSTED! HELP!**

## Creating a New generation

- (defun new-population (population fitness nonterms terms)
- (loop for x in population collect
- (let ((parent (roulette population fitness)))
- (case (roulette '(cross mutate pick) \*PARAMETERS\*)
- ((cross) (crossover parent (pick population)))
- ((mutate) (mutate parent nonterms terms))
- ((pick) parent))))))
- 
- 
- 

## The whole thing...

- (defun genprog (pop gens fitfunc nonterms terms)
- (let\* ((fitness (mapcar fitfunc pop))
- (bestfit (loop for x in fitness minimize x))
- (bestone (nth (index bestfit fitness) pop)))
- ;print info about current generation
- (if (and (not (< bestfit 0.1)) (not (zerop gens)))
- (genprog
- (new-population pop (normalize fitness) nonterms terms)
- (- gens 1) fitfunc nonterms terms))
- pop)))

## Pros and Cons of GP

- Powerful methodology, engineering flexibility
- Can be parallelized
- Can lead to solutions which can be further refined
- Uses a lot of computer time
- Embeds a lot of human bias
  - E.g. did it work because human intelligence in "setup"?
- Too many tricks and parameters make it go.

## Simple GP problems

Problem	Trigonometric Identities
Objective	$\cos^2(x)$
Terminals	x, 1.0
Functions	+ - * / SIN

## Problems

Problem	Symbolic Integration
Objective	$\cos(x)+2x+1$
Terminals	x
Functions	+ - * / SIN COS EXP

## Genetic Library Builder (GLIB) Angeline/Pollack

- Learned to play TicTacToe
- GP with additional Mutation:
  - Compression by Defining new functions

## 3 by 3 game primitives

- POS00 ...POS22 (Position Tokens)
- OPEN (pos) returns pos or NIL
- MINE (pos)
- YOURS (pos)
- PLAY (pos) (side-effecting)
- AND (LISP control function)
- OR (LISP control function)

00	01	02
10	11	12
20	21	22

## Evaluating Fitness for TicTacToe

- Plays a game by iterative eval (no loops).
- Fitness by playing imperfect symbolic player
- Cumulative Score (Halt on error):
  - +1 for legal move
  - +1 for blocking
  - +4 for draw
  - +12 for win

## Sample Game by Evolved Player

	X	

## Sample Game by Evolved Player

O		
		X
	X	

## Sample Game by Evolved Player

O		
	X	X
	X	O

## Sample Game by Evolved Player

O	O	X
	X	X
	X	O

## Cyberpath

O	O	X	Maximizes Score
<hr/>			winQ 12
X	X	X	Block 3
<hr/>			legal 5
O	X	O	

## Cyberpath led to insights

- TTT Learning needed a "fallable" opponent
  - Learning against random player failed
  - learning against total expert failed
- The "programming cost" of the heuristic player should be counted as inductive bias
- Is there a way to learn without a teacher?
  - YES: Co-evolution...

# Co-evolution

## ■ Co-evolution

- evaluation of a strategy in a complex competitive environment
  - ▶ Basis for Self-learning (autodidactic systems)
  - ▶ Exploits "arms-race" phenomena to solve "chicken & egg" problems
  - ▶ Absolute versus Relative fitness
  - ▶ Suffers from Measurement Problems
    - The Red Queen Effect
    - Mediocre Stable States
    - Death Spirals