



Evolutionary Algorithms



Background

- Evolution, as the natural selection of variations of types, is a powerful form of learning
 - optimization (speed, energy usage, vision)
 - Discovery of novelty (vision, flight, cognition)
 - Organization of complexity
- Biology is a consistent source of analogies and ideas for weak methods and theories:
 - hill-climbing search, generate and test, society of mind, neural networks, etc.



The Flowchart

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

■ * sometimes we don't know the best ans




Population

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

■ ■ ■ ■ ■ ■ ■ ■ ■



Example: Vector of Real Numbers

■ Basic procedure:

- Establish range (-10,10)
- Establish Granularity (.02)
- Round range/granularity to 2^n (1024)
- Use n-bit Grey Code to represent number
- Concatenate codes for several numbers

■ Why?

- So mutations will be like adding a little noise
- so enough variation of numbers can be searched without REAL variation

Fitness

- Computable function applicable to members of the population, which evaluate the genotypes and returns a NUMBER

■ Tuned precisely to the task domain

- Lots of human intuition as "inductive bias"

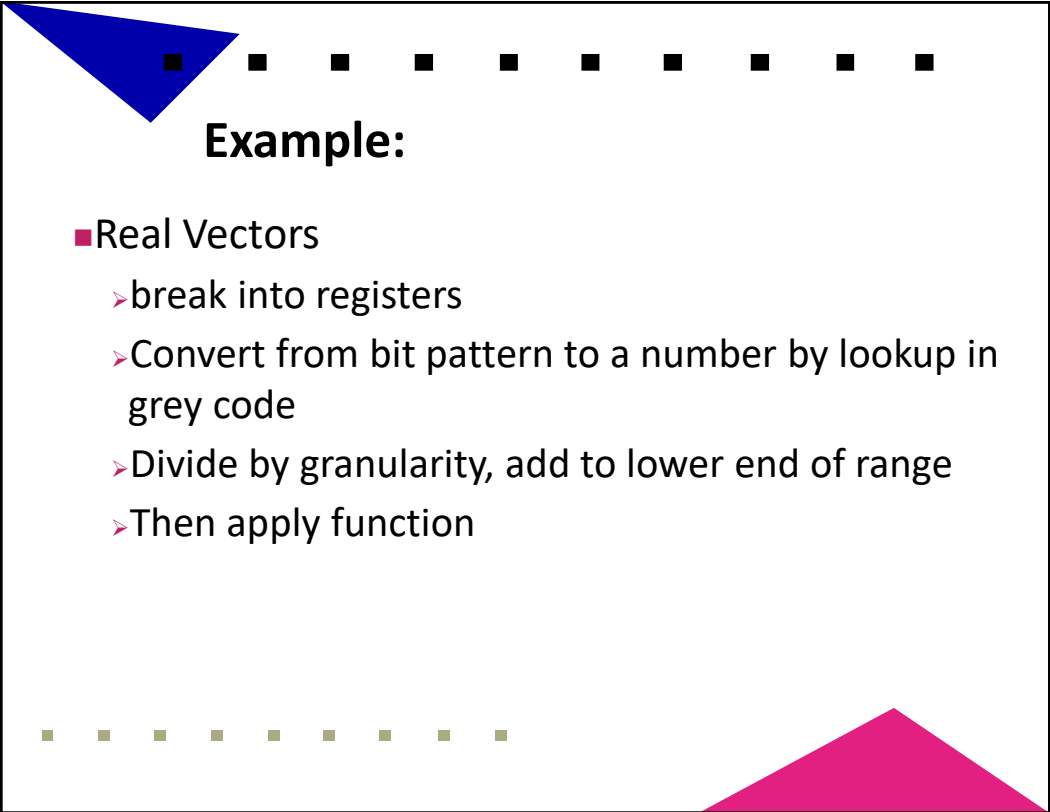
■ Slowest part of the GA paradigm

- Evaluation of a mathematical function
- Calculation over a large set of data
- Building a model from parameters then testing the model
- construction of a machine from the code, and running it



Example:

■ Real Vectors

- break into registers
 - Convert from bit pattern to a number by lookup in grey code
 - Divide by granularity, add to lower end of range
 - Then apply function
- 



Example: 6 bit genotype

■ Fitness function: 25 17 -5 -12 -3 2 times each bit (+ 20)

- 110011 -> 61
- 010101 -> 27
- 100111 -> 32
- 010010 -> 34
- 001101 -> 5
- 101001 -> 42

■ Total over population: 201

■ Average Fitness: 33.5



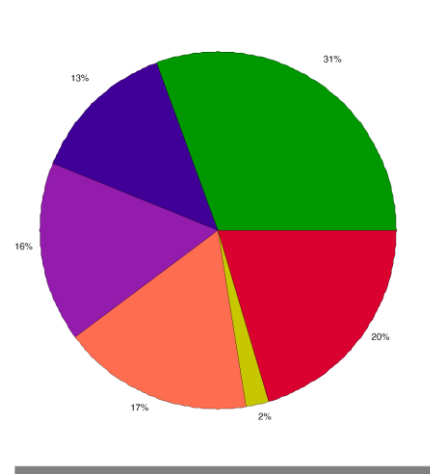
Selection

- Once the members of the population are evaluated, the "fittest" should multiply, while the weakest get diluted or deleted.
- Default: Proportional Selection based on Relative Fitness
- "Roulette Wheel"

Example: Relative Fitness

- Fitness function: 25 17 -5 -12 -3
2 times each bit (+20)

- 110011 -> 61/201 -> 30%
- 010101 -> 27/201 -> 13%
- 100111 -> 32/201 -> 16%
- 010010 -> 34/201 -> 17%
- 001101 -> 5/201 -> 2%
- 101001 -> 42/201 -> 20%

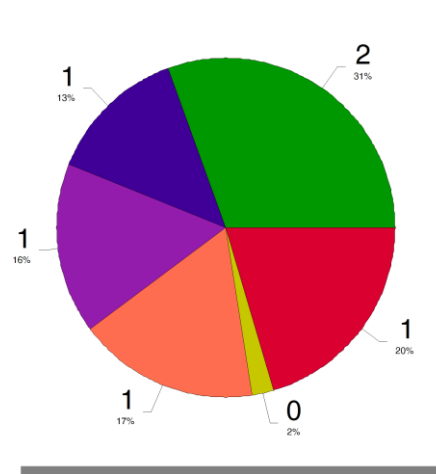


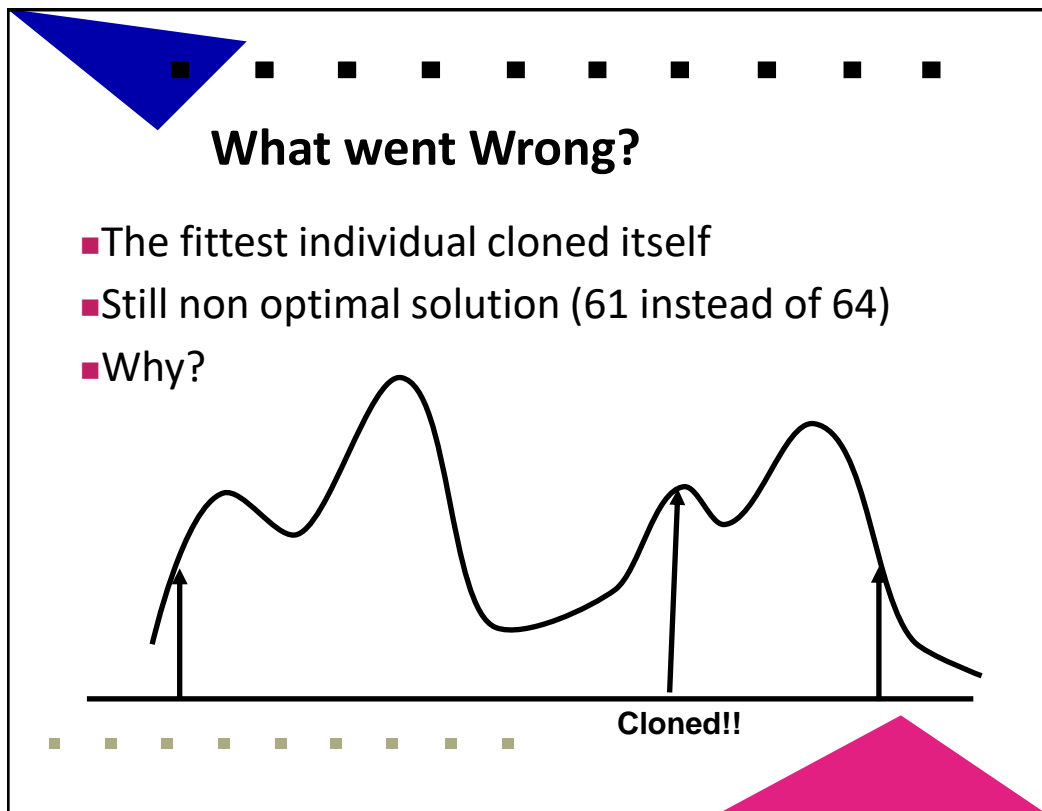
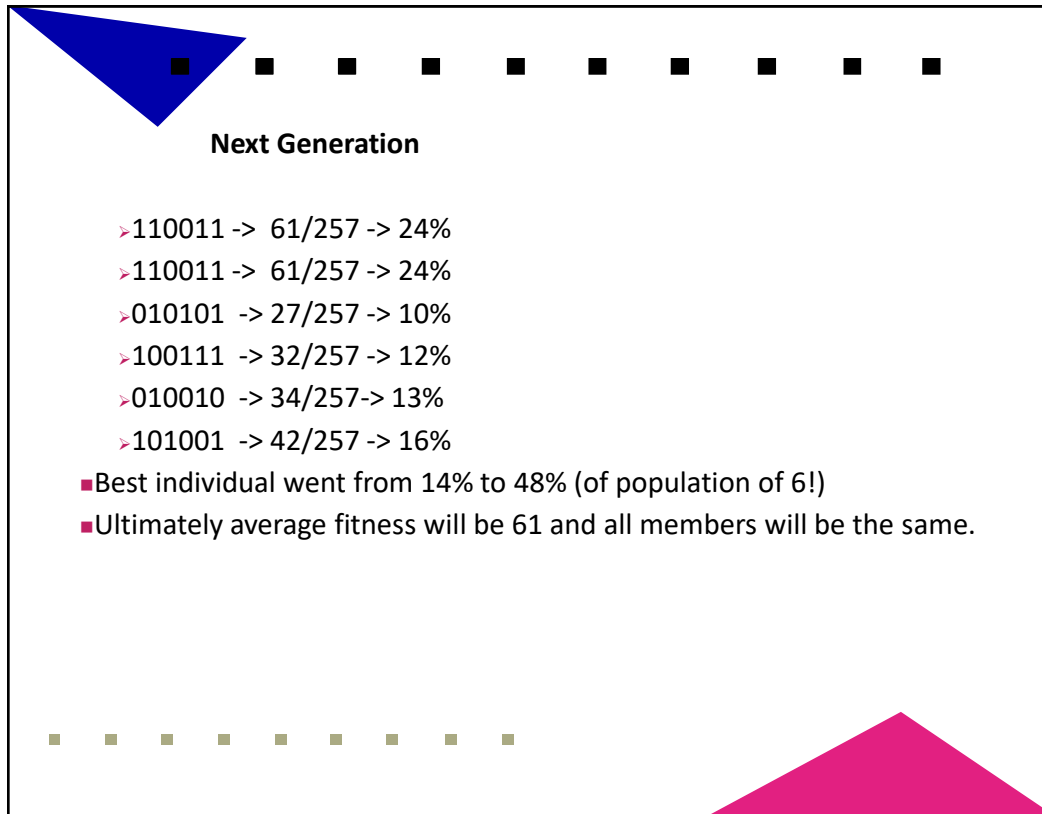
Reproduction

- Next Generation should have higher average fitness if above average members gain and below average members lose
- Default: Create a new population of the same size by selecting members of the current population in integer rounding of roulette wheel
- Variants
 - Synchronous replacement of entire population versus asynchronous, change group through adds and deletes

Example

- 110011 -> 30% -> 2 copies
- 010101 -> 13% -> 1 copy
- 100111 -> 16% -> 1 copy
- 010010 -> 17% -> 1 copy
- 001101 -> 2% -> DEAD
- 101001 -> 20% -> 1 copy
- New Total fitness 257
- Average Fitness: 43








Loss of Diversity

- The act of changing the population by increasing good members and decreasing bad members will increase the average fitness over the population.
- If there is no additional members added to the population, the search will converge to the best initial member (given an absolute fitness function).
 - How to get positive variation?





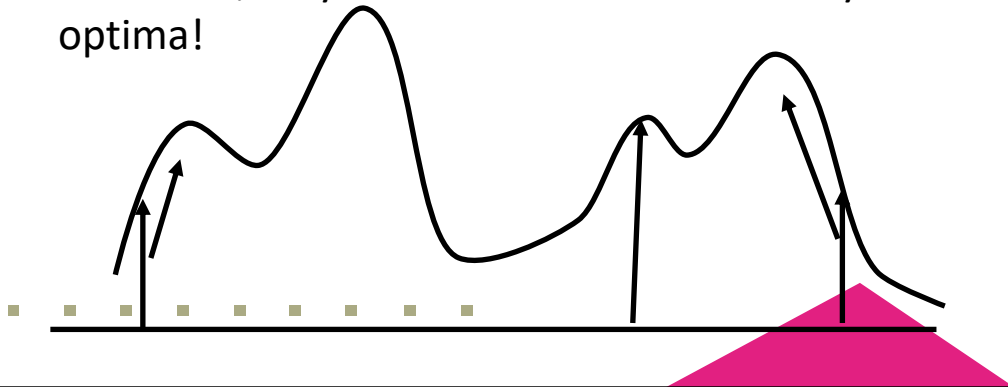
Mutation

- Mutagens randomly applied to members of the population might create new members with higher fitness?
- Mutation consists of minor perturbations to a genotype:
 - flipping a bit
 - adding random noise to a number
 - reversing orbits in a permutation
 - adding/deleting edges in a graph



When does mutation help?

- when it is "small," such as when a gray level code bit changes an infinitesimal input to a continuous function.
- Not often, only when a member has nearby optima!



Would OnlyMutation work?

- a MutationOnly paradigm is like Parallel Hill Climbing
- A little better than hillclimbing with a restart to different initial conditions.
 - or use gaussian mutation (with small chance to make large jump)
- We need a way to expand the search space to not get caught in multiple local maxima

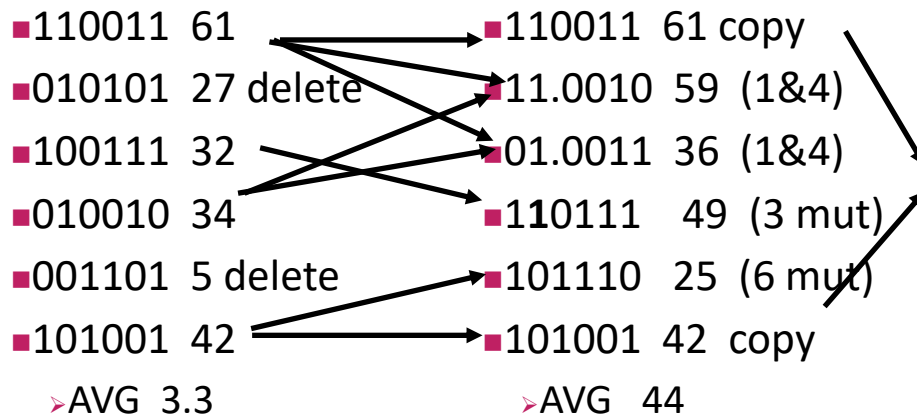


Cross-breeding

- If two parents are good at different things, maybe one of their children will be good at both!
- Default 1 point crossover of bitstrings
 - abcdef $\swarrow \searrow$ abcDEF
 - ABCDEF $\nwarrow \nearrow$ ABCdef
- Variants
 - Uniform crossover, flipping a coin for each bit
 - 2 point crossover ABcdeF
 - Knowledge-based crossover: using knowledge of the task to help make viable children

Example

25 17 -5 -12 -3 2 times each bit (+20)



110011 + 101001 crossed at 4 or 5th position
yields 110001, optimal fitness = 64



PedagoGAcod 1

- (defun pick (lst)
- (nth (random (length lst)) lst))

- (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))))



create pop of bitstrings

- (defun make-rand-member (n)
- (loop for i from 1 to n collect (random 2)))

- (defun make-population (width count)
- (loop for i from 1 to count collect
- (make-rand-member width)))

Mutation and Crossover

```

■ (defvar *mutrate* .1)

■ (defun mutate (e1)
■   (loop for bit in e1 collect (if (< (random 1.0) *mutrate*) (- 1 bit) bit)))

■ (defun crossover (e1 e2)
■   (let ((k (random (length e1))))
■     (n (length e1))
■     (if (zerop (random 2))
■       (append (subseq e1 0 k)(subseq e2 k n))
■       (append (subseq e2 0 k)(subseq e1 k n))))

```

cheap trick to reuse roulette to define next generation

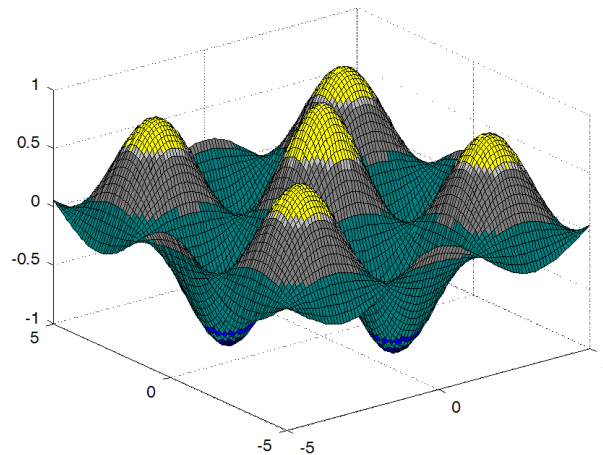
```

■ (defvar *PARAMETERS* '(.8 .1 .1)) ;how much do you cross, mutate,
■   elitism (keep parent as is)?

■ (defun new-population (population fitness)
■   (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))
■         ((pick) parent))))

```


two reals $-5 < x < 5$



■ $z = \cos(x) \cdot \cos(y) \cdot \exp(-\sqrt{x^2 + y^2}/20)$

funner fitness function

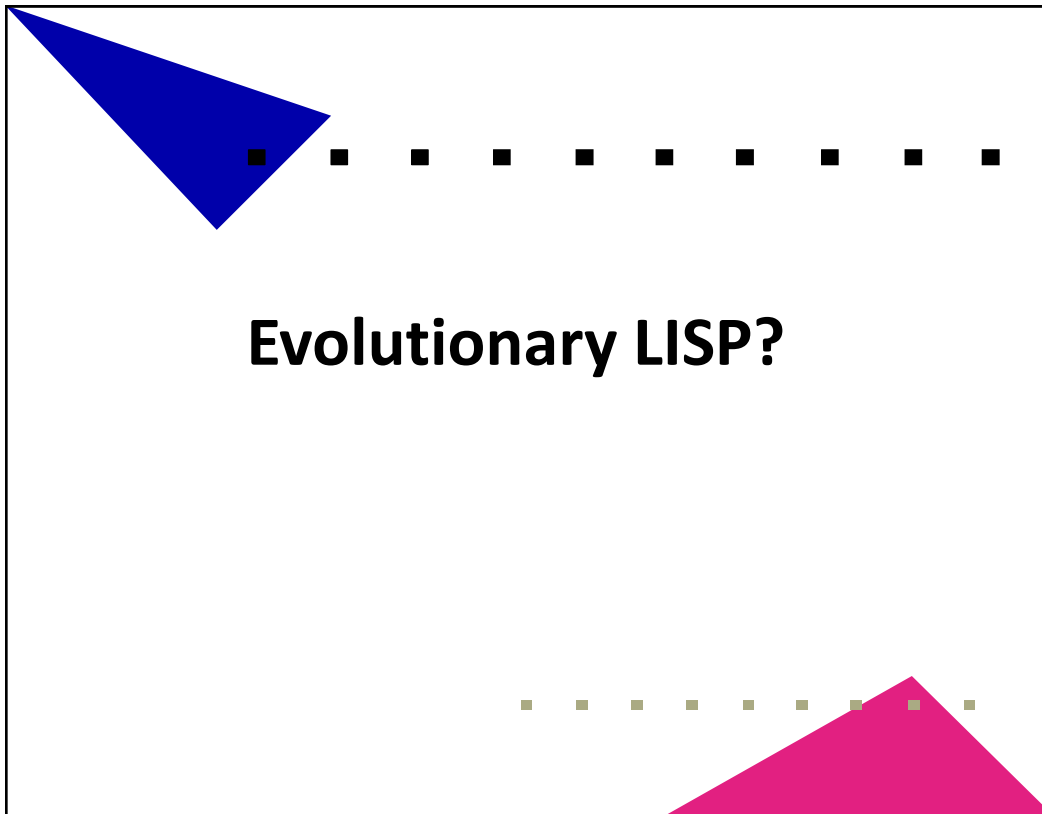
```

■ (defun bin2dec (string)
■   (loop for bit in (reverse string) as i from 0 sum (* bit (expt 2 i))))

■ ;take 2 10-bit strings as 0-1023 --> -5 to 5 -> function

■ (defun funfit (string20)
■   (let ((x (+ -5 (/ (bin2dec (subseq string20 0 10)) 103)))
■         (y (+ -5 (/ (bin2dec (subseq string20 10 20)) 103))))
■     (* (cos x)(cos y)(exp (/ (- (sqrt (+ (expt x 2)(expt y 2)))) 20))))))

```




Dynamics of GA Search

- There are several competing forces which must be coordinated in GA's, each with a positive and negative effect:
 - Proportional Reproduction raises average fitness, but can lead to loss of diversity and convergence of the population.
 - Must have "new ideas" or MAX fitness cannot increase
 - Adding diversity through mutation and crossover can help climb local hills, but too much could lead to a drop in average fitness




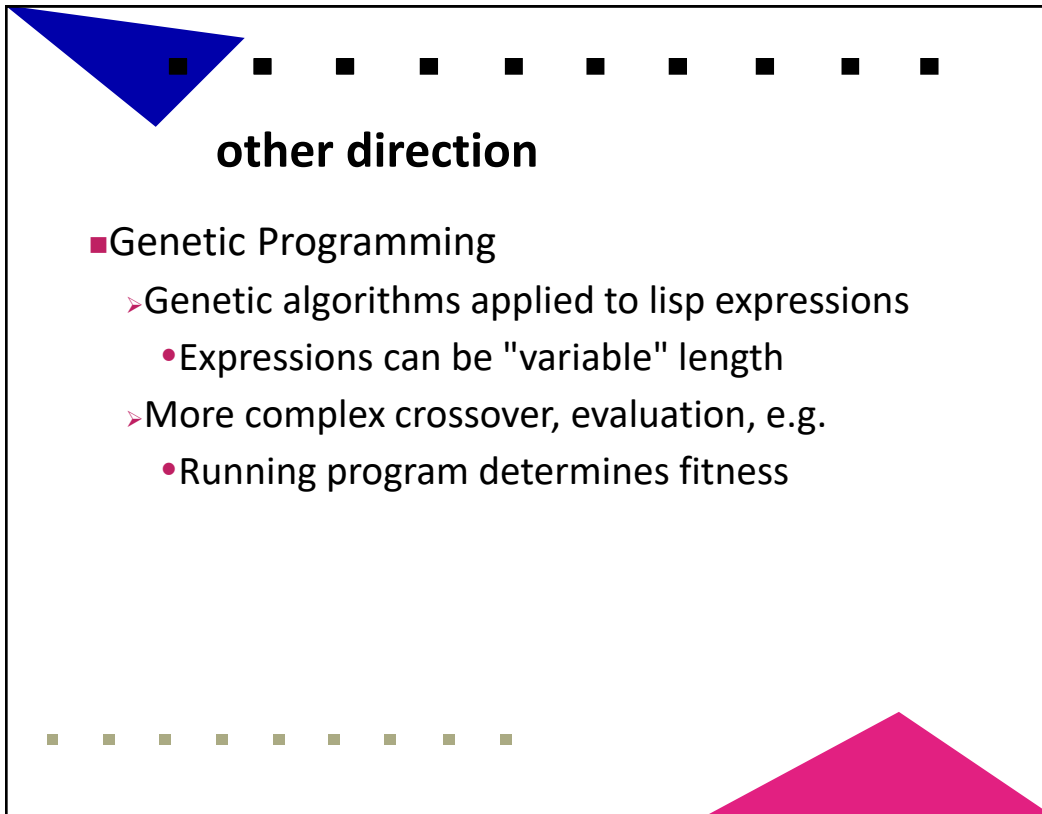
More Lingo

- Loss of Diversity
 - Premature Convergence
 - Elitism
 - keeping some good guys around
 - Steady State
 - Add one at a time, instead of generations
 - Fitness Sharing
 - Crowding
 - hacks to maintain diversity
- 



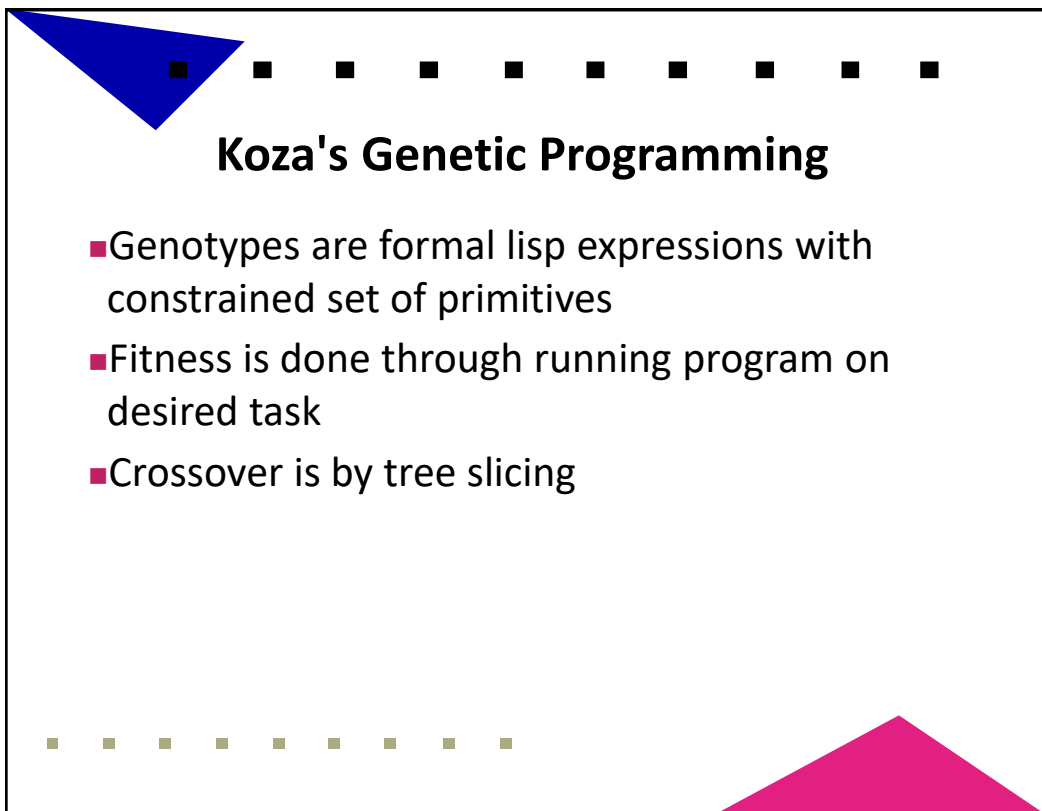
Exploration versus Exploitation

- These forces form a race condition, which need to be balanced by experimenting with parameters such as the population size, mutation and crossover rates, and ultimately by hacking directly on the representations and the functions which perform (asexual) mutation and (sexual) crossover.
 - In a new application, you have to watch the average and maximum fitness, and the diversity to control parameters of search.
 - What does this "black art" mean for our question about "inductive bias"?
- 



other direction

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




similar parts to GA


- (defun pick (lst)
 - (nth (random (length lst)) lst))

 - (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))))
- 




similar parts to GA (*continued*)

- (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)))))
- 



initial function generator

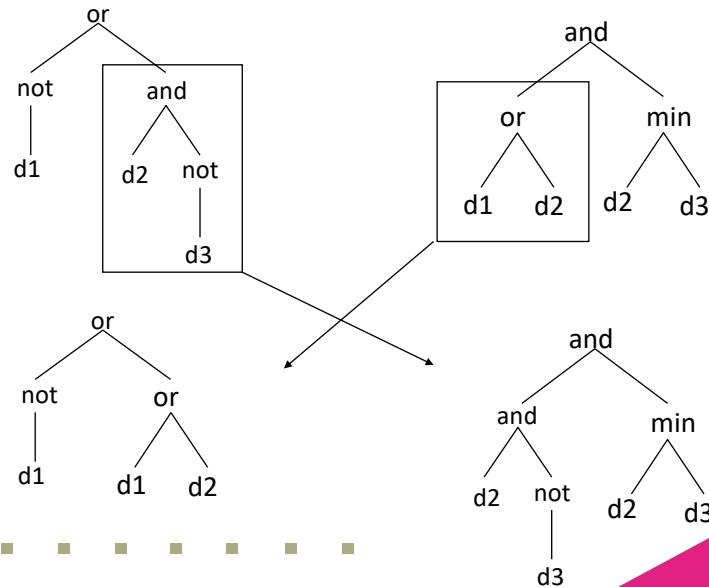
- (defun make-rand-expr (nonterms terms depth)
 - (if (< depth 2)(pick terms)
 - (let ((n (pick nonterms)))
 - (cons (car n)
 - (loop for i from 1 to (cdr n) collect
 - (make-rand-expr nonterms terms (random depth))))))
- (defun make-init-pop (nonterms terms depth n)
 - (loop for i from 1 to n collect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



GP Hacks

- functions which are protected from overflow
 - Divide which checks denominator for 0
 - Square root which uses Absolute value
 - etc
- Initial Function Generator
 - Which biases for diversity
- LIMIT CUTOFF
 - Stop programs from growing and slowing

Crossover Operator



crossover and mutation hack

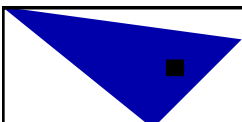
- (defun replace-subexpr (expr subexpr newexpr)
- (flet ((rplace (expr1) (replace-subexpr expr1 subexpr newexpr)))
- (cond ((equal expr subexpr) newexpr)
- ((not (consp expr)) expr)
- (t (cons (car expr) (mapcar #'rplace (cdr expr))))))
- ;cheap crossover to replace one expression by another
- (defun crossover (e1 e2)
- (replace-subexpr e1 (pick-subexpr e1)(pick-subexpr e2)))
- ;cheap mutate is to cross with something novel
- (defun mutate (expr nonterms terms)
- (crossover expr (make-rand-expr nonterms terms 4)))

What is needed

- 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)
 - ACT (side-effecting action)
- Fitness Function
 - How to evaluate an evolved function


gp framework is that of GA

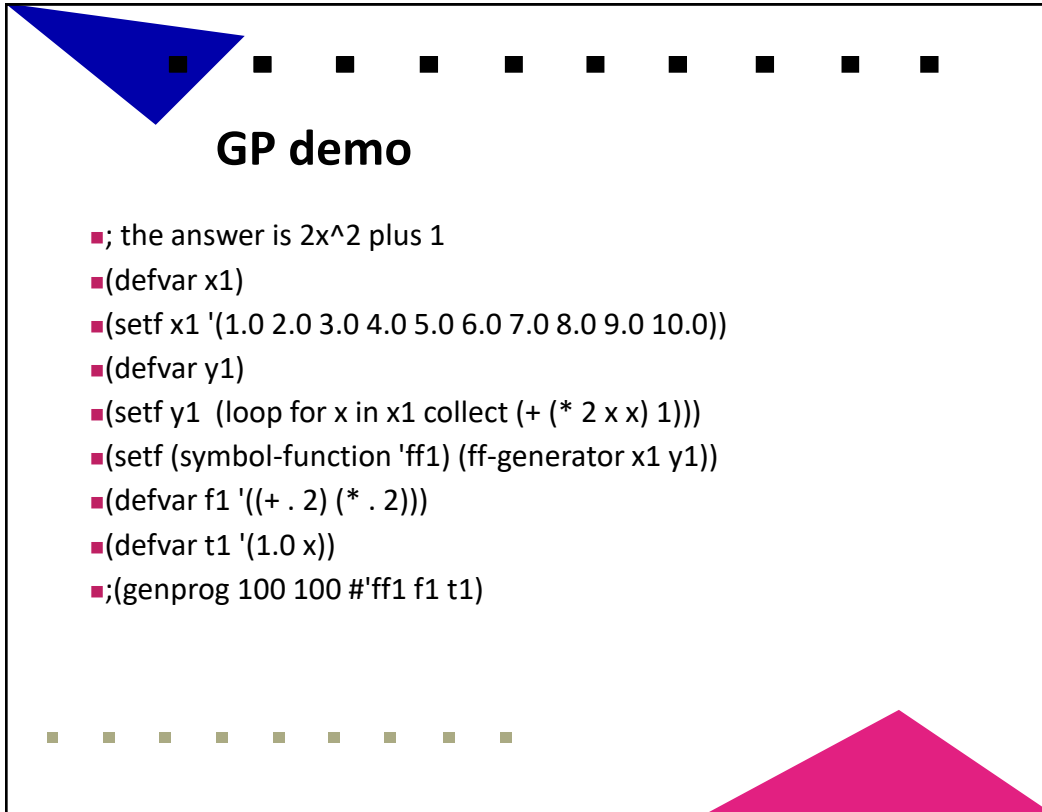
- (defvar *PARAMETERS* '(.8 .1 .1)) ;how much do you cross, mutate, keep?
- (defun new-population (population fitness nonterms terms)
 - (loop for x in population collect
 - (let ((parent (roulette population fitness)))
 - (case (roulette '(cross mutate pick) *PARAMETERS*)
 - ((cross) (crossover parent (pick population)))
 - ((mutate) (mutate parent nonterms terms))
 - ((pick) parent))))
- (defun index (x l)
 - (cond ((or (null l) (equal x (car l))) 0)
 - (t (1+ (index x (cdr l)))))



Things which have been GP'ed

- Control of pendulums
- Logical problems (multiplexor)
- symbolic integration
- induction of sequences
- game learning
- classification
- etc.



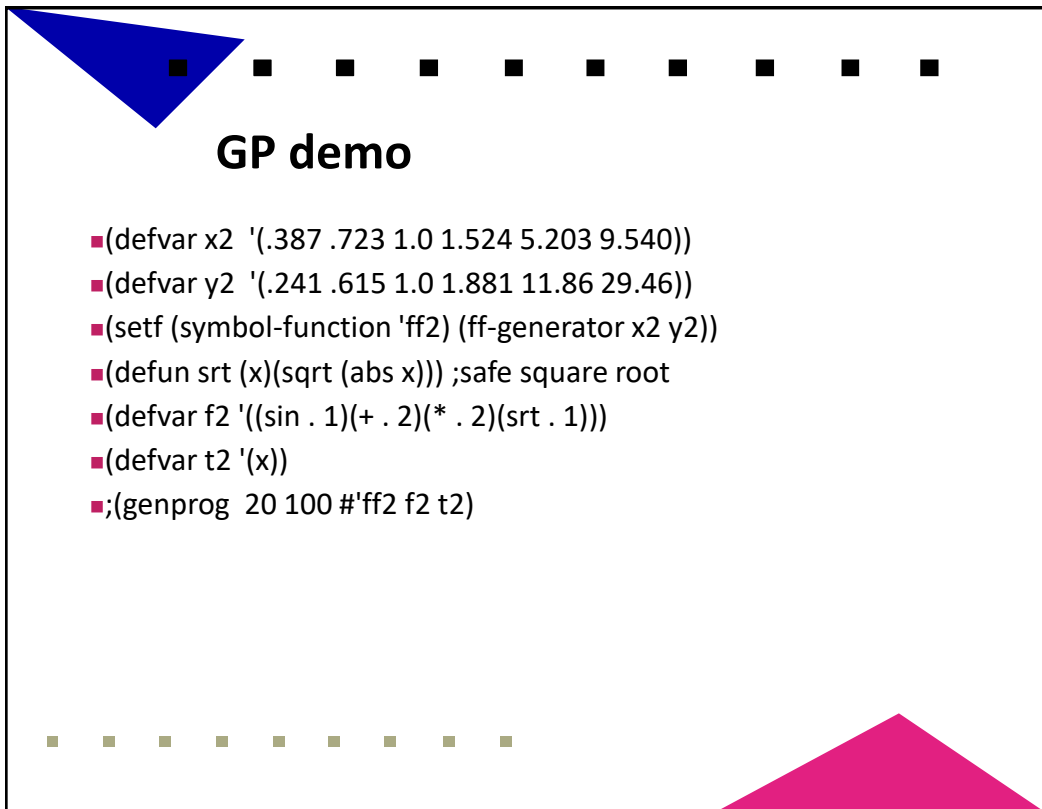


GP demo

```

■; the answer is 2x^2 plus 1
■(defvar x1)
■(setf x1 '(1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0))
■(defvar y1)
■(setf y1 (loop for x in x1 collect (+ (* 2 x x) 1)))
■(setf (symbol-function 'ff1) (ff-generator x1 y1))
■(defvar f1 '((+ . 2) (* . 2)))
■(defvar t1 '(1.0 x))
■;(genprog 100 100 #'ff1 f1 t1)

```



GP demo

```

■(defvar x2 '(.387 .723 1.0 1.524 5.203 9.540))
■(defvar y2 '(.241 .615 1.0 1.881 11.86 29.46))
■(setf (symbol-function 'ff2) (ff-generator x2 y2))
■(defun srt (x)(sqrt (abs x))) ;safe square root
■(defvar f2 '((sin . 1)(+ . 2)(* . 2)(srt . 1)))
■(defvar t2 '(x))
■;(genprog 20 100 #'ff2 f2 t2)

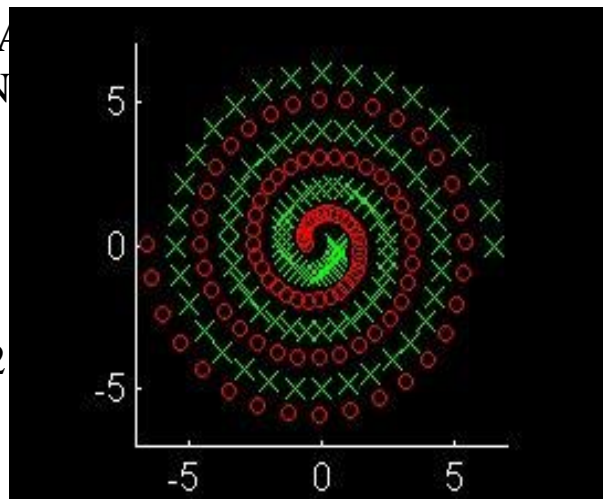
```

auxiliary functions

- ;;creates a fitness function(x) which sums abs difference between y
- ;; and popmember(x) for a vector of x and y's
- (defun ff-generator (x-data y-data)
- #'(lambda (expr)
- (loop for i from 1 to (length x-data) do
- (setf x (nth (- i 1) x-data))
- (setf y (nth (- i 1) y-data))
- sum (abs (- y (eval expr))))))
- (defun normalize (fitness) ; seeking 0 as the best fitness
- (let* ((maxfit (loop for x in fitness maximize x))
- (relfit (loop for x in fitness collect (+ 1 (- maxfit x))))
- (sumrel (* 1.0 (loop for x in relfit sum x))))
- (loop for y in relfit collect (/ y sumrel))))

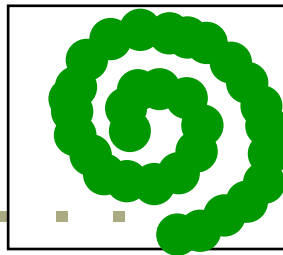
Intertwined Spirals Problem

- Originated by A. J. C. Cook and J. D. Schaffer and J. D. Schaffer and J. D. Schaffer as NN challenge
- Non-convex Classification Problem
- 194 points in 2 classes



Spiral Problem

- NN Solutions by: Fahlman, Witbrock, Grossberg, Fritzke, etc.
- Ungainly descriptions which generalize poorly



GP approach to Spirals

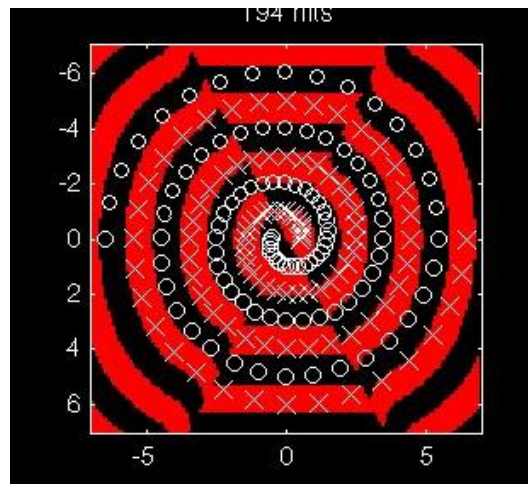
- Koza92
 - SIN, COS, *, /, +, -
 - X , Y , R
- Juille & Pollack 1995
 - parallel processing implementation
 - niche preservation

Results and Performance

■ Problem: Intertwined Spirals

➤ $(\sin (/ (\text{iflte } (- (- (- (* A A) (\sin (/ (\text{iflte } -0.52381 B$
 $(\sin -0.33333) -0.33333) -0.33333))) (* B B)) (/ A (/$
 $-0.33333 A))) -0.80952 B (\sin (/ (/ A (- (\cos(\sin (*$
 $(\cos (\sin -0.52381))(/ B (/ A (- (\cos -0.33333$
 $0.04762)))))) 0.04762)) (\sin (\sin -0.33333)))) -$
 $0.33333))$

Image of co-evolved Spiral





Discuss the Question!

- Are genetic algorithms, and other forms of search and learning based on the evolution metaphor merely another weak method with a creative name?
 - Or are there deep scientific secrets that biology has discovered in its own search which we can turn into a powerful technology, more powerful than the strong method of knowledge engineering?
- 