Genetic Programming

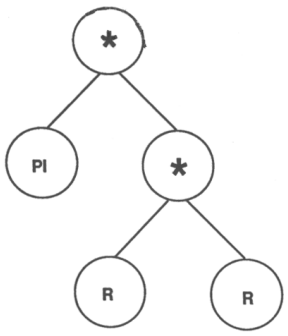
(John Koza, 1992)

- "Genetic Programming": Evolve populations of programs (rather than bit strings).

Any computer program can be expressed as a "parse tree":

(* PI (* R R))

```
PROGRAM AREA-OF-CIRCLE
      R = 45
      PI = 3.1415
      AREA = PI * (R * R)
      PRINT AREA
END AREA-OF-CIRCLE
```

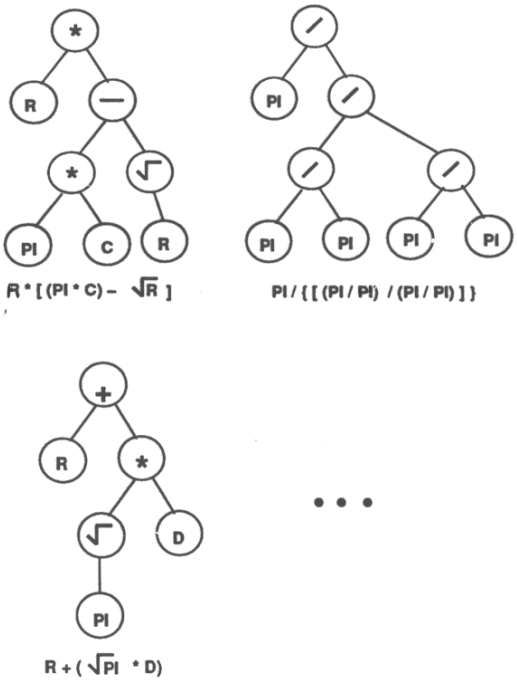


Koza's genetic programming algorithm:

1. Choose a set of functions and terminals for the program, e.g.,

{+, -, *, /, *sqrt, sin, cos, abs, pow,* R, PI, D, C, rand()}

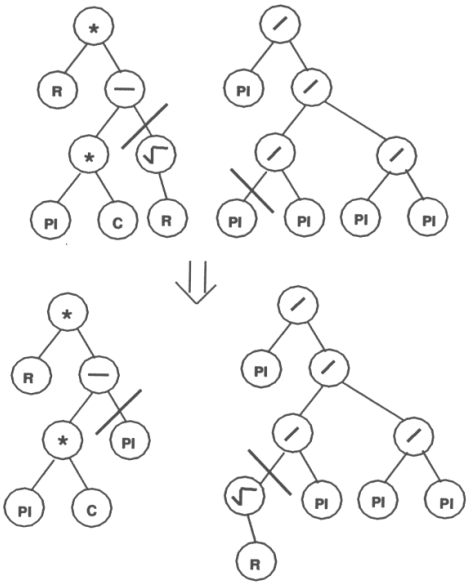
2. Generate an initial population of random programs (trees), each up to some maximum depth.



Koza's genetic programming algorithm:

3. Run the GA:

- **Fitness:** Run each program on "training data". Fitness is how many training cases the program gets right (or how close it gets to correct answer).
- **Selection:** Select parents probabilistically, based on fitness.
- **Crossover:** Exchange subtrees of parents.
- **Mutation:** Replace subtree with a random tree.



To represent a decision tree, create functions representing the possible values of each attribute.

$F_{Outlook}(a_1, a_2, a_3)$

$= a_1$ if *Outlook = Sunny*

a_2 if *Outlook = Overcast*

a_3 if *Outlook = Rain*

$F_{Humidity}(a_1, a_2)$

$= a_1$ if *Humidity = High*

a_2 if *Humidity = Normal*

$F_{Wind}(a_1, a_2)$

$= a_1$ if *Wind = Strong*

a_2 if *Wind = Weak*

$F_{Temp}(a_1, a_2, a_3)$

$= a_1$ if *Temp = Hot*

a_2 if *Temp = Mild*

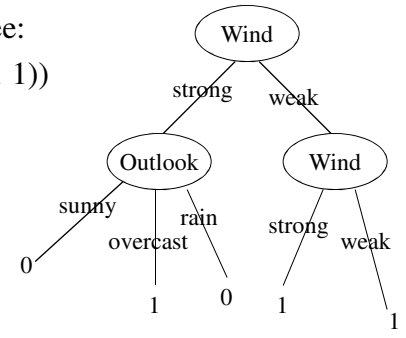
a_3 if *Temp = Cool*

- Function set: $F_{Outlook}$, $F_{Humidity}$, F_{Wind}, F_{Temp}

  3 args, 2 args, 2 args, 3 args

- Terminal set: 0, 1

- Example of random decision tree:
  $(F_{Wind}\ (F_{Outlook}\ 0\ 1\ 0)\ (F_{Wind}\ 1\ 1))$



Algorithm for evolving decision trees

1. Generate initial population of *N* random decision trees
   - Parameter: max size of tree
2. Calculate fitness of each tree in population
   - Fraction of correct classifications on training set
3. Rank population by fitness, select *N/2* pairs of parents probabilistically according to rank.
4. Cross over parents to create children. Mutate children.
   - Parameters: crossover and mutation probabilities
5. Put children in new population. Go to step 2.

- Has been shown in some studies to be as good or better than ID3 (and descendents) for decision tree induction. (Though not always...)
- However, no theoretical or thorough empirical comparison has been done yet.

Using GP to Construct Features for Machine Learning Algorithms (Krawiec, 2002)

- Learning from examples paradigm
  - "Classifier" to be trained
    - e.g., neural network, decision tree
  - Training examples
    - Vectors of "features", plus classification for each vector
  - Test cases
    - tests generalization abilities
    - e.g., separate test data set, or can use cross validation

Example: Predicting stock prices

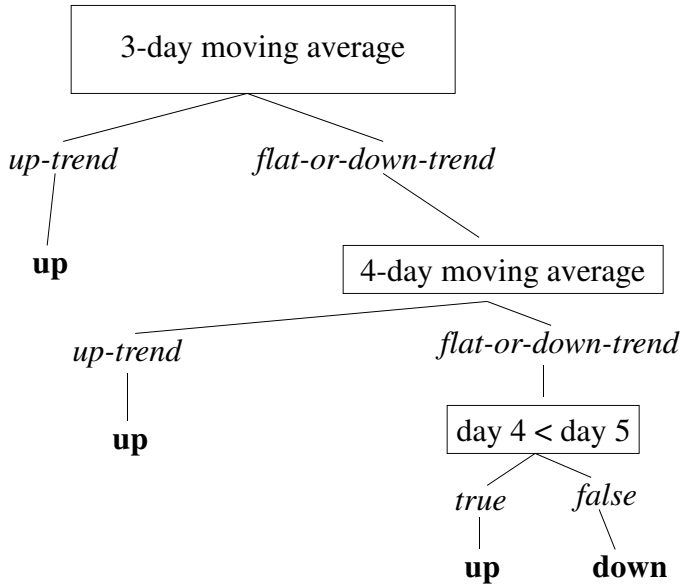
Each training example is the closing price of stock for a five-day window, plus the class up or down (a prediction of what will happen on the sixth day).

(20.2, 20.5, 21.1, 22.6, 20.0, down)

(25.1, 25.4, 25.6, 26.2, 27.2, up)

etc.

Simple decision tree



Question of this paper:

How to discover these higher-level functions of features?

Approach

- Evolve new features by combining functions of old ones in various ways.
- Each individual in GP population represents a vector of feature definitions (here fixed at 4), each represented as a parse tree.
  - Function set: +, -, *, % (protected division), log, LT, GT, EQ, IF, EQap, rand_int()
  - Terminal set: x_1, x_2, ..., x_n (original features)
- Give example of individual

Approach

- Fitness of an individual:
  - For each training example, calculate the four new features.
  - Use new training set to train decision tree (using cross validation).
  - Fitness is resulting average accuracy of classification.
- Standard GP selection, crossover, and mutation
- Population size: 500, number of generations: 50
- Results: Significant improvement in generalization ability for decision trees on several problems

Advanced GP Techniques

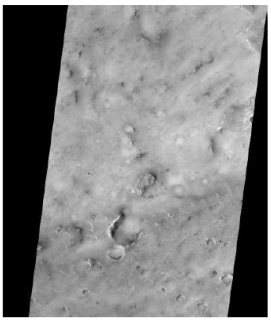
- Encapsulation
- Automatic definition of functions
- Incorporating development (genotype → development → phenotype)

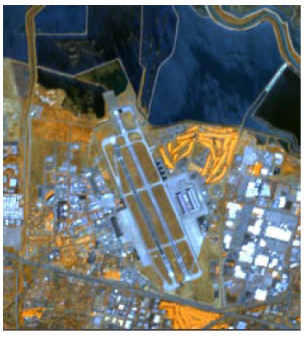
Evolving image-processing algorithms

- GENIE project (GENetic Image Exploitation), Los Alamos National Laboratory

Brumby, Harvey, Perkins, Porter, Bloch, Szymanski, et al.

Consider a large library of multi-spectral photographic images from satellites and high-flying airplanes.





- How to automatically find images containing desired features?

  E.g., water, clouds, craters on Mars, forest fire damage, vegetation, urban areas, roads, airports, bridges, beaches,...

Special-purpose algorithms can (possibly) be developed for any particular feature, but goal of GENIE project is to have a general-purpose system:

A user can specify *any* feature (via training examples), and system will quickly find images containing that feature.

Some possible applications:

Remote sensing

Ecological and environmental modeling
planetary image analysis

Medical imaging

Image database search

Training Examples for GENIE

- Input images consist of ~10-100 spectral planes.
- User brings up training image(s) in GUI
- User paints examples of “true” and “false” pixels.

From Brumby et al., 2000

Figure 3. Training scene is taken from an AVIRIS overflight of NASA Moffet Field Air Station.

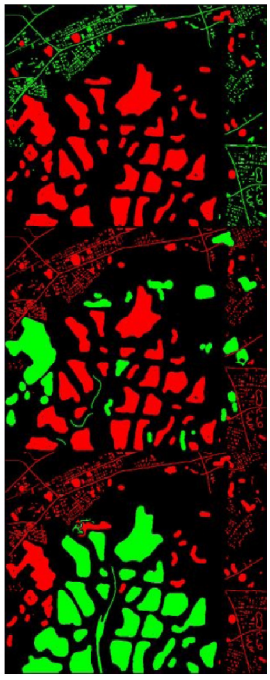
From Brumby et al., 2000

Human created training images

Green = "feature"

Red = "non-feature"

Black = "not classified"

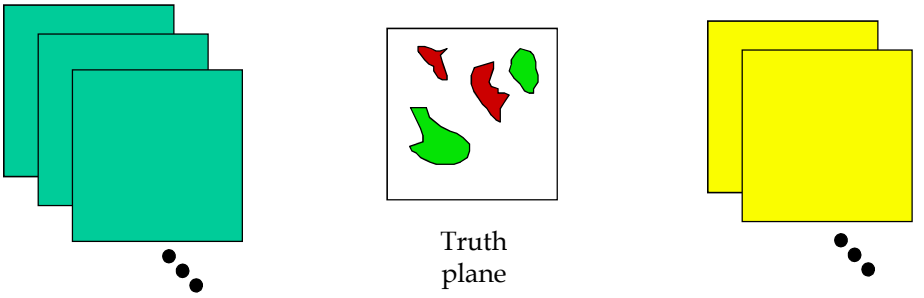


Roads

Vegetation

Water

Data and Feature Planes



Data planes
(spectral channels)

Truth
plane

"Feature" planes

What GENIE Evolves

- GENIE evolves an image processing algorithm that operates on data planes to produce a new set of *feature planes.*
- New feature planes are added to data planes to train a conventional classifier ("Fisher Linear Discriminant")

GENIE's Gene's

- *Genes* are elementary image-processing operations that read from data planes (spectral channels) or feature planes and write to feature planes

  e.g.,

  addp(d1, d2, s2)
  linear_comb(s2, s1, s1, 0.9)
  erode(s1, s1, 3, 1)

- Elementary operators include spectral, morphological, logical, and thresholding operators

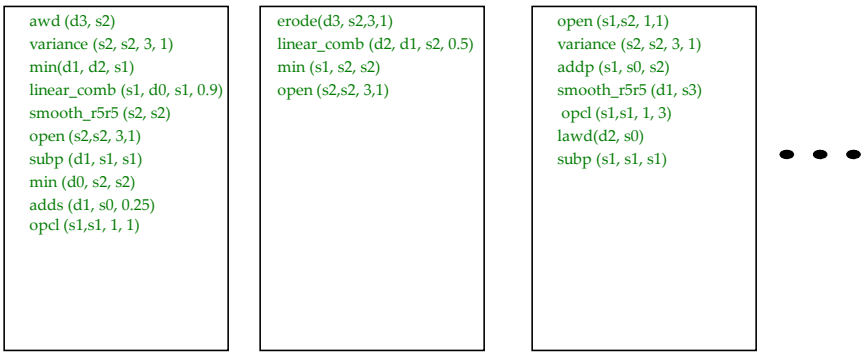
GENIE's Chromosomes

- *Chromosomes* are algorithms made up of genes

```
lawd (d3, s2)                    ; Law's textural operator
variance (s2, s2, 3, 1)          ; local variance in circular neighb.
min(d1, d2, s1)                  ; min at each pixel location
linear_comb (s1, d0, s1, 0.9)
smooth_r5r5 (s2, s2)             ; textural operator r5r5
open (s2,s2, 3,1)                ; erode, then dilate
subp (d1, s1, s1)                ; subtract planes
min (d0, s2, s2)
adds (d1, s0, 0.25)              ; add scalar
opcl (s1,s1, 1, 1)               ; erode, dilate, dilate, erode
h_basin (s0, s0, 18)             ; regional minima
linear_comb (s0, d9, s0, 0.2)
linear_comb (d4, s2, s2, 0.9)
addp (d6, s0, s0)                ; add planes
```

GENIE's Population

- *Initial population* is a set of 20-50 randomly generated chromosomes:



GENIE's Fitness

- *Fitness:*
  - Run chromosome on image data to produce feature planes.
  - Find linear combination of planes that maximizes spectral separation between labeled "true" and "false" pixels.
  - Threshold resulting plane to get binary image (each pixel is 1 or 0):
    - 1 = feature, 0= non-feature
    - Use threshold that will maximize fitness

GENIE's Fitness

– Fitness:

$$F = 500(R_d + (1 - R_f))$$

where

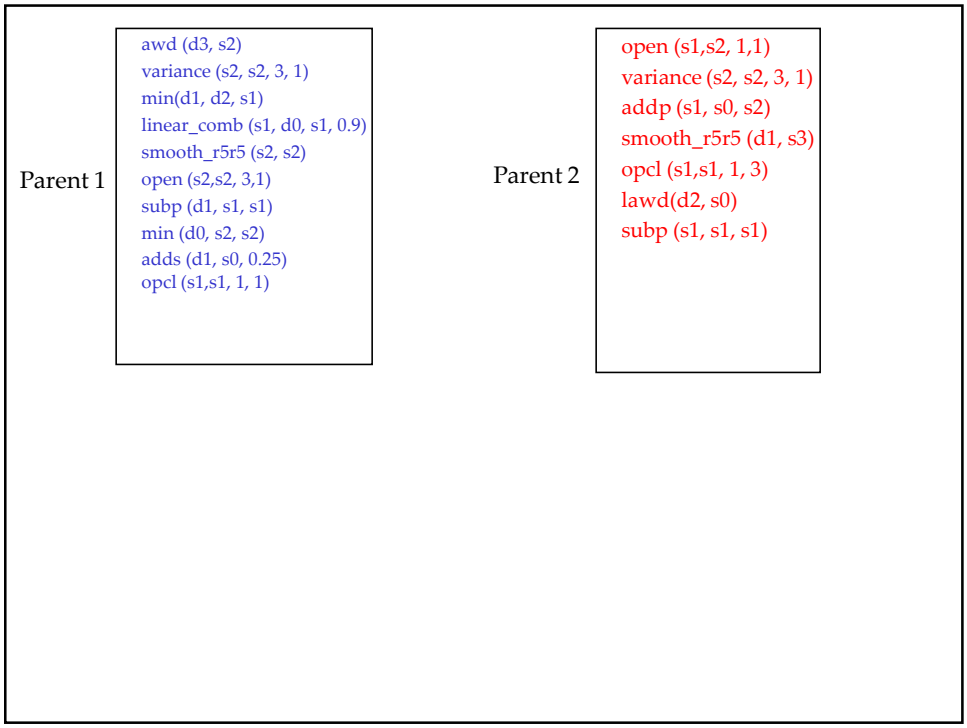
R_d is **detection rate**: fraction of "feature" pixels (green) classified correctly

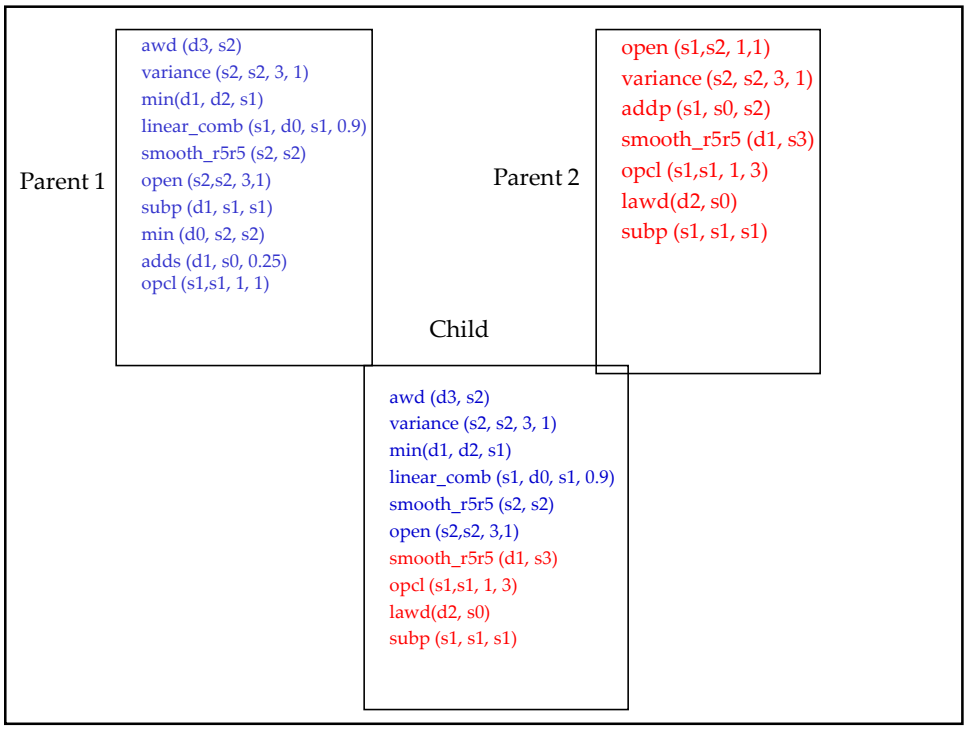
R_f is **false alarm rate**: fraction of non-feature pixels (red) classified incorrectly

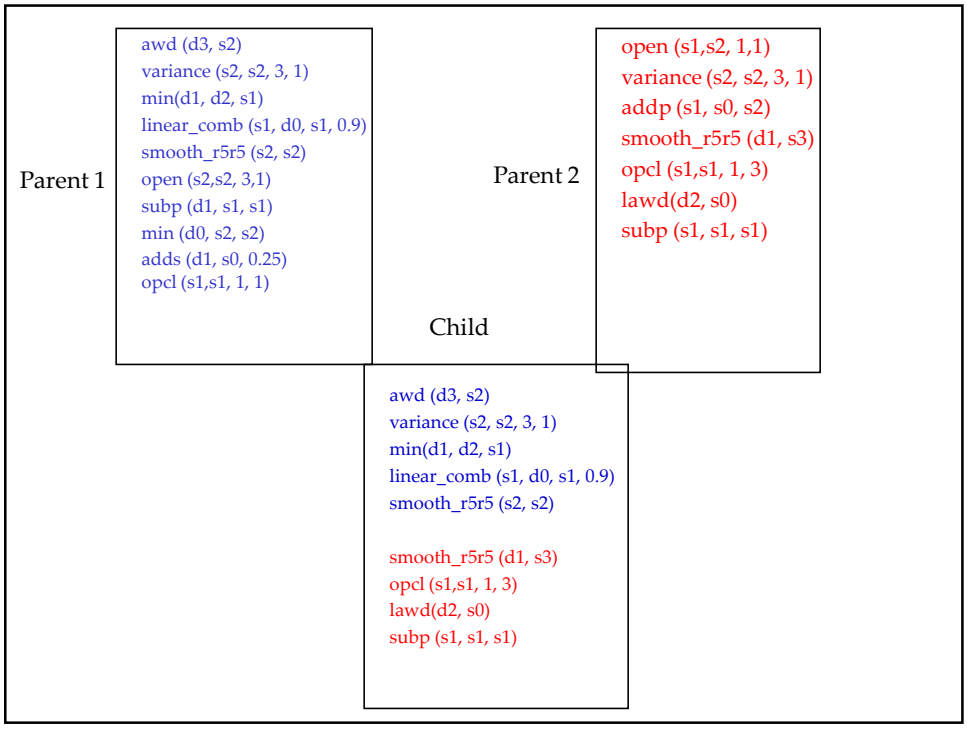
F=1000 is perfect fitness.

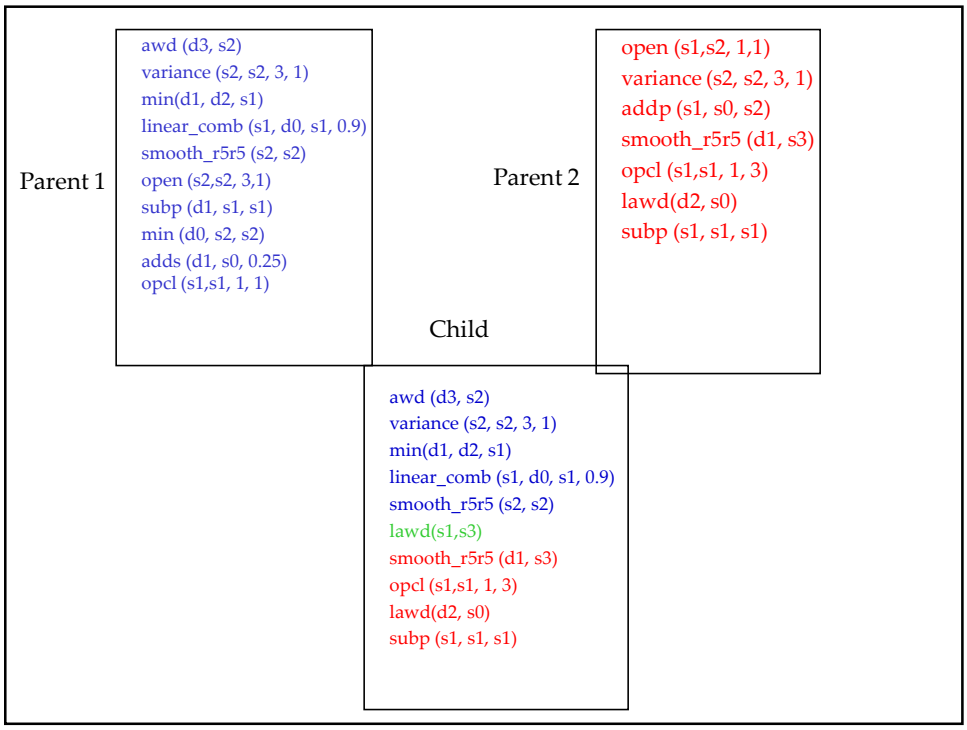
The GA

- *Selection*: Selection some number of best in population
- *Crossover*: Recombine different genes from two different parents
- *Mutation*: Randomly change a chosen gene


Parent 1
awd (d3, s2)
variance (s2, s2, 3, 1)
min(d1, d2, s1)
linear_comb (s1, d0, s1, 0.9)
smooth_r5r5 (s2, s2)
open (s2,s2, 3,1)
subp (d1, s1, s1)
min (d0, s2, s2)
adds (d1, s0, 0.25)
opcl (s1,s1, 1, 1)
Parent 2
open (s1,s2, 1,1)
variance (s2, s2, 3, 1)
addp (s1, s0, s2)
smooth_r5r5 (d1, s3)
opcl (s1,s1, 1, 3)
lawd(d2, s0)
subp (s1, s1, s1)


Parent 1
awd (d3, s2)
variance (s2, s2, 3, 1)
min(d1, d2, s1)
linear_comb (s1, d0, s1, 0.9)
smooth_r5r5 (s2, s2)
open (s2,s2, 3,1)
subp (d1, s1, s1)
min (d0, s2, s2)
adds (d1, s0, 0.25)
opcl (s1,s1, 1, 1)
Parent 2
open (s1,s2, 1,1)
variance (s2, s2, 3, 1)
addp (s1, s0, s2)
smooth_r5r5 (d1, s3)
opcl (s1,s1, 1, 3)
lawd(d2, s0)
subp (s1, s1, s1)
Child
awd (d3, s2)
variance (s2, s2, 3, 1)
min(d1, d2, s1)
linear_comb (s1, d0, s1, 0.9)
smooth_r5r5 (s2, s2)
open (s2,s2, 3,1)
smooth_r5r5 (d1, s3)
opcl (s1,s1, 1, 3)
lawd(d2, s0)
subp (s1, s1, s1)


Parent 1
awd (d3, s2)
variance (s2, s2, 3, 1)
min(d1, d2, s1)
linear_comb (s1, d0, s1, 0.9)
smooth_r5r5 (s2, s2)
open (s2,s2, 3,1)
subp (d1, s1, s1)
min (d0, s2, s2)
adds (d1, s0, 0.25)
opcl (s1,s1, 1, 1)
Parent 2
open (s1,s2, 1,1)
variance (s2, s2, 3, 1)
addp (s1, s0, s2)
smooth_r5r5 (d1, s3)
opcl (s1,s1, 1, 3)
lawd(d2, s0)
subp (s1, s1, s1)
Child
awd (d3, s2)
variance (s2, s2, 3, 1)
min(d1, d2, s1)
linear_comb (s1, d0, s1, 0.9)
smooth_r5r5 (s2, s2)
smooth_r5r5 (d1, s3)
opcl (s1,s1, 1, 3)
lawd(d2, s0)
subp (s1, s1, s1)


Parent 1
awd (d3, s2)
variance (s2, s2, 3, 1)
min(d1, d2, s1)
linear_comb (s1, d0, s1, 0.9)
smooth_r5r5 (s2, s2)
open (s2,s2, 3,1)
subp (d1, s1, s1)
min (d0, s2, s2)
adds (d1, s0, 0.25)
opcl (s1,s1, 1, 1)
Parent 2
open (s1,s2, 1,1)
variance (s2, s2, 3, 1)
addp (s1, s0, s2)
smooth_r5r5 (d1, s3)
opcl (s1,s1, 1, 3)
lawd(d2, s0)
subp (s1, s1, s1)
Child
awd (d3, s2)
variance (s2, s2, 3, 1)
min(d1, d2, s1)
linear_comb (s1, d0, s1, 0.9)
smooth_r5r5 (s2, s2)
lawd(s1,s3)
smooth_r5r5 (d1, s3)
opcl (s1,s1, 1, 3)
lawd(d2, s0)
subp (s1, s1, s1)

- The GA is run until acceptable performance on the training examples is found.
- Generalization performance is evaluated, and user then creates new training example, based on generalization performance.

Some Results

From Brumby et al., 2000

Figure 3. Training scene is taken from an AVIRIS overflight of NASA Moffet Field Air Station.

From Brumby et al., 2000

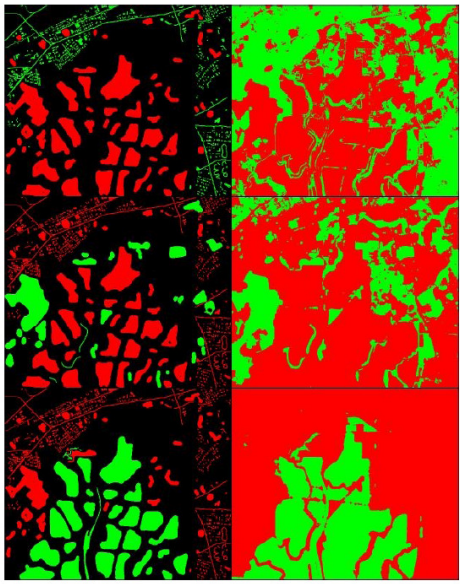


Figure 4. Analyst supplied training data (on left) and evolved tool output (on right) for, from top: roads/buildings, vegetation, water. "True" pixels appear light grey, "false" pixels appear as medium grey. Unclassified pixels appear black.

From Brumby et al., 2000

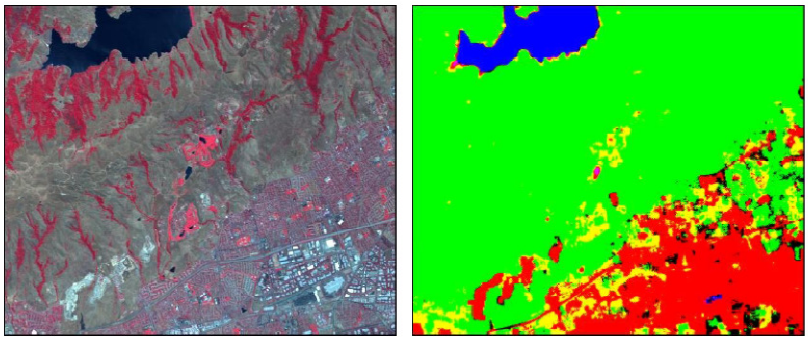
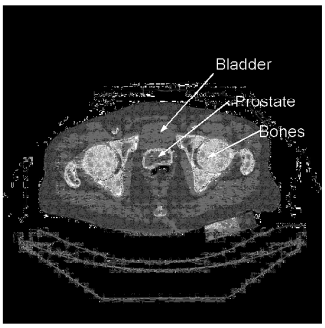


Figure 6. Result of application of the evolved tools to a new scene from the same AVIRIS campaign.

Recent work by Ghosh & Mitchell

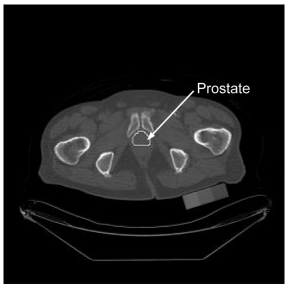
- Combine GENIE's texture-learning ability with genetic shape-learning algorithm.
- Application: Find prostate in CAT scans of pelvic region

From Ghosh & Mitchell, 2006

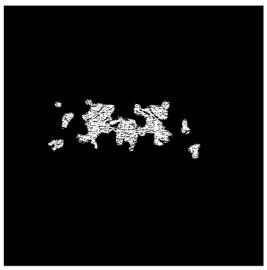


Training image, contour drawn by radiologist

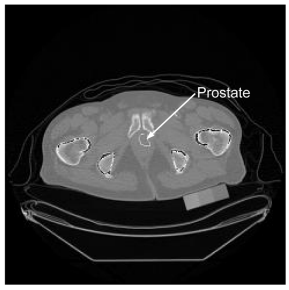
(Prostate pixels painted green for GENIE input; samples of non-prostate pixels painted red)



Test image, with hand-segmentation by a radiologist



Results from GENIE alone



Results from our hybrid algorithm

Genetic Art and Computer Graphics (Karl Sims, 1993)

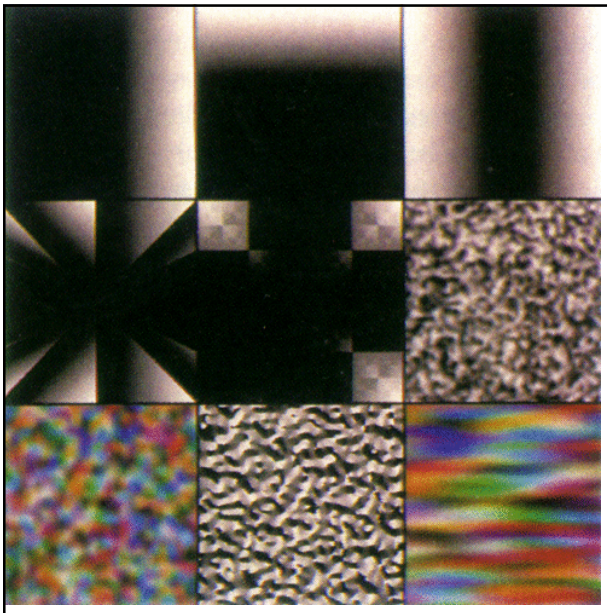
- See http://www.hedweb.com/biomorph/biomorph.htm
- GA individuals: equations that generate a color for each pixel coordinate
- Function set:

  +, -, /, mod, round, min, max, abs, expt, log, and, or, xor, sin, cos, atan, if, dissolve, hsv-to-rgb, vector, transform-vector, bw-noise, color-noise, warped-bw-noise, warped-color-noise, blur, band-pass, grade-mag, grad-dir, bump, ifs, warped-ifs, warp-abs, warp-rel, warp-by-grad

- Terminal set:

  X, Y, rand_scalar(), rand_vector(n)

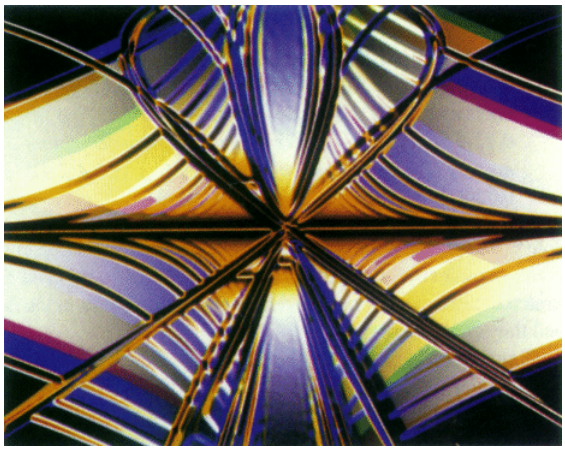
- Each function returns an image (an array of pixel colors)
- Could add a third dimension *Z* to allow volume, and a fourth dimension *T* to allow time (animations)



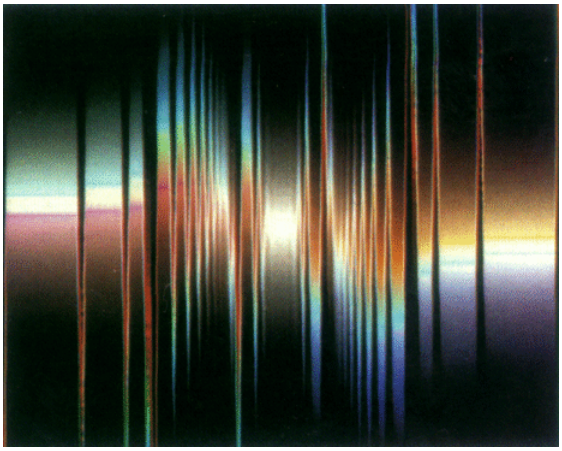
Left to right, top to bottom:

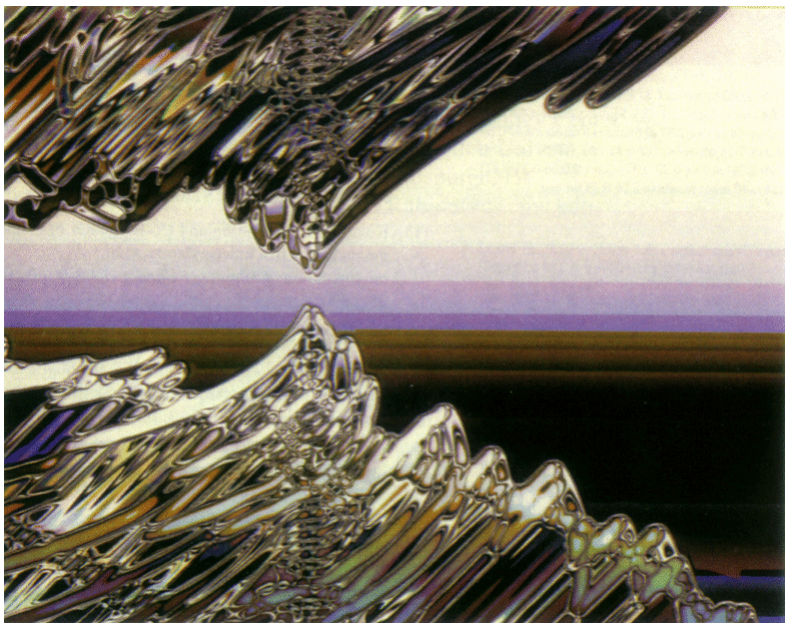
a. *Y*
b. *X*
c. *(abs X)*
d. *(mod X (abs Y))*
e. *(and X Y)*
f. *(bw-noise .2 2)*
g. *(color-noise .1 2)*
h. *(grad-direction (bw-noise .15 2) 0 0)*
i. *(warped-color-noise (* X .2) Y .1 2)*

Some Results



```
(round (log (+ y
(color-grad (round
(+ (abs (round (log
(+ y (color-grad
(round (+ y (log
(invert y) 15.5)) x)
3.1 1.86 #(0.95 0.7
0.59) 1.35)) 0.19)
x)) (log (invert y)
15.5)) x) 3.1 1.9
#(0.95 0.7 0.35)
1.35)) 0.19) x)
```



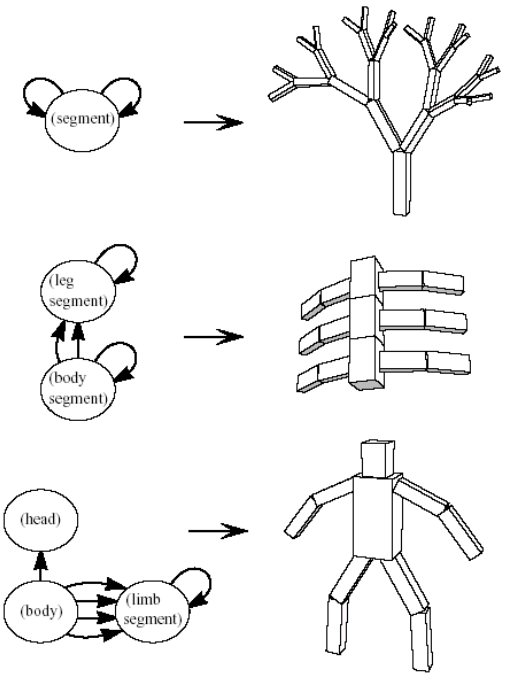


Evolving Virtual Creatures
Sims, 1994

- Goal is to evolve life-like simulated “creatures”.
- Simultaneously evolve morphology and control systems of simulated creatures that can “walk”, swim, jump, follow light.
- Related to Dawkins’ Blind Watchmaker program, which evolved morphologies.
- Environment is detailed simulation of physics.

Creature morphology

- Morphology developed from directed graph of nodes and links.
- Each node of the graph: body part and neural network controlling that body part



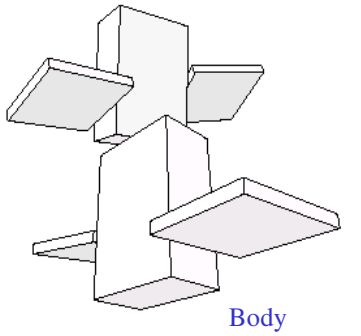
Genotype graphs and corresponding morphologies

Creature control

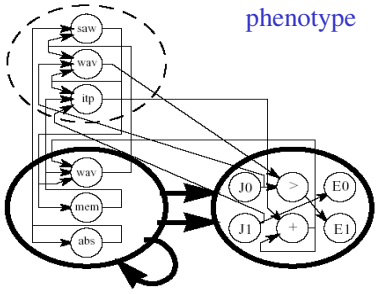
- Neural network controlling each body part:
  - Function that inputs sensor values and outputs effector values (forces or torques at joints)
- Sensors:
  - Joint-angle sensors: returns value of angle
  - Contact sensors: 1.0 if contact is made (with anything), -1.0 if no contact
  - Photosensors: return coordinates of light source direction relative to orientation of part

Creature control

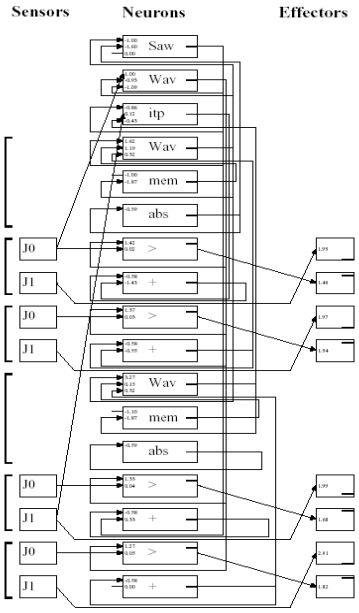
- Neurons: can have different activation functions (function of inputs):
  - sum, product, divide, sum-threshold, greater-than, sign-of, min, max, abs, if, interpolate, sin, cos, atan, log, expt, sigmoid, integrate, differentiate, smooth, memory, oscillate-wave, oscillate-saw.



Body phenotype



Genotype



Neural network phenotype

Creature control

- At each simulated time step, every neuron computes its output value from its inputs.
  - Two brain time steps performed for each dynamic simulation time step.
- Effectors: get input from a single neuron or sensor, output joint force (like a muscle force).
  - Each effector controls a degree of freedom of a joint.

Physical Simulation

- Complicated physical simulation is used:
  - articulated body dynamics, numerical integration, collision detection, collision response, friction, viscous fluid effect.
- Parallel implementation on a Connection Machine

Evolution of Creatures

- Creature grown from its genetic description.
- Run in simulation for some period of time
  - Sensors provide data about world and creature's body to brain
  - brain produces effector forces which move parts of creature
  - Fitness: how successful was desired behavior by end of allotted time

Video:
http://video.google.com/videoplay?docid=7219479512410540649

Fitness

- Swimming:
  - Environment: Turn off gravity, turn on viscous water effect.
  - Fitness: Swimming speed (distance of center of mass/time from initial center of mass position)
    - Special tweaks to prevent circling, initial push versus constant motion
    - Notice that tweaks on all of these make clear the trial and error aspect of this project
- Walking:
  - Environment: Turn on gravity, turn off viscous water effect.
  - Fitness: Horizontal speed.
    - Tweak: To prevent tall creatures that simply fall over, run simulation with no friction and no effector forces until height of center of mass reaches a stable minimum.

Fitness

- Jumping:
  - Environment: Turn on gravity, turn off viscous effect.
  - Fitness: Maximum height above the ground of lowest part of creature.
-
- Following a light source:
  - Environment: Enable photosensors.
  - Fitness: Average speed following different light sources over several trials.

Evolution of Creatures

- Create initial population of genotypes
- Best 20% survives. Rest are replaced by offspring of best 20%.
- Mutation:

  Many different types, depending on object, e.g.:
  - Numerical parameters changed by small random amounts
  - New random nodes are added
  - Connections randomly moved
  - New random connections generated

Applications?