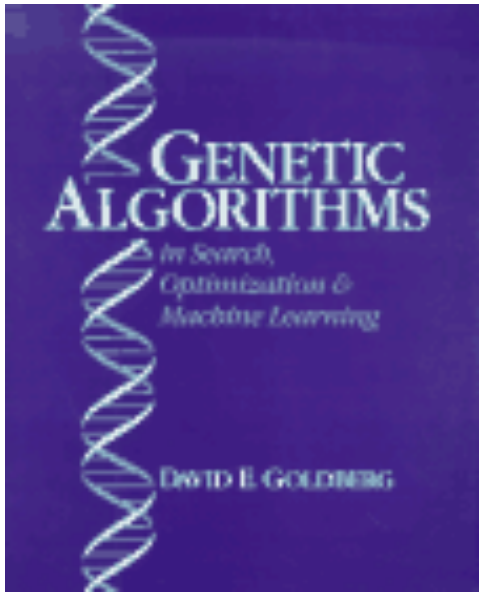
Evolutionary Computation

Robert Keller
Harvey Mudd College
December 2007

Some Early References



David Goldberg

Genetic Algorithms in
Search,
Optimization and
Machine Learning, 1989.

Zbigniew Michalewicz

Genetic Algorithms + Data Structures
= Evolution Programs
3rd edition, Springer-Verlag, 1993

Evolutionary Computation

- Evolutionary algorithms
- Genetic algorithms
- Genetic programming
- Artificial life (“alife”)

Historical

- 1966, Fogel, Owens, and Walsh, ***Artificial Intelligence through Simulated Evolution***, John Wiley & Sons

- Looked at derivation of
  - finite-state machines
  - controllers
  - data reduction

  through successive mutations

Historical

- 1975, John H. Holland
  Adaptation in natural and artificial systems, MIT Press
  - Focus was on natural systems, simulation
  - Introduced current genetic algorithm idea
  - Mostly theory, some applications to:
    - game-playing
    - search programs

Among Mr. Holland's Opuses

- "A universal computer capable of executing an arbitrary number of programs simultaneously." In Proceedings 1959 Eastern Joint Computer Conference. IEEE. pp108-13.

- See http://www.pscs.umich.edu/jhhfest/jhh-pub.html for more.

A Diagram from Holland's Paper

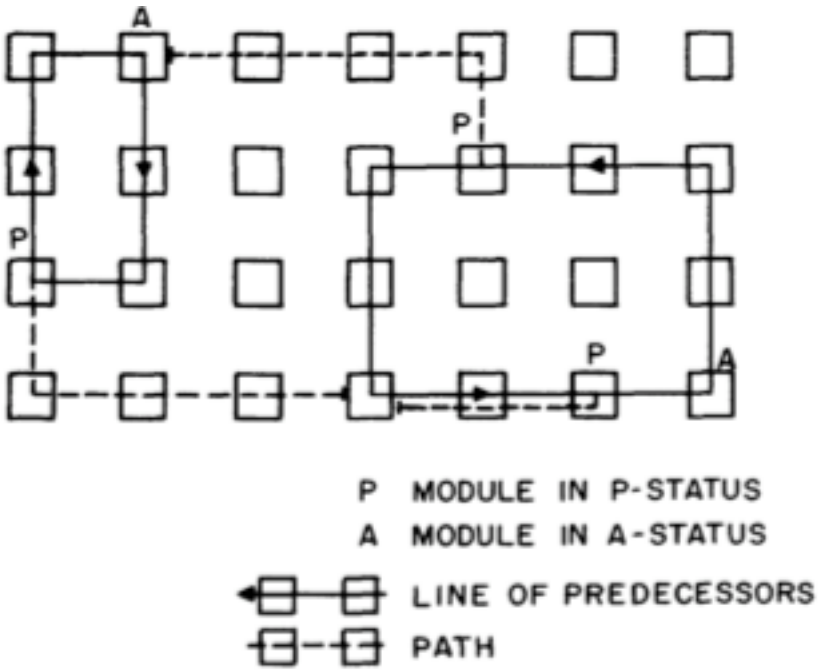


Fig. 5—Two interacting subroutines.

Mr. Holland's Progeny

- David Goldberg
- Stephanie Forrest
- Kenneth DeJong
- Melanie Mitchell
- John Koza
- many others
- See http://www.pscs.umich.edu/jhhfest/schedule-closed.html for Festscrhift papers.

Today

The MIT Press

IEEE TRANSACTIONS ON
NEURAL NETWORKS

IEEE TRANSACTIONS ON
FUZZY SYSTEMS

IEEE TRANSACTIONS ON
EVOLUTIONARY
COMPUTATION

Genetic Algorithms

- An approach to difficult optimization problems (TSP, etc.)
- Heuristic, not guaranteed to find true optima
- Finds good approximations fast

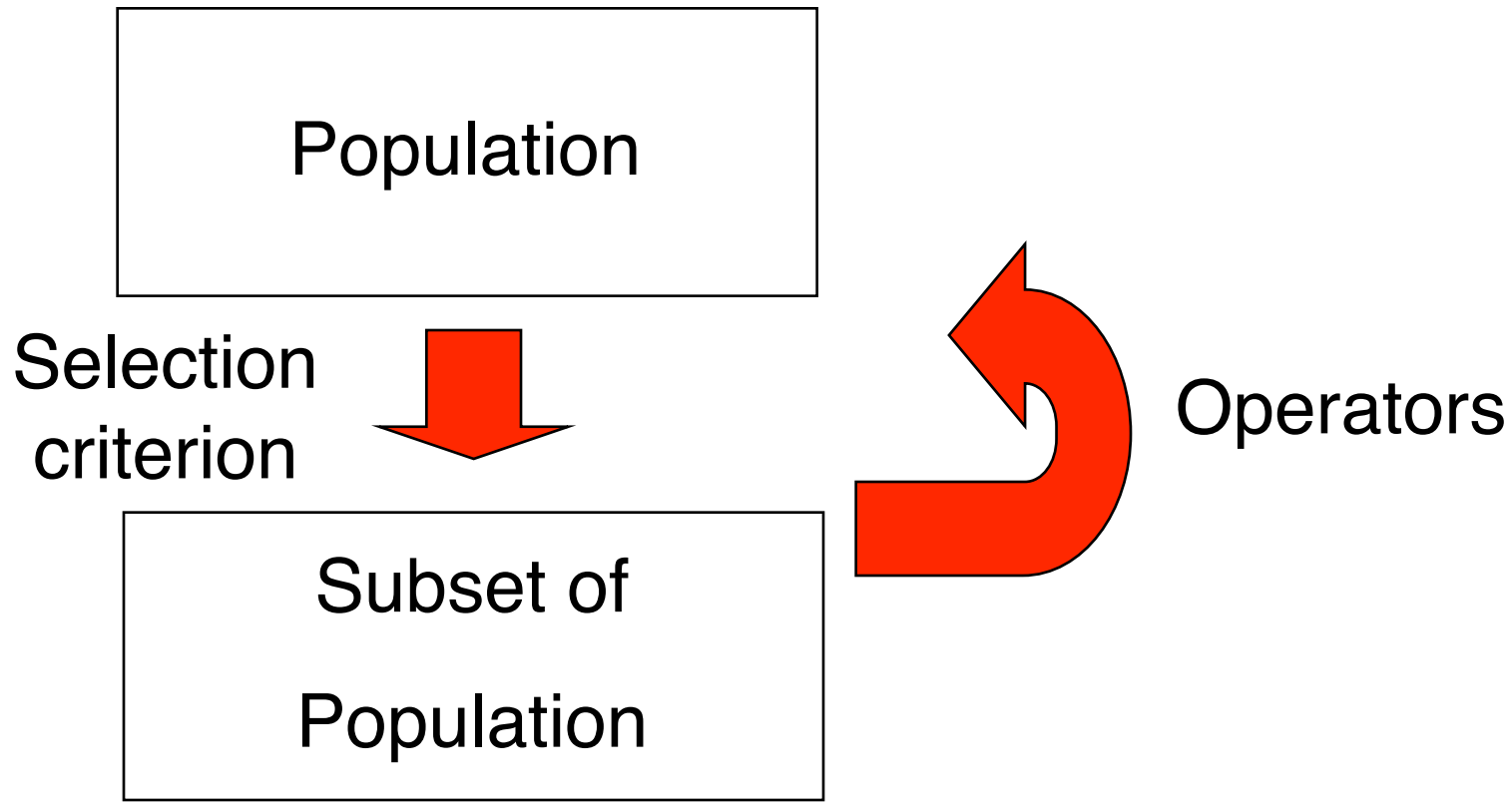
GA Applications

- Hundreds, if not thousands
- All manner of optimization problems in engineering, science, finance, etc.
- Application to neural networks:
  - *Evolve* the structure and/or weights of a neural network, rather than train it.

Principles of Natural Selection

- Concentrate on **population** rather than a *single* individual
- Individuals that are fit enough to survive will reproduce
- Create new individuals from existing ones
  - Crossover
  - Mutation

Genetic Flow Diagram



until an individual exists satisfying performance criterion or resources are exhausted

Common Operators

- **Copy** (aka Reproduce): An individual from the current generation is copied into the next generation.
- **Crossover**: Two (or more) individuals from the current generation are used to form an individual in the next generation.
- **Mutate**: A single individual from the current generation is mutated to form an individual in the next generation.

Individuals

- Individuals are represented by their genome, or **genotype**.
- The genotype may be the “program” for the actual individual, or **phenotype**.

Genetic Algorithm Example

- Consider the “subset sum” problem:
  - Given a set of integers S and a target value T, find a subset of S that the maximum sum **without exceeding** T.

Genetic Algorithm Example

- "maximal subset sum" problem:
  - Given a **set** of integers S and a **target** value T, find a subset of S that the maximum sum **without exceeding** T.
- This is an *optimization* problem. The related *decision* problem ("subset sum problem"), find whether there is a subset of a set of integers summing to *exactly* T, is known to be NP-complete.
- A related problem arises in cryptography.

Genetic Approach

- As the genome, use a bit-vector.
- There is one bit for each element in the set S.
- The bit is 1 iff the element is in the subset.

Example

- S = {19, 23, 35, 52, 61, 68, 76, 84, 92}
- T = 200
- Genome is an element of $\{0, 1\}^9$
- Possible individuals:

  010001001, sum{23, 68 , 92} = 183

  101000010, sum{19, 35, 84} = 138

What can we try to produce more fit individuals?

- **Mutation**: change a random bit:

  101000010, sum{19, 35, 84} = 138

  111100010, sum{19, 23, 35, 84} = 161

Fatal Mutations

- Note that a mutation could be “fatal”, resulting in a totally unfit individual. The “carcass” of this individual could still be in the next generation, however.

What can we try to produce more fit individuals?

- **Crossover**: Combine two individuals

  001|110000, sum{35, 52, 61} = 148

  100|000110, sum{19, 76, 84} = 179

crossover point selected at random

- New genomes:

  001000110, sum{35, 76, 84} = 195 ← better

  100110000, sum{19, 52, 61} = 132

Crossover Variations

- Sometimes **two crossover points** are chosen, rather than one, and the sub-sequences between them are swapped.
- **Uniform crossover** means to pick each gene independently of the others, based on a probability value.
- See: http://www.nd.com/genetic/crossover.html for other possibilities.
- Just as with mutation, any type of crossover could produce one or more individuals that are totally unfit.

Sample GA Program

- The program /cs/cs152/ga/subsum/subsum.java carries out the genetic algorithm on this problem.

- Examples:

  go ss1.in
  go ss2.in
  go ss3.in

Main Loop of the Subset Sum GA Program (1)

```
public void evolve(int generations)
  {
  for( generation = 0; generation < generations; generation++ )
    {
     retain();    // retain the more fit individuals
     crossover(); // perform crossover on those retained
     mutate();    // mutate the resulting population
     sort();      // sort by fitness
    }
  }
```

Sample Run of subsum (1)

```
Subset sum problem
generations = 100
population size = 10
retain size = 5
immutable = 5
mutation rate = 0.1

target = 200.0
values = (19 23 35 52 61 68 76 84 92)
```

Sample Run of subsum (1)

```
generation 0, average fitness = 33.7:
181/200 (r 19) 000111000 52 61 68
164/200 (r 36) 100010010 19 61 84
-1/200 (r 201) 000011110 61 68 76 84
-1/200 (r 201) 010111000 23 52 61 68
-1/200 (r 201) 000010110 61 76 84
-1/200 (r 201) 000111100 52 61 68 76
-1/200 (r 201) 101101101 19 35 52 68 76 92
-1/200 (r 201) 000011110 61 68 76 84
-1/200 (r 201) 101101101 19 35 52 68 76 92
-1/200 (r 201) 001100101 35 52 76 92
```

only two fit individuals

Sample Run of subsum (2)

```
generation 1, average fitness = 64.4:
181/200 (r 19) 000111000 52 61 68
164/200 (r 36) 100010010 19 61 84
160/200 (r 40) 000000110 76 84
145/200 (r 55) 000010010 61 84
-1/200 (r 201) 001100101 35 52 76 92
-1/200 (r 201) 000011110 61 68 76 84
-1/200 (r 201) 010111000 23 52 61 68
-1/200 (r 201) 000011110 61 68 76 84
-1/200 (r 201) 100011110 19 61 68 76 84
-1/200 (r 201) 000010110 61 76 84
```

four fit individuals

Sample Run of subsum (3)

```
generation 2, average fitness = 114.3:
197/200 (r 3) 000110010 52 61 84
181/200 (r 19) 000111000 52 61 68
164/200 (r 36) 100010010 19 61 84
160/200 (r 40) 000000110 76 84
148/200 (r 52) 100011000 19 61 68
145/200 (r 55) 000010010 61 84
96/200 (r 104) 001010000 35 61
54/200 (r 146) 101000000 19 35
-1/200 (r 201) 000110011 52 61 84 92
-1/200 (r 201) 001100101 35 52 76 92
```

eight fit individuals

Sample Run of subsum (4)

```
generation 5, average fitness = 157.7:
200/200 (r 0) 100111000 19 52 61 68
197/200 (r 3) 000110010 52 61 84
197/200 (r 3) 000110010 52 61 84
181/200 (r 19) 000111000 52 61 68
181/200 (r 19) 000111000 52 61 68
179/200 (r 21) 001100001 35 52 92
164/200 (r 36) 100010010 19 61 84
164/200 (r 36) 100010010 19 61 84
115/200 (r 85) 101010000 19 35 61
-1/200 (r 201) 000111010 52 61 68 84
best possible fit reached in generation 5
```

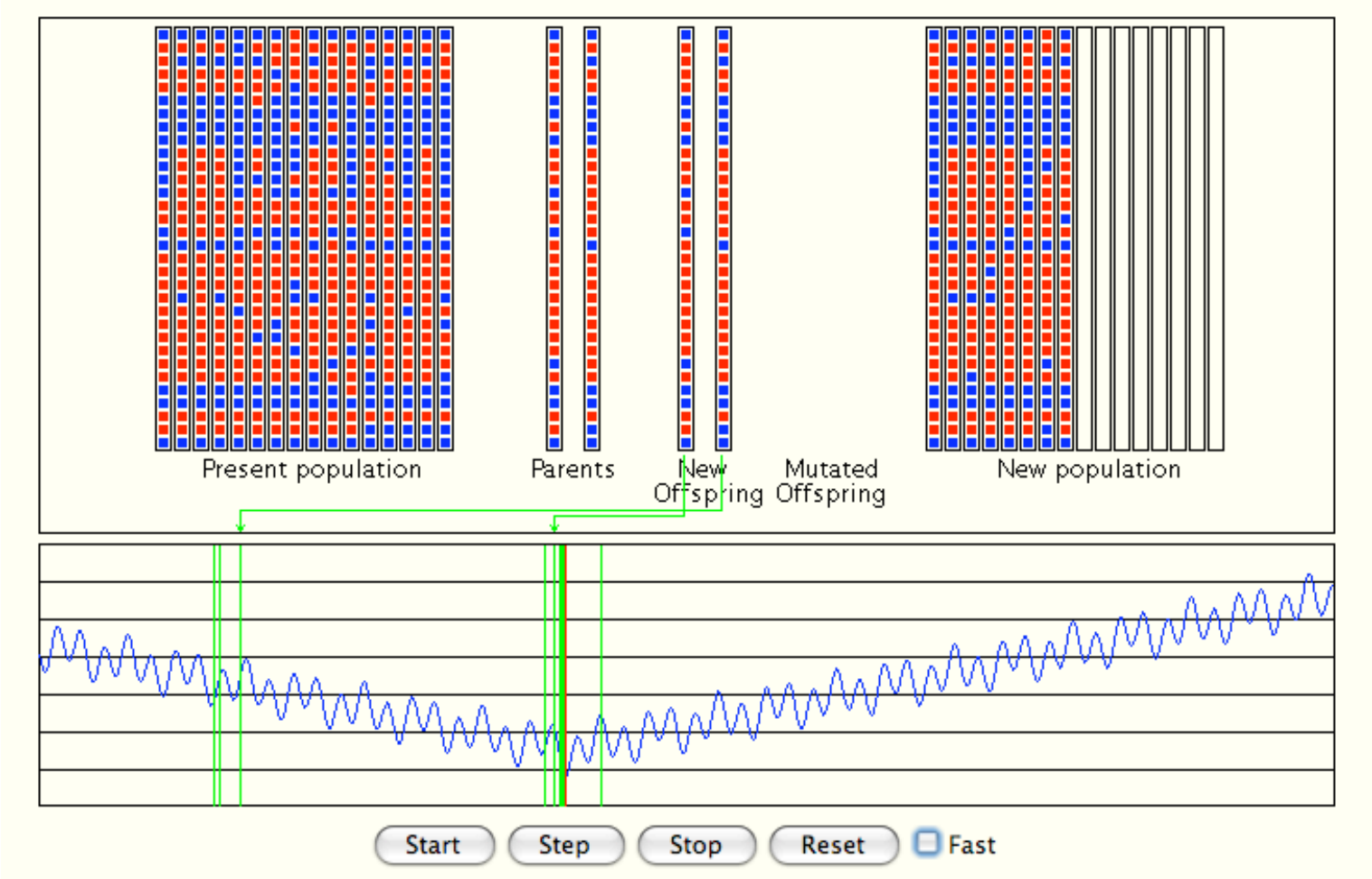
perfect individual

Complexity Considerations

- All possible subsets of n values can be **enumerated** in 2^n steps.
- For low n (say in the 20's or less), this might be feasible.
- For larger n, it is not (2^{32} = about 5 billion).
- For large n, the genetic algorithm can produce good, if not optimal, answers in much less time than enumeration.

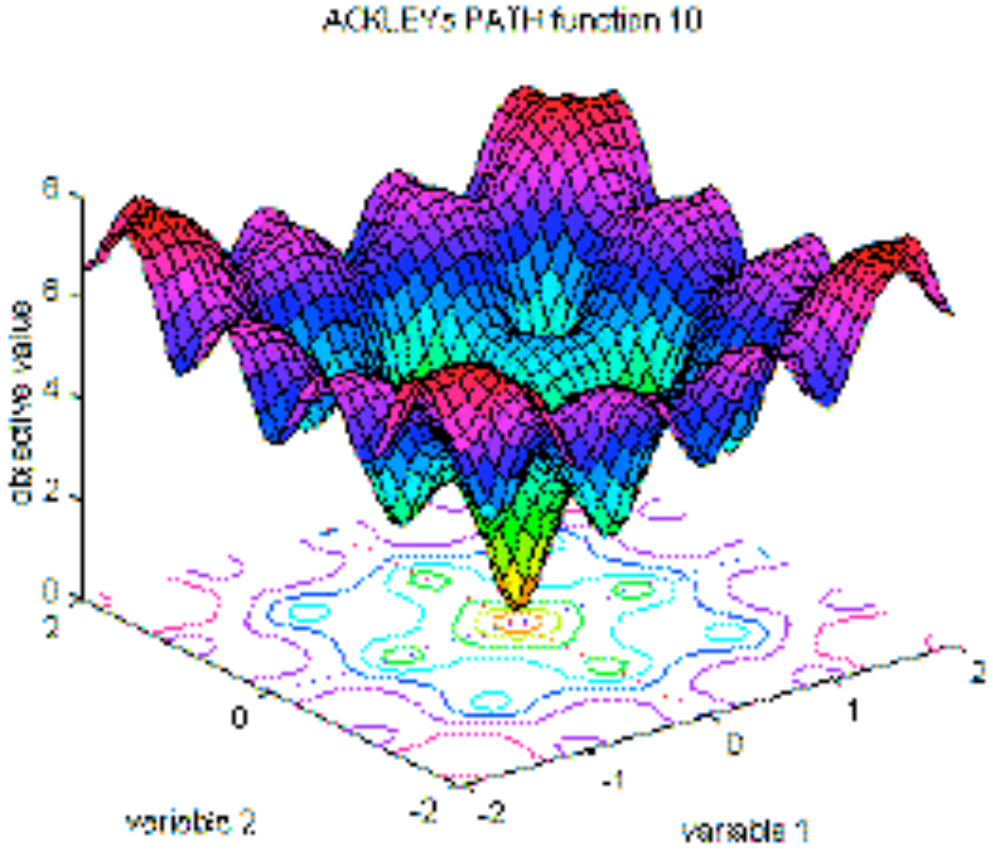
Function Minimization Demo

(http://www.obitko.com/tutorials/genetic-algorithms/example-function-minimum.php)



Rugged Landscapes

(http://www.geatbx.com/docu/fcnindex-01.html)



Other Genomes

- A bit vectors is not the only way, or necessarily the best way, to represent a genome.
- Other possibilities:
  - A list or matrix of integers or floats

Challenges in representing Genomes: Traveling Salesperson Problem

- Not every optimization problem has a genome encoding that will allow naïve mutations and crossovers.
- Consider the TSP:
  - An instinctive way to represent a genome is as a **permutation** of the cities on a tour.

Crossing two Permutations

- [1, 3, | 2, 6, 5, | 4]
  [2, 3, | 4, 1, 6, | 5]

- Result of naïve crossing:
  [1, 3, 4, 1, 6, 5]
  [2, 3, 2, 6, 5, 4]
- Unfortunately, these sequences are **not** permutations.

Reinterpreting permutation crossings

- [1, 3, | 2, 6, 5, | 4]
  [2, 3, | 4, 1, 6, | 5]

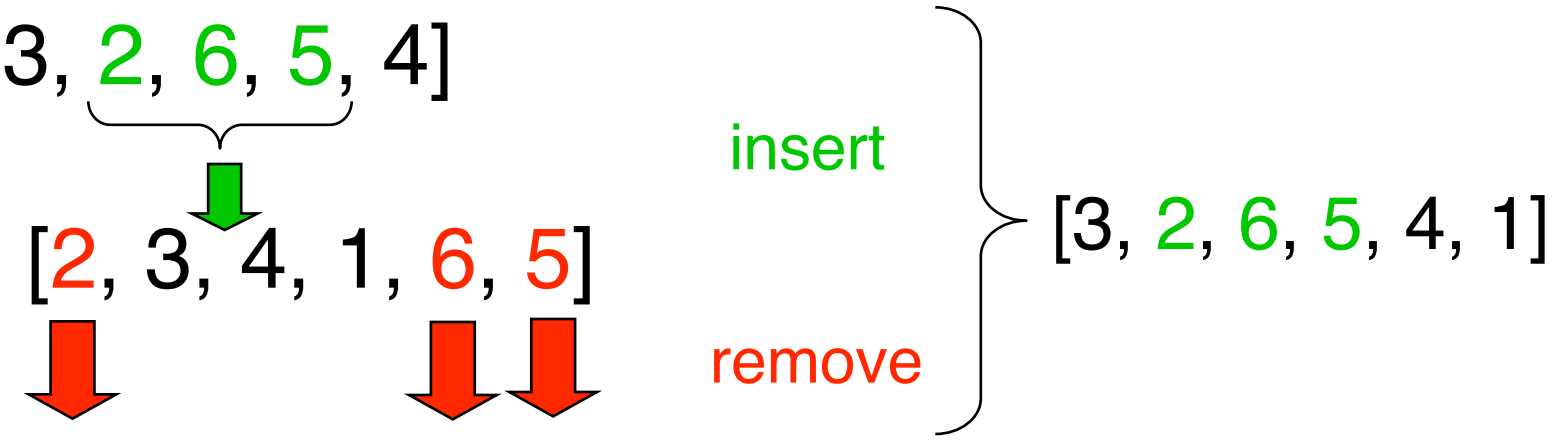
- Interpret as:
  - Insert 2,6,5 in the second genome at the first crossover point and remove those elements from wherever they occurred in the second genome.

Reinterpreting permutation crossings

- [1, 3, | 2, 6, 5, | 4]
  [2, 3, | 4, 1, 6, | 5]

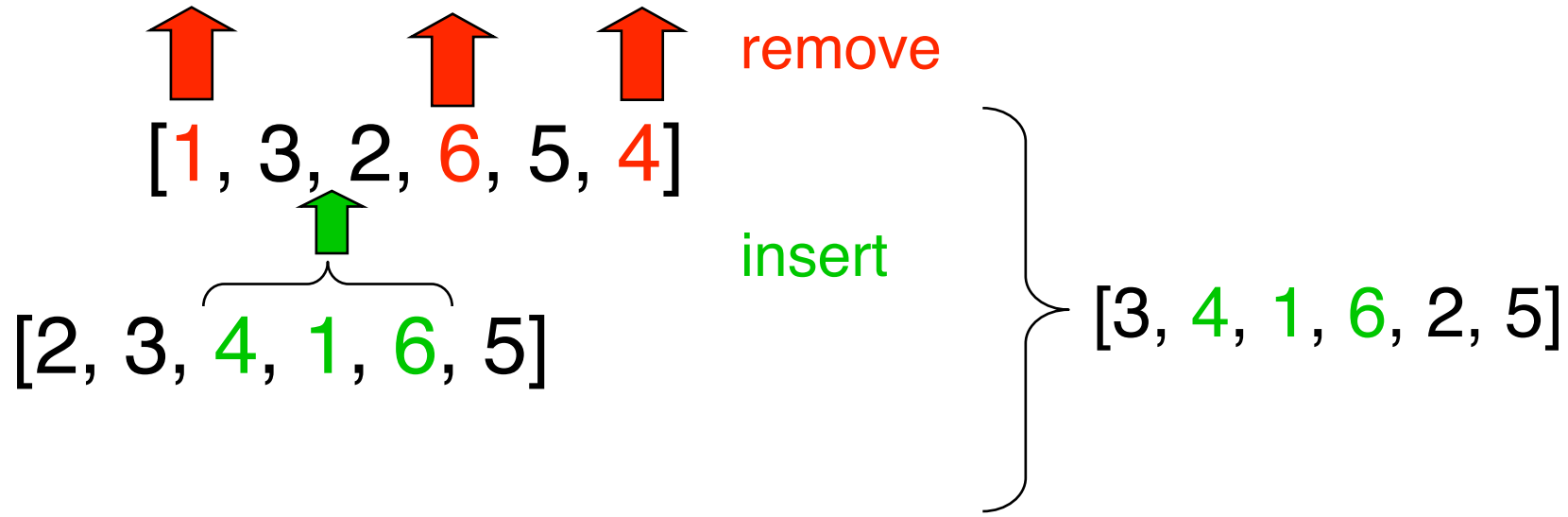
crossover points selected at random

- [1, 3, 2, 6, 5, 4]



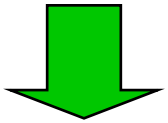
Similarly, to produce the second new genome

- [1, 3, | 2, 6, 5, | 4]
  [2, 3, | 4, 1, 6, | 5]

- 

Net effect of Crossover

- [1, 3, | 2, 6, 5, | 4]
  [2, 3, | 4, 1, 6, | 5]



- [3, 4, 1, 6, 2, 5]
  [3, 2, 6, 5, 4, 1]

- The results share some of the structure of both parents, which is desirable.

Keller's TSP GA

- /cs/cs152/ga/tsp
- Uses same overall loop as the subset sum algorithm.
- The genome is now a permutation vector.
- Crossover is as described.
- Mutation consists of swapping two random elements of the permutation.

TSP GA in operation (1)

```
Traveling Salesperson  Problem
generations = 1000
population size = 50
retain size = 25
immutable = 15
mutation rate = 0.1
```

TSP GA in operation (2)

```
Costs:
0.0  1.0  2.0  4.0  9.0  8.0  3.0  2.0  1.0  5.0  7.0  1.0  2.0  9.0  3.0
1.0  0.0  5.0  3.0  7.0  2.0  5.0  1.0  3.0  4.0  6.0  6.0  6.0  1.0  9.0
2.0  5.0  0.0  6.0  1.0  4.0  7.0  7.0  1.0  6.0  5.0  9.0  1.0  3.0  4.0
4.0  3.0  6.0  0.0  5.0  2.0  1.0  6.0  5.0  4.0  2.0  1.0  2.0  1.0  3.0
9.0  7.0  1.0  5.0  0.0  9.0  1.0  1.0  2.0  1.0  3.0  6.0  8.0  2.0  5.0
8.0  2.0  4.0  2.0  9.0  0.0  3.0  5.0  4.0  7.0  8.0  3.0  1.0  2.0  5.0
3.0  5.0  7.0  1.0  1.0  3.0  0.0  2.0  6.0  1.0  7.0  9.0  5.0  1.0  4.0
2.0  1.0  7.0  6.0  1.0  5.0  2.0  0.0  9.0  4.0  2.0  1.0  1.0  7.0  8.0
1.0  3.0  1.0  5.0  2.0  4.0  6.0  9.0  0.0  3.0  3.0  5.0  1.0  6.0  4.0
5.0  4.0  6.0  4.0  1.0  7.0  1.0  4.0  3.0  0.0  9.0  1.0  8.0  5.0  2.0
7.0  6.0  5.0  2.0  3.0  8.0  7.0  2.0  3.0  9.0  0.0  2.0  1.0  8.0  1.0
1.0  6.0  9.0  1.0  6.0  3.0  9.0  1.0  5.0  1.0  2.0  0.0  5.0  4.0  3.0
2.0  6.0  1.0  2.0  8.0  1.0  5.0  1.0  1.0  8.0  1.0  5.0  0.0  9.0  6.0
9.0  1.0  3.0  1.0  2.0  2.0  1.0  7.0  6.0  5.0  8.0  4.0  9.0  0.0  7.0
3.0  9.0  4.0  3.0  5.0  5.0  4.0  8.0  4.0  2.0  1.0  3.0  6.0  7.0  0.0
```

TSP GA in operation (3)

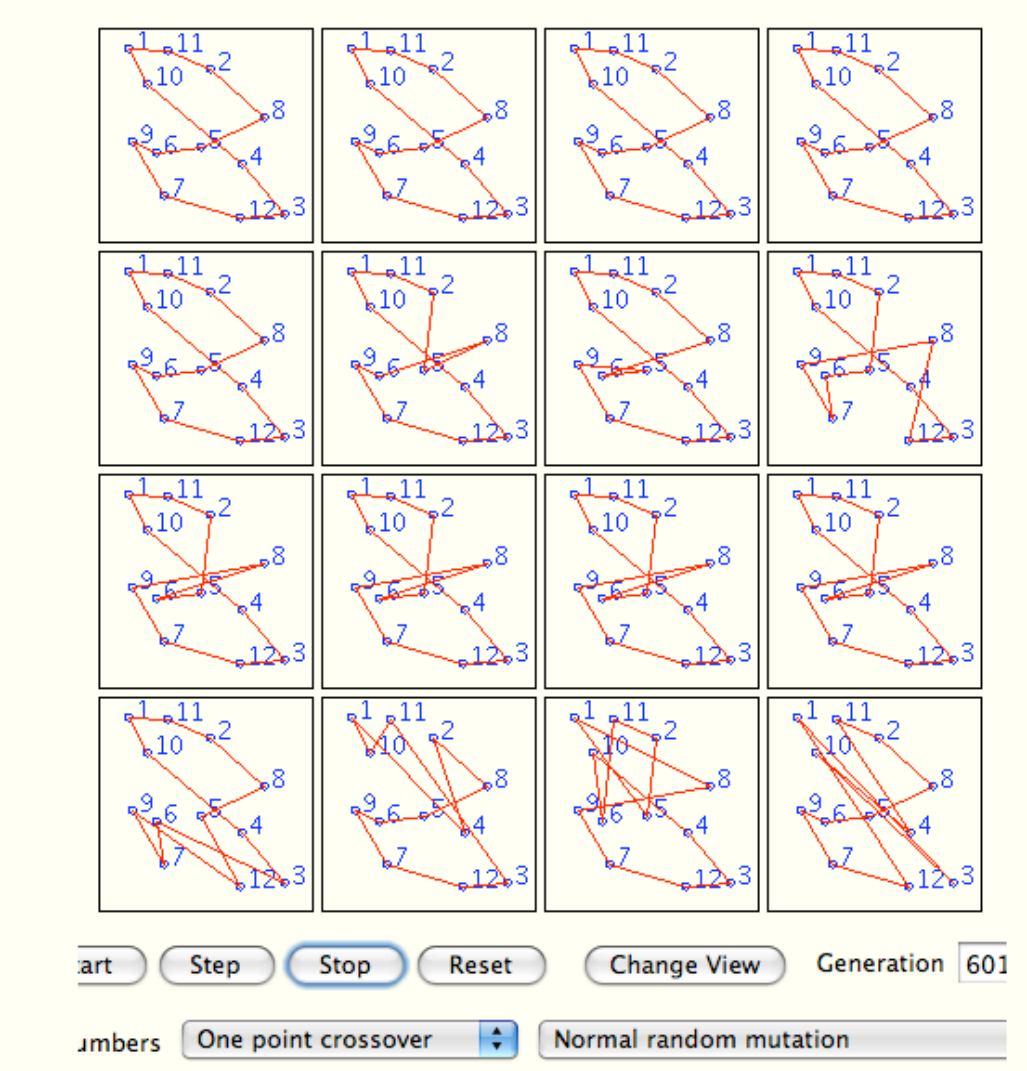
Improvement in generation 1: 25: 4 6 10 14 9 11 7 12 3 5 13 1 0 8 2
Improvement in generation 9: 22: 4 6 14 9 11 7 12 10 3 5 13 1 0 8 2
Improvement in generation 10: 20: 4 6 9 14 10 11 7 12 3 5 13 1 0 8 2
Improvement in generation 30: 18: 4 6 9 14 10 12 7 11 3 5 13 1 0 8 2
Improvement in generation 317: 17: 4 6 9 14 10 12 5 13 3 11 7 1 0 8 2

best after 1000 generations, 17: 4 6 9 14 10 12 5 13 3 11 7 1 0 8 2

verification: 4 (1.0) 6 (1.0) 9 (2.0) 14 (1.0) 10 (1.0) 12 (1.0) 5 (2.0) 13 (1.0) 3 (1.0) 11 (1.0) 7 (1.0) 1 (1.0) 0 (1.0) 8 (1.0) 2 (1.0) 4

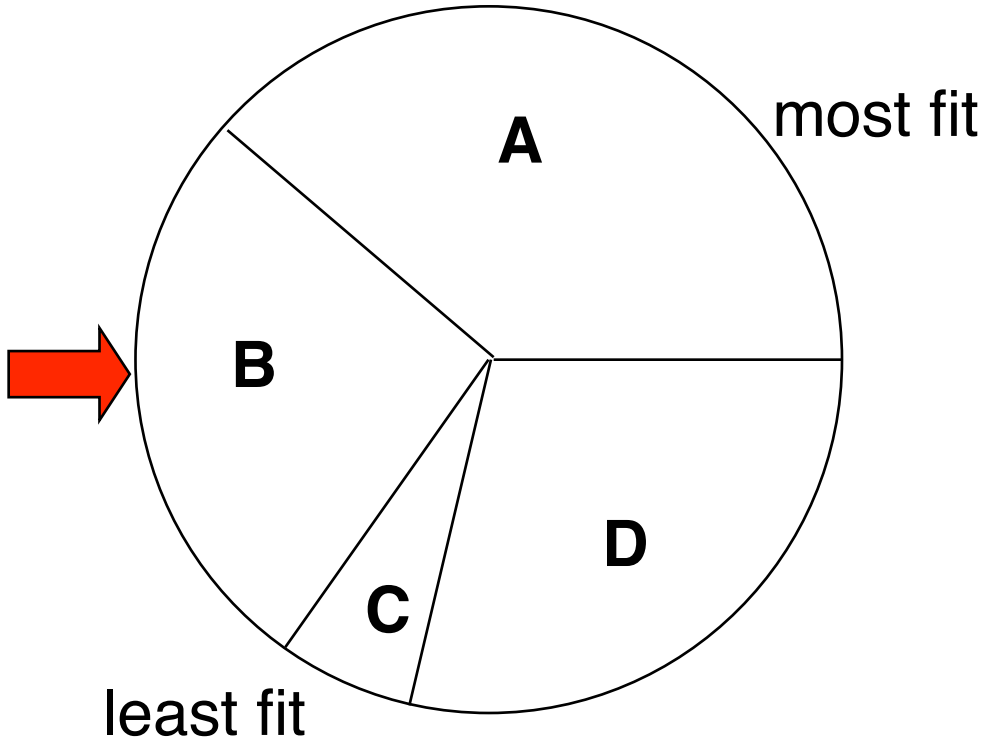
Graphic TSP Demo

(http://www.obitko.com/tutorials/genetic-algorithms/tsp-example.php)



Roulette-Wheel Optimization

Give individuals in the population a probability of being picked **in proportion** to their fitness.



Features from a GA Library

Features	
Genetic Algorithms • Generational • Steady-State	**Data Types** • Binary • Integer • Float
Mutation Operators • Flip Bit • Boundary • Uniform • Gaussian	**Crossover Operators** • One Point • Two Point • Uniform • Arithmetic • Heuristic
Selection Operators • Roulette • Tournament • Top Percent • Best • Random	**Termination Methods** • Generation Number • Evolution Time • Fitness Convergence • Population Convergence • Gene Convergence

GA Perspective

- Like gradient descent, GA's can also get stuck in local fitness extrema.
- The space is different; for GA's, a stuck point corresponds to a population from which crossover does not yield any better individuals.
- Mutation is one hope for leaving such an extremum. Other possibilities are simulated annealing, random restarts.

Skeptical Opinion

- The building block hypothesis of GA's has been sharply criticized on the grounds that it lacks theoretical justification and experimental results have been published that draw its veracity into question. On the theoretical side, for example, Wright et al. state:

  "The various claims about GAs that are traditionally made under the name of the building block hypothesis have, to date, no basis in theory and, in some cases, are simply incoherent"

- On the experimental side uniform crossover was seen to outperform one-point and two-point crossover on many of the fitness functions studied by Syswerda. Summarizing these results, Fogel remarks:

  "Generally, **uniform crossover yielded better performance** than two-point crossover, which in turn yielded better performance than one-point crossover".

- Syswerda's results contradict the building block hypothesis because **uniform crossover is extremely disruptive of short schemata** whereas one and two-point crossover are more likely to conserve short schemata and combine their defining bits in children produced during recombination.

Evolving NNs using GAs

- The structure could be fixed and just the weights evolved.
- The structure could be evolved too, by making the whole network be completely connected, but letting some weights evolve toward 0.
- Recurrent networks could be handled without special techniques, such as RTRL.

A Few Examples

- http://www.cs.bgu.ac.il/~omri/NNUGA/

  evolving non-linear 2D classifications
- http://www.msm.cam.ac.uk/phase-trans/2006/ga_html_files/ga.html

  GAs for NNs.

Other Evolution Options

- Since we are simulating using a computer, not actually evolving species, there is no reason why **Lamarckian**, rather than **Darwinian**, evolution could not be used.
- Most results to date are Darwinian.
- Lamarckian could integrate other learning models to breed new species from individuals that have learned.
- Lamarckian also goes by the name Memetic Algorithms, after the term “meme” introduced by Richard Dawkins.

Lamarckian Leads

- Grefenstette, J. 1991. *Lamarckian learning in multi-agent environments*. In Proceedings of the Fourth International Conference of Genetic Algorithms, 303-310. Morgan Kaufmann.

- Houck, C., Joines, J., and Kay, M., Utilizing Lamarckian Evolution and the Baldwin Effect in Hybrid Genetic Algorithms, NCSU-IE TR 96-01, 1996.

Baldwin Effect (1896)

- Traits acquired during an organisms lifetime cannot be passed on genetically.

- However, **ability to learn** useful traits can be passed on indirectly, as it enhances survivability.

Genetic Programming

- Genetic programming is the GA idea applied to evolving ***programs*** (as opposed to just numbers).
- The prime mover of this field is/was John R. Koza.

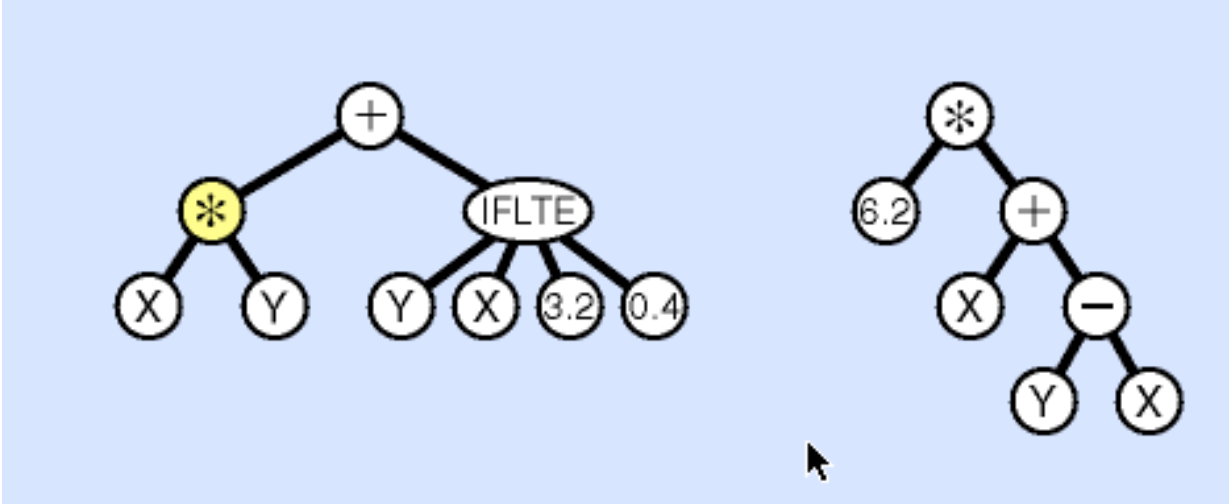
Reference

John R. Koza

Genetic Programming :
On the Programming of Computers
by Means of Natural Selection

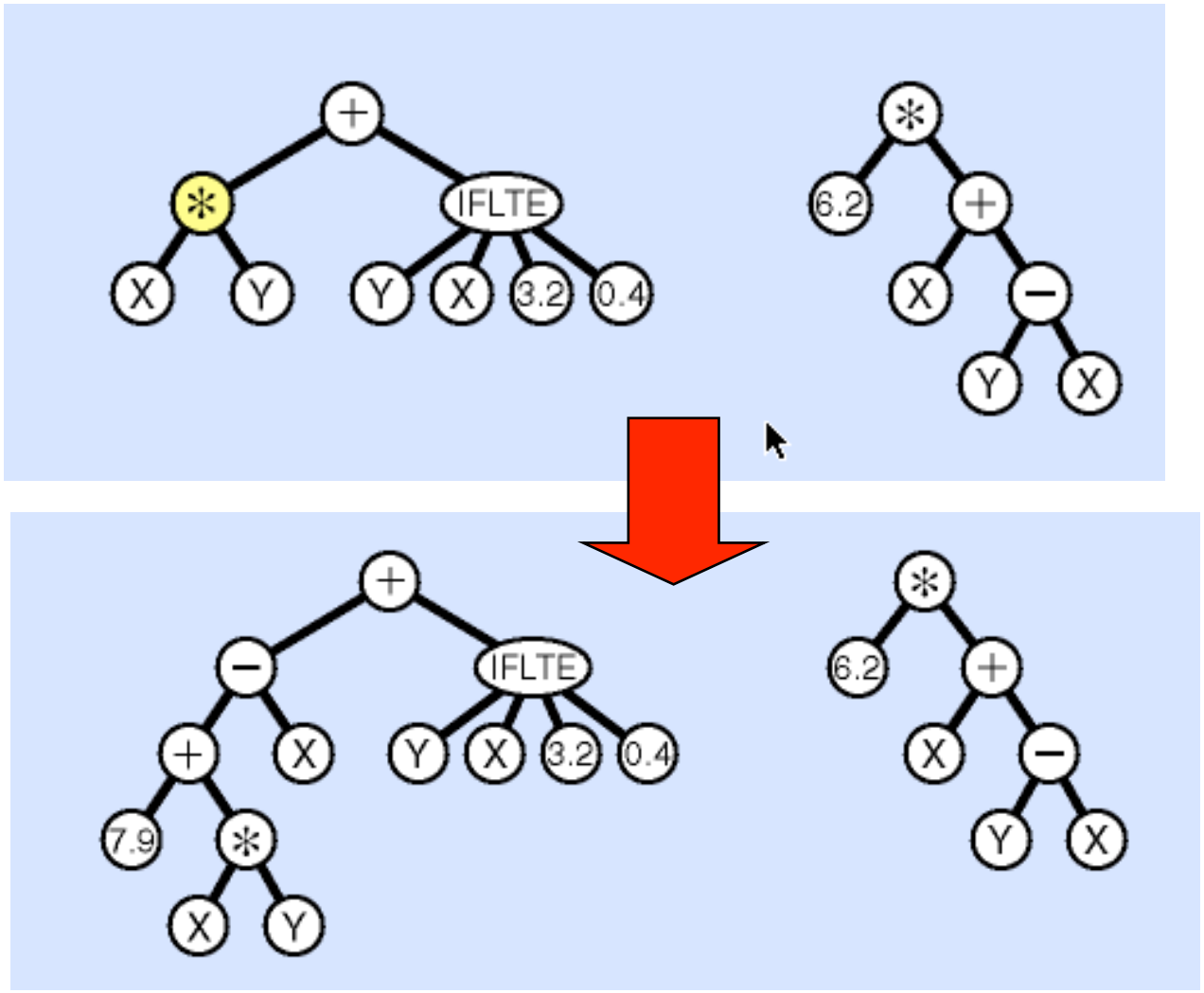
MIT Press, 1996

Genetic Programming Genomes = Syntax Trees

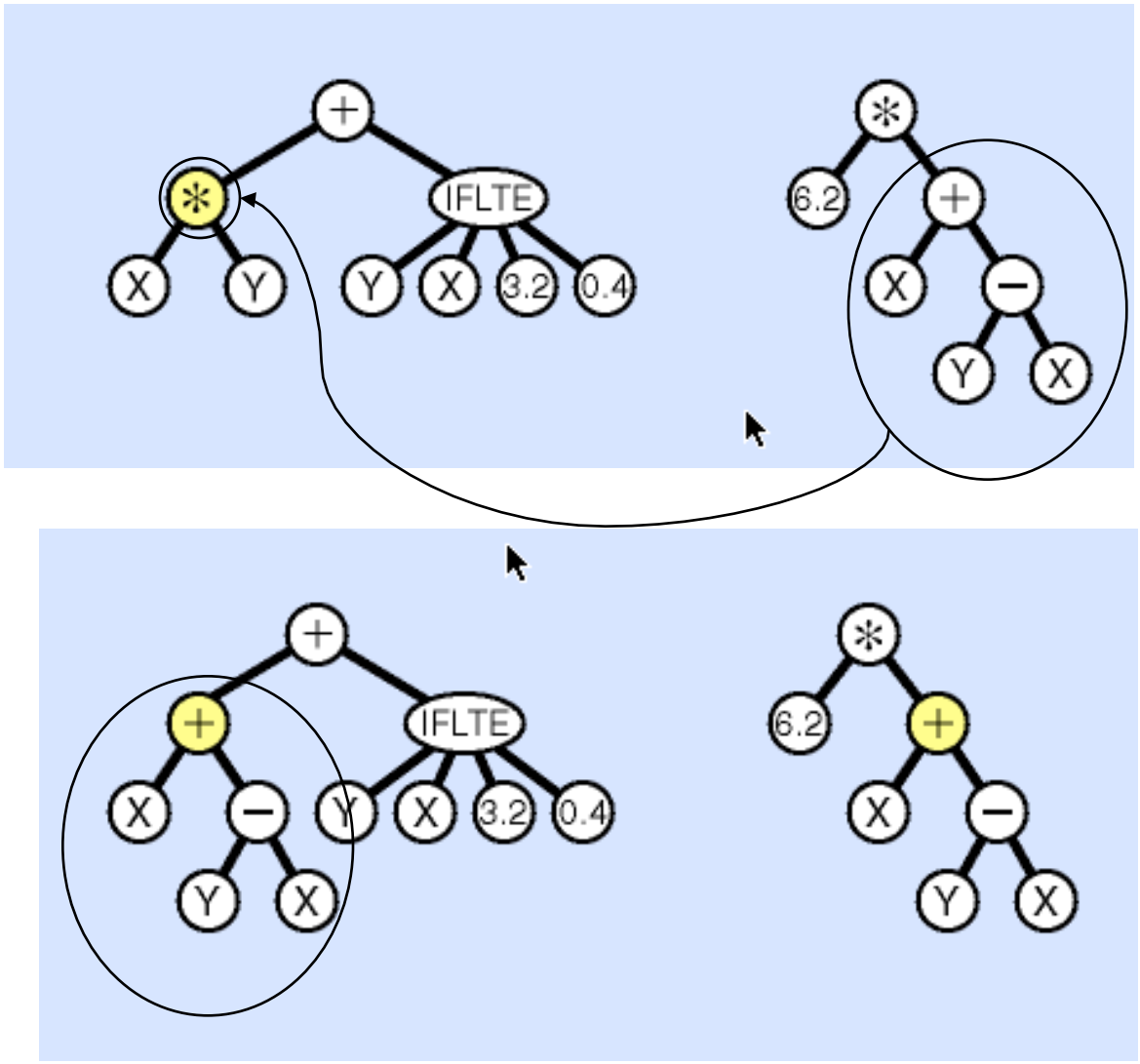


For animated tutorial, please see:
http://www.genetic-programming.com/gpanimatedtutorial.html

Mutation of a Progam

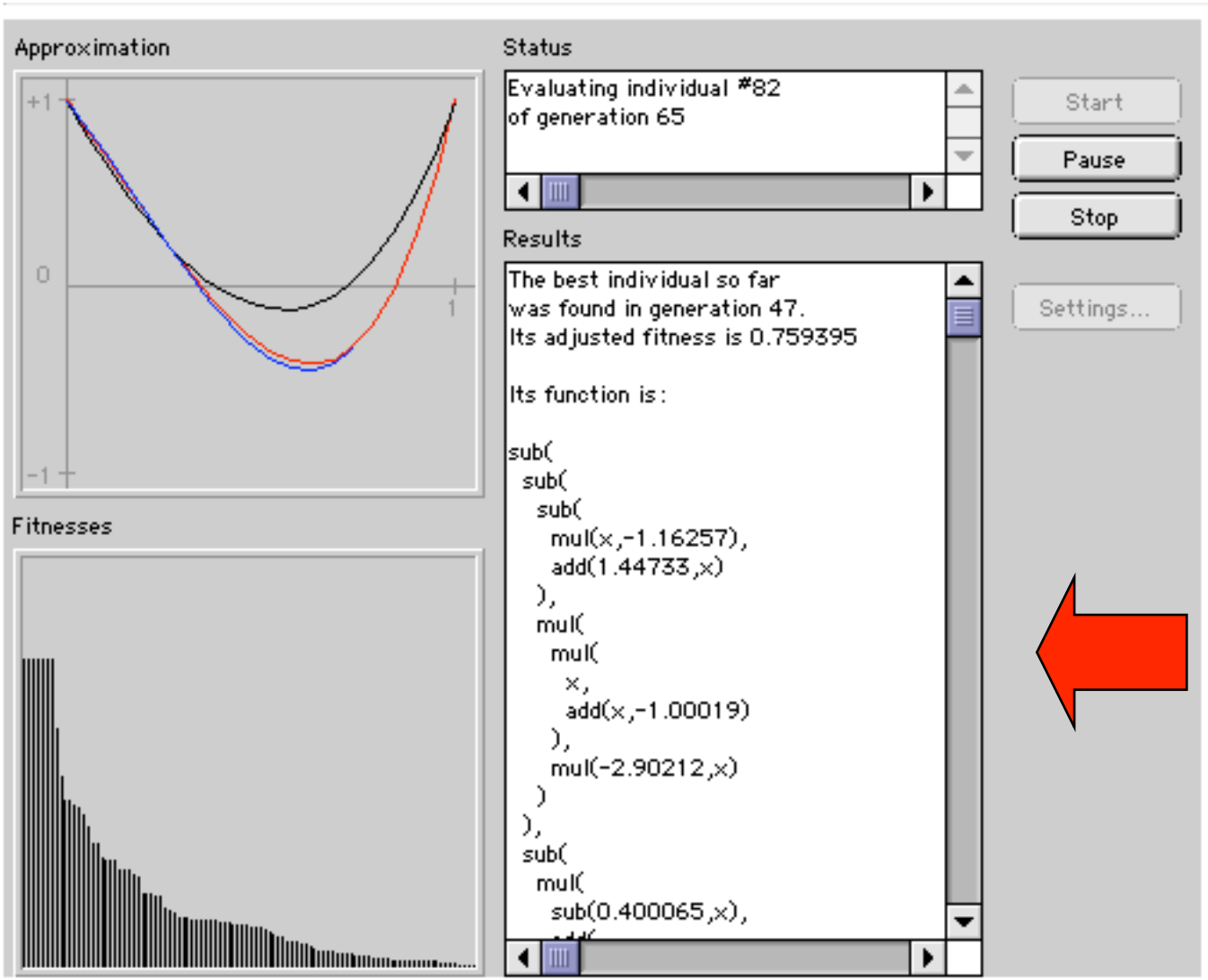


Crossover of Two Programs



Genetic Programming Demo: Symbolic Regression (authored by Koza)

(http://alphard.ethz.ch/gerber/approx/default.html)



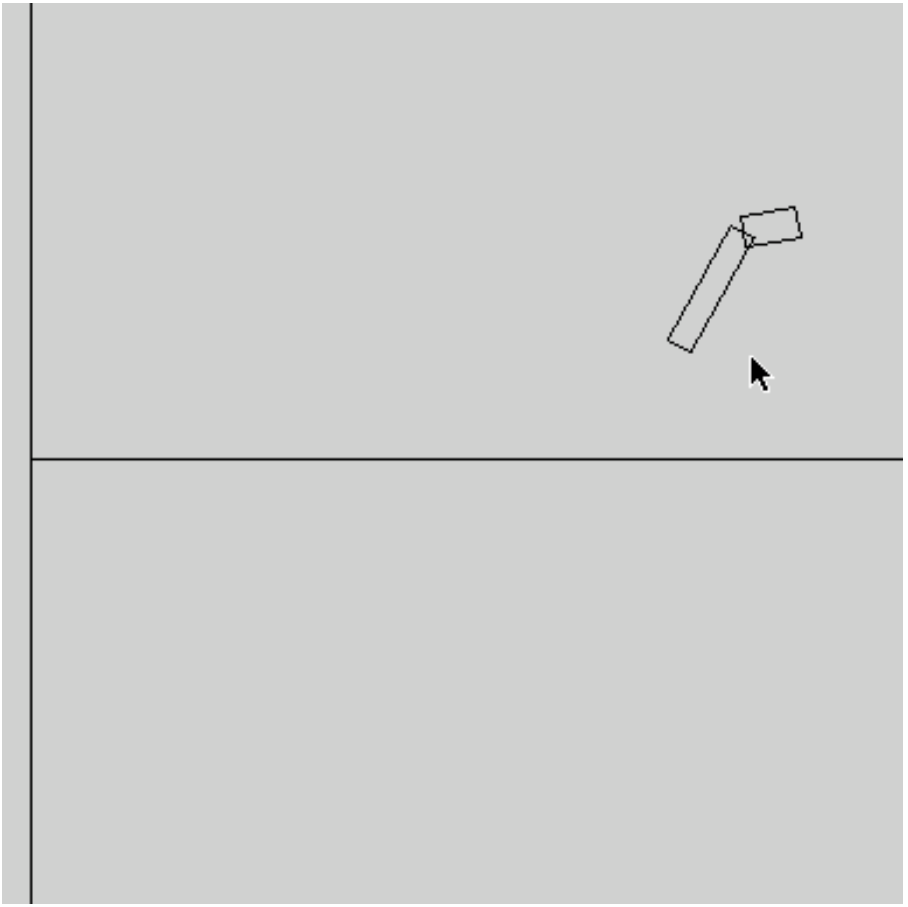
The evolved program

Truck-Backer using GP

http://www.handshake.de/user/blickle/Truck/index.html

The evolved program

```
PLUS[MINUS[y,MUL[x,trailert]],IFLTZ[PLUS[MINUS[y
MUL[x,trailert]],x],PLUS[PLUS[0.124600,cabt],
ATG[-0.798000,0.220800]],
PLUS[PLUS[cabt,cabt],PLUS[MINUS[y,MUL[MINUS[y,
MUL[MINUS[y,MUL[MINUS[y,MUL[x,trailert]],
trailert]],trailert]],trailert]],PLUS[PLUS[cabt
```



GP Caution

- Typically fitness of genetically-evolved programs is established by working on a large number of test cases.
- We know this is not completely sound, however it often works pragmatically.
- **A possible fruitful area** is to evolve a proof of correctness along with the program itself. I know of no work in this area.

Engineering Applications: Evolving Hardware (Analog & Digital)

- The evolved program is a ***list of instructions*** for ***constructing the circuit***, rather than the graph of the circuit itself.
- **This approach has also been used to construct neural networks.**
- The results for circuit design are competitive with, or superior to, human engineering.
- Numerous patents on electrical systems have been reinvented using GP.

Generating Analog Circuits

EETIMES
June 03, 1996, Issue: 904
Section: Technology

Genetic program auto-designs analog circuits

R. Colin Johnson

Palo Alto, Calif. - Genetic algorithms (GA) can sometimes rescue software engineers stymied by problems that don't lend themselves to conventional programming techniques. But hardware engineers should be using GAs for designing analog circuits too, claims Stanford professor John Koza.

"Genetic programming can design what-you-want-is-what-you-get electronic circuitry without any prior knowledge about electrical engineering," said Koza. He went on to contrast his approach with that of conventional artificial intelligence: "AI says that the power is in the knowledge, but I say that knowledge is the enemy.

"What we want to use is nature's method to evolve optimal solutions without the hindrance of preconceived knowledge."

http://www.genetic-programming.com/published/eetimes060396.html

A Genetically-Evolved Amplifier

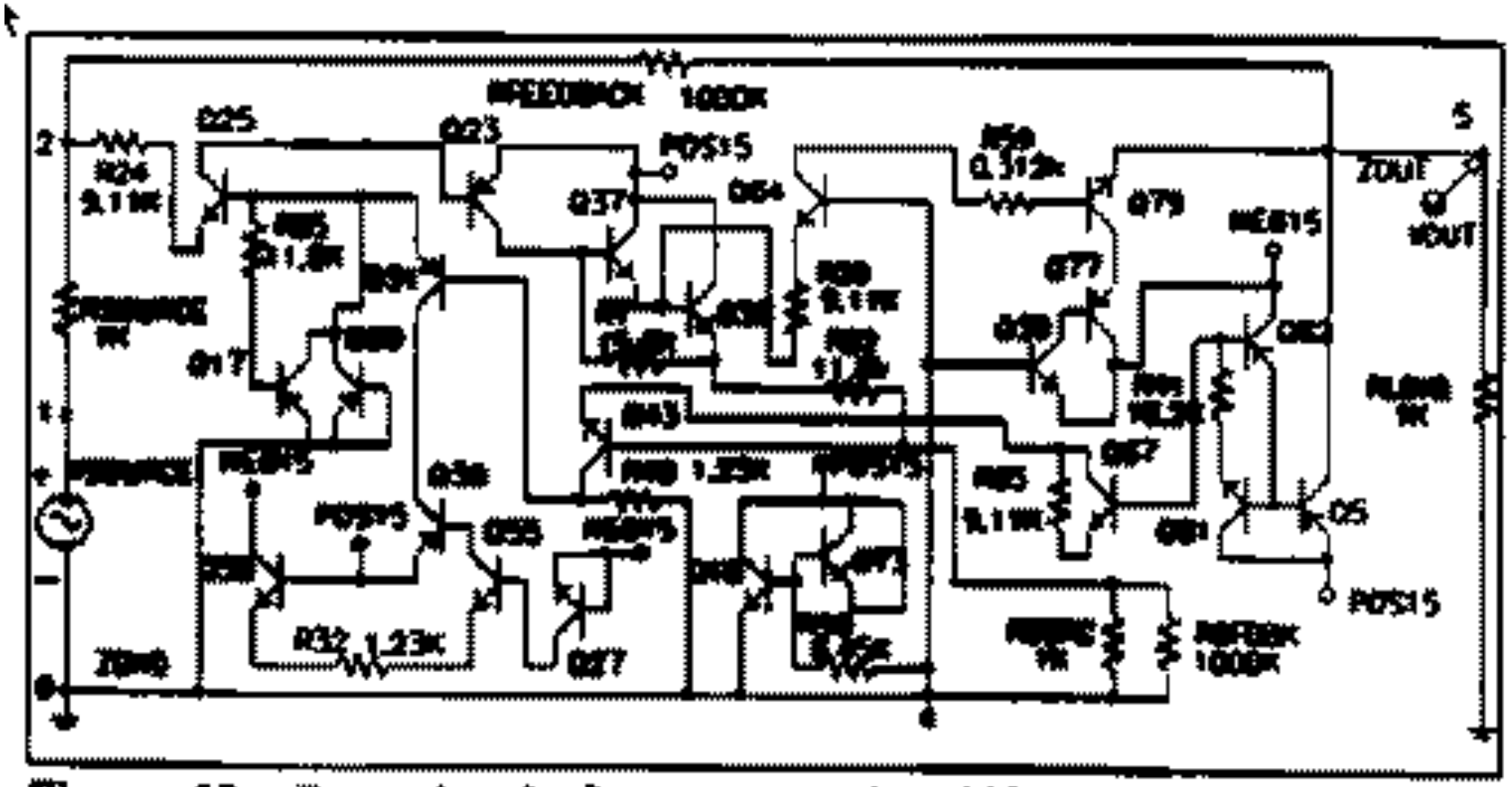
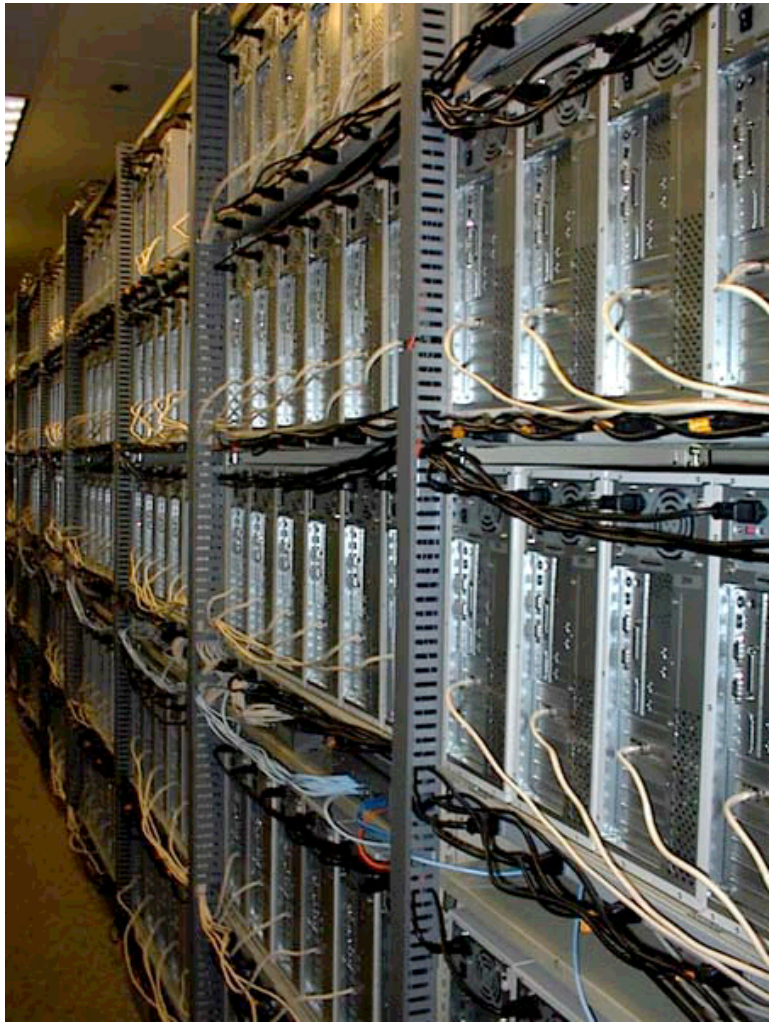


Figure 13 Best circuit from generation 109.

Koza (striped shirt) with 70-node Beowulf cluster

Koza's 1000 node Beowulf used for genetic programming



Other Opportunities

- Parallelism in Computation (local work of Beda & Margileth, and Tom Johnson, CS 152)
- Parallelism optimizing transformations
- Music
- Robotics

GA and Artificial Life (alife)

- Artificial life demonstrates mechanisms that could underlie real life.
- It is not limited to life as we know it, but rather plausible life, including robotics.

Cellular Automata

- It was observed long ago that certain cellular automata can be observed to have interesting “growth” and “reproductive” characteristics.

- This lead to von Neumann’s “Self-Reproducing Automata”, Conway’s “Game of Life”, etc.

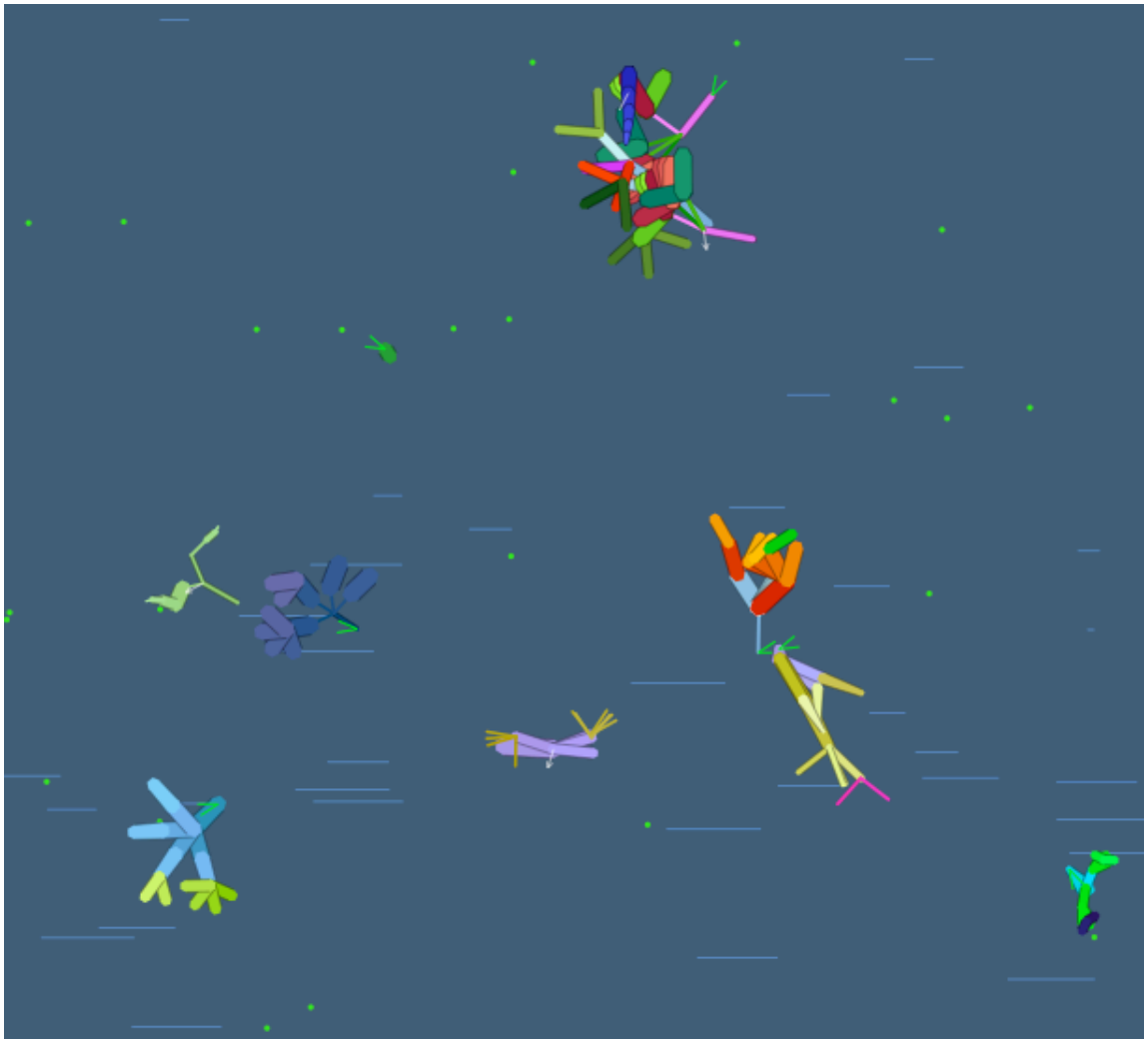
Possible Benefits from Alife

- Not only might we better understand how living things work (DNA/RNA replication, etc.),
- we can also use our observations to improve algorithmic processes not intended to represent life:
  - Swarm intelligence (1989, used in animation, robotics)
  - Ant colony optimization (1992, used for TSP, routing)
  - Artificial immune systems (used for intrusion prevention)
  - etc.

Example of an Alife Program: GenePool, by Jeffrey Ventrella

- Two-dimensional space
- Swimbot: multi-jointed organisms
- Food, energy, growth
- Genetics
  - Sexual reproduction
  - Feature preferences
- GUI features
  - Tracking
  - Zooming and panning
  - Graphing of population
  - Loadable genomes

GenePool Screen Excerpt



Swimbot Mental States

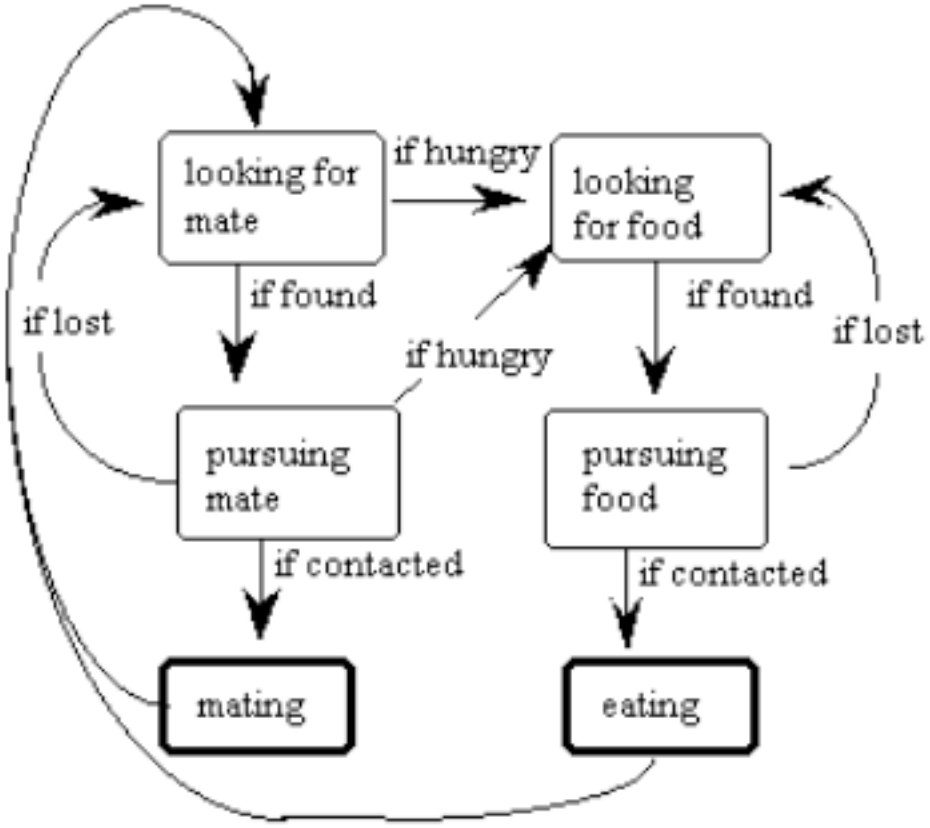


Figure 1.6. Swimbot mental states

Mating

1.2.11 Mating and Birth

When two swimbots mate (i.e., at least one of them is pursuing the other, and the distance between their genitals is less than the length of the genital vector) one offspring appears in-between them, which inherits genetic building blocks from both parents. A standard genetic-algorithm crossover technique is used, and some random mutation occurs in random genes.

Mini-Dramas

- Most Loved: shows the swimbot who has produced the most offspring (as pursued)
- Best at Mating: shows the swimbot who has produced the most offspring (as pursuer)
- Biggest Eater: shows swimbot who has eaten the most food bits
- Mutual Love: shows two swimbots pursuing each other as mates (if found)
- Love Triangle: shows three swimbots in a circular loop of mate pursuit (if found)
- Competition for food: shows group of swimbots pursuing a common food bit (if found)

Discoveries

1.4.2 Celebrating Diversity

One of the attractiveness criteria is "similar color". When this is turned on, swimbots will choose mates whose bodies contain the closest spectrum of colors to their own. One experiment was to encourage interracial mating by adding a new attractiveness criterion called "opposite color" — as shown above. Not surprisingly, when this is turned on, the population converges on a perpetual state of psychedelic diversity.

Bifurcation

1.4.1 Sexual Dimorphism?

In specific simulation runs, an attraction criterion was chosen which was intentionally in conflict with normal pressures for efficient swimming: attraction was set to "still" (i.e., swimbots exhibiting the least amount of motion become the most attractive). The prediction was that this would cause mass extinction. But many populations actually thrived, converging on a distinct bifurcation among body types, with the majority being small and nearly motionless, and a small minority being similar with the exception of having whip-like tails enabling them to swim rapidly. These rapid swimbots (the "breeders") are largely responsible for propagating the genes throughout the population, while the majority of swimbots simply lie around being attractive (the "sitters"). The breeders expend more energy and eat more food bits, while the sitters eat very little and expend very little energy.

1.6 Similar Simulations

A number of alife software simulations share common features with Gene Pool:

- Framsticks [3]: far exceeds Gene Pool in functionality and physical simulation, including features for many variations of 3D simulation and user-manipulation. Like GenePool, Framsticks creatures consist of jointed body parts which rotate against each other.
- SodaPlay [11]: demonstrates great variety of form and motion using 2D graphics, in an entertaining format. SodaPlay uses a more "molecular" style of physics modeling, base on spring forces, to affect the positions and orientations of potentially-large-scale spring structures having semi-coherent positions and orientations.
- LifeDrop [6]: shows intriguing biomorphs breeding in an ethereal setting, with ways to interactively change the view. Like Gene Pool, LifeDrop shows multiple biomorphs interacting at once.