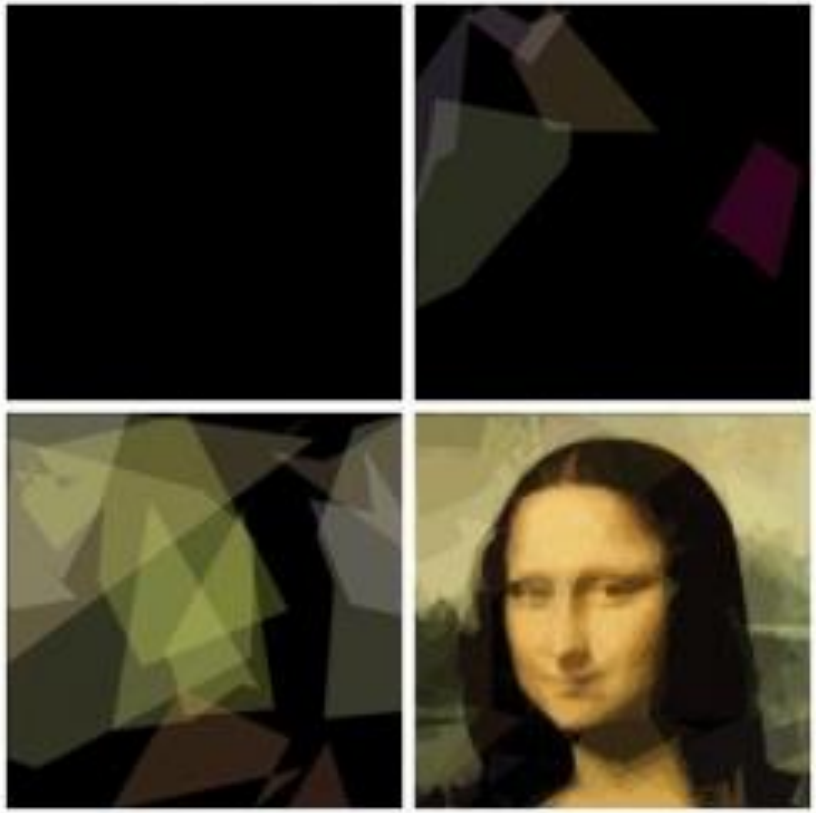
Evolving the Mona Lisa

- Uses only 50 polygons of 6 vertices each.
- Population size of 1, no crossover – parent compared with child, and superior image kept.
- Assuming each polygon has 4 bytes for color (RGBA) and 2 bytes for each of 6 vertices, this image only requires 800 bytes.
- However, compression time is prohibitive and storage is cheaper than processing time. ☹

Examples - GA in the wild

- Image compression – evolving the Mona Lisa

Generation 1



Generation 301

Generation 2716

Generation 904314

http://rogeralsing.com/2008/12/07/genetic-programming-evolution-of-mona-lisa/

https://rogerjohansson.blog/2008/12/07/genetic-programming-evolution-of-mona-lisa/

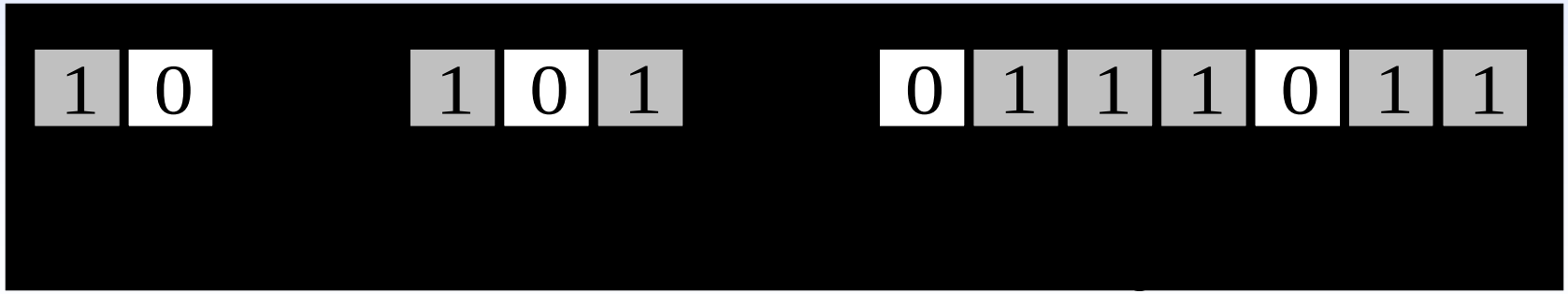
Genetic algorithms: case study

- Suppose it is desired to find the maximum of the "peak" function of two variables:

$$f(x,y) = (1-x)^2 e^{-x^2-(y+1)^2} - (x - x^3 - y^3)\, e^{-x^2-y^2}$$

where parameters x and y vary between –3 and 3.

- The first step is to represent the problem variables as a chromosome – parameters x and y as a concatenated binary string:



- We also choose the size of the chromosome population, for instance 6, and randomly generate an initial population.
- The next step is to calculate the fitness of each chromosome. This is done in two stages.
- First, a chromosome, that is a string of 16 bits, is partitioned into two 8-bit strings:

- Then these strings are converted from binary (base 2) to decimal (base 10):

$$(10001010)_2 = 1\times 2^7 + 0\times 2^6 + 0\times 2^5 + 0\times 2^4 + 1\times 2^3 + 0\times 2^2 + 1\times 2^1 + 0\times 2^0 = (138)_{10}$$

and

$$(00111011)_2 = 0\times 2^7 + 0\times 2^6 + 1\times 2^5 + 1\times 2^4 + 1\times 2^3 + 0\times 2^2 + 1\times 2^1 + 1\times 2^0 = (59)_{10}$$

- Now the range of integers that can be handled by 8-bits, that is the range from 0 to $(2^8 - 1)$, is mapped to the actual range of parameters *x* and *y*, that is the range from –3 to 3:

$$\frac{6}{256-1} = 0.0235294$$

- To obtain the actual values of *x* and *y*, we multiply their decimal values by 0.0235294 and subtract 3 from the results:

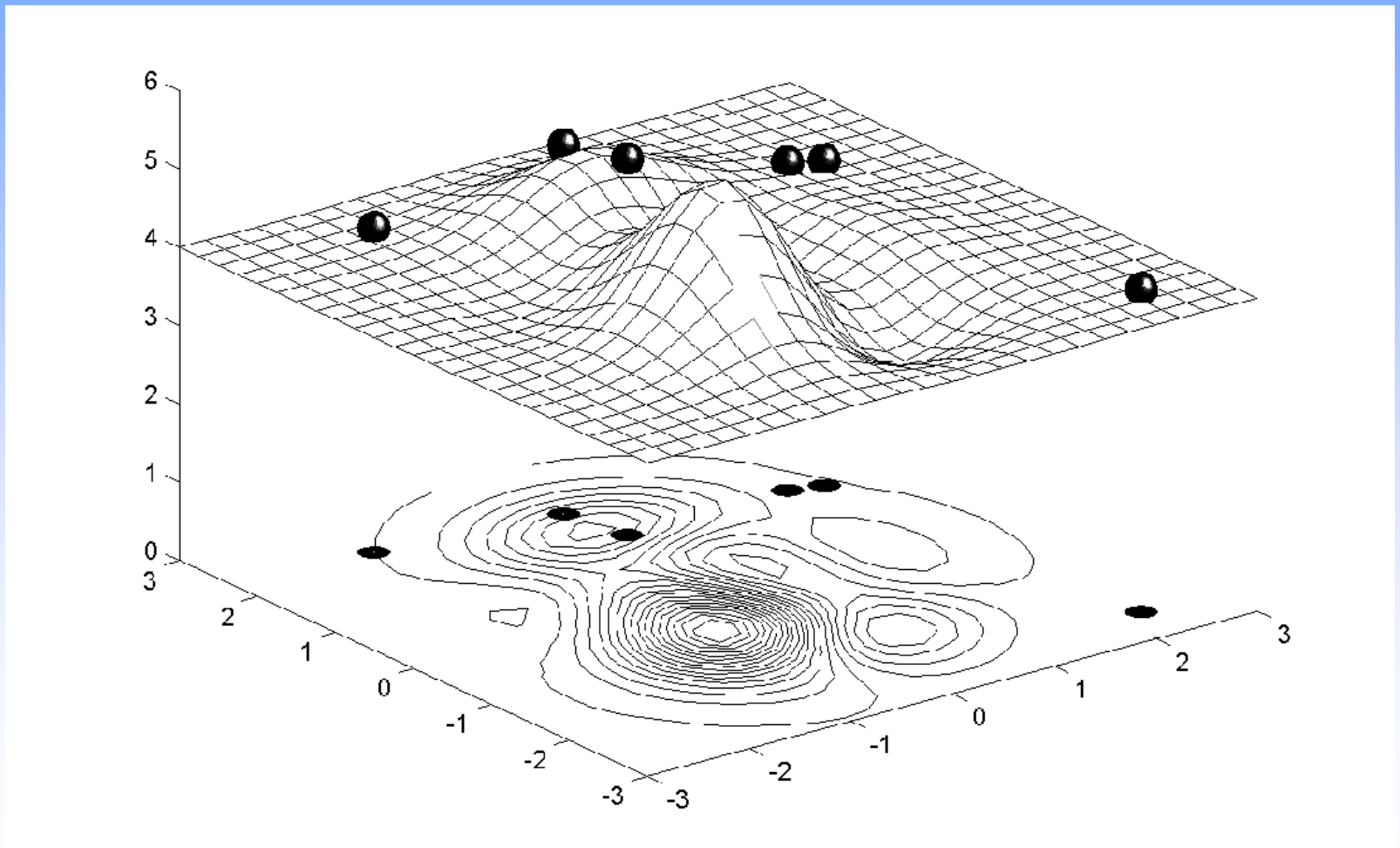
$$(10001010)_2 = 1\times 2^7 + 0\times 2^6 + 0\times 2^5 + 0\times 2^4 + 1\times 2^3 + 0\times 2^2 + 1\times 2^1 + 0\times 2^0 = (138)_{10}$$

and

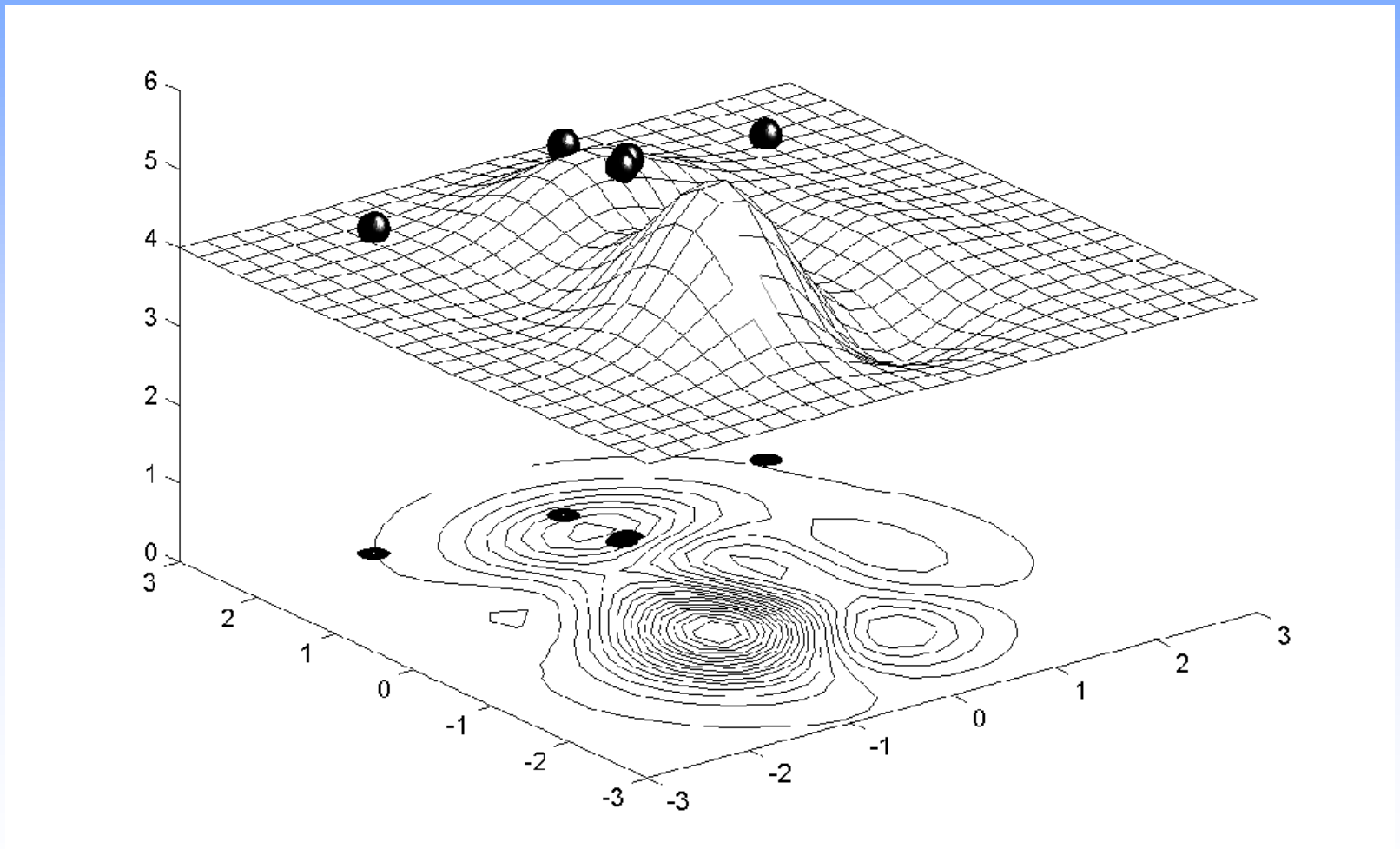
$$(00111011)_2 = 0\times 2^7 + 0\times 2^6 + 1\times 2^5 + 1\times 2^4 + 1\times 2^3 + 0\times 2^2 + 1\times 2^1 + 1\times 2^0 = (59)_{10}$$

- Using decoded values of x and y as inputs in the mathematical function, the GA calculates the fitness of each chromosome.
- To find the maximum of the "peak" function, we will use crossover with the probability equal to 0.7 and mutation with the probability equal to 0.001. As we mentioned earlier, a common practice in GAs is to specify the number of generations. Suppose the desired number of generations is 100. That is, the GA will create 100 generations of 6 chromosomes before stopping.

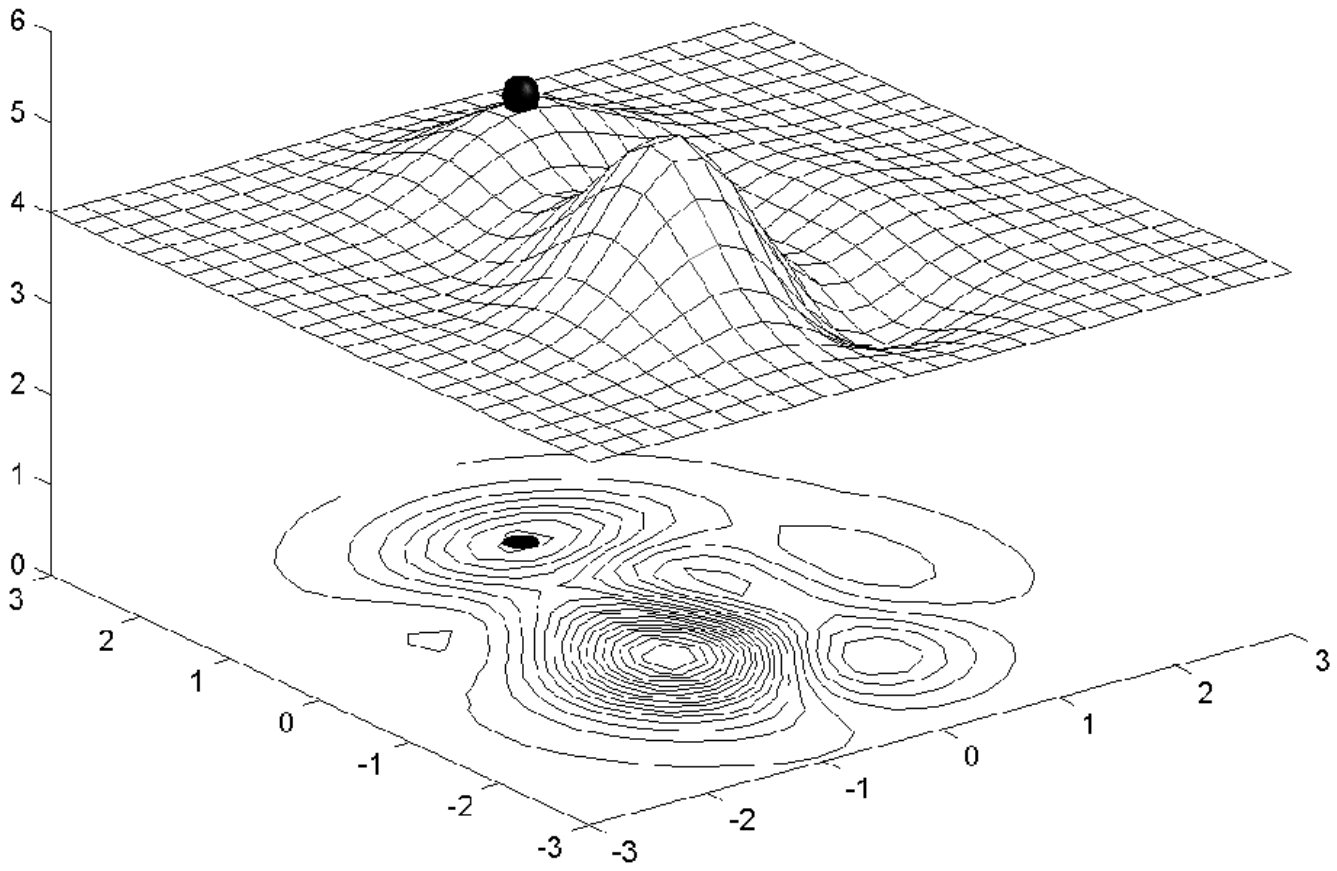
Chromosome locations on the surface of the "peak" function: initial population



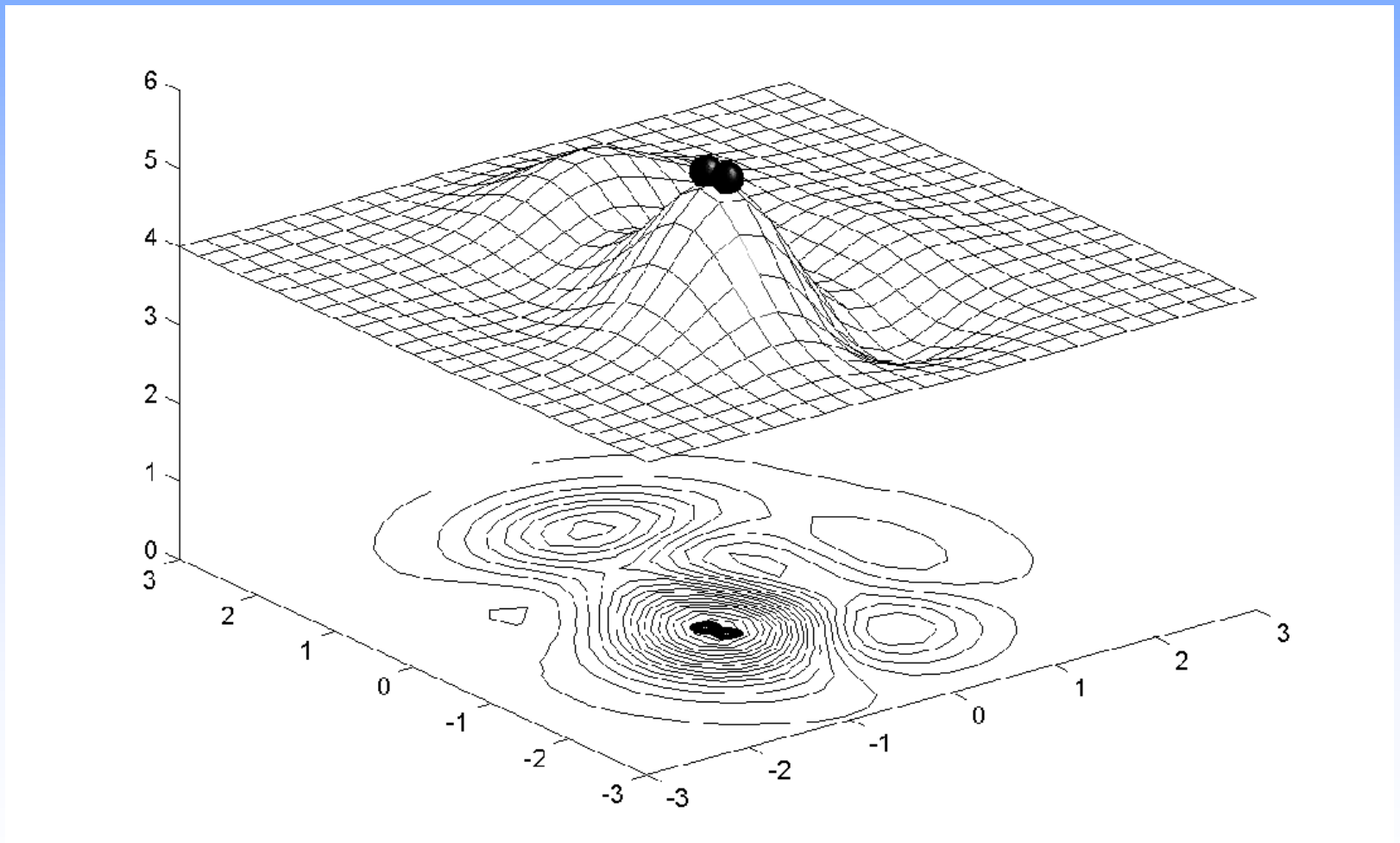
Chromosome locations on the surface of the "peak" function: first generation



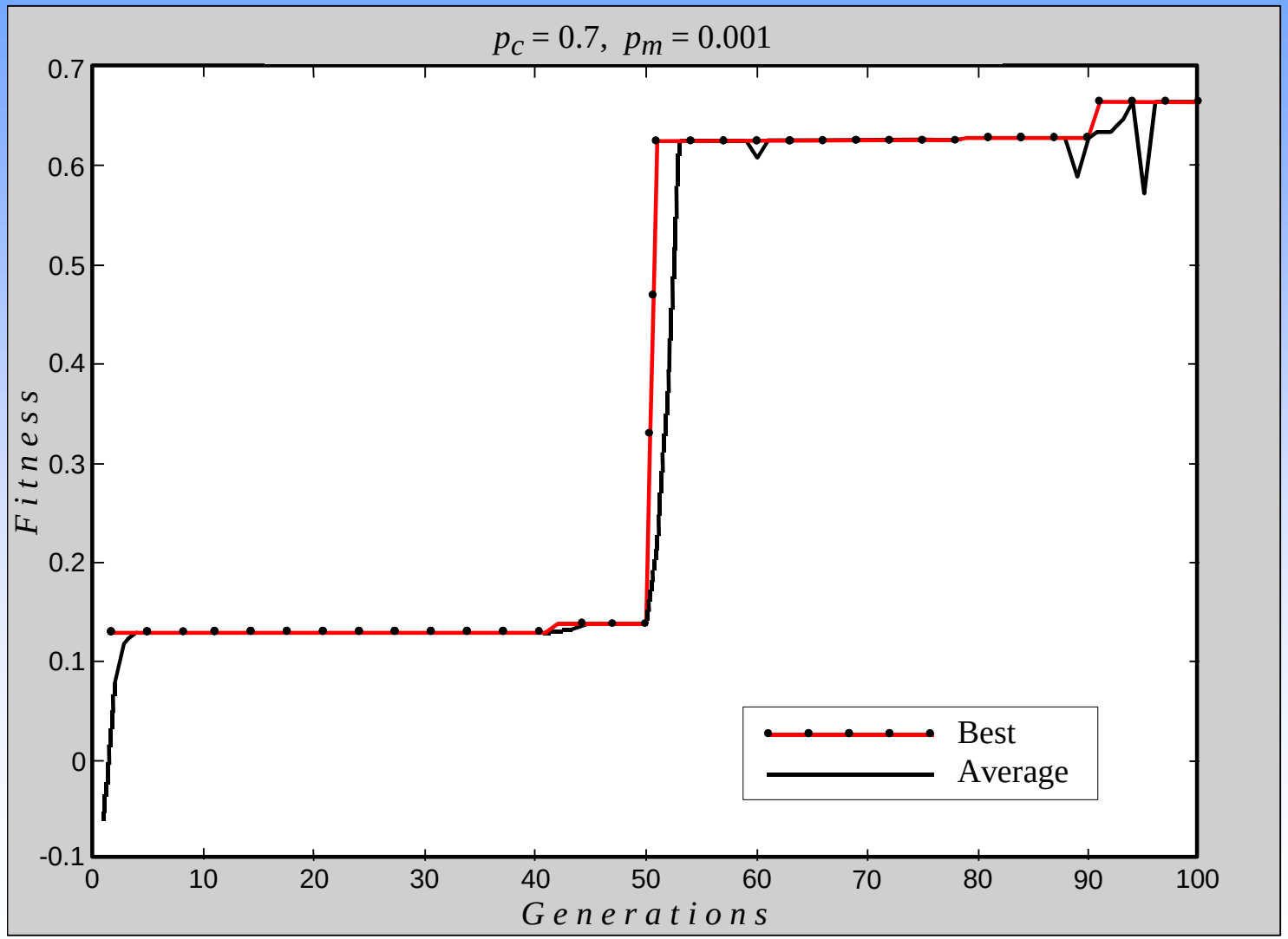
Chromosome locations on the surface of the "peak" function: local maximum



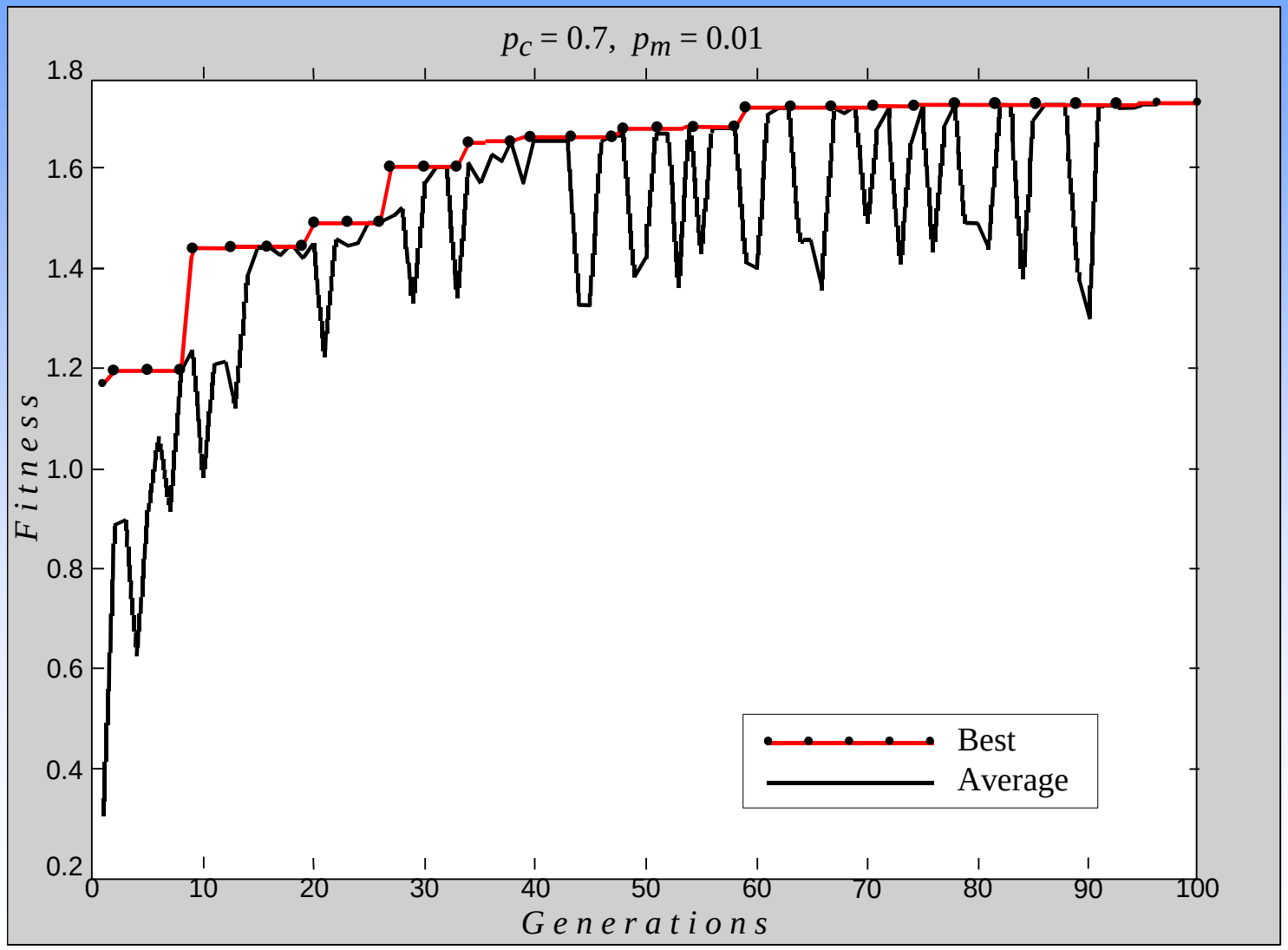
Chromosome locations on the surface of the "peak" function: global maximum



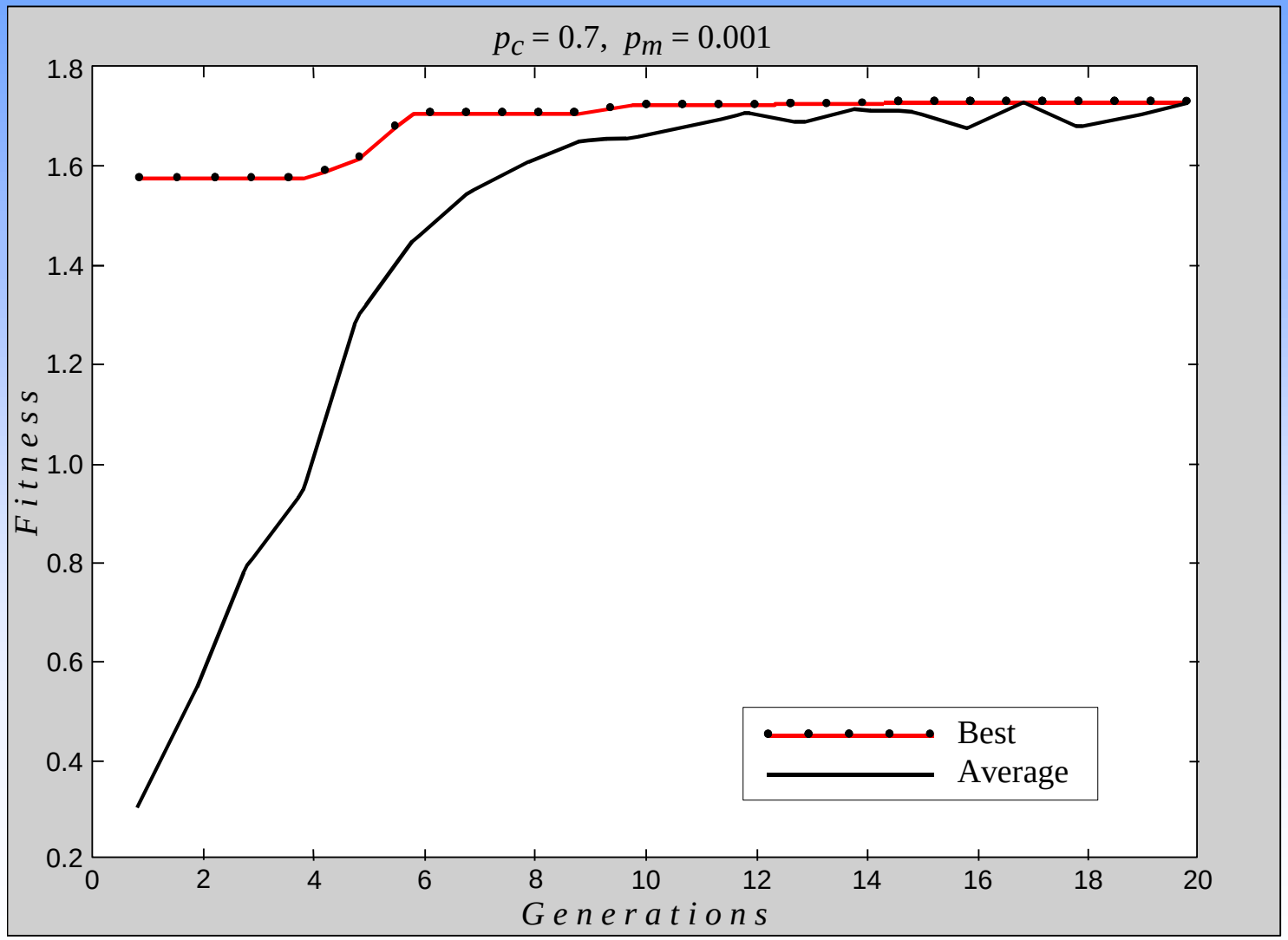
Performance graphs for 100 generations of 6 chromosomes: local maximum



Performance graphs for 100 generations of 6 chromosomes: global maximum



Performance graphs for 20 generations of 60 chromosomes



Applying Genetic Algorithms in Software Architecture Design

Outi Räihä (TaY)

Motivation and problem statement

- Software systems becoming more complex and dynamic ->

  more intricate architectures that need to be able to evolve with changing demands
- How to ease the building of such architectures? A similar problem has been solved in nature through evolution
- Genetic algorithms can be used to simulate evolution in SE
- The main research problem: *Is it possible to build a quality software architecture from high-level requirements, using a genetic algorithm*?

Genetic algorithms - intro

- Non-deterministic algorithms to search through exceptionally large search spaces
- Analogies to biology and evolution – chromosome, mutations, crossover, population, generation, survival of the fittest

Genetic algorithms - execution

- Model the solution as a chromosome
- Create initial population with size p
- Administer mutations at chromosome or gene level
- Administer crossover to create offspring
- Evaluate individuals with fitness function
- Select population for next generation
- Iterate for g generations

Modeling

- Responsibility-based approach
  - A high-level definition of the architecture
  - Straightforward modeling
  - Size of input kept under control
- Each responsibility ri represented as a supergene SGi
  - Basic info: name ni, type di, parameter size pi, execution time ti, frequency fi, call cost ci, variability vi, depending responsibilities Ri = {rj, rk, …rm}
  - Architecture info: class(es) Ci that belongs to, interface Ii implemented, dispatcher Di used, responsibilities communicating through dispatcher RDi, pattern Pi that belongs to

Modeling

Ri	ni	di	ti	pi	fi	ci	vi	Ci	Ii	Di	Pi	RDi

Supergene SGi representing responsibility ri

SG1	SG2	…	SGm-1	SGm

Chromosome with m responsibilities

Mutation and crossover operations

- Mutations reform the architecture, crossovers attempt to combine good parts of two different solutions
- Patterns and dispatcher provide challenges, as many mutations, e.g., merging two classes may unintentionally break a design pattern -> corrective operations must be implemented to ensure the architecture stays coherent
- Mutations are assigned a probability with which they are executed. A roulette wheel selection is made for a mutation and the mutation index is chosen randomly
- Null mutation and crossover also included in the ”wheel”

Genetic Programming

GENETIC PROGRAMMING

THE CHALLENGE

"How can computers learn to solve problems without being explicitly programmed? In other words, how can computers be made to do what is needed to be done, without being told exactly how to do it?"

— Attributed to Arthur Samuel (1959)

CRITERION FOR SUCCESS

"The aim [is] ... to get machines to exhibit behavior, which if done by humans, would be assumed to involve the use of intelligence.“

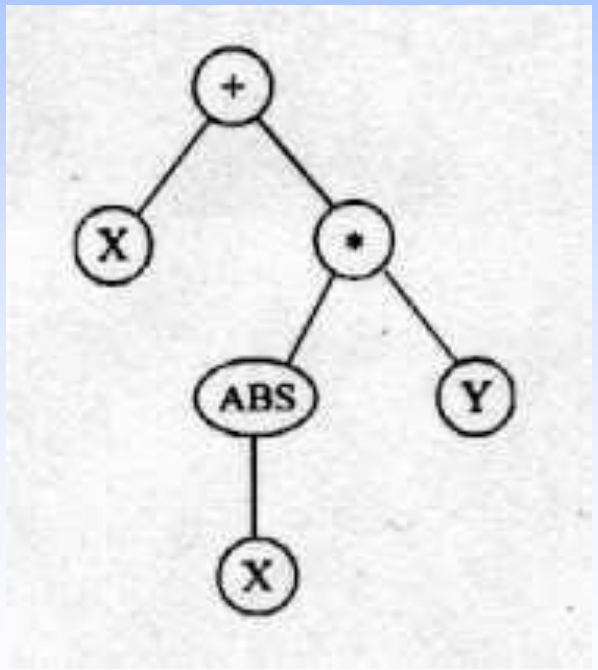
— Arthur Samuel (1983)

Genetic Programming

- John Koza, 1992
- Evolve **program** instead of bitstring
- **Lisp** program structure is best suited
  - Genetic operators can do simple replacements of sub-trees
  - All generated programs can be treated as legal (no syntax errors)

Genetic Programming

- Specialized form of GA
- Manipulates a very specific type of solution using modified genetic operators
- Original application was to design computer programs
- Now applied in alternative areas eg. Analog Circuits
- Does not make distinction between search and solution space.
- Solution represented in very specific hierarchical manner.



Background/History

- By John R. Koza, Stanford University.
- 1992, Genetic Programming Treatise - “Genetic Programming. On the Programming of Computers by Means of Natural Selection.” - Origin of GP.
- Combining the idea of machine learning and evolved tree structures.

Why Genetic Programming?

- It saves time by freeing the human from having to design complex algorithms.
- Not only designing the algorithms but creating ones that give optimal solutions.
- Again, Artificial Intelligence.

What Constitutes a Genetic Program?

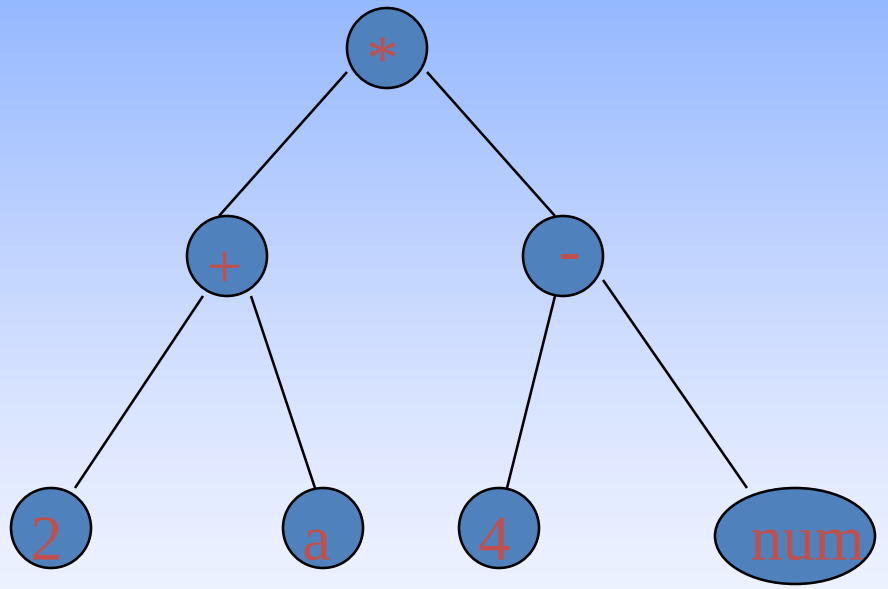
- Starts with "What needs to be done"
- Agent figures out "How to do it"
- Produces a computer program - “Breeding Programs”
- Fitness Test
- Code reuse
- Architecture Design - Hierarchies
- Produce results that are competitive with human produced results

How are Genetic Principles Applied?

- “Breeding” computer programs.
- Crossovers.
- Mutations.
- Fitness testing.

Computer Programs as Trees

- Infix/Postfix
- (2 + a)*(4 - num)



“Breeding” Computer Programs

Hmm hmm heh.
Hey butthead. Do computer programs actually score?

“Breeding” Computer Programs

- Start off with a large “pool” of random computer programs.
- Need a way of coming up with the best solution to the problem using the programs in the “pool”
- Based on the definition of the problem and criteria specified in the fitness test, mutations and crossovers are used to come up with new programs which will solve the problem.

Mutations

Mutations in Nature

Properties of mutations

- Ultimate source of genetic variation.
- **Radiation, chemicals** change genetic information.
- Causes new genes to be created.
- One chromosome.
- Asexual.
- Very rare.

Before:

acgtactggctaa

After:

acatactggctaa

Genetic Programming

- **1.** Randomly generate a combinatorial set of computer programs.
- **2.** Perform the following steps iteratively until a termination criterion is satisfied
  - a. Execute each program and assign a fitness value to each individual.
  - b. Create a new population with the following steps:
    - i. Reproduction: Copy the selected program unchanged to the new population.
    - ii. Crossover: Create a new program by recombining two selected programs at a random crossover point.
    - iii. Mutation: Create a new program by randomly changing a selected program.
- **3.** The best sets of individuals are deemed the optimal solution upon termination

Mutations in Programs

- Single parental program is ***probabilistically selected*** from the population based on fitness.
- **Mutation** point randomly chosen.
  - the subtree rooted at that point is deleted, and
  - a **new subtree is grown** there using the same random growth process that was used to generate the initial population.
- **Asexual operations** (mutation) are typically performed sparingly:
  - with a *low probability* of mutations,
  - probabilistically selected from the population based on fitness.

Crossovers in Programs

Crossovers in Programs

1. Two ***parental programs*** are selected from the population based on fitness.

2. A crossover point is randomly chosen in the first and second parent.
   1. The first parent is called receiving
   2. The second parent is called contributing

3. The subtree rooted at the crossover point of the first parent is deleted

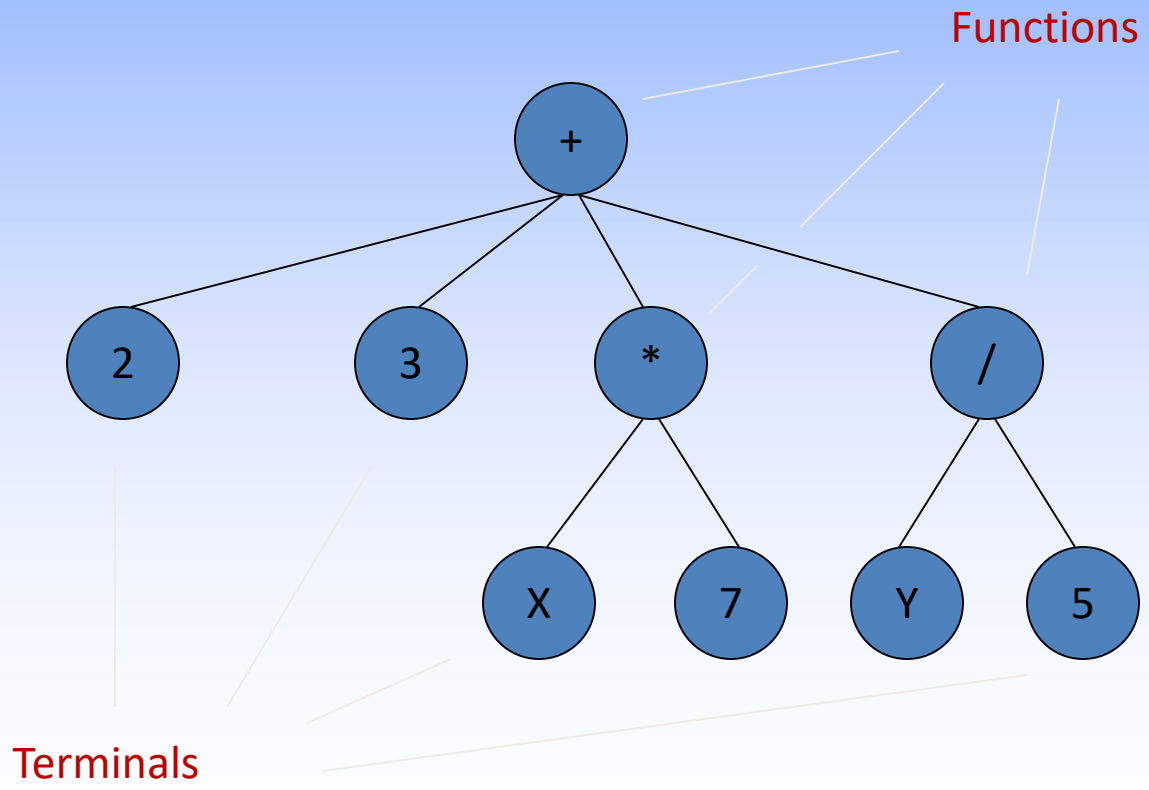
4. It is replaced by the subtree from the second parent.

5. **Crossover** is the *predominant operation* in genetic programming (and genetic algorithm) research

6. It is performed with a high probability (say, 85% to 90%).

Using Trees To Represent Computer Programs

(+ 2 3 (* X 7) (/ Y 5))



Randomly Generating Programs

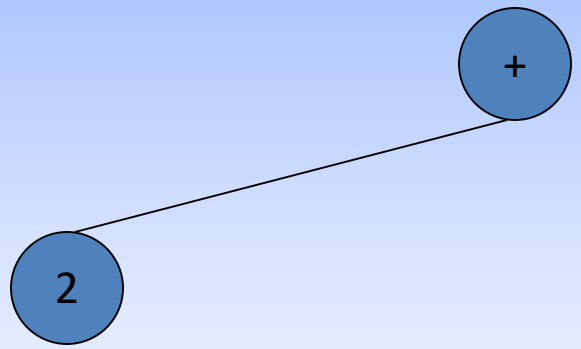
- Randomly generate a program that takes two arguments and uses basic arithmetic to return an answer
  - Function set = {+, -, *, /}
  - Terminal set = {integers, X, Y}
- Randomly select either a function or a terminal to represent our program
- If a function was selected, recursively generate random programs to act as arguments

Randomly Generating Programs

(+ …)

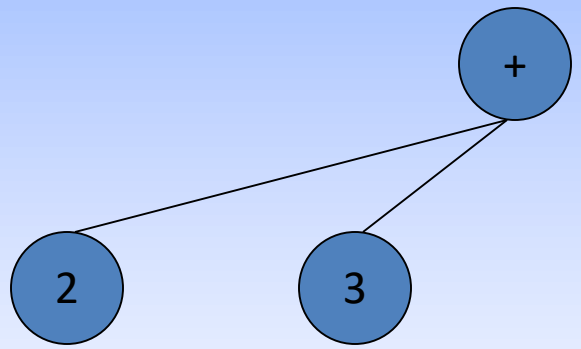
Randomly Generating Programs

(+ 2 …)



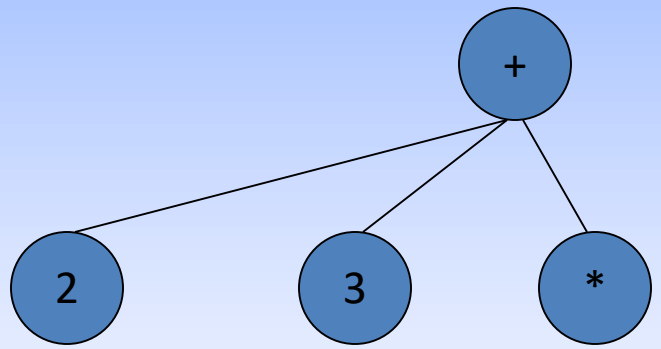
Randomly Generating Programs

(+ 2 3 …)



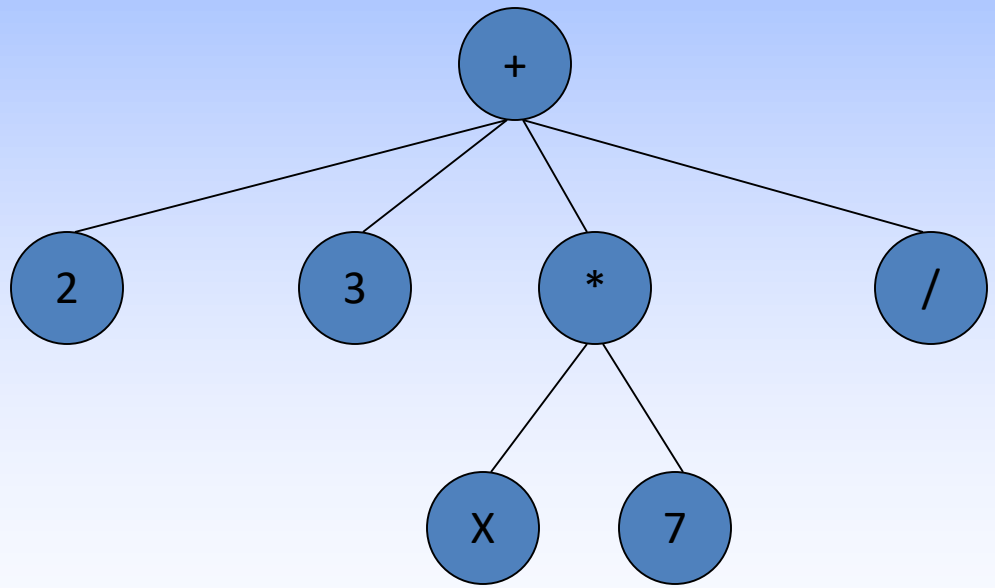
Randomly Generating Programs

(+ 2 3 (* …) …)



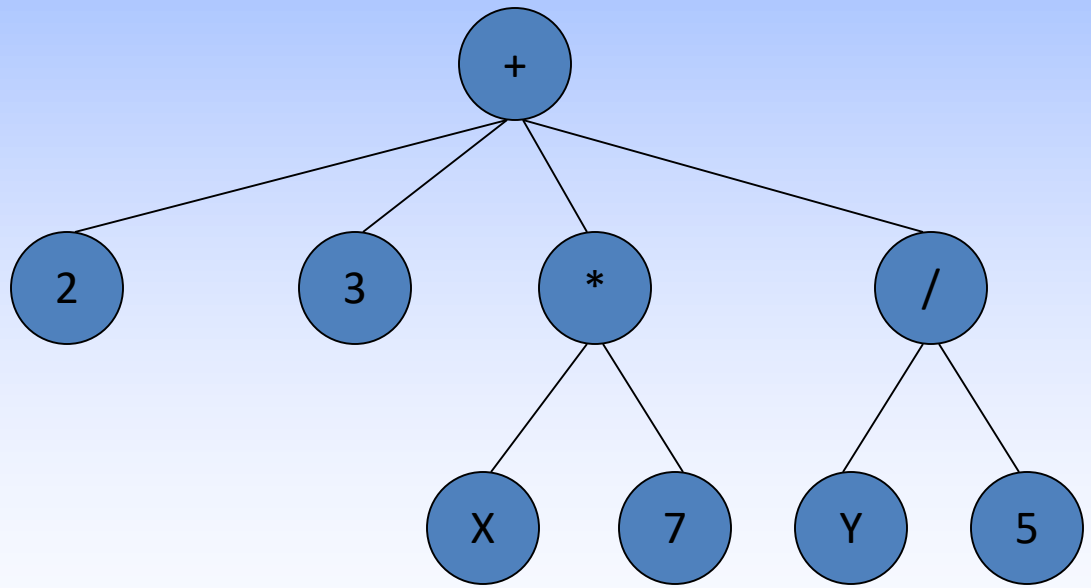
Randomly Generating Programs

(+ 2 3 (* X 7) (/ …))



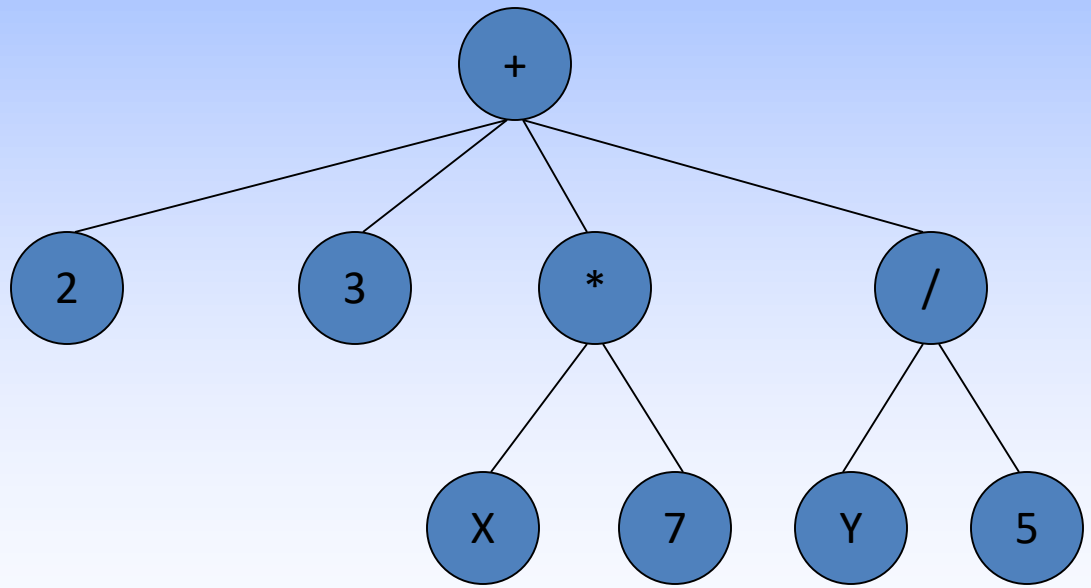
Randomly Generating Programs

(+ 2 3 (* X 7) (/ Y 5))



Mutation

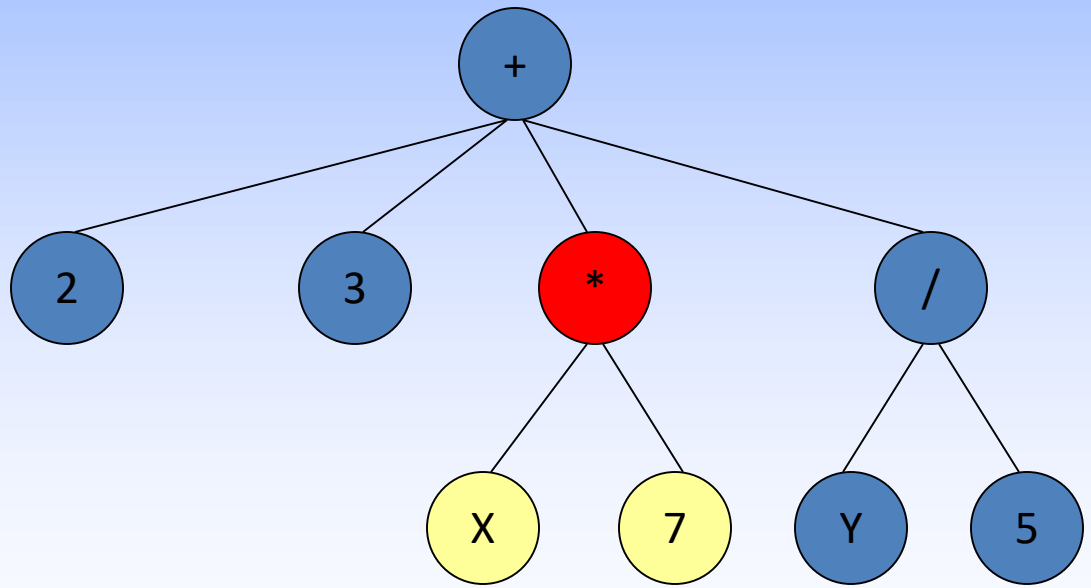
(+ 2 3 (* X 7) (/ Y 5))



Mutation

First pick a random node

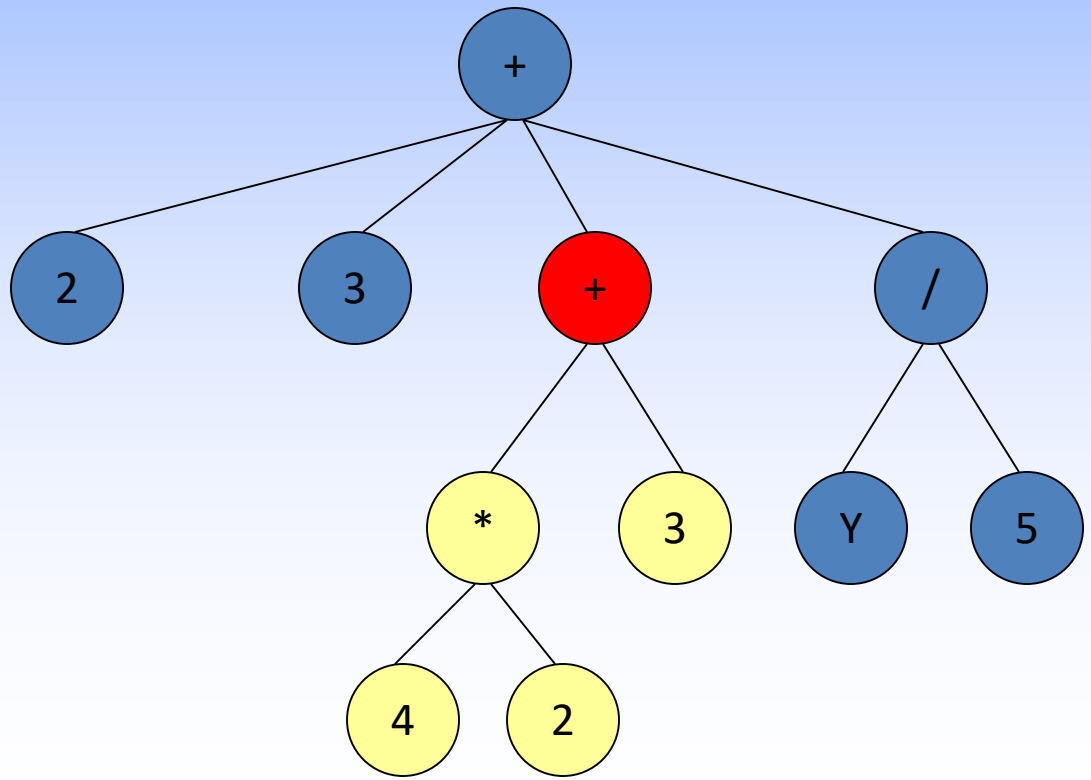
(+ 2 3 (* X 7) (/ Y 5))



Mutation

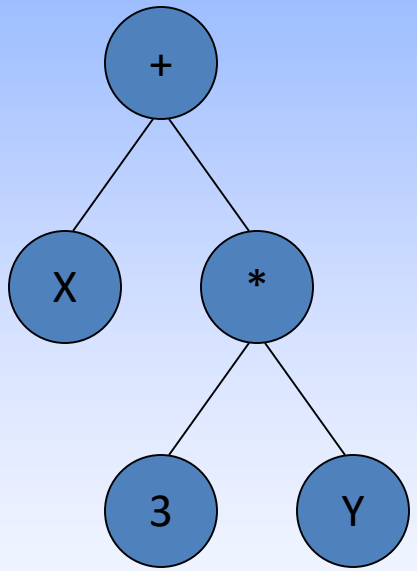
(+ 2 3 (+ (* 4 2) 3) (/ Y 5))

Delete the node and its children, and replace with a randomly generated program

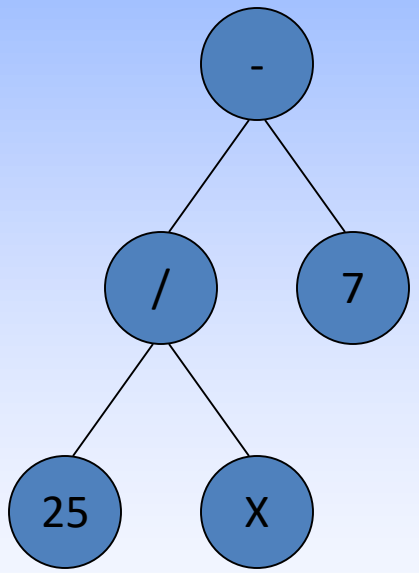


Crossover

(+ X (* 3 Y))

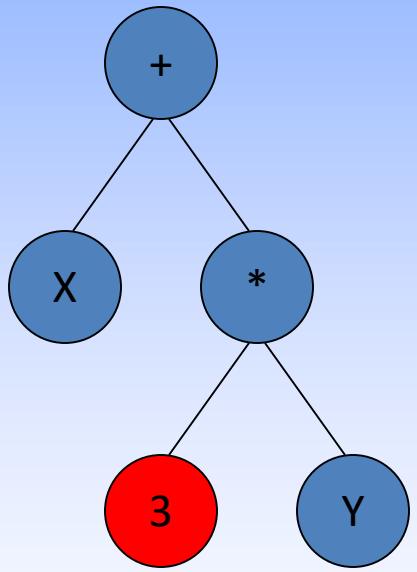


(- (/ 25 X) 7)



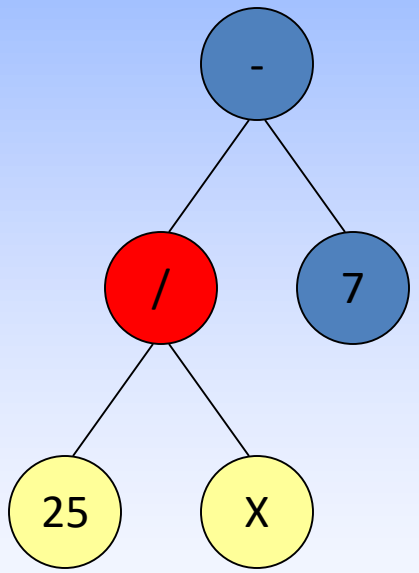
Crossover

(+ X (* 3 Y))



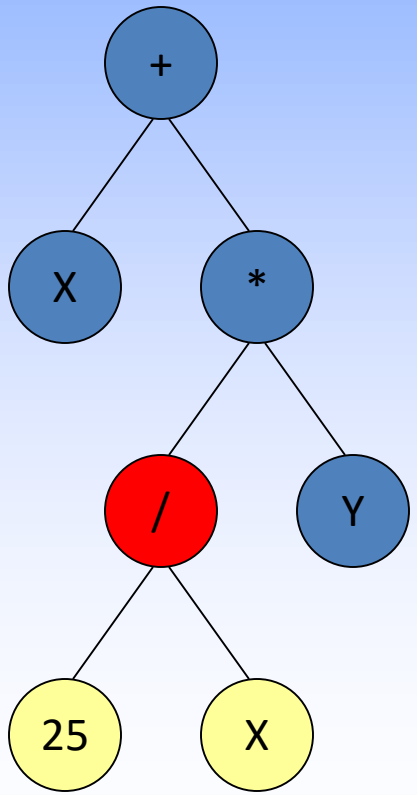
Pick a random node in each program

(- (/ 25 X) 7)



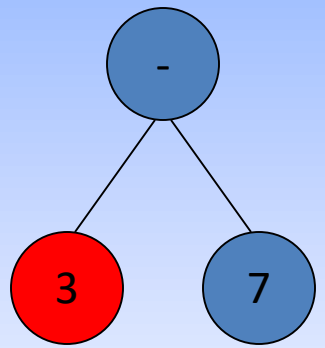
Crossover

(+ X (* (/ 25 X) Y))



Swap the two nodes

(- 3 7)



What About Just Randomly Generating Programs?

- Is Genetic Programming really better than just randomly creating new functions?
- **Yes!**
  - Pete Angeline compared the result of evolving a tic-tac-toe algorithm for 200 generations, with a population size of 1000 per generation, against 200,000 randomly generated algorithms
  - The best evolved program was found to be significantly superior to the best randomly generated program [Genetic Programming FAQ, 2002]
- The key lies in using a fitness measure to determine which functions survive to reproduce in each generation

Applications of GP in robotics

1. Wall-following robot – Koza
   - Behaviors of subsumption architecture of Brooks. Evolved a new behavior.
2. Box-moving robot – Mahadevan
3. Evolving behavior primitives and arbitrators
   - for subsumption architecture
4. Motion planning for hexapod – Fukuda, Hoshino, Levy PSU.
5. Evolving communication agents Iba, Ueda.
6. Mobile robot motion control – Walker.
   - for object tracking
7. Soccer
8. Car racing

Population sizes from 100 to 2000

Real World Applications

- Lockheed Martin Missiles and Space Co. - Near-Minimum-Time Spacecraft Maneuvers [Howley, 96]
- GP applied to the problem of rest-to-rest reorientation maneuvers for satellites
- Optimal time solution is a vector of nonlinear differential equations, which are difficult to solve
- An approximate solution is necessary for a real-time controller
- Results: Rest-to-Rest Maneuver Times (8 test cases)
  - Optimal Solution: 287.93 seconds
  - Expert Solution: 300.3 seconds
  - GP Solution: 292.8 seconds

Real World Applications

- Symbolic Regression
- Problem: Given a set of data points, find a mathematical model

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

Real World Applications

- Neural Network Optimization [Zhang, Mühlenbein, 1993]
- Image Analysis [Poli, 1996a]
- Generation of a knowledge base for expert systems [Bojarczuk, Lopes, Freitas, 2000]
- Fuzzy Logic Control [Akbarzadeh, Kumbla, Tunstel, Jamshidi, 2000]
- Hardware Evolution (Field-Programmable Gate Array) [Thompson, 1997]

What's Next?

- Parallel Distributed Genetic Programming [Poli, 1996b]
  - Operates on graphs rather than parse trees
- Finite State Automata
- Asymetric Recurrent Neural Networks [Pujol, Poli, 1997]

References

- *Genetic Programming FAQ,* 2002. http://www.cs.ucl.ac.uk/research/genprog/gp2faq/gp2faq.html
- Akbarzadeh, M.R., Kumbla, K., Tunstel, E., Jarnshidi, M., "Soft Comuting for Autonomous Robotic Systems", *Computers and Electrivcal Engineering*, 2002: 26, pp. 5-32.
- Bojarczuk, C.C., Lopes, H.S., Freitas, A.A., "Genetic Programming for Knowledge Discovery In Chest-Pain Diagnosis", *IEEE Engineering in Medicine and Biology Magazine,* 2000: 19, v. 4, pp. 38-44.
- Howley, B., "Genetic Programming of Near-Minimum-Time Spacecraft Attitude Maneuvers", *Proceedings of Genetic Programming 1996,* Koza, J.R. et al. (Eds), MIT Press, 1996, pp. 98-109.
- Koza, J., "Genetic Programming as a Means for Programming Computers by Natural Selection", 1994.
- Poli, R., "Genetic Programming for Image Analysis", *Proceedings of Genetic Programming 1996,* Koza, J.R. et al. (Eds), MIT Press, 1996, pp. 363-368.
- Poli, R., "Parallel Distributed Genetic Programming", *Technical* report, The University of Birmingham, School of Computer Science, 1996.
- Pujol, J.C.F., Poli, R., "Efficient Evolution of Asymetric Recurrent Neural Networks Using a Two-dimensional Representation", 1997.
- Thompson, A., "Artificial Evolution in the Physical World", *Evolutionary Robotics: From Intelligent Robots to Artificial Life*, Gomi, T. (Ed.), AAI Books, 1997, pp. 101-125.
- Zhang, B.T., Mühlenbein, H., "Genetic Programming of Minimal Neural Nets Using Occam's Razor", *Proc. Of 5th Int. Conf. On Genetic Algorithms,* 1993, pp. 342-349.

Subset of LISP for Genetic Programming

This is a very very small subset of Lisp

Lisp

- Lisp does not distinguish between **data** and **programs**
  i.e. each object can be data or a program
  → **S-expression**

- Examples for S-expressions:

```
(+ 1 2)              → 3
(+ (* 8 5) 2)        → (+ 40 2) → 42
```

LISP = Famous first language of Artificial Intelligence and Robotics

Lisp

- Every object is either an **atom** or a **list**
- Examples for atoms:
  `7, 123, obj_size`
- Examples for lists:
  `(1 2 3), (+ obj_size 1), (+ (* 8 5) 2)`

- More functions
- Atom, list, cons, car, cdr, numberp, arithmetic, relations. Cond.
- Copying example

A very important concept – Lisp does not distinguish data and programs

Lisp

- Lisp does not distinguish between **data** and **programs** i.e. each object can be data or a program
  → **S-expression**

- Examples for S-expressions:

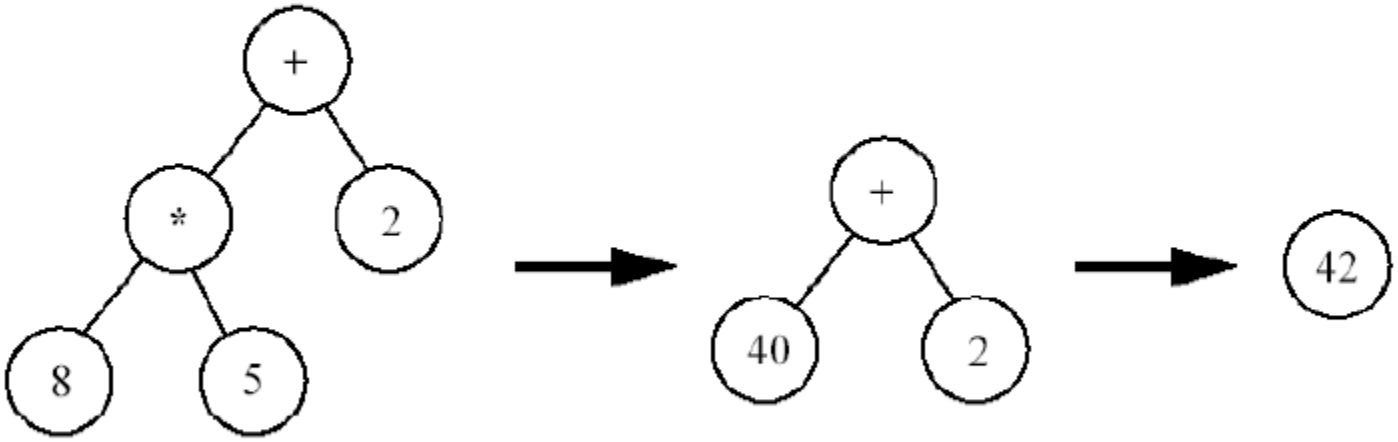
```
(+ 1 2)              → 3
(+ (* 8 5) 2)        → (+ 40 2) → 42
```

Programs in Lisp are trees that are evaluated (calculated)

:

Lisp

Tree-reduction semantics



Lisp allows to define special languages in itself

Robot-Lisp

- Define subset of **Lisp** with robot functions
- Implement **generator** that produces Robot-Lisp programs from population with crossover and mutation
- Implement **interpreter** for Robot-Lisp programs, so these programs can be run (on simulator or real robot) and evaluated (determine fitness)

Robot-Lisp

value

Name	Kind	Semantics
zero	atom, int, constant	0
low	atom, int, constant	20
high	atom, int, constant	40
(INC v)	list, int, function	**Increment** v+1
obj_size	atom, int, image sensor	search image for color object, return height in pixels (0..60)
obj_pos	atom, int, image sensor	search image for color object, return x-position in pixels (0..80) or return –1 if not found
psd_left	atom, int, distance sensor	measure distance in mm to left (0..999)
psd_right	atom, int, distance sensor	measure distance in mm to right (0..999)
psd_front	atom, int, distance sensor	measure distance in mm to front (0..999)
psd_back	atom, int, distance sensor	measure distance in mm to back (0..999)

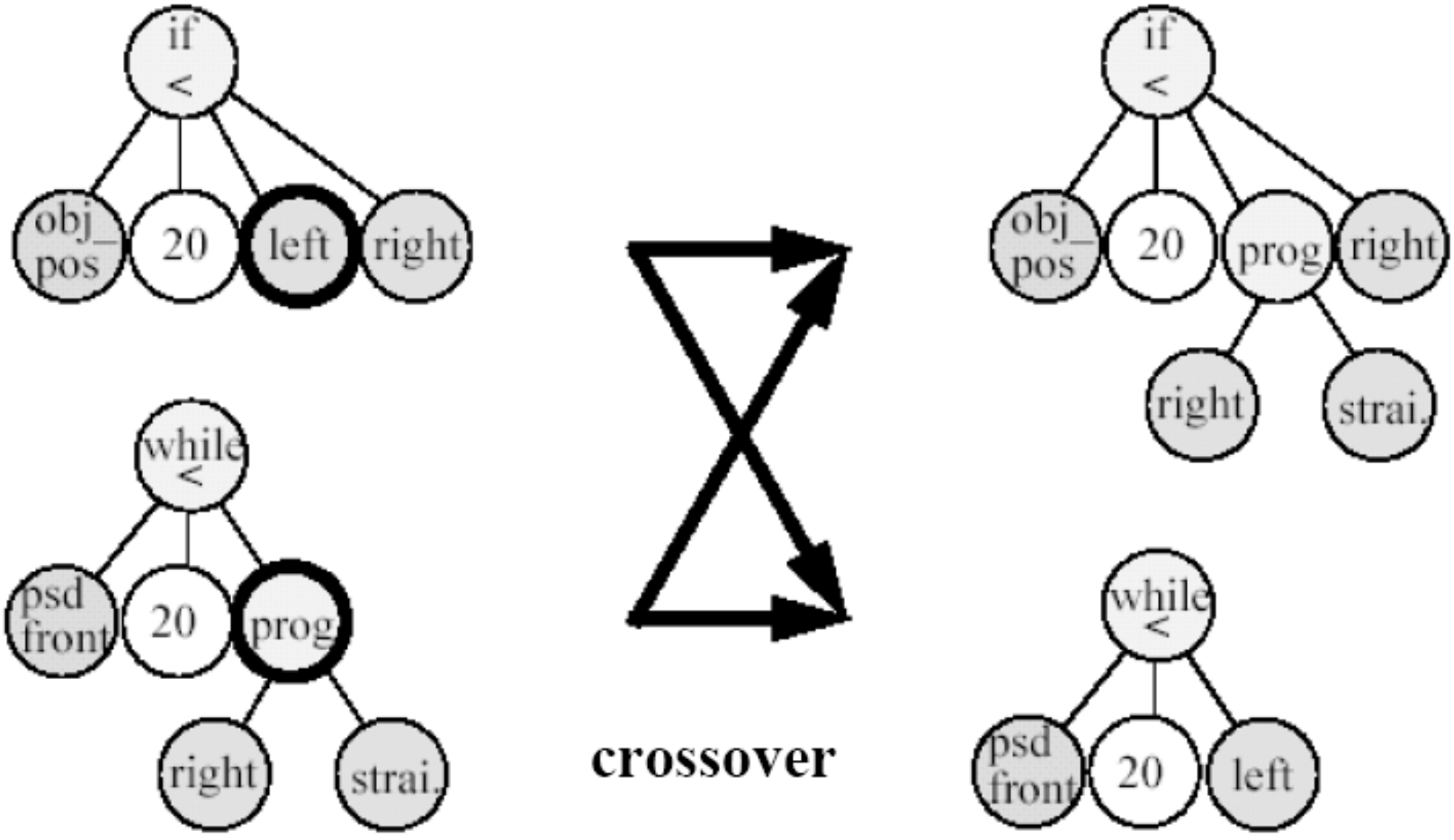
Robot-Lisp cont

statement

`turn_left`	atom, statem., act.	rotate robot 10° to the left
`turn_right`	atom, statem., act.	rotate robot 10° to the right
`drive_straight`	atom, statem., act.	drive robot 10cm forward
`drive_back`	atom, statem., act.	drive robot 10cm backward
`(IF_LESS v1 v2 s1 s2)`	list, statement, program construct	**Selection** if (v_1<v_2) s_1; else s_2;
`(WHILE_LESS v1 v2 s)`	list, statement, program construct	**Iteration** while (v_1<v_2) s;
`(PROGN2 s1 s2)`	list, statement, program construct	**Sequence** s_1; s_2;

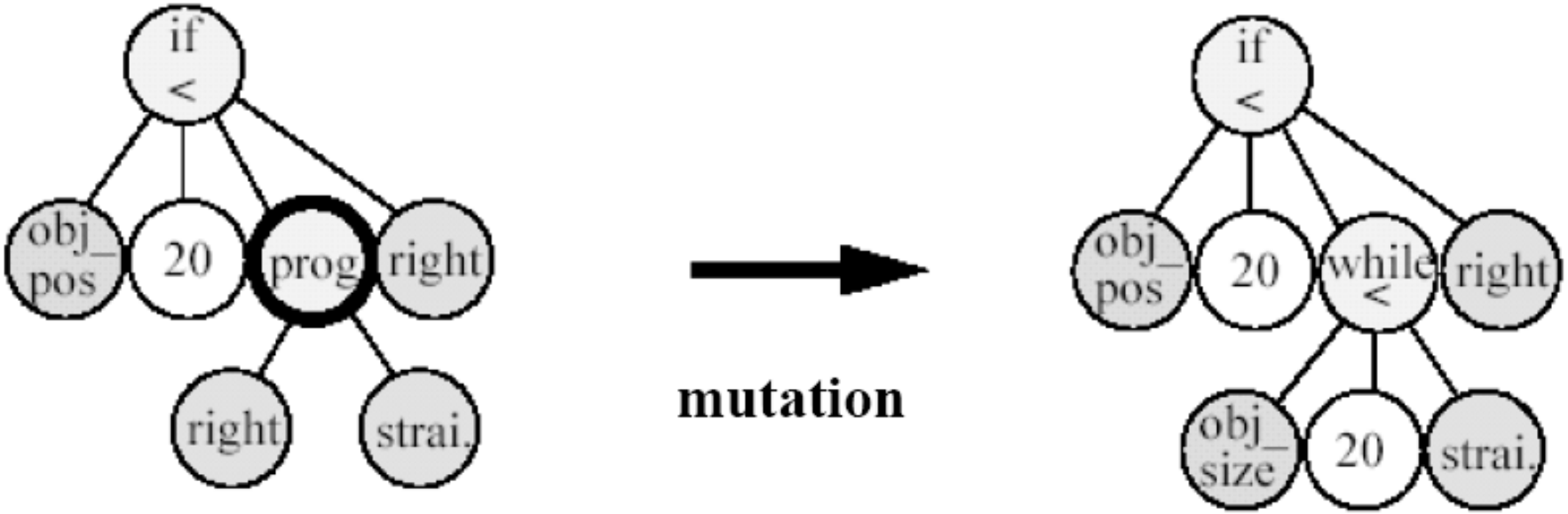
- This subset of Lisp was defined for a mobile robot
- You can define similar subsets for a humanoid robot, robot hand, robot head or robot theatre

Robot-Lisp Generator



As an exercise, you can think about behaviors of all Braitenberg Vehicles described by such programs

Robot-Lisp Generator



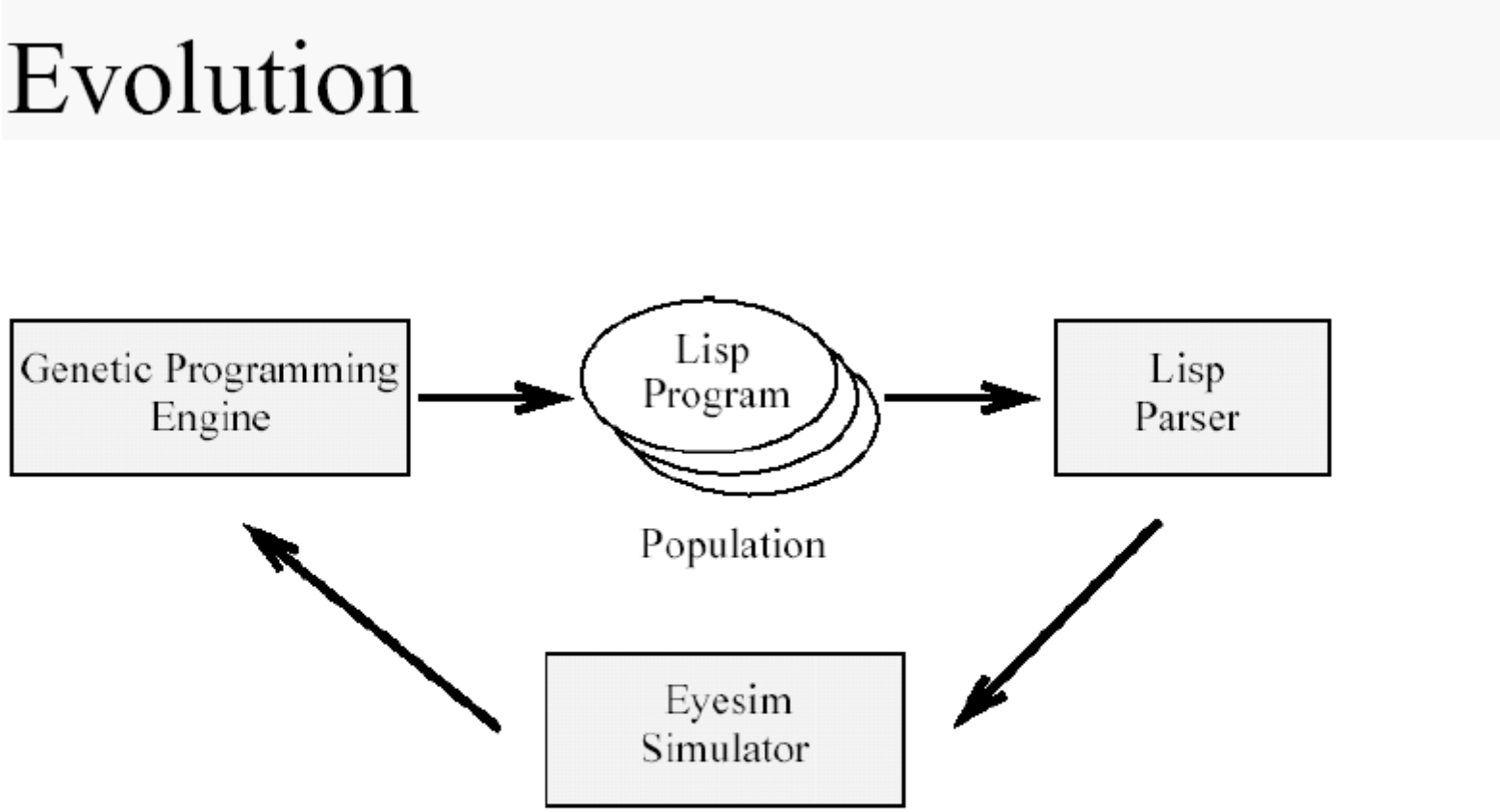
- You can define much more sophisticated mutations that are based on constraints and in general on your knowledge and good guesses (heuristics)

Robot-Lisp Interpreter

```
int compute(Node n)
{ int ret, return_val1, return_val2;
  ...
  switch(n->symbol) {
  case PROGN2:
    compute(n->children[0]);
    compute(n->children[1]);
    break;
  case IF_LESS:
    return_val1 = compute(n->children[0]);
    return_val2 = compute(n->children[1]);
    if (return_val1 <= return_val2) compute(n->children[2]);
      else                          compute(n->children[3]);
    break;
  case turn_left:  turn_left(&vwhandle);
    break;
  case turn_right: turn_right(&vwhandle);
    break;
```

1. You can define your own languages in Lisp
2. You can write your own Lisp-like language interpreter in C++
3. Many on Web

EyeSim simulator allows to simulate robots, environments and learning strategies together, with no real robot



Evolution here takes feedback from simulated environment (simulated)

In our project with teens the feedback is from humans who evaluate the robot theatre performance (subjective)

In our previous project with hexapod the feedback was from real measurement in environment (objective)

Evolution principles are the same for all evolutionary algorithms.

Evolution

Choose percentage of

- Keep best individuals
- Apply crossover
- Apply mutation

Selection Methods

- Fitness proportionate
- Tournament selection
- Linear rank selection
- Truncation selection

- Each individual (Lisp program) is executed on the EyeSim simulator for a limited number of program steps.
- The program performance is rated continually or when finished.