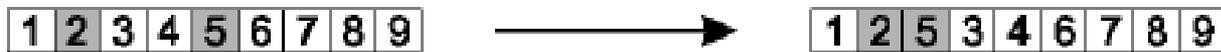


Mutations for Permutations

- Insert mutation:
 - Pick two allele values at random
 - Move the second to follow the first, shifting the rest along to accommodate
 - Note: this preserves most of the order and adjacency information; changes the position of numbers a lot



Removed Adjacency: (2,3), (4,5), (5,6)

Added Adjacency: (2,5), (4,6), (5,3)

Removed orders: 3->5, 4->5

Added orders: 5->3, 5->4

Changed positions: 3, 4, 5

Mutations for Permutations

- Swap mutation:
 - Pick two alleles at random and swap their positions
 - Disrupts adjacency information and order more; preserves positions



Removed Adjacency:	(1,2), (2,3), (4,5), (5,6)
Added Adjacency:	(1,5), (2,6), (4,2), (5,3)
Removed order:	2->3, 2->4, 2->5, 3->5, 4->5
Added order:	5->3, 5->4, 3->2, 4->2, 5->2
Changed positions:	2, 5

Mutations for Permutations

- Inversion mutation:
 - Pick two alleles at random and then invert the substring between them.
 - Preserves most adjacency information (only breaks two links) but disruptive for order information

1 2 3 4 5 6 7 8 9



1 5 4 3 2 6 7 8 9

Mutations for Permutations

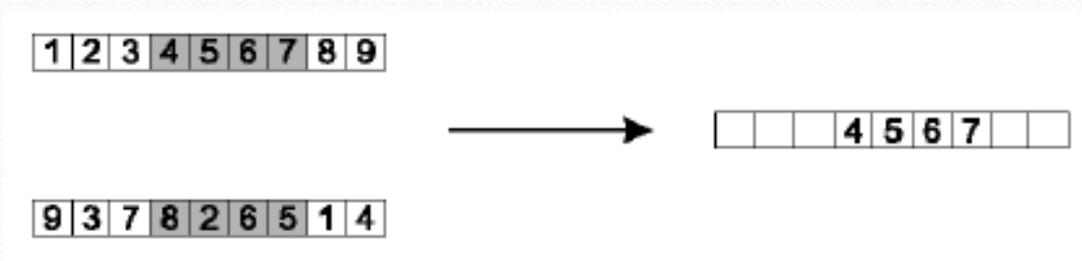
- Scramble mutation:
 - Pick a subset of genes at random (not necessarily consecutive)
 - Randomly rearrange the alleles in those positions



Crossover for Permutations

- Order one crossover:

- Choose an arbitrary part from the first parent, copy this part to the first child



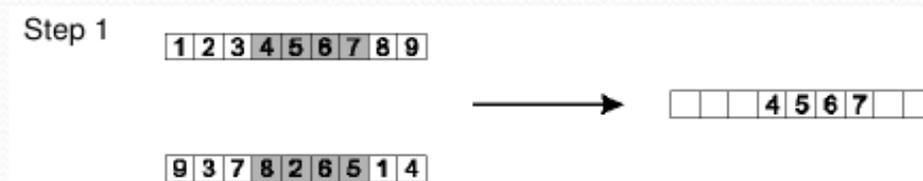
- Copy the numbers that are not in the first part, to the first child:
 - starting right from cut point of the copied part,
 - using the order of the second parent and wrapping around at the end



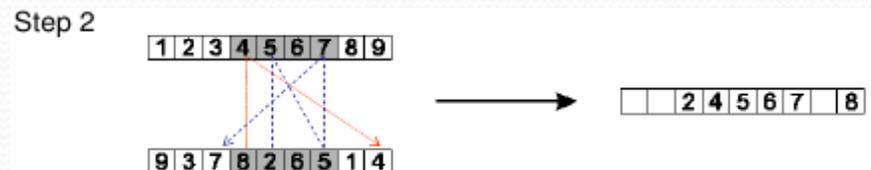
- Analogous for the second child, with parent roles reversed

Crossover for Permutations

- Partially Mapped Crossover (PMX):
 - Choose random segment and copy it from P_1

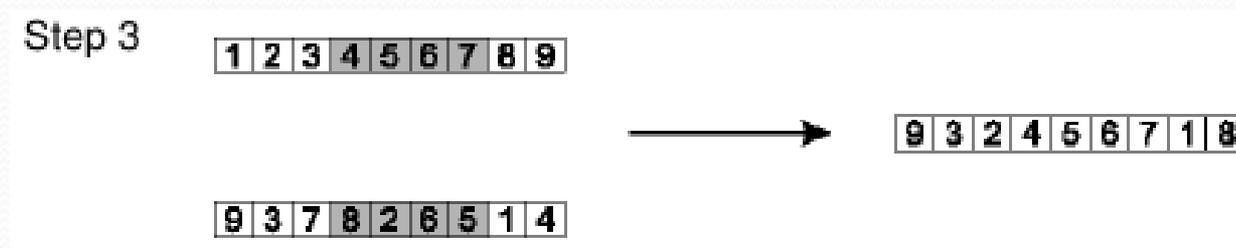


- Starting from the first crossover point look for elements in that segment of P_2 that have not been copied
- For each of these i look in the offspring to see what element j has been copied in its place from P_1
- Place i into the position occupied j in P_2 , since we know that we will not be putting j there (as is already in offspring)
- If the place occupied by j in P_2 has already been filled in the offspring k , put i in the position occupied by k in P_2



Crossover for Permutations

- Partially Mapped Crossover (PMX):
 - Having dealt with the elements from the crossover segment, the rest of the offspring can be filled from P₂.



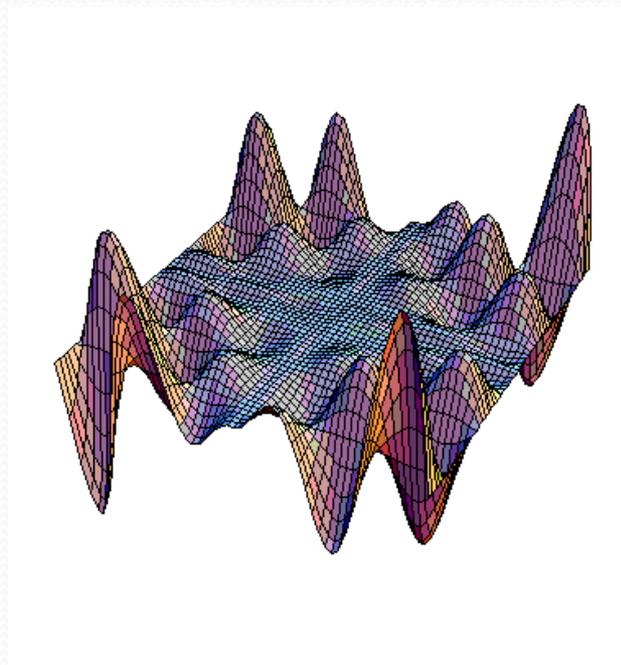
- Idea: maintain position

Order vs Position in Permutations

- Order, but not position of numbers is important in problems such as the traveling salesman problem (visiting all cities in a certain order)
- Position, but not order of numbers is important in problems such as allocating visitors in hotels to rooms (visitors have to be allocated once to one room, but the order of the allocation does not matter)

Evolutionary Strategies

- Numerical optimization problems:
 - **Given** a function f from real numbers to a real number
 - **Find** coordinates at which f is maximized



Evolutionary Strategies

- Main idea:
individuals consist of vectors of real numbers
(not binary)
- Redefinitions of
 - selection
 - crossover
 - mutation
- Operations executed in the order
crossover → mutation → selection

ES: Selection

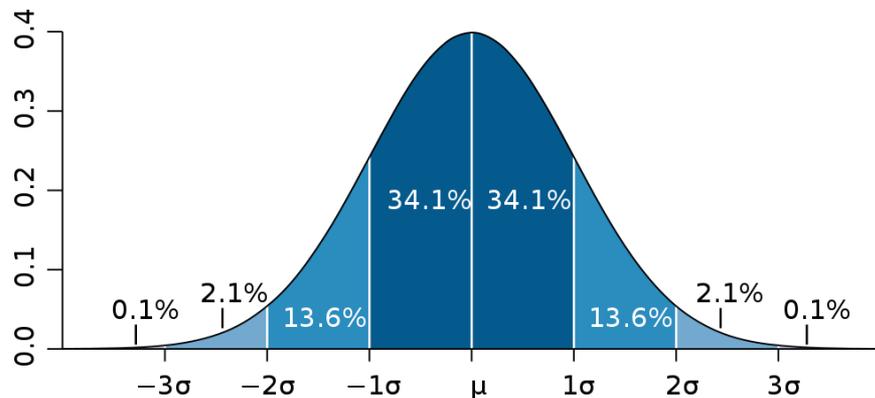
- Not performed *before* mutation and crossover, but after these operations
- It is assumed mutation (& crossover) generate $\lambda > \mu$ individuals (where μ is population size) (typically $\lambda \approx 7\mu$)
- Deterministically eliminate worst individuals from
 - children only: (μ, λ) -ES \rightarrow escapes from local optima more easily (Notational convention)
 - parents and children: $(\mu + \lambda)$ -ES \rightarrow doesn't forget good solutions (“elitist selection”)

ES: Basic Mutation

- An individual is a vector $\vec{h} = (x_1, \dots, x_n)$
- Mutate each x_i by sampling a change from a normal distribution:

$$x_i \leftarrow x_i + \Delta x_i \quad \text{where} \quad \Delta x_i \sim N(0, \sigma)$$

“sampled from”



Simple modification:
mutation rate for each x_i

Major question:
How to set σ or σ_i ?

MAIN IDEA: make search more efficient
by increasing mutation rate if this seems safe

ES: Basic Mutation

- An algorithm for setting global σ : Improved fitness

- Count the number G_s of successful mutations

- Compute the ratio of successful mutations

$$p_s = G_s / G$$

- Update strategy parameters according to

$$\sigma_i = \begin{cases} \sigma_i / c & \text{if } p_s > 0.2 \\ \sigma_i \cdot c & \text{if } p_s < 0.2 \\ \sigma_i & \text{if } p_s = 0.2 \end{cases}$$

$c \in [0.8, 1.0]$

Increase mutation rate as it appears better solutions are far away
“1/5 rule”

until termination

Basic (1+1) ES

- Common use of the 1/5 rule

```
t := 0;
initialize P(0) := {x̄(0)} ∈ I, I = IRn, x̄ = (x1, ..., xn);
evaluate P(0) : {f(x̄(0))}
while not terminate(P(t)) do
  mutate: x̄'(t) := m(x̄(t))
    where x̄'_i := x_i + σ(t) · N_i(0, 1) ∀i ∈ {1, ..., n}
  evaluate: P'(t) := {x̄'(t)} : {f(x̄'(t))}
  select: P(t + 1) := s(1+1)(P(t) ∪ P'(t));
  t := t + 1;
  if (t mod n = 0) then
    σ(t) := 
$$\begin{cases} \sigma(t - n)/c & , \text{ if } p_s > 1/5 \\ \sigma(t - n) \cdot c & , \text{ if } p_s < 1/5 \\ \sigma(t - n) & , \text{ if } p_s = 1/5 \end{cases}$$

    where ps is the relative frequency of successful
      mutations, measured over intervals of,
      say, 10 · n trials;
    and 0.817 ≤ c ≤ 1;
  else
    σ(t) := σ(t - 1);
  fi
od
```

ES Mutation:

Strategy Parameters

- An individual is a vector $\vec{h} = (x_1, \dots, x_n, \sigma)$
or $\vec{h} = (x_1, \dots, x_n, \sigma_1, \dots, \sigma_n)$
where the σ_i are the standard deviations
- Mutate strategy parameter(s) first
Order is important!
- If the resulting child has high fitness, it is assumed that:
 - quality of phenotype is good
 - quality of strategy parameters that led to this phenotype is good

ES Mutation: Strategy Parameters

- Mutation of one strategy parameter

$$\bar{a} = ((x_1, \dots, x_n), \sigma)$$

$$\bar{a}' = ((x'_1, \dots, x'_n), \sigma')$$

$$\sigma' = \sigma \cdot \exp(\tau_0 \cdot N(0,1))$$

$$x'_i = x_i + \sigma' \cdot N_i(0,1)$$

Individual before mutation

Individual after mutation

1.: Mutation of step sizes

2.: Mutation of objective variables

Here the new σ' is used!

ES Mutation: Strategy Parameters

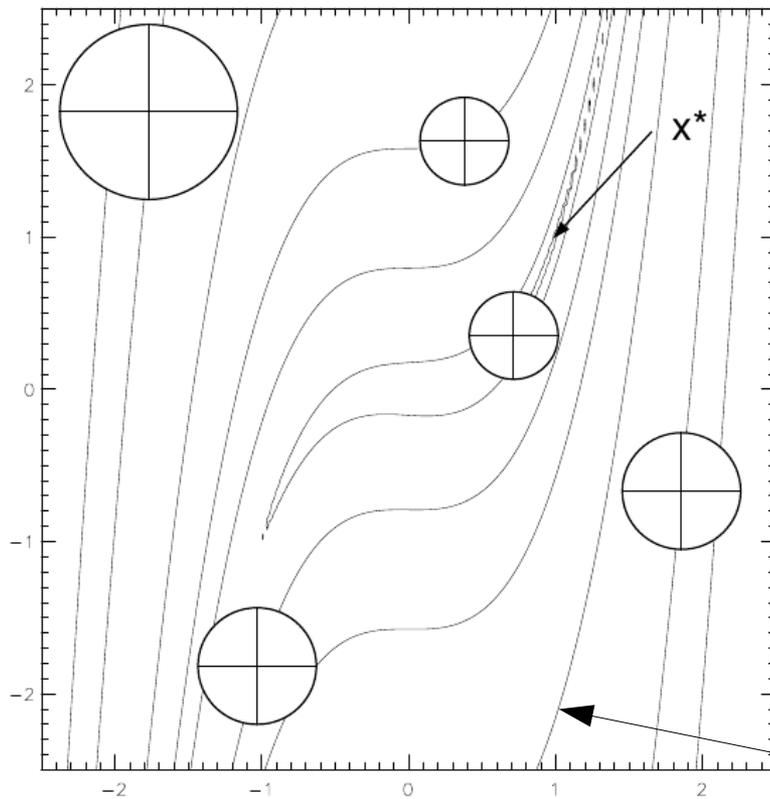
- Here τ_0 is the mutation rate
 - τ_0 bigger: faster but more imprecise
 - τ_0 smaller: slower but more imprecise
- Recommendation for setting τ_0 :

$$\tau_0 = \frac{1}{\sqrt{n}}$$

*H.-P. Schwefel: Evolution and Optimum Seeking, Wiley, NY, 1995.

ES Mutation: Strategy Parameters

⊕ equal probability to place an offspring



- One parameter for each individual
- 2 dimensional genotype

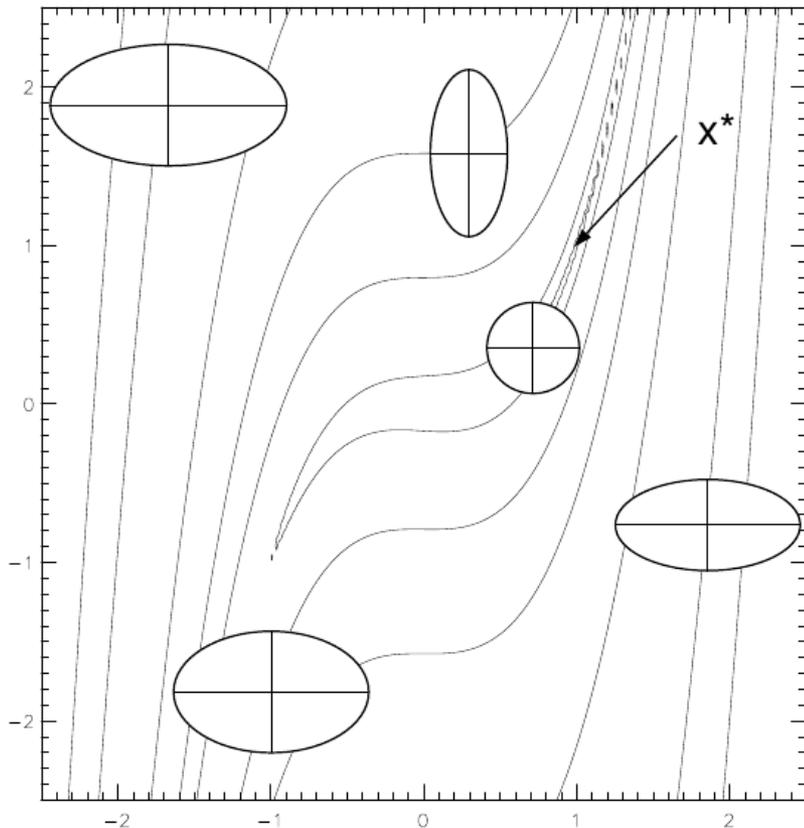
$$\vec{h} = (x_1, x_2, \sigma)$$

- 5 individuals

Line indicates points with equal fitness

ES Mutation: Strategy Parameters

 equal probability to place an offspring



- One parameter for each dimension
- 2 dimensional genotype
$$\vec{h} = (x_1, x_2, \sigma_1, \sigma_2)$$
- 5 individuals

ES Mutation: Strategy Parameters

- Mutation of all strategy parameters

$$\begin{aligned}\sigma'_i &= \sigma_i \cdot \exp(\tau' \cdot N(0, 1) + \tau \cdot N_i(0, 1)) \\ x'_i &= x_i + \sigma'_i \cdot N_i(0, 1)\end{aligned}$$

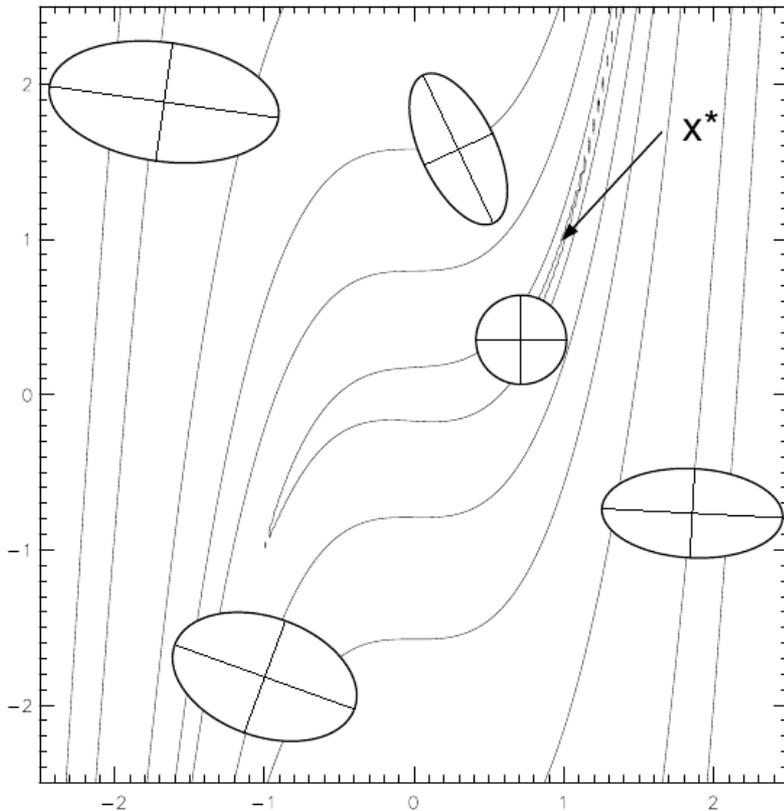
Sample from normal distribution,
the same for all parameters

Update for this specific parameter

ES Mutation: Strategy Parameters



equal probability to place an offspring



- An individual is a vector

$$\vec{h} = (x_1, \dots, x_n, \sigma_1, \dots, \sigma_n, \alpha_1, \dots, \alpha_m)$$

where α_i encode angles

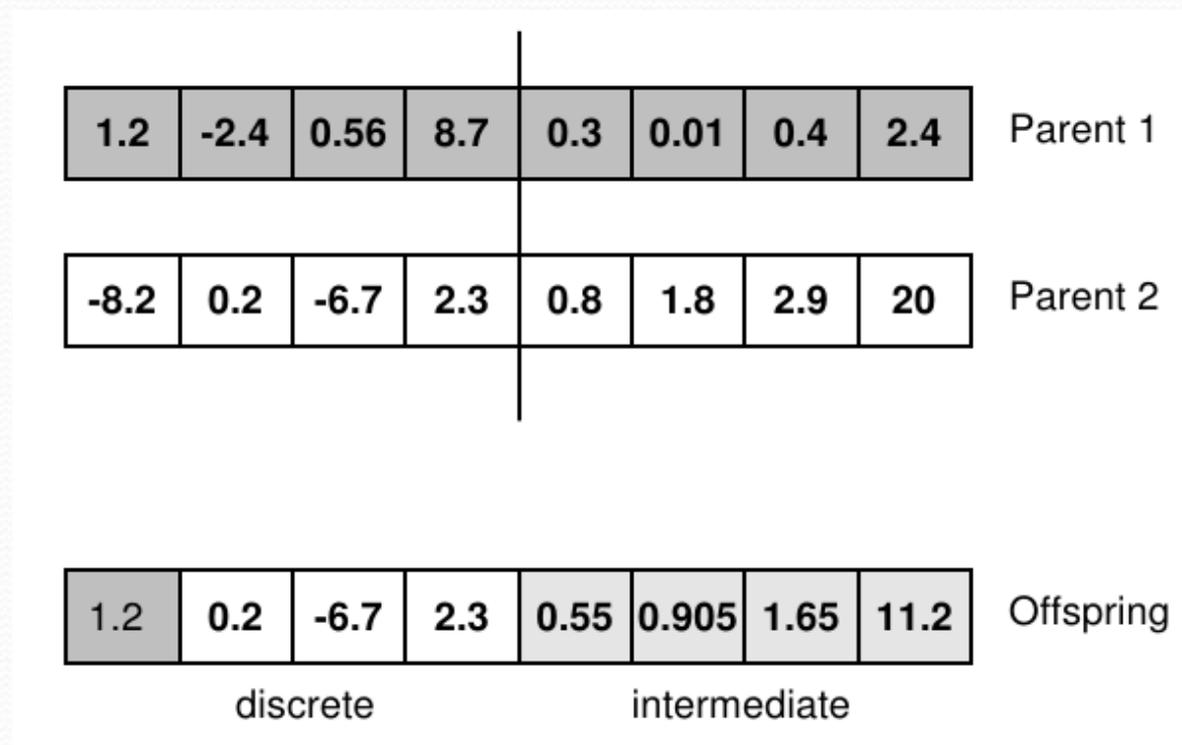
- Also here mutation can be defined
- Mathematical details skipped

ES Crossover / Recombination

- Application of operator creates **one** child (not two)
- Is applied λ times to create an offspring population of size λ (on which then mutation and selection is applied)
- Per offspring gene two parent genes are involved
- Choices:
 - combination of two parent genes:
 - average value of parents (*intermediate recombination*)
 - value of one randomly selected parent (*discrete recombination*)
 - choice of parents:
 - a different pair of parents for each gene (*global recombination*)
 - the same pair of parents for all genes

ES Crossover / Recombination

- Default choice: discrete recombination on phenotype, intermediate recombination on strategy parameters



GAs vs. ES

Genetic algorithms

- Crossover is the main operator
- Uses also mutation
- Encoding for problem representation
- Biased selection of the parents
- Algorithm parameters often fixed
- Selection → Crossover → Mutation

Evolution strategies

- Mutation is the main operator
- Uses also crossover (recombination)
- No encoding needed for problem representation
- Random selection of the parents
- Adaptive set of algorithm parameters (strategy parameters)
- Crossover → Mutation → Selection

Genetic Programming

- Goal: to learn computer programs from examples (like in machine learning and data mining)
- Main idea:
represent (simple) *computer programs* in individuals of arbitrary size
- Redefinitions of
 - selection
 - crossover
 - mutation

Individuals are Program Trees / Parse Trees

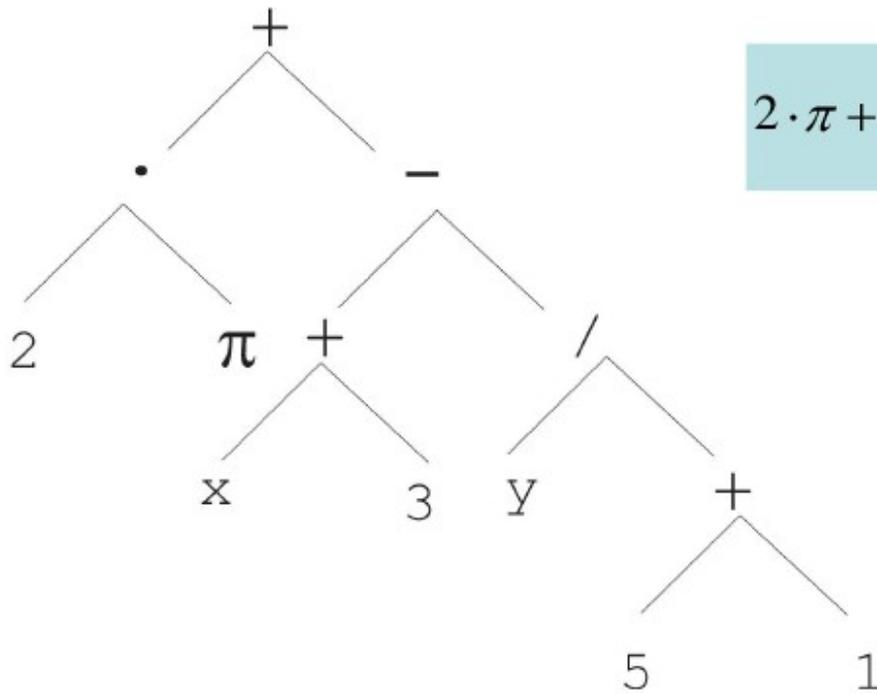
- Representation of
 - Arithmetic formulas
 - Logical formulas
 - Computer programs

$$2 \cdot \pi + \left((x + 3) - \frac{y}{5 + 1} \right)$$

$$(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$$

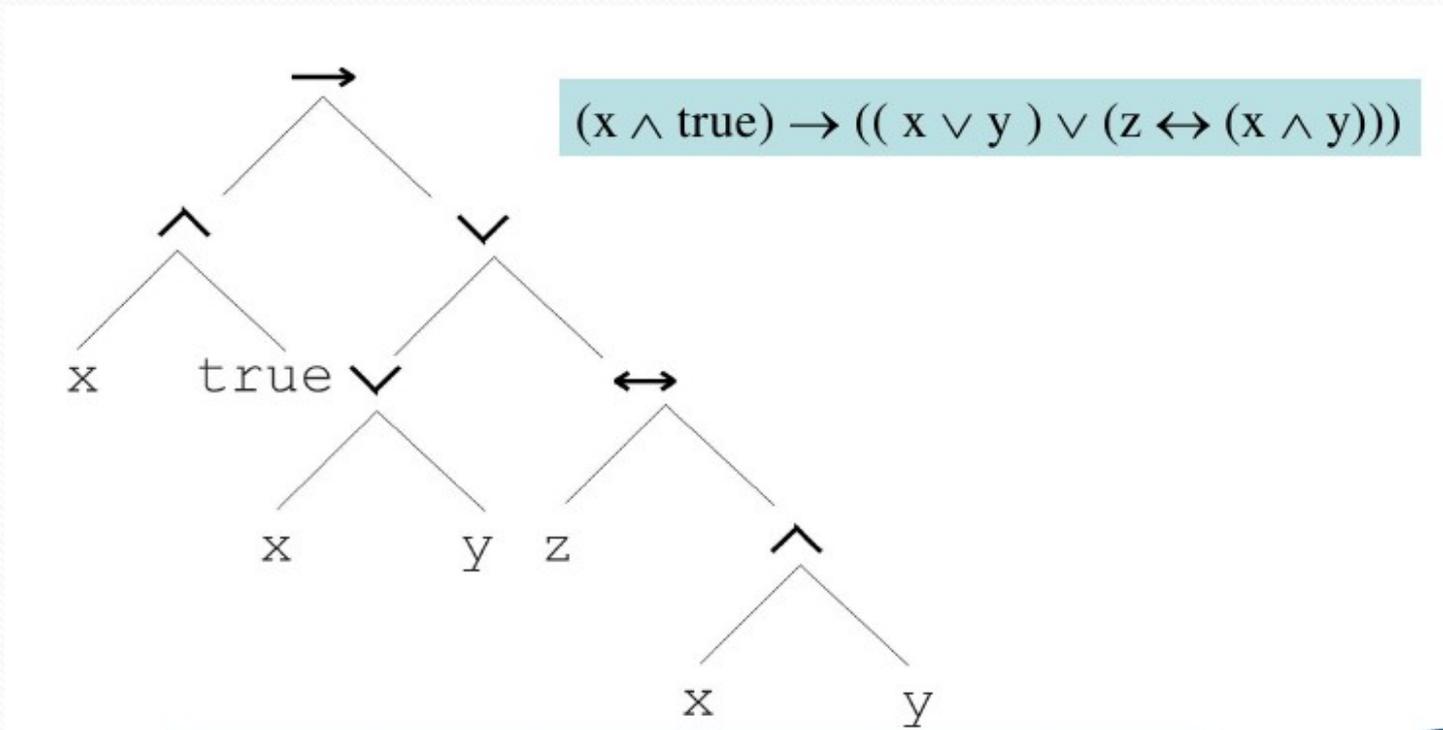
```
i = 1;  
while (i < 20)  
{  
    i = i + 1  
}
```

Representation of Arithmetic Formula

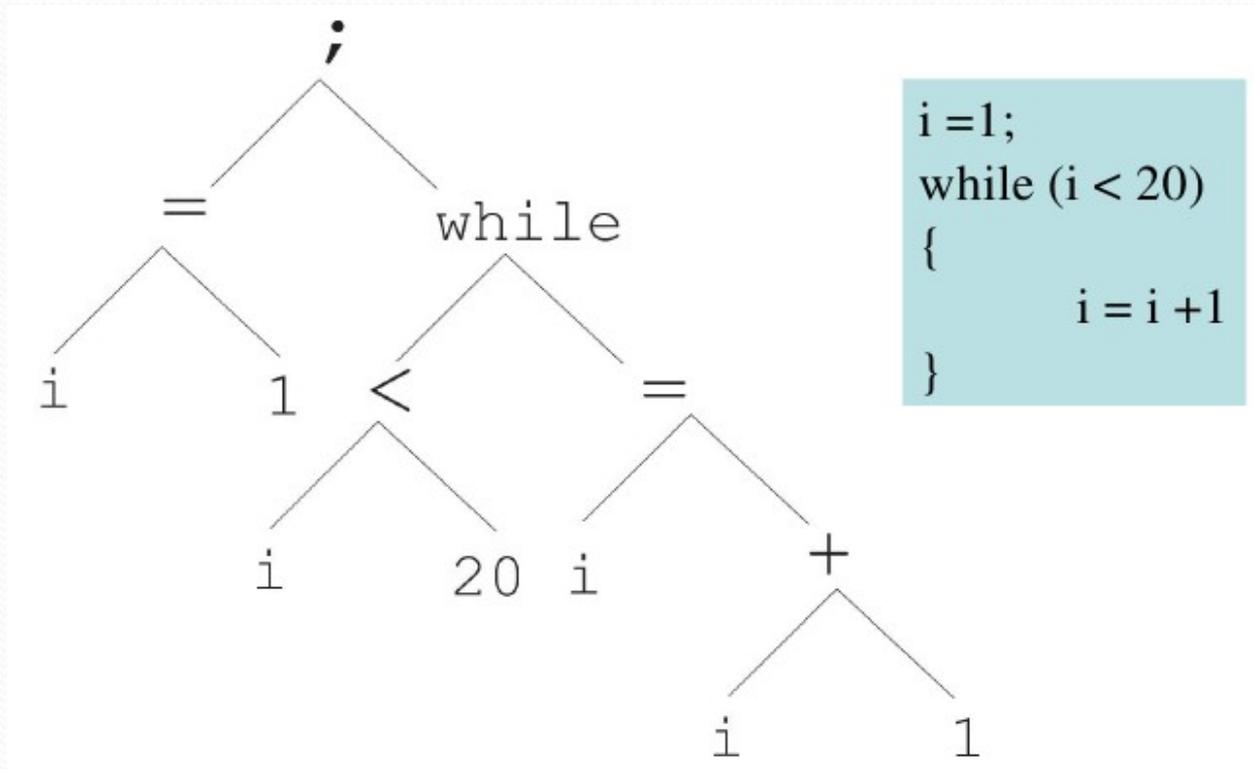


$$2 \cdot \pi + \left((x + 3) - \frac{y}{5 + 1} \right)$$

Representation of Logical Formula



Representation of Computer Programs



Representation

- Trees consisting of:
 - terminals (leaves)
 - constants
 - variables (inputs to the program/formula)
 - functions of fixed arity (internal nodes)

Considerations in Function Selection

- **Closure:** any function should be well-defined for all arguments

Example: $\{ *, / \}$ is not closed as division is not well defined if the second argument is 0 \rightarrow redefine $/$.

- **Sufficiency:** the function and terminal set should be able to represent a desirable solution

Evolutionary Cycle

- Fixed population size
- Create a new population by randomly selecting an operation to apply, each of which adds one or two individuals into the new population, starting from one or two fitness proportionally selected individuals:
 - reproduction (copying)
 - one of many crossover operations
 - one of many mutation operations

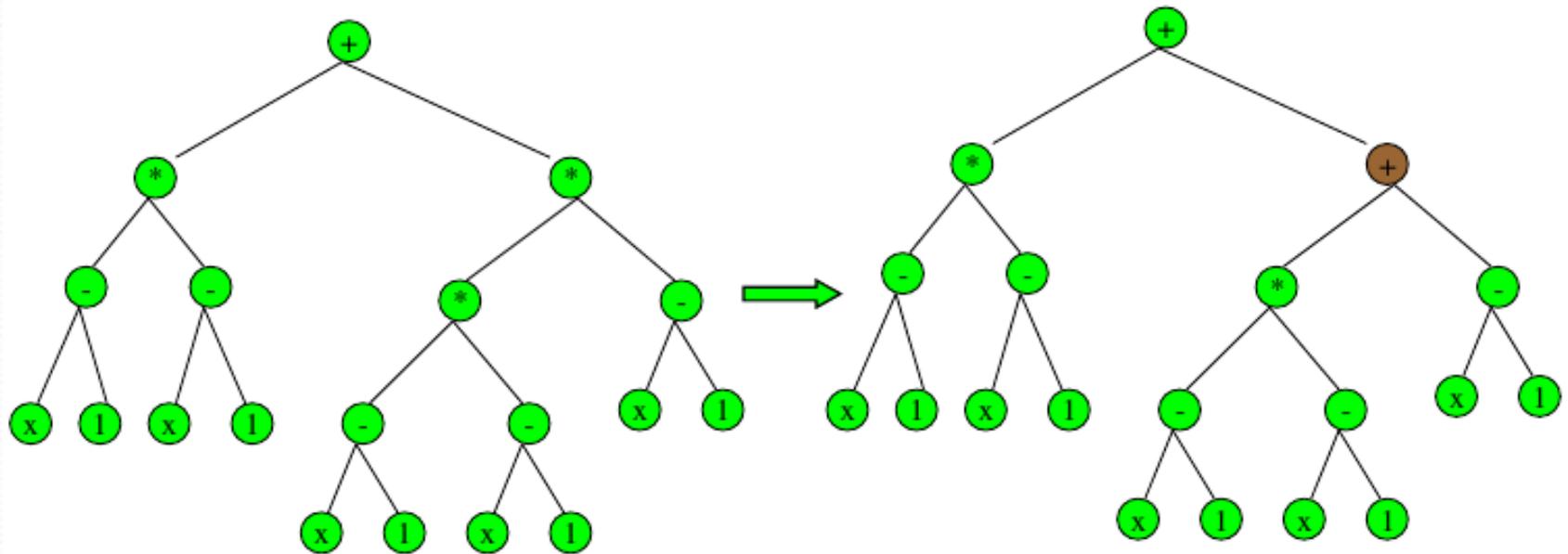
Initialization

- Given is a maximum depth on trees D_{max}
- Full method:
 - for each level $< D_{max}$ insert a node with function symbol (recursively add children of appropriate types)
 - for level D_{max} insert a node with a terminal
- Grow method:
 - for each level $< D_{max}$ insert a node with either a terminal or a function symbol (and recursively add children of appropriate types to these nodes)
 - for level D_{max} insert a node with a terminal
- Combined method: half of the population full, the other grown

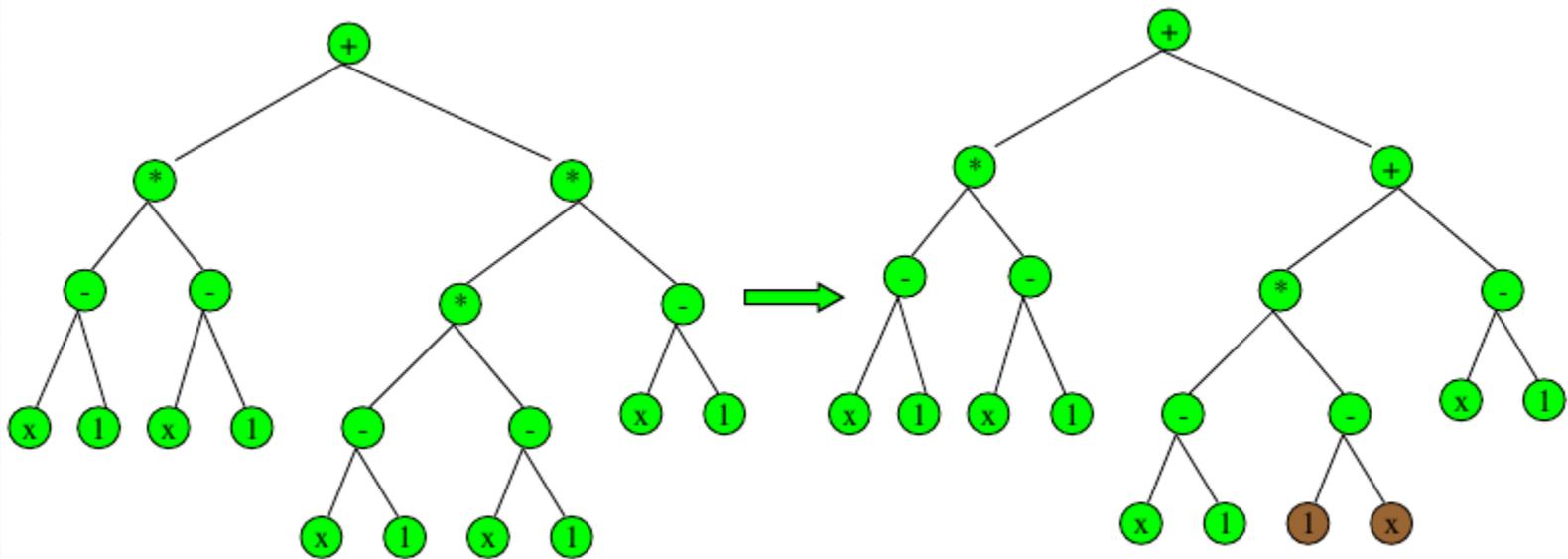
Mutation

Operator name	Description
Point mutation	single node exchanged against random node of same class
Permutation	arguments of a node permuted
Hoist	new individual generated from subtree
Expansion	terminal exchanged against random subtree
Collapse subtree	subtree exchanged against random terminal
Subtree mutation	subtree exchanged against random subtree

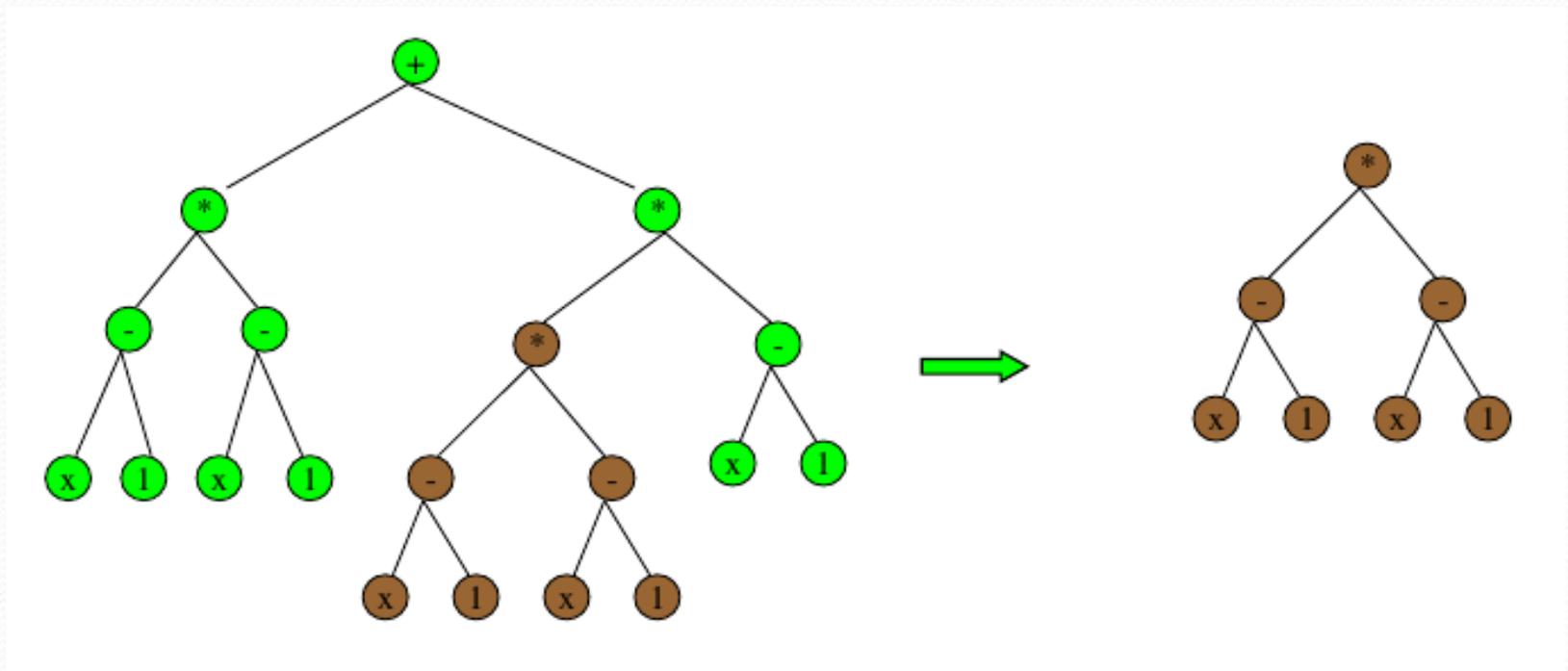
Point Mutation



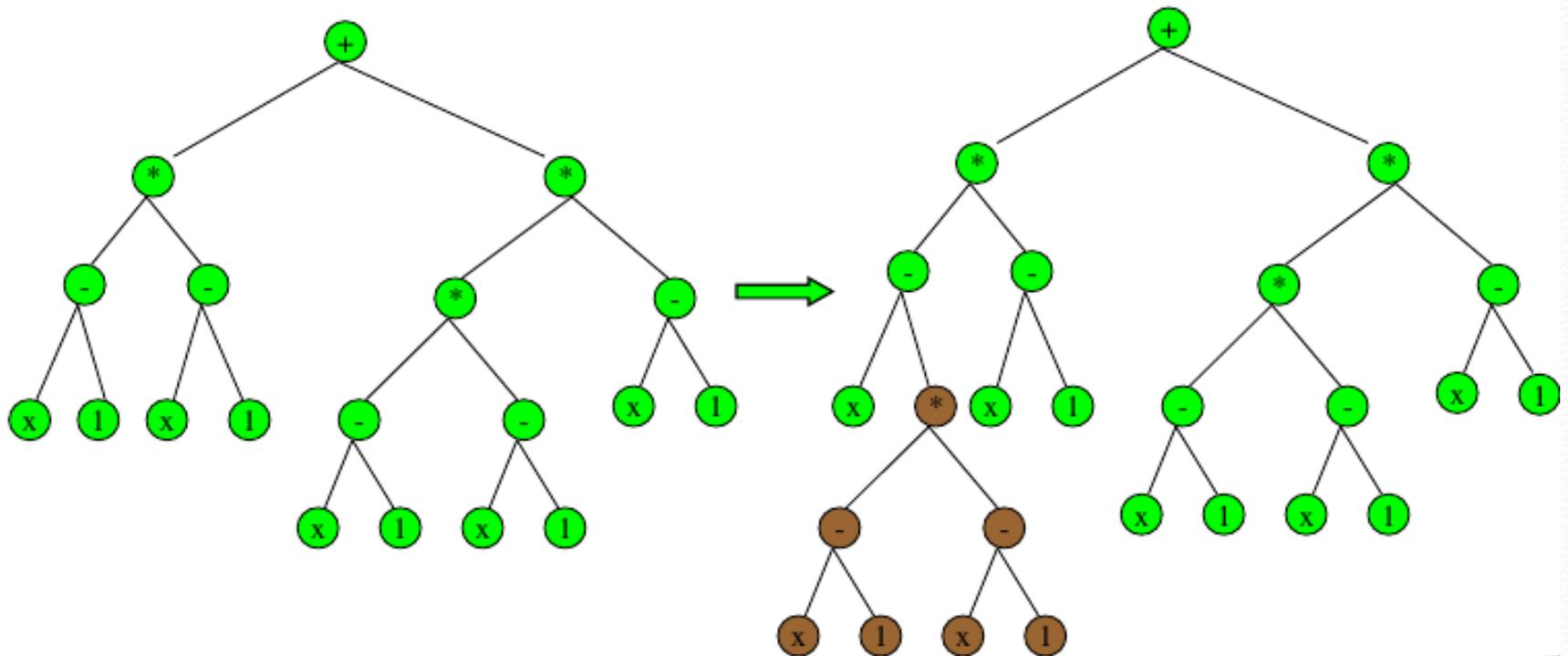
Permutation



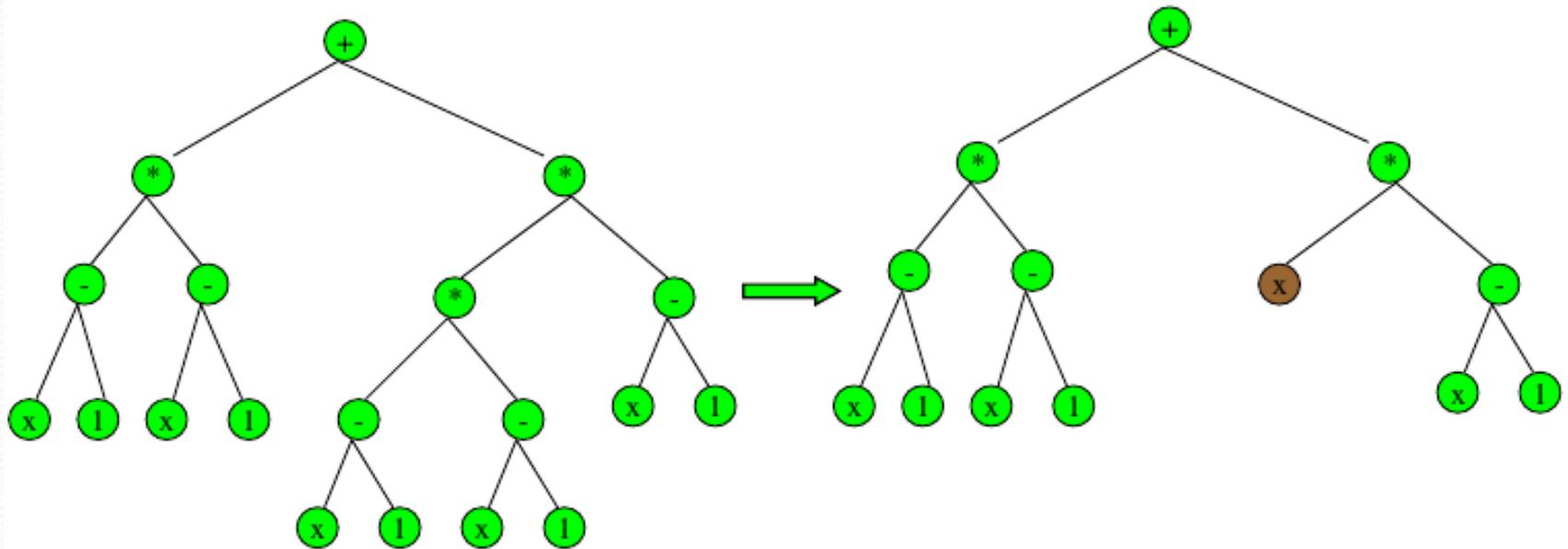
Hoist



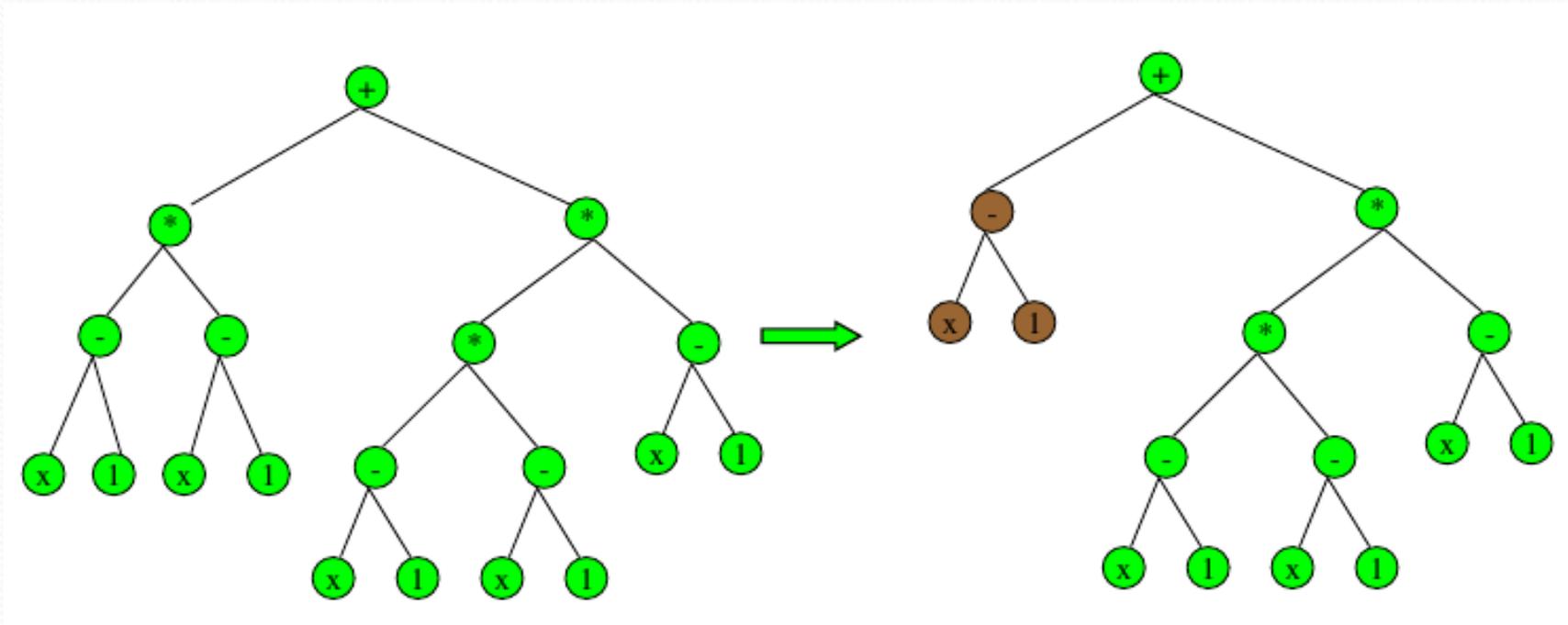
Expansion Mutation



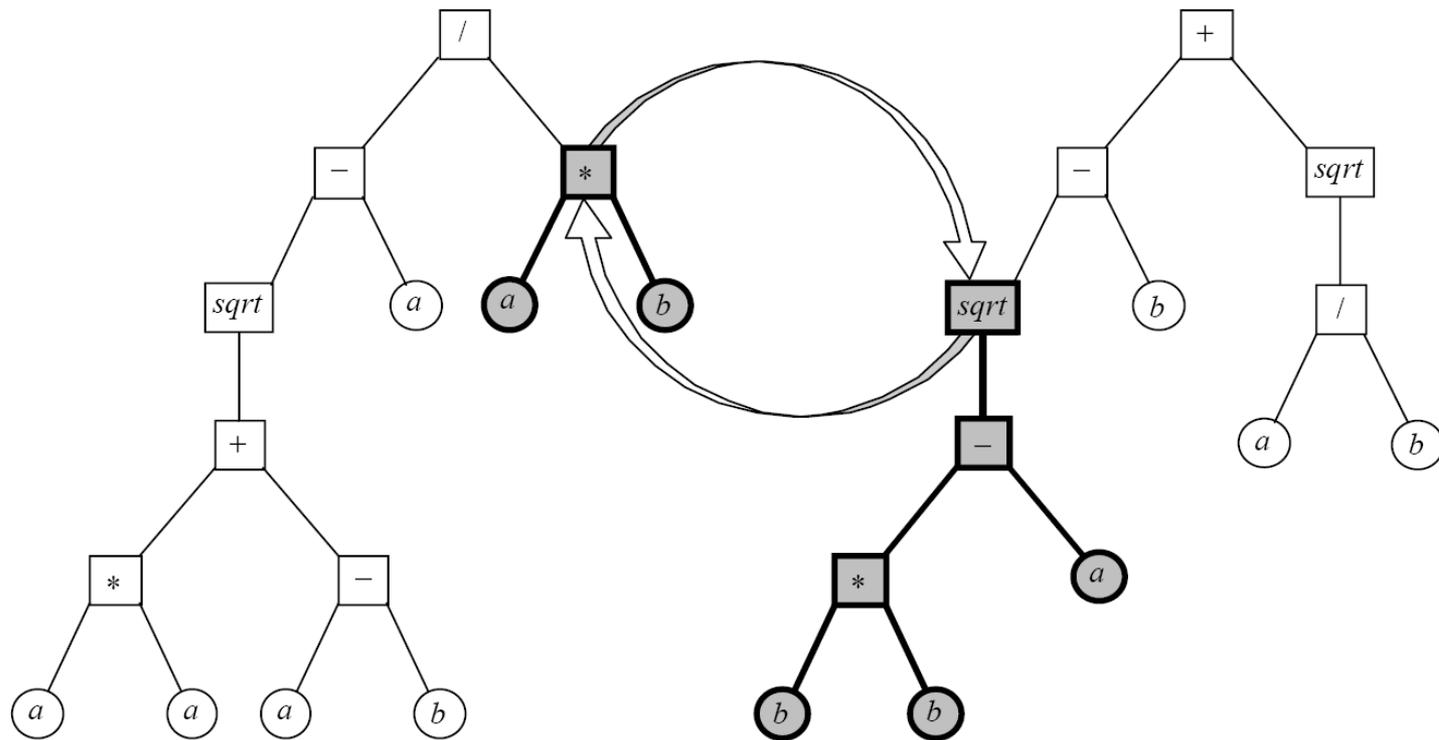
Collapse Subtree Mutation



Subtree Mutation



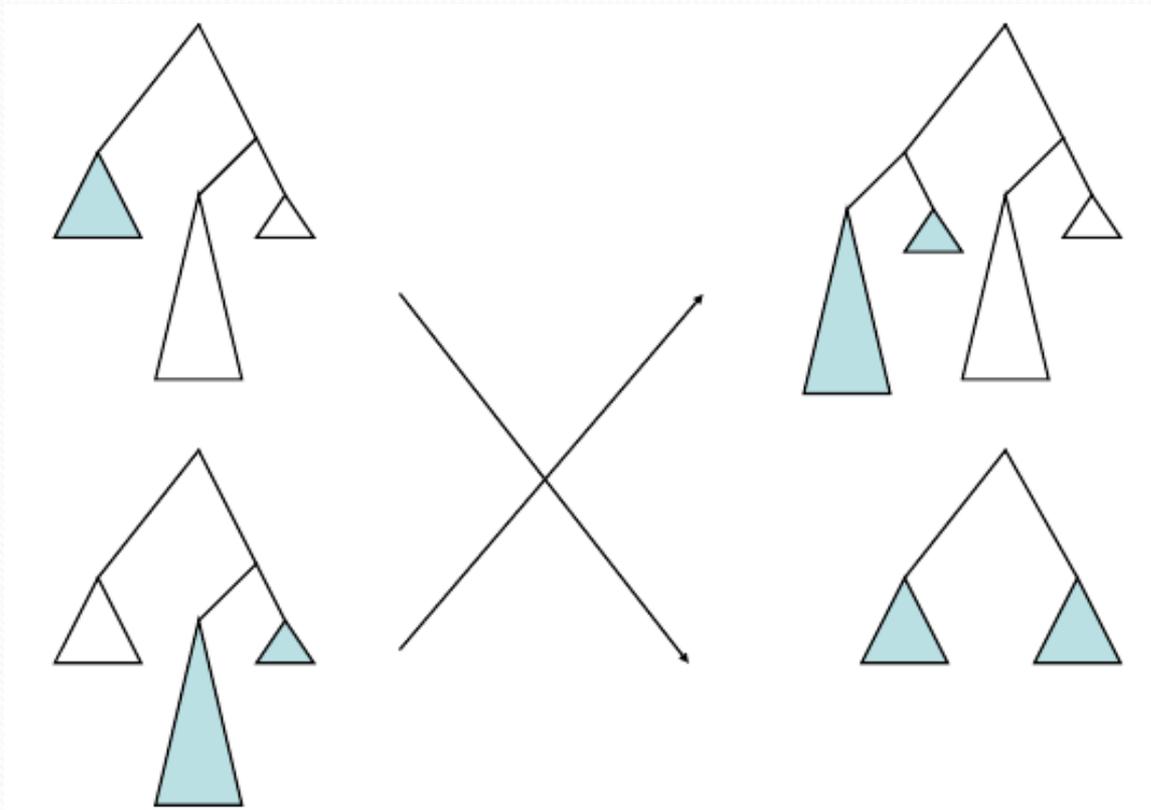
Crossover



$(/ (- (sqrt (+ (* a a) (- a b))) a) (* a b))$

$(+ (- (sqrt (- (* b b) a)) b) (sqrt (/ a b)))$

Self-Crossover



Bloat

- “Survival of the fittest”, i.e. the tree sizes in the populations increase over time
- Countermeasures:
 - simplification
 - penalty for large trees
 - hard constraints on the size of trees resulting from operations

Editing Operator

- An operation that simplifies expressions
- Examples:
 - $X \text{ AND } X \rightarrow X$
 - $X \text{ OR } X \rightarrow X$
 - $\text{NOT}(\text{NOT}(X)) \rightarrow X$
 - $X + 0 \rightarrow X$
 - $X . 1 \rightarrow X$
 - $X . 0 \rightarrow 0$
 -

Example – Symbolic Regression Pythagorean Theorem

Not (necessarily)
linear

Underlying function: $c = \sqrt{a^2 + b^2}$

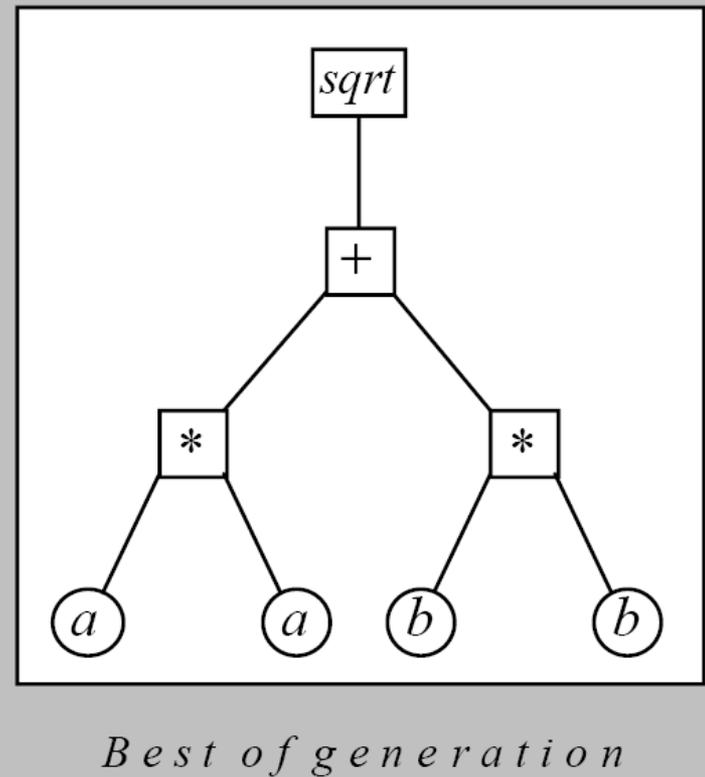
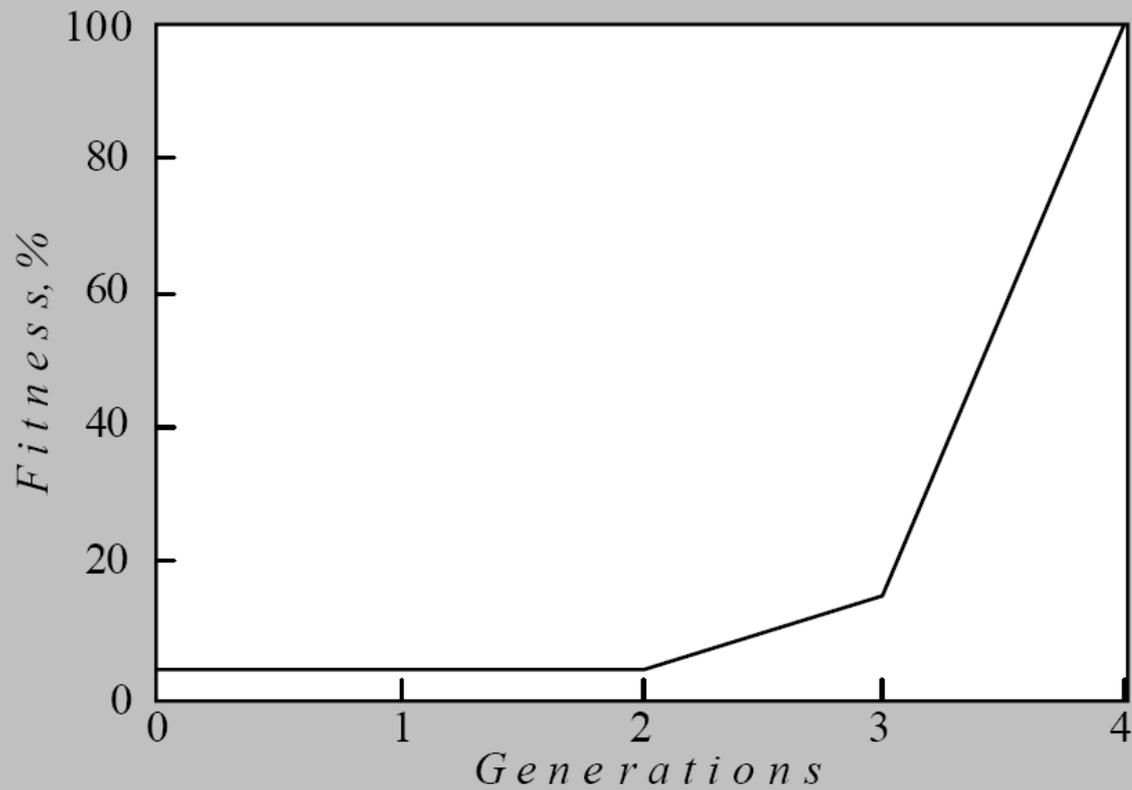
Negnevitsky 2004

Fitness cases:

Side a	Side b	Hypotenuse c	Side a	Side b	Hypotenuse c
3	5	5.830952	12	10	15.620499
8	14	16.124515	21	6	21.840330
18	2	18.110770	7	4	8.062258
32	11	33.837849	16	24	28.844410
4	3	5.000000	2	9	9.219545

Language elements: +, -, *, /, sqrt, a , b

Results



Example – Symbolic Regression

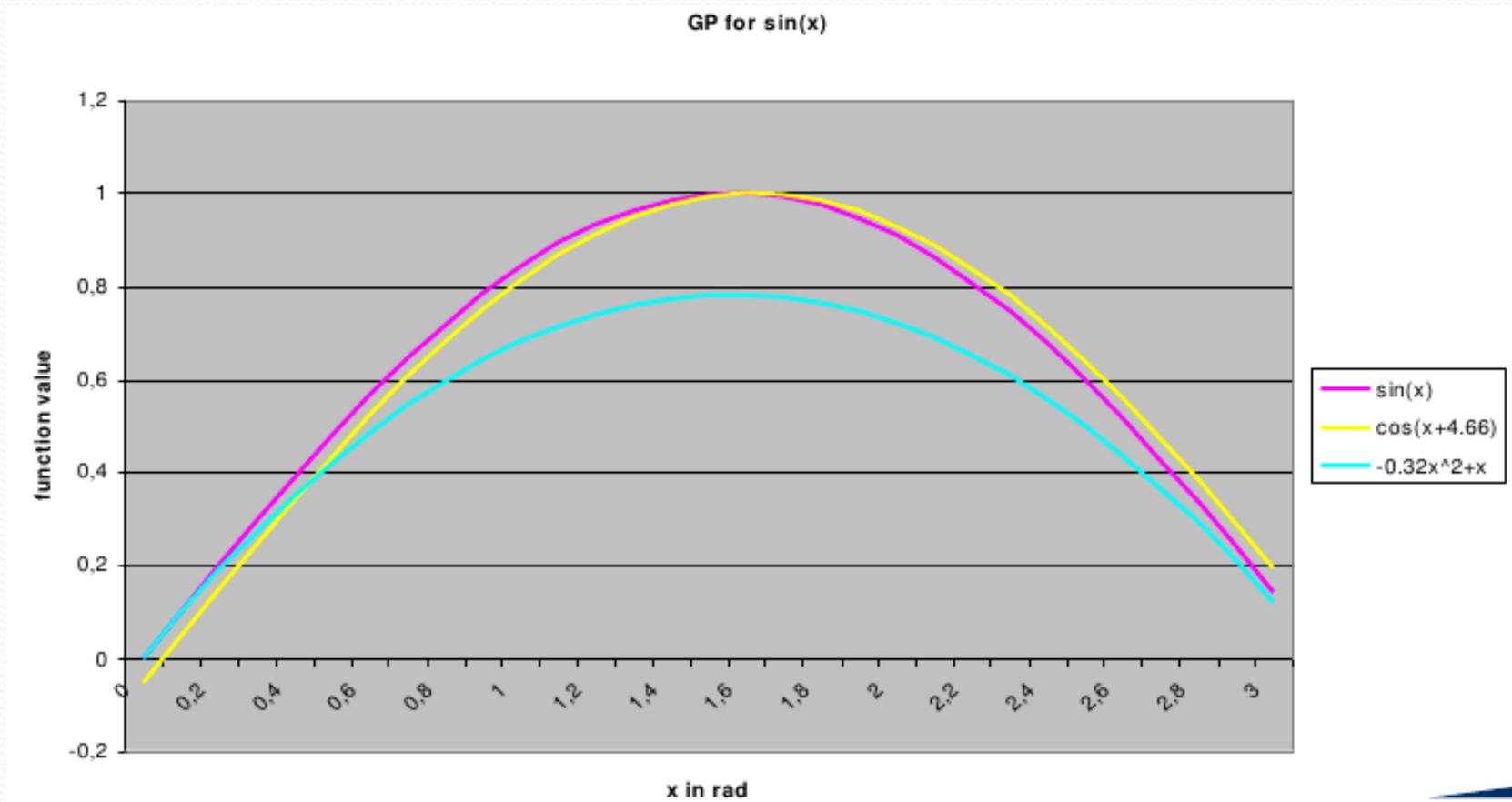
Approximation of $\sin(x)$

- **Given** examples $(x, \sin(x))$ with x in $\{0, 1, \dots, 9\}$
- **Find** a good approximation of $\sin(x)$

Function Sets	Result	Generation	Error (final)
$F_1: \{ +, -, *, /, \sin \}$	$\sin(x)$	0	0.00
$F_2: \{ +, -, *, /, \cos \}$	$\cos(x + 4.66)$	12	0.40
$F_3: \{ +, -, *, / \}$	$-0.32x^2 + x$	29	1.36

Example – Symbolic Regression

Approximation of $\sin(x)$



GAs vs. GP

Genetic algorithms

- Chromosomes represent coded solutions
- Fixed length chromosomes
- A small set of well-defined genetic operators
- Conceptually simple

Genetic programming

- Chromosomes represent executable code
- Variable length chromosomes
- More complex genetic operators required
- Conceptually complex