Evolutionary algorithms

- Simple genetic algorithms
- Evolutionary Strategies
- Genetic Programming

Partially based on slides by Thomas Bäck

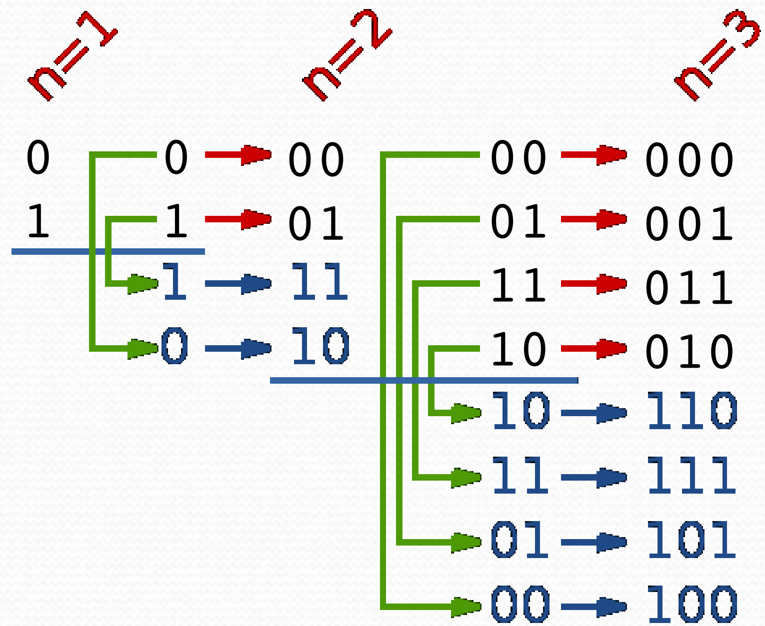
Gray Coding

- Aim: binary coding of integers such that integers x and y for which $|x-y|=1$ only differ in one bit

Dec	Gray	Binary
0	000	000
1	001	001
2	011	010
3	010	011
4	110	100
5	111	101
6	101	110
7	100	111

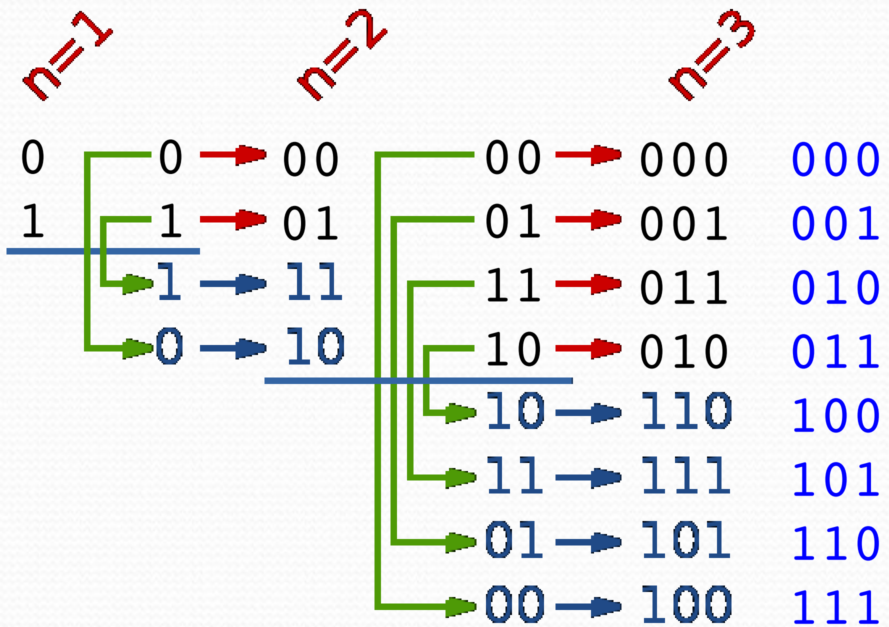
Gray Coding

- Codes for *n*=1: (i.e., integers 0, 1)
  0 1
- Codes for *n*=2: (i.e., integers 0, 1, 2, 3)
  Reflected entries for *n*=0:
  1 0
  Prefix old entries with 0:
  00 01
  Prefix reflected entries with 1:
  11 10
  Codes hence:
  00 01 11 10
- Codes for *n*=3: (i.e., integers 0, 1, 2, …, 7)
  Reflected entries for *n*=2:
  10 11 01 00
  Codes hence:
  000 001 011 010 110 111 101 100



Gray Coding

- Given a "normal" bit representation, how to calculate the Gray code?



bitstring → Gray

10100 → 11110

10101 → 11111

10110 → 11101

11001 → 10101

A bit flips in the Gray code iff the bit before it has value 1 in the original code.

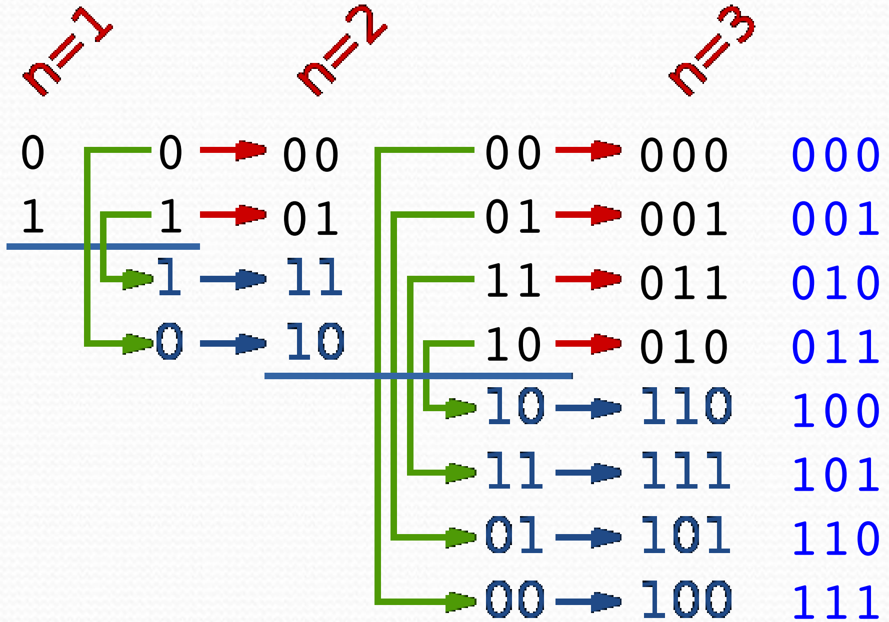
Gray Coding

- Source code in Python for calculating Gray code:

```
def binaryToGray(num):
        return (num >> 1) ^ num
```

Gray Coding

- Given a Gray code, how to calculate a “normal” bit representation?



bitstring → Gray
10100 → 11110
10101 → 11111
10110 → 11101
11001 → 10101

A bit flips in the “normal” code (as compared to the Gray code) iff the bit before it has value 1 in the “normal” code.

Gray Coding

- Gray coding does not avoid that integers far away from each other can have similar codes

  00000=0

  10000=31

  → Mutation can still change numbers a lot

- Gray coding only ensures that there always is a one-bit mutation to transform integer x into integer $x+1$ or $x-1$.

Constraints

- Examples:
  - "A string of numbers should represent a permutation" (1,2,3) is valid; (1,1,3) is not
  - "The sum of numbers should not be lower than a threshold"
- Possibility 1: fitness function modification
  - setting fitness of unfeasible solutions to zero (search may be very inefficient due to unfeasible solutions)
  - penalty function (negative terms for violated constraints)
  - barrier function (already penalty if "close to" violation)

Constraints

- Possibility 2 (preferred method): special encoding
  - GA searches always through allowed solutions
  - smaller search space
  - ad hoc method, may be difficult to find
- Example: permutations (see AI course)

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

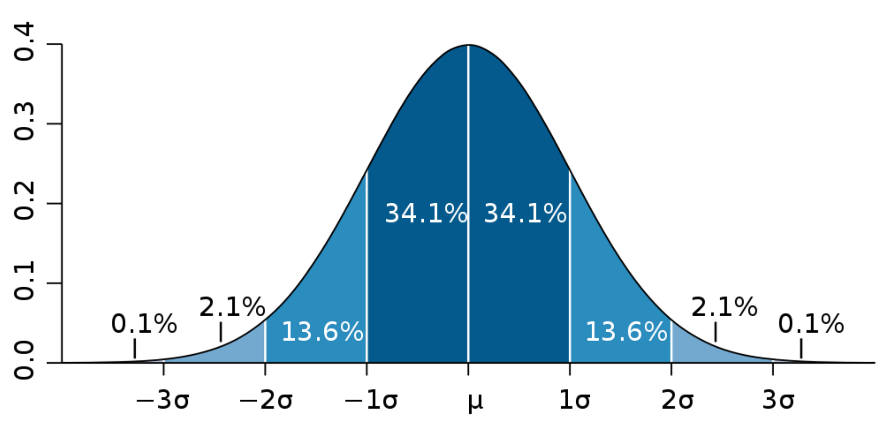
- Not performed *before* mutation and crossover, but *after* these operations
- It is assumed mutation (& crossover) generate $\lambda > \mu$ individuals (where μ is population size) (typically $\lambda \approx 7\mu$)
- Deterministically eliminate worst individuals from
  - children only: (μ,λ)-ES →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

ES: Basic Mutation

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

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

"sampled from"



Simple modification: mutation rate for each x_i

Major question: How to set σ or σ_i ?

MAIN IDEA: <u>make search more efficient</u>
by increasing mutation rate if this seems safe

ES: Basic Mutation

- An algorithm for setting global σ :
  - Count the number G_s of successful mutations
  - Compute the ratio of successful mutations $p_s = G_s / G$
  - Update strategy parameters according to

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

$$c \in [0.8, 1.0]$$

until termination

Improved fitness

Increase mutation rate as it appears better solutions are far away

"1/5 rule"

Basic (1+1) ES

- Common use of the 1/5 rule

$t := 0;$
initialize $P(0) := \{\vec{x}(0)\} \in I,\ I = I\!R^n,\ \vec{x} = (x_1, \ldots, x_n);$
evaluate $P(0) :\ \{f(\vec{x}(0))\}$
while not *terminate*$(P(t))$ **do**
  mutate: $\vec{x}'(t) := m(\vec{x}(t))$
    where $x_i' := x_i + \sigma(t) \cdot N_i(0,1)\ \forall i \in \{1, \ldots, n\}$
  evaluate: $P'(t) := \{\vec{x}'(t)\} :\ \{f(\vec{x}'(t))\}$
  select: $P(t+1) := s_{(1+1)}(P(t) \cup P'(t));$
  $t := t + 1;$
  if $(t \bmod n = 0)$ **then**

$$\sigma(t) := \begin{cases} \sigma(t-n)/c & ,\ \textbf{if } p_s > 1/5 \\ \sigma(t-n) \cdot c & ,\ \textbf{if } p_s < 1/5 \\ \sigma(t-n) & ,\ \textbf{if } p_s = 1/5 \end{cases}$$

    where p_s *is the relative frequency of successful mutations, measured over intervals of, say,* $10 \cdot n$ *trials;*
    and $0.817 \le c \le 1;$
  else
    $\sigma(t) := \sigma(t-1);$
  fi
od

ES Mutation: Strategy Parameters

- An individual is a vector $\vec{h} = (x_1, \ldots, x_n, \sigma)$ or $\vec{h} = (x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n)$
  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

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

Individual before mutation

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

Individual after mutation

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

1.: Mutation of step sizes

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

2.: Mutation of objective variables

Here the new σ' is used!

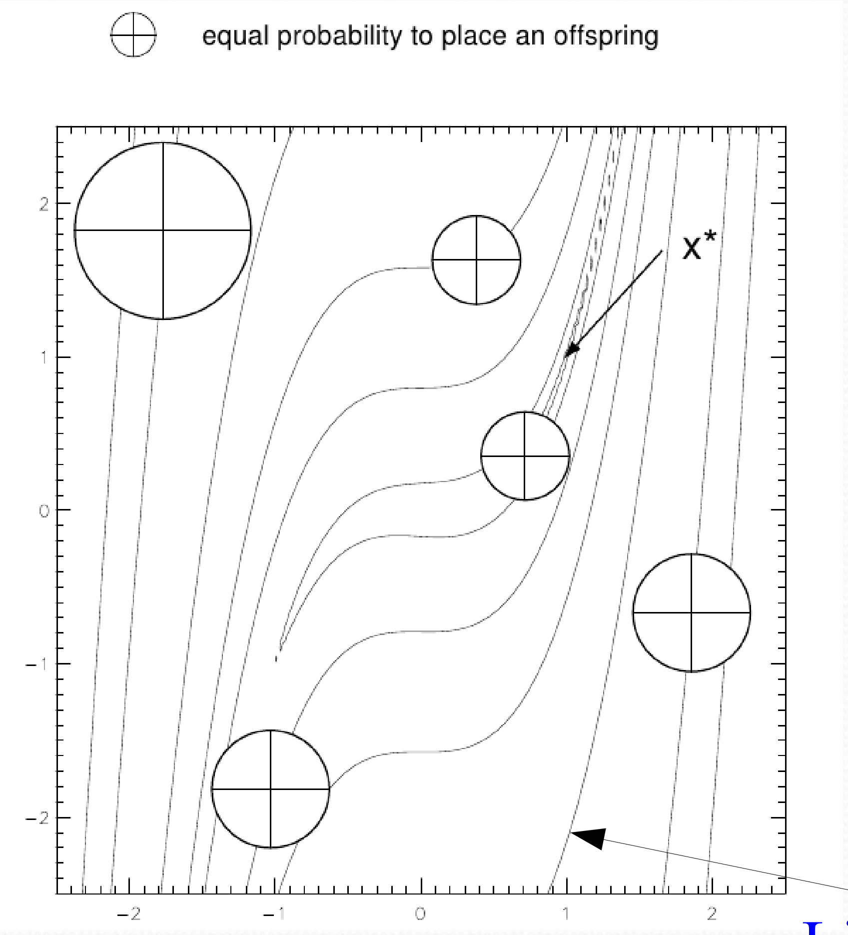
ES Mutation: Strategy Parameters

- 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



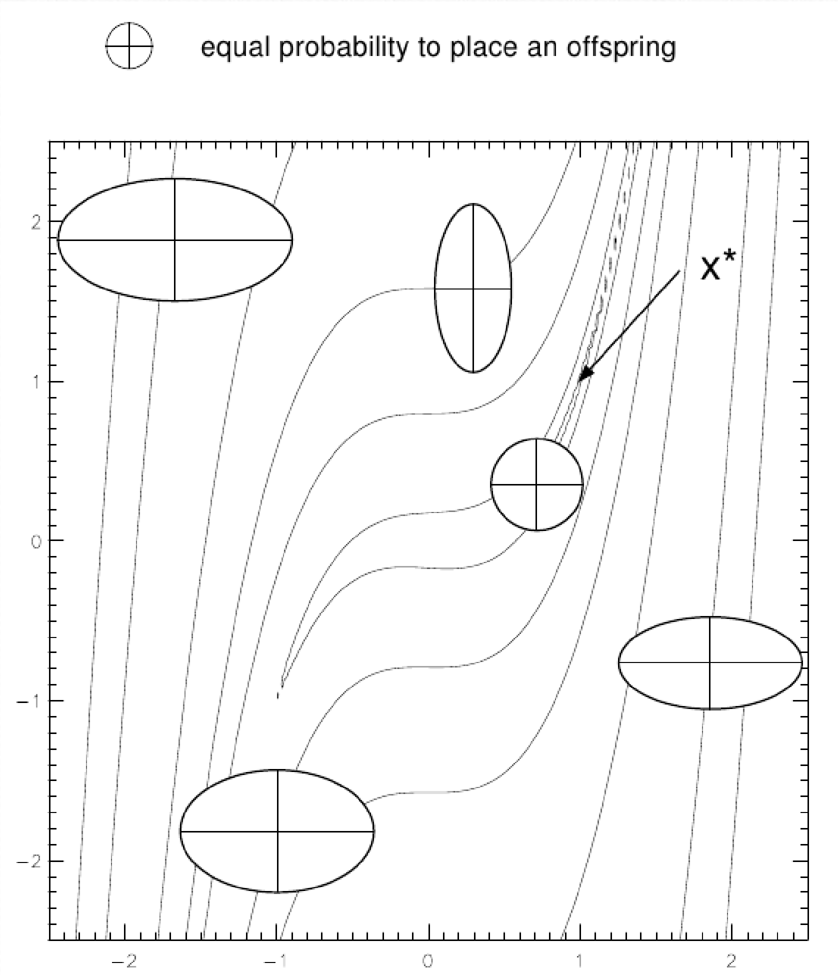
- One parameter for each individual
- 2 dimensional genotype

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

- 5 individuals

Line indicates points with equal fitness

ES Mutation: Strategy Parameters



- One parameter for each dimension
- 2 dimensional genotype

$$\vec{h} = (x_1, x_2, \sigma_1, \sigma_2)$$

- 5 individuals

ES Mutation: Strategy Parameters

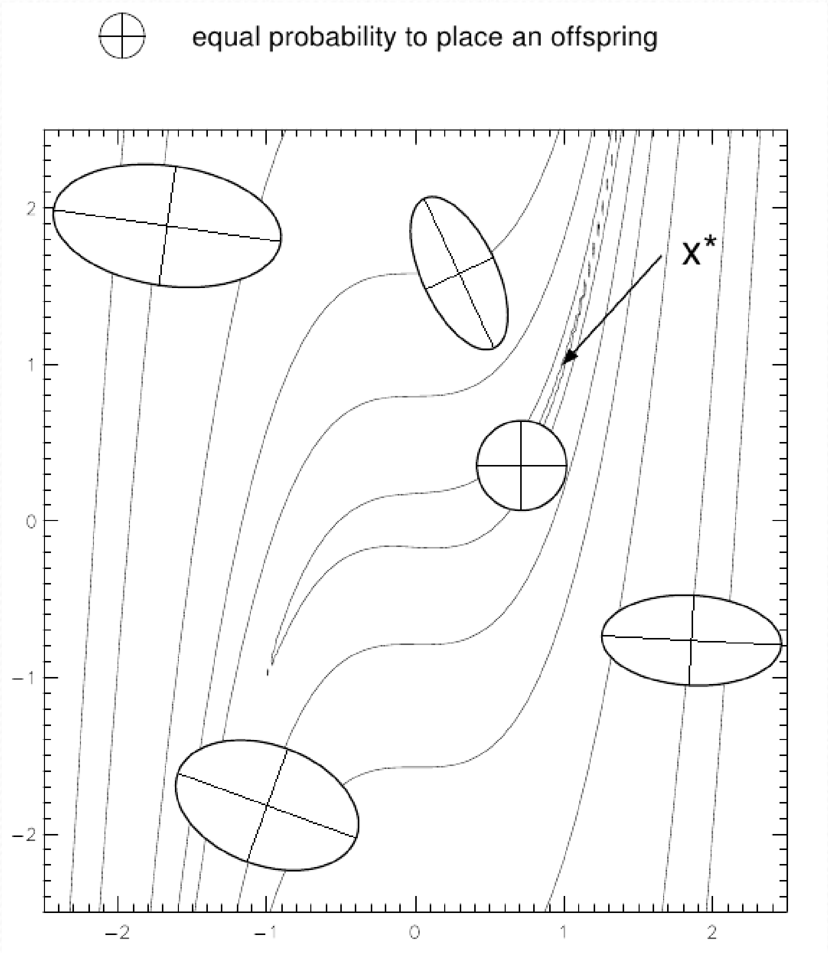
- Mutation of all strategy parameters

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

Sample from normal distribution, the same for all parameters

Update for this specific parameter

ES Mutation: Strategy Parameters



- An individual is a vector

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

where α_i encode angles

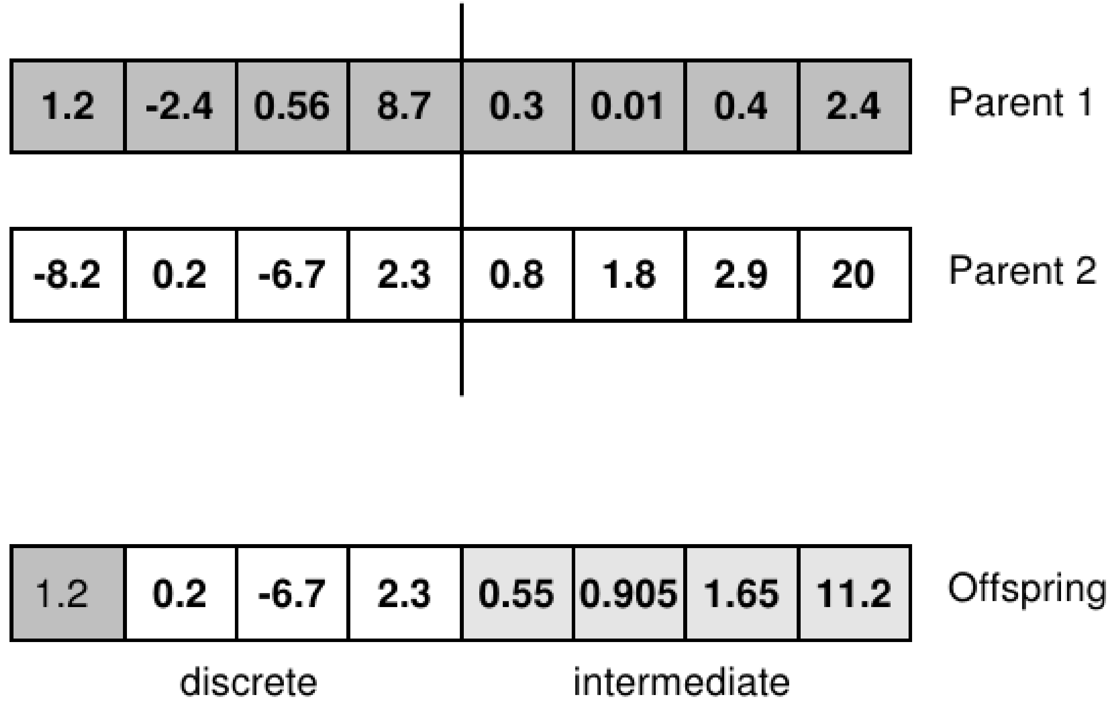
- Also here mutation can be defined
- Mathematical details skipped

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

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

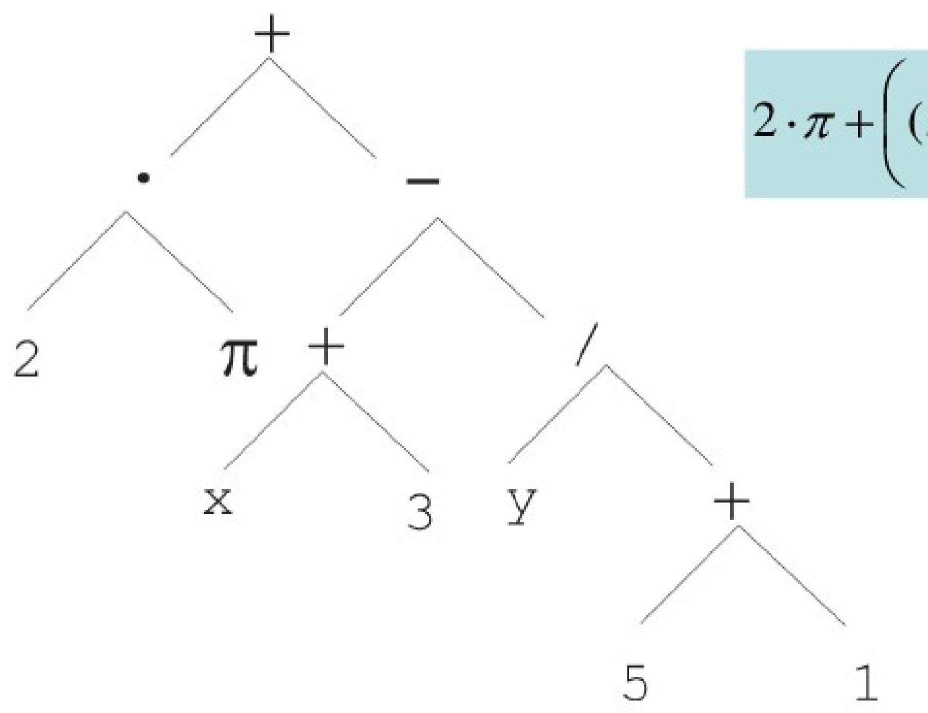
  - Logical formulas

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

  - Computer programs

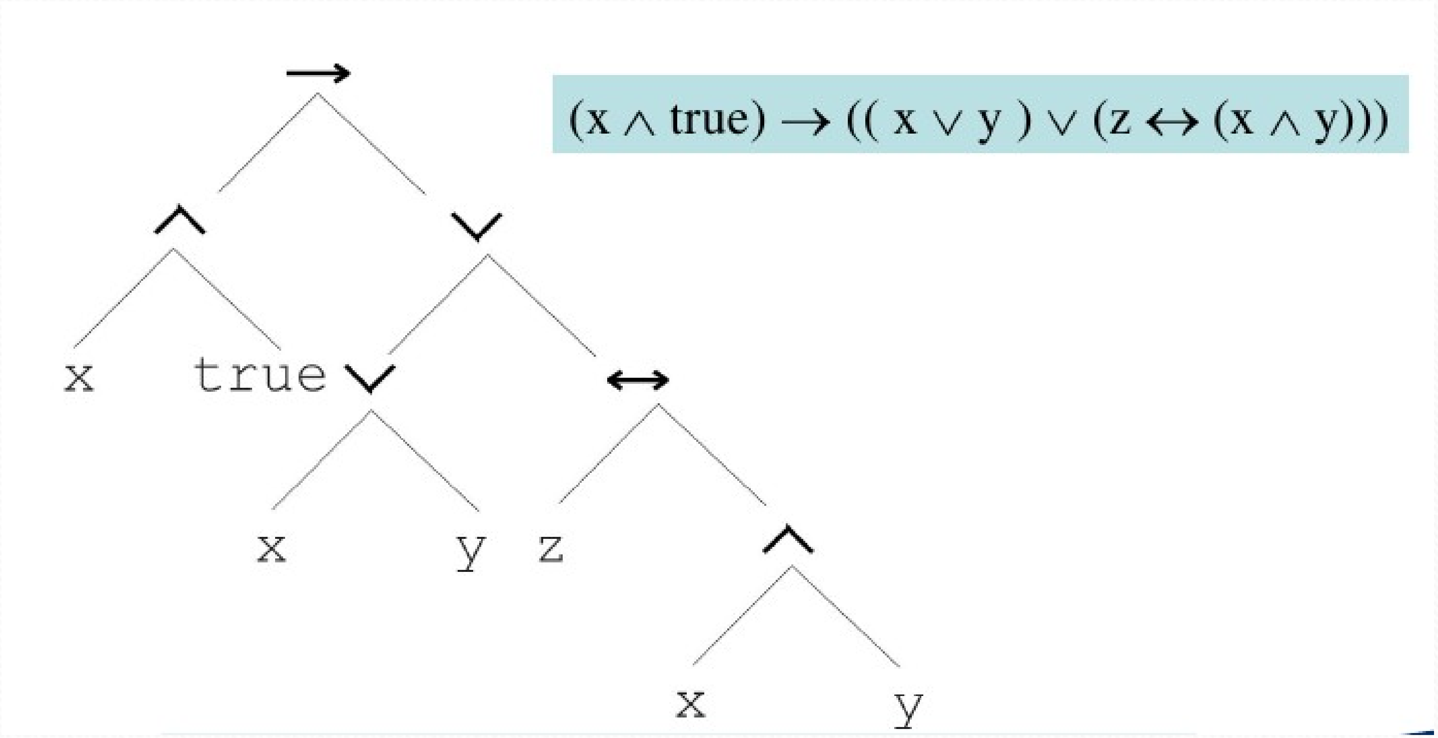
```
i =1;
while (i < 20)
{
        i = i +1
}
```

Representation of Arithmetic Formula

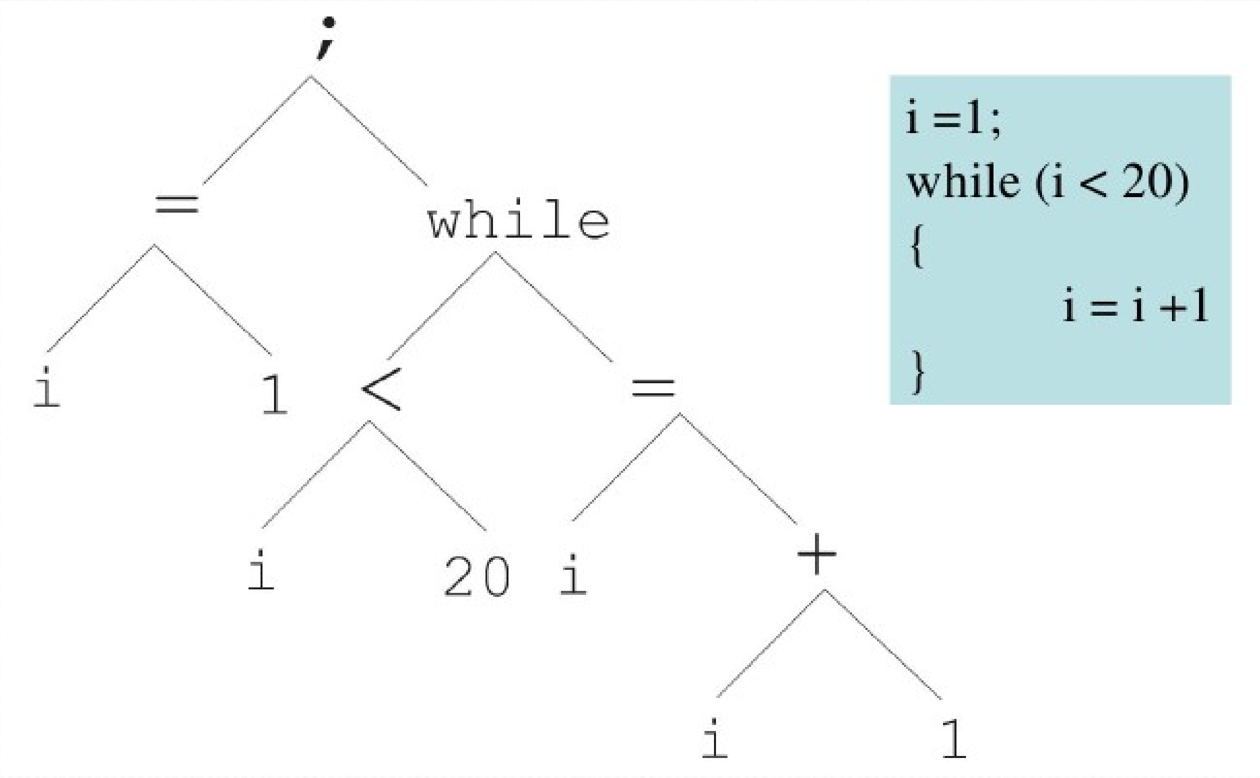


$$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$$

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 → 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

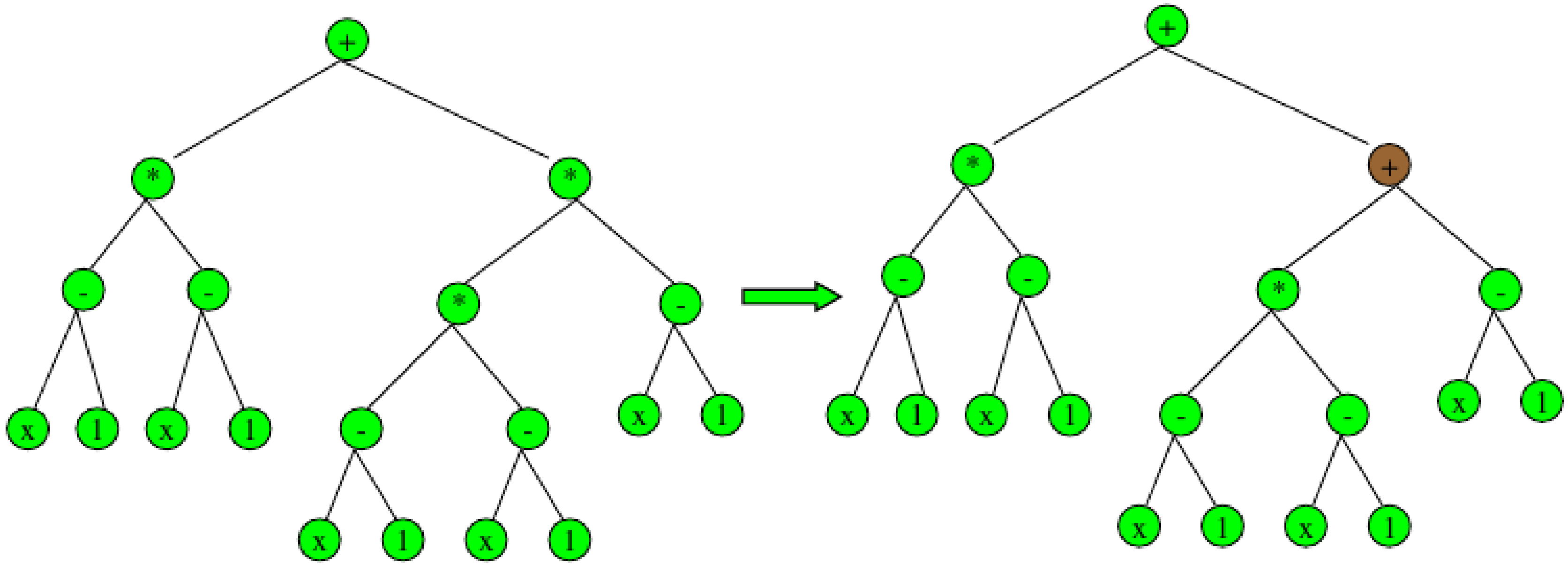
Initialization

- 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

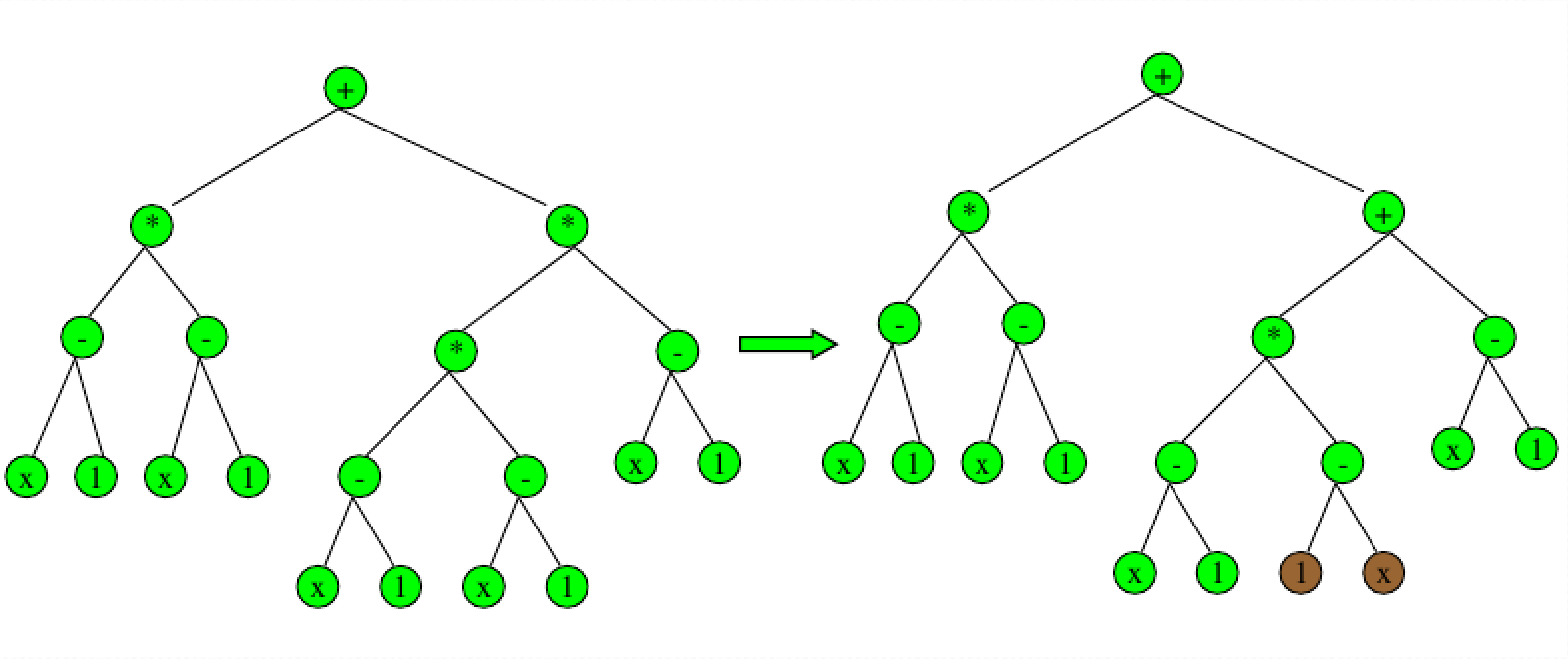
Mutation

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

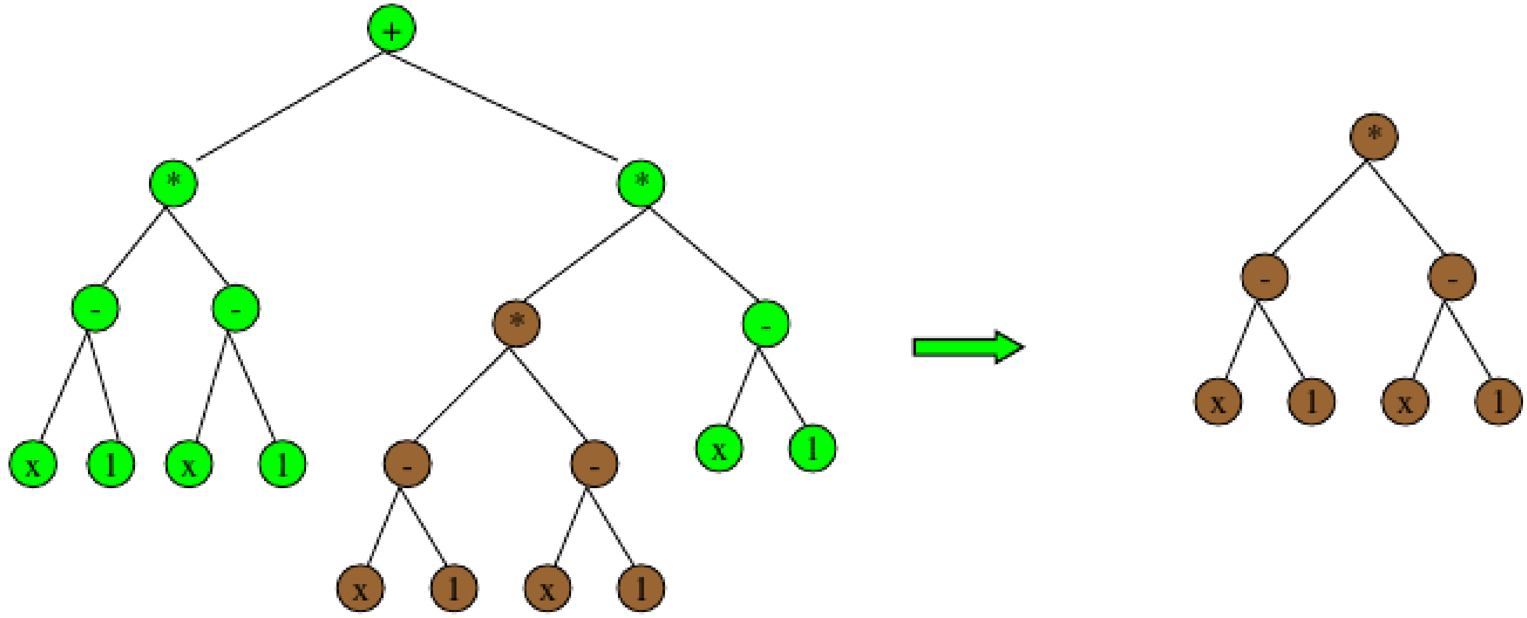
Point Mutation



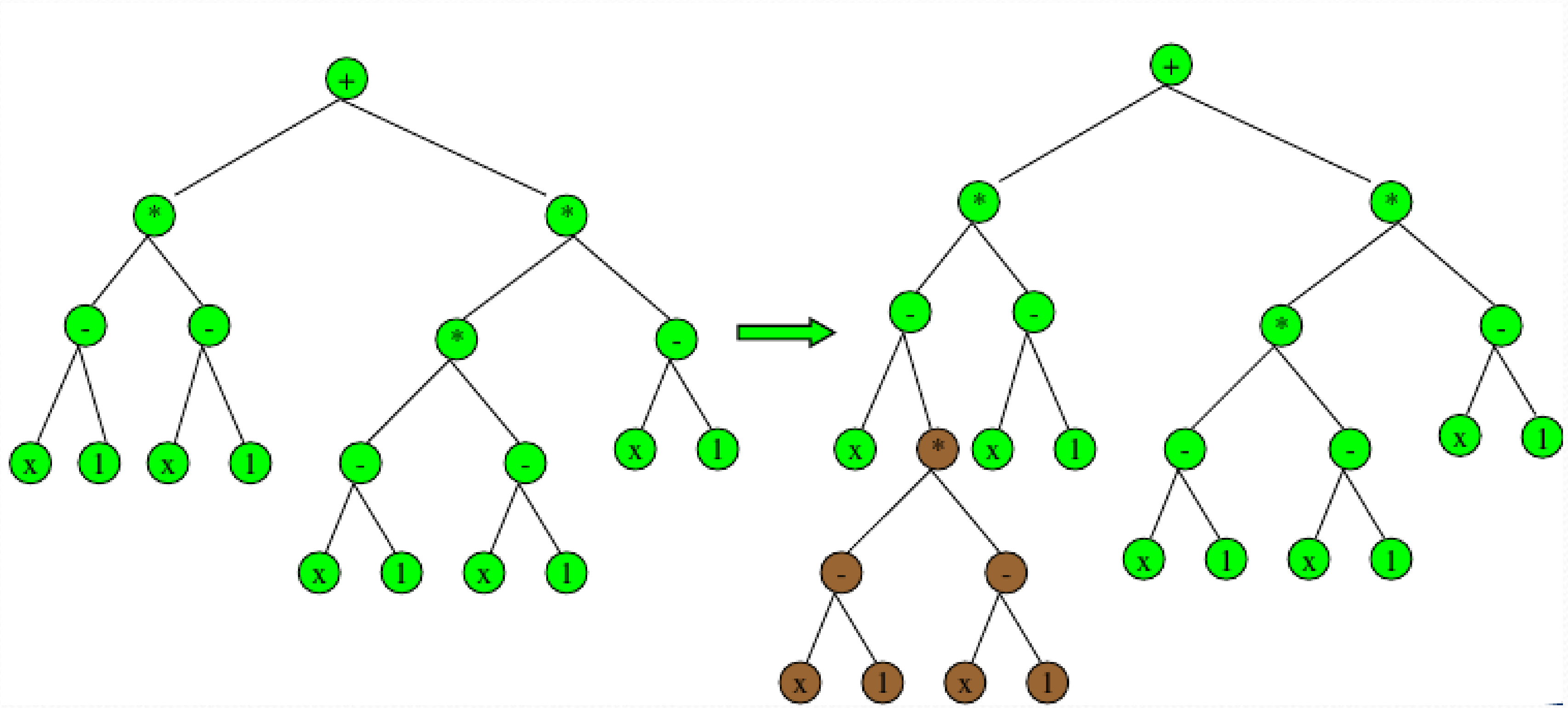
Permutation



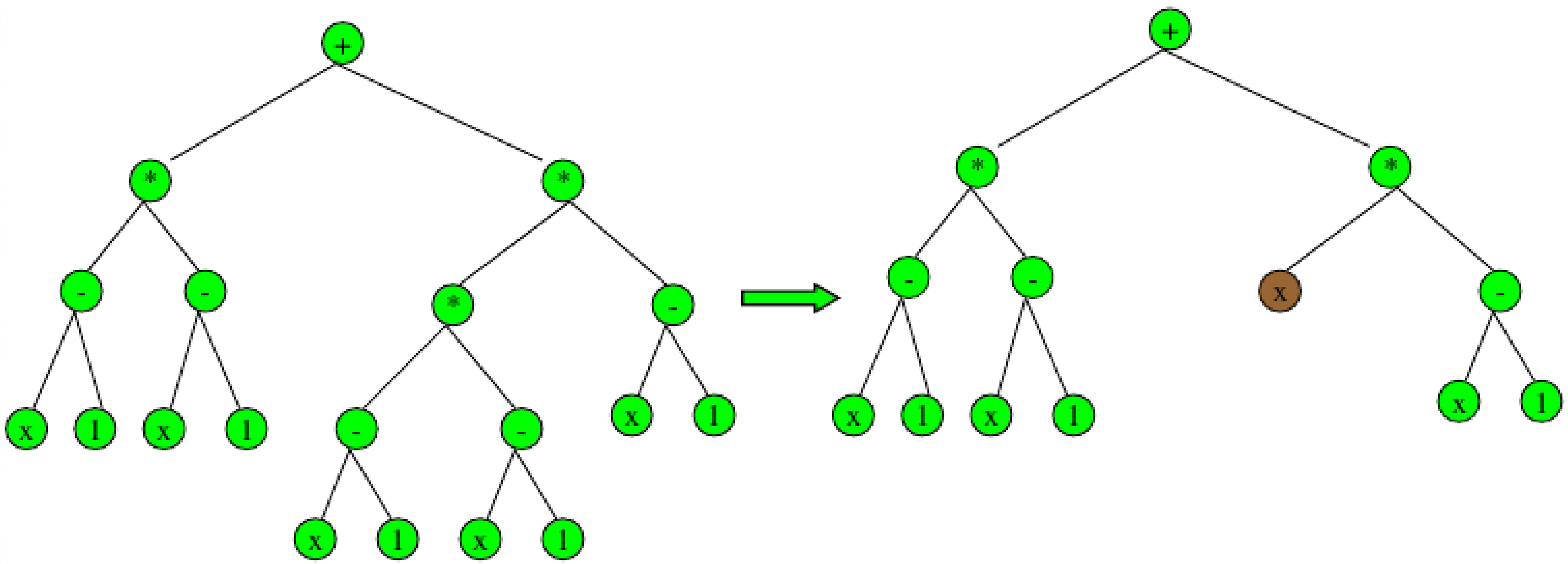
Hoist



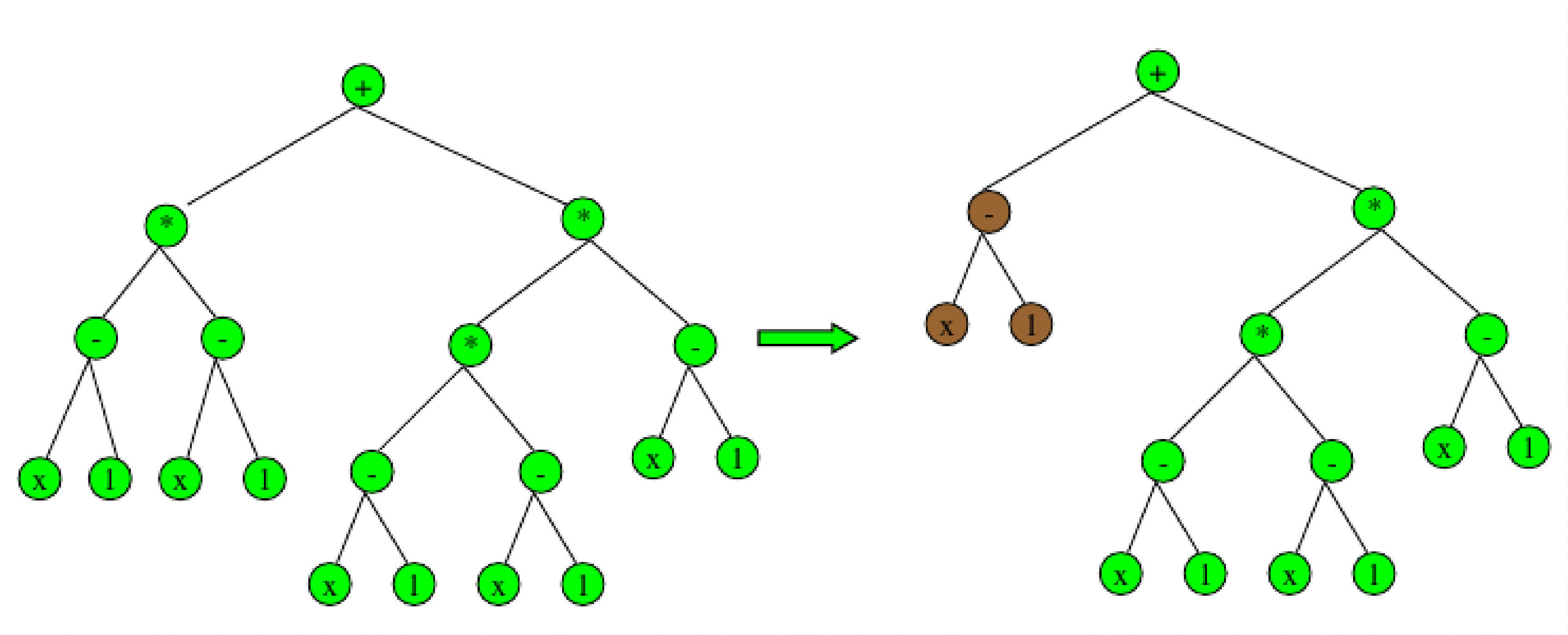
Expansion Mutation



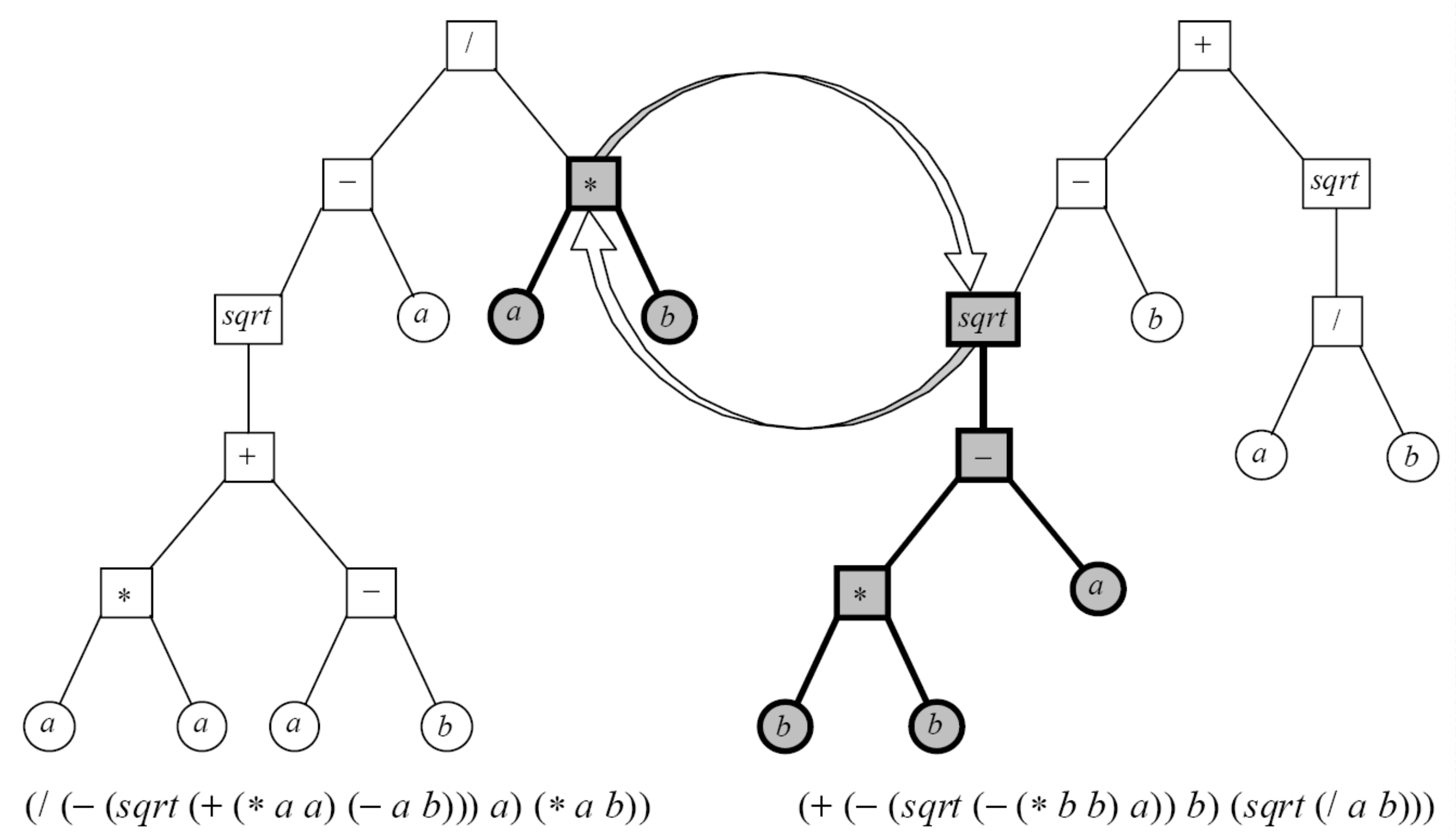
Collapse Subtree Mutation



Subtree Mutation

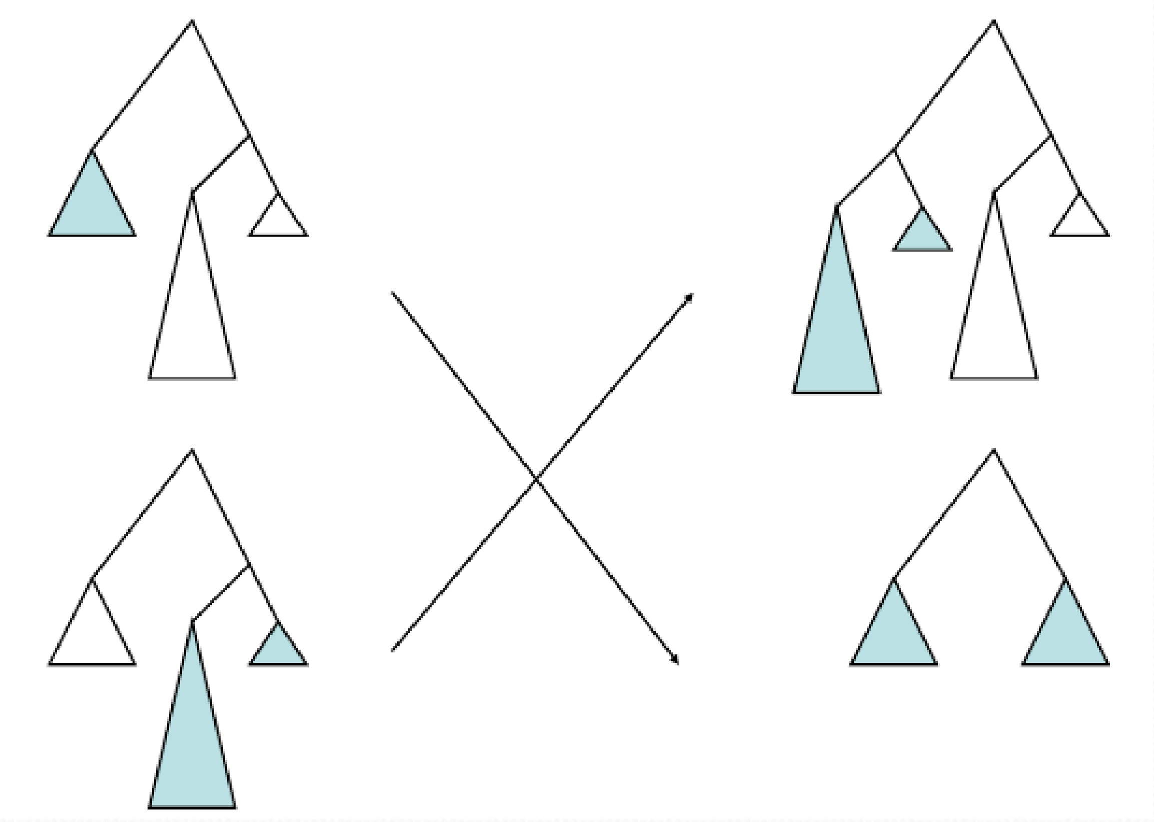


Crossover



/
−
∗
sqrt
a
a
b
+
∗
−
a
a
a
b
+
−
sqrt
sqrt
b
−
/
∗
a
a
b
b
b
(/ (− (sqrt (+ (∗ a a) (− a b))) a) (∗ a b))
(+ (− (sqrt (− (∗ b b) a)) b) (sqrt (/ a b)))

Self-Crossover



Bloat

- “Survival of the fattest”, 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 AND X → X
  - X OR X → X
  - NOT(NOT(X)) → X
  - X + 0 → X
  - X . 1 → X
  - X . 0 → 0
  - ….

Example – Symbolic Regression Pythagorean Theorem

Not (necessarily) linear

Negnevitsky 2004

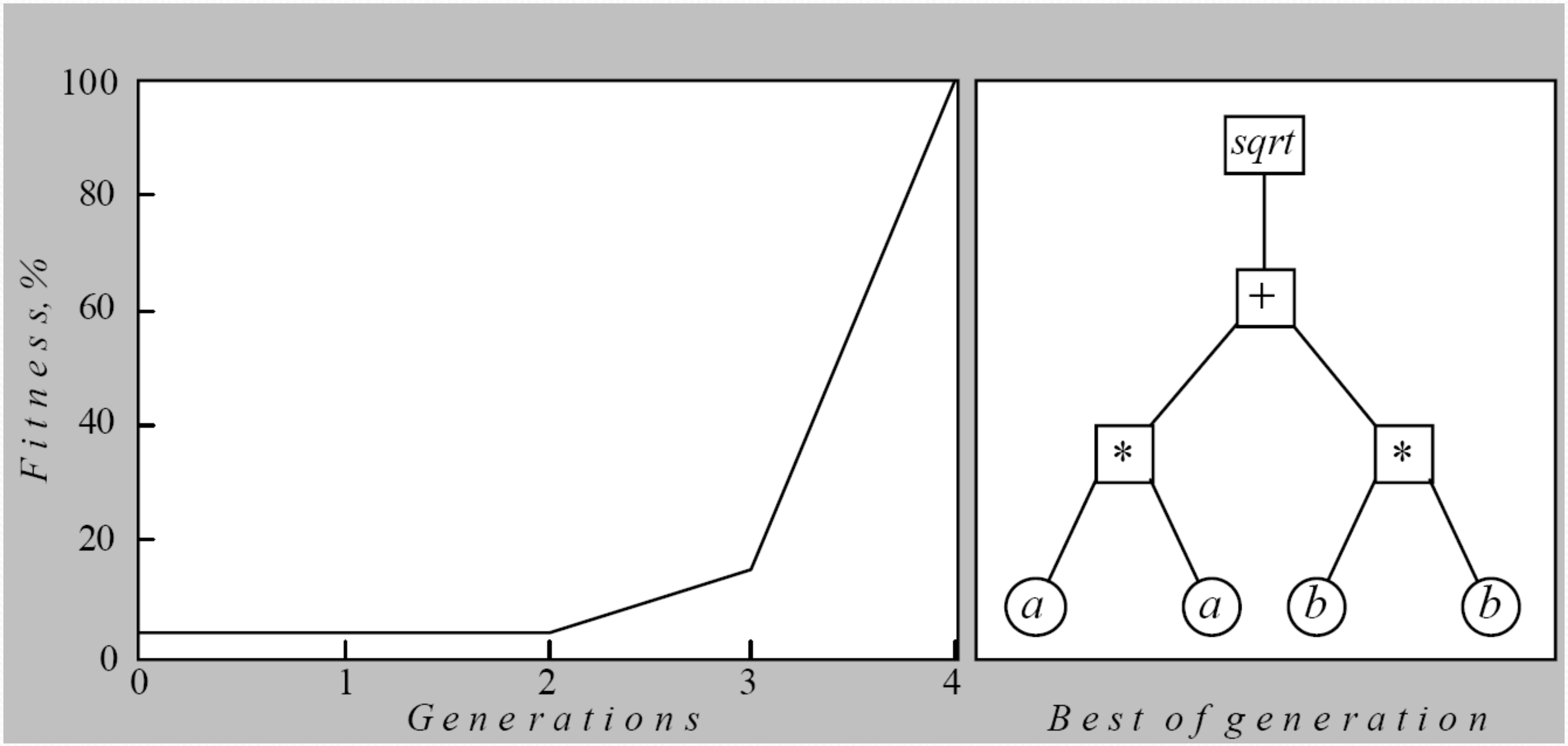
Underlying function: $c = \sqrt{a^2 + b^2}$

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



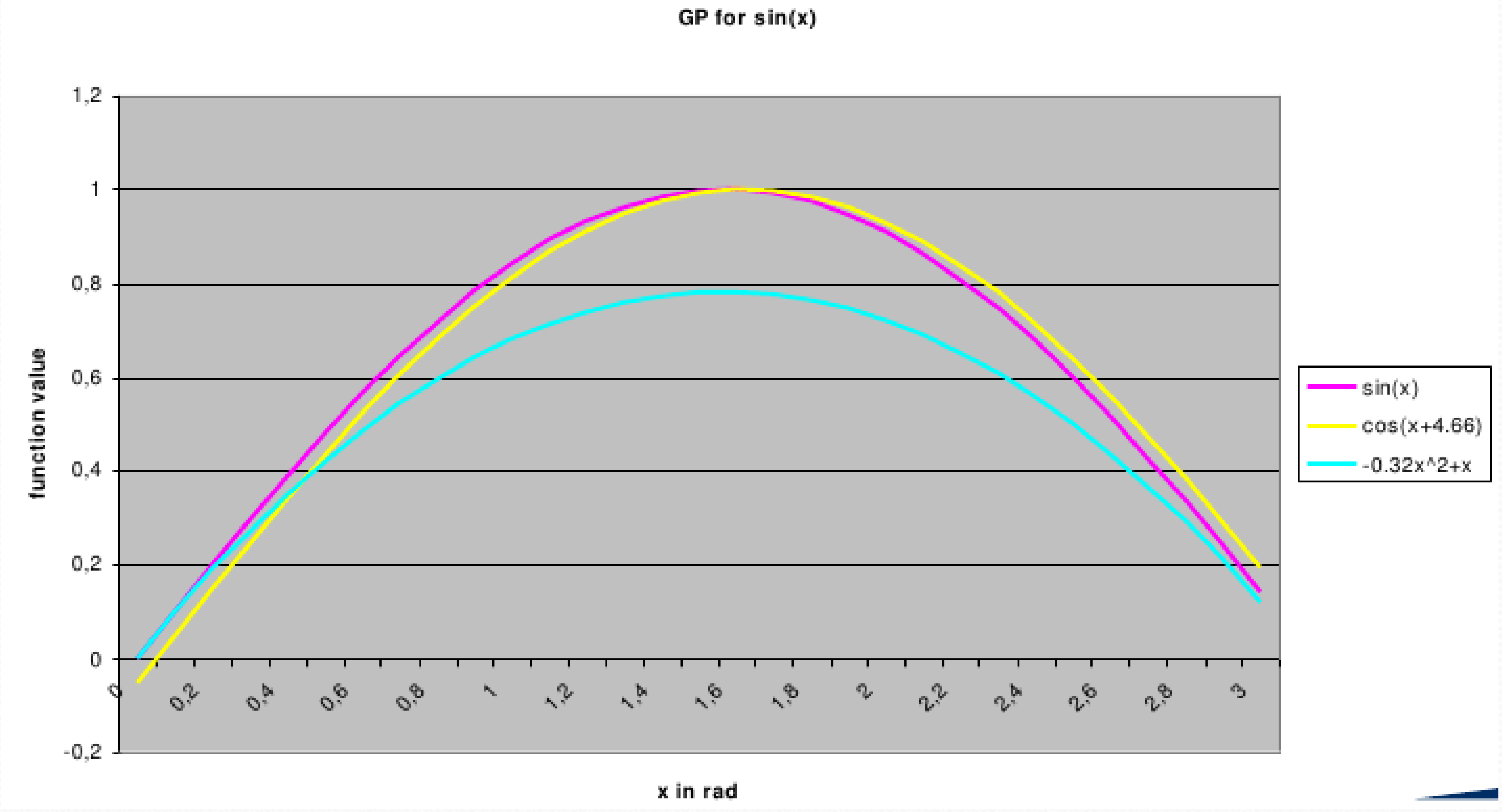
100
80
60
40
20
0
F i t n e s s, %
0
1
2
3
4
G e n e r a t i o n s
sqrt
+
*
*
a
a
b
b
B e s t o f g e n e r a t i o n

Example – Symbolic Regression Approximation of sin(x)

- **Given** examples (x,sin(x)) with x in {0,1,…,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: { +, -, *, / }	$-0.32\,x^2 + x$	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