

Evolutionary algorithms

- Simple genetic algorithms
- Evolutionary Strategies
- Genetic Programming

Partially based on slides by Thomas Bäck

Genetic algorithms

- Randomized search algorithms based on mechanics of natural selection and genetics
- Principle of natural selection through 'survival of the fittest' with randomized search
- Search efficiently in large spaces
- Robust w.r.t. to the complexity of the search problem
- Many applications in hybrid systems (Fuzzy logic, neural networks)

Advantages of GAs

- Evolution and natural selection has proven to be a robust and flexible method for adaptation
- GAs can be used to represent complex problems relatively easily compared to traditional approaches
- GAs can be easily parallelized and run on multiple machines

Basic elements

- Solutions require encoding
- Encoding is called a *chromosome* or *genotype*
- Solution represented by chromosome is the *phenotype*
- A number of *individuals* form a *population*
- Population is updated iteratively; each iteration is called a *generation*
- Objective function is called the *fitness function*
- Evaluate multiple solutions (in parallel) for the next search step

Some definitions

- **Population**: a collection of solutions for the studied (optimization) problem
- **Individual**: a single solution in a GA
- **Chromosome (genotype)**: representation for a single solution
- **Gene**: part of a chromosome, usually representing a variable as part of the solution

Some definitions

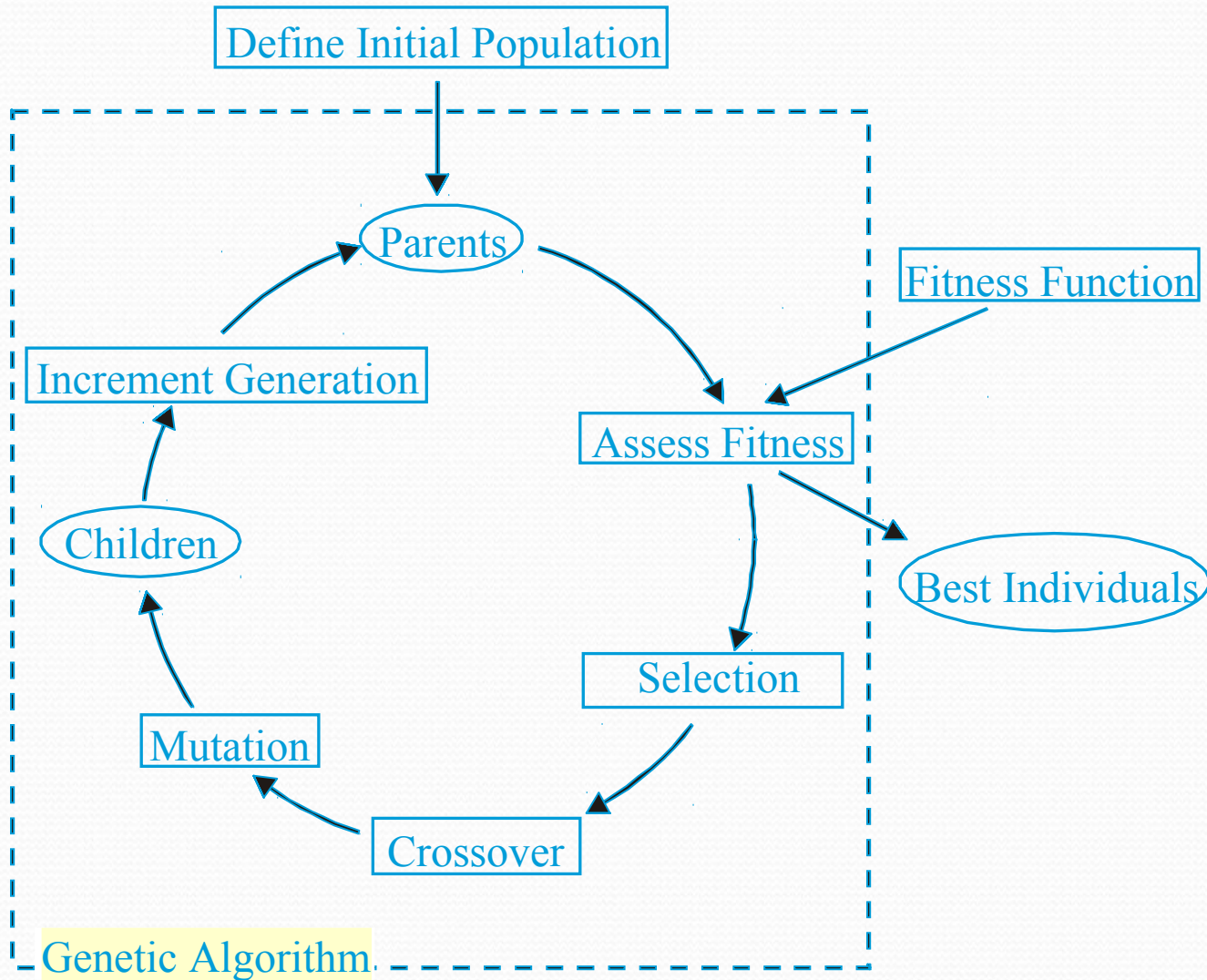
- **Encoding**: conversion of a solution to its equivalent representation (chromosome)
- **Decoding**: conversion of a chromosome (**genotype**) to its equivalent solution (phenotype)
- **Fitness**: scalar value denoting the suitability of a solution

GA terminology

Generation t

| | | x | y | | solution | fitness | | |
|------------|---|---|---|------|----------|------------|-------|---|
| population | } | 1 | 0 | 0 | 0 | individual | (2,0) | 4 |
| | | 0 | 1 | 0 | 1 | (1,1) | 2 | |
| | | 0 | 0 | 1 | 1 | (0,3) | 3 | |
| | | 0 | 1 | 1 | 0 | (1,2) | 3 | |
| | | 0 | 1 | 0 | 1 | (1,1) | 2 | |
| | | } | | gene | | | | |
| | | } | | | | chromosome | | |

Genetic algorithm



Pseudo code

- Initialize population P :
 - E.g. generate random p solutions
- Evaluate solutions in P :
 - determine for all $h \in P$, $\text{Fitness}(h)$
- **While** terminate is FALSE
 - Generate new generation P using genetic operators
 - Evaluate solutions in P
- **Return** solution $h \in P$ with the highest Fitness

Termination criteria

- Number of generations
(restart GA if best solution is not satisfactory)
- Fitness of best individual
- Average fitness of population
- Difference of best fitness (across generations)
- Difference of average fitness (across generations)

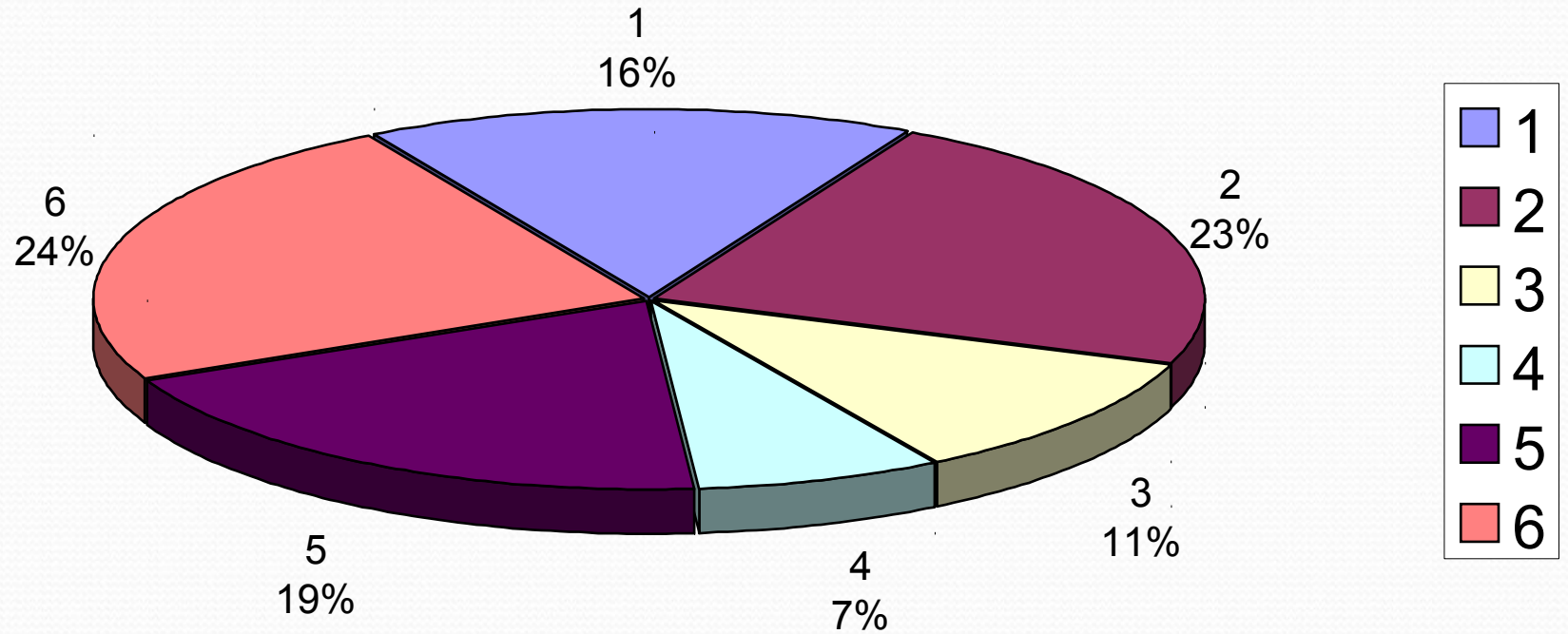
Reproduction

Three steps:

- Selection
- Crossover
- Mutation

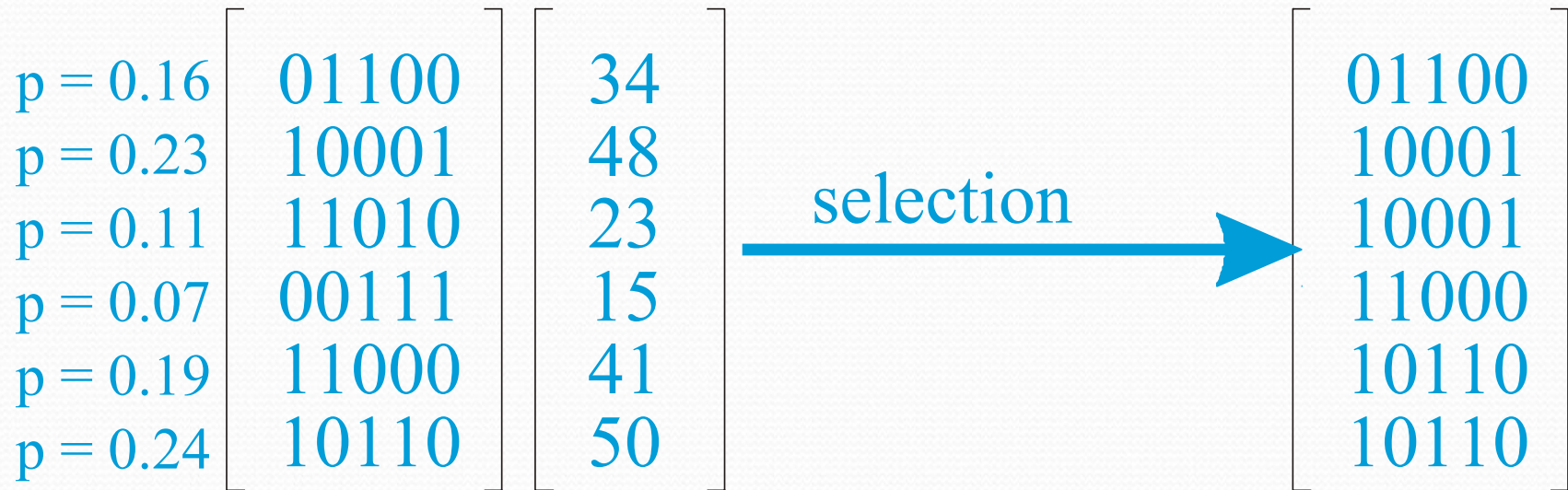
In GAs, the population size is often kept constant. User is free to choose which methods to use for all three steps.

Roulette-wheel selection



Roulette-wheel selection

individuals fitness



Sum = 211

Cumulative probability: 0.16, 0.39, 0.50, 0.57, 0.76, 1.00

Tournament selection

- Select pairs randomly
- Fitter individual wins
 - deterministic
 - probabilistic
 - constant probability of winning
 - probability of winning depends on fitness

It is also possible to combine tournament selection with roulette-wheel

Crossover

- Exchange parts of chromosome with a crossover probability (p_c is usually about 0.8)
- Select crossover points randomly

One-point crossover:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|



crossover point

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|

Uniform crossover

- Exchange bits using a randomly generated mask

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|

mask

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|



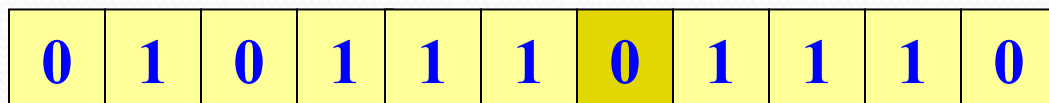
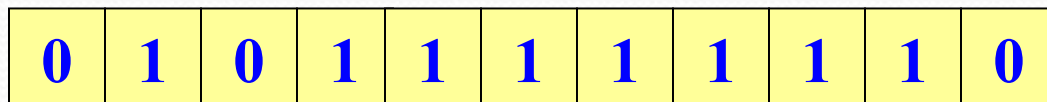
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|

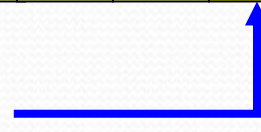
Mutation

- Crossover is used to search the solution space
- Mutation is needed to escape from local optima
- Introduces genetic diversity
- Mutation is rare (p_m is about 0.005)

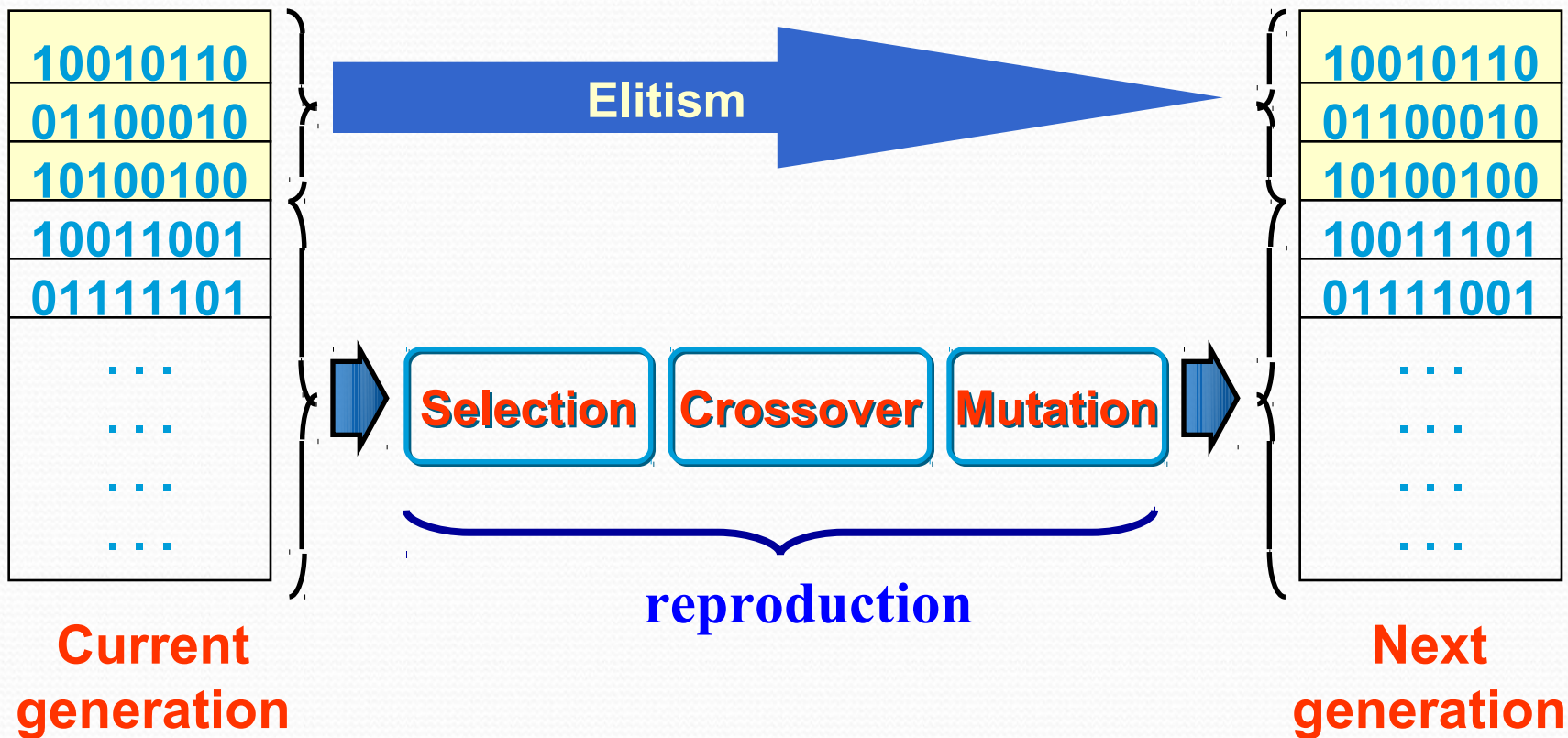
Uniform mutation:



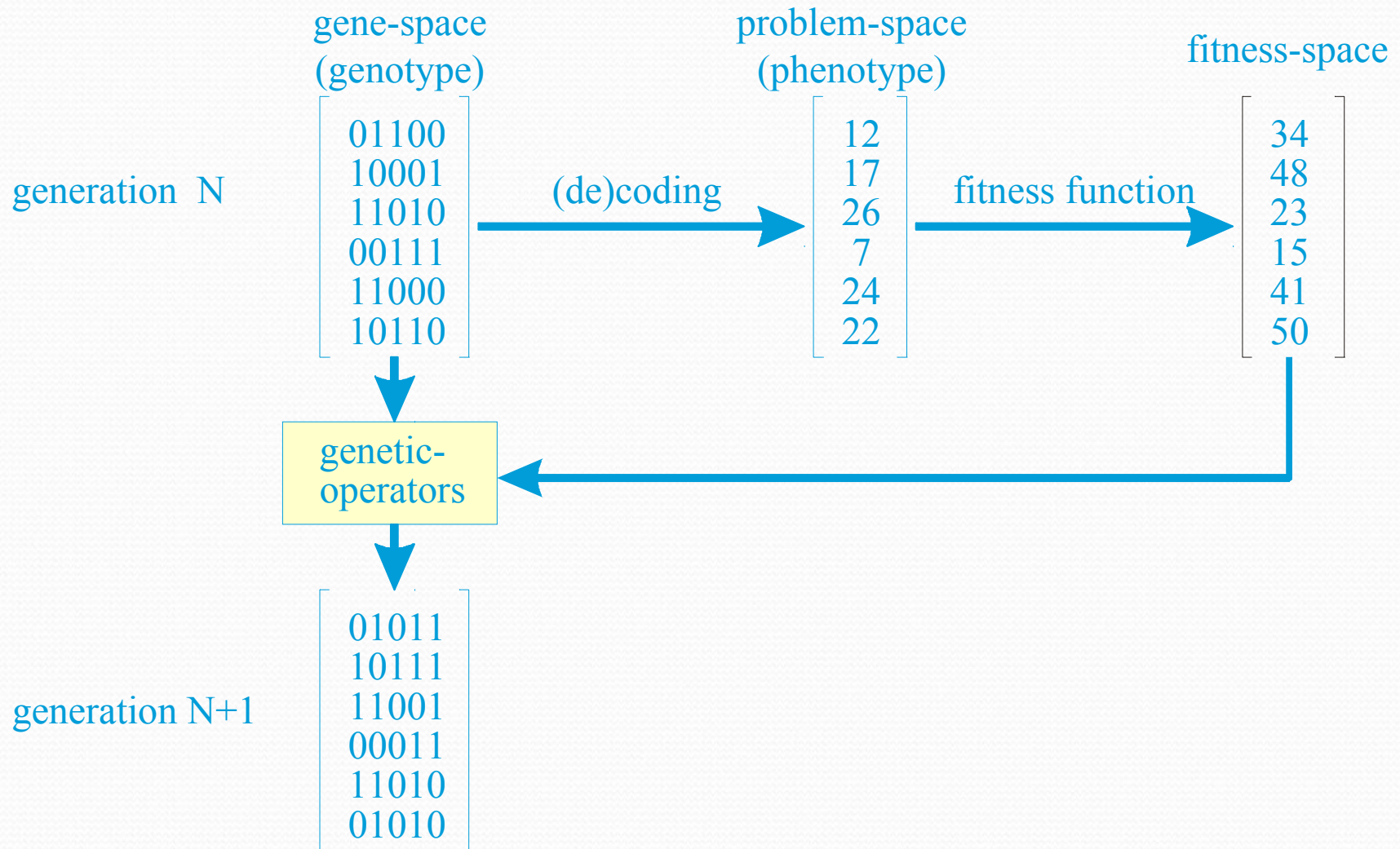
mutated bit



GA iteration



Spaces in GA iteration



Encoding and decoding

- Common coding methods for numbers
 - simple binary coding
 - Gray coding (binary)
 - real valued coding (*evolutionary strategies*)

Constraints

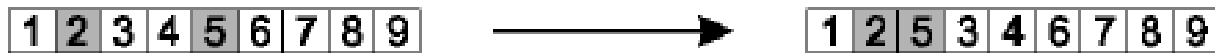
- Examples:
 - “A string of numbers should represent a permutation”
(1,2,3) is valid; (1,1,3) is not
 - “The sum of numbers should not be lower than a threshold”
- Explicit handling: fitness function modification
 - setting fitness of unfeasible solutions to zero
(search may be very inefficient due to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

Constraints

- Implicit (preferred method): special encoding
 - GA searches always for allowed solutions
 - smaller search space
 - ad hoc method, may be difficult to find
- Example: permutations

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

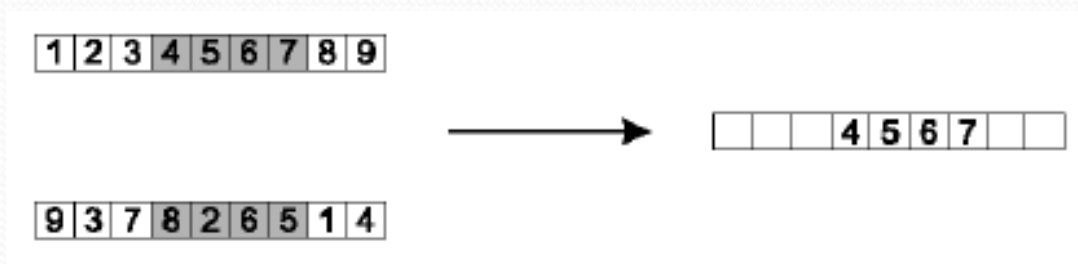
Mutations for Permutations

- 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



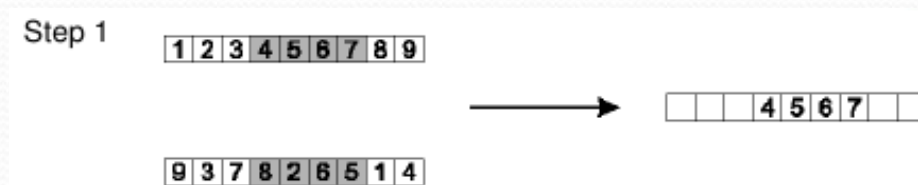
- Copy the numbers that are not in the first part, to the first child:
 - starting right from cut point of the copied part,
 - using the order of the second parent and wrapping around at the end



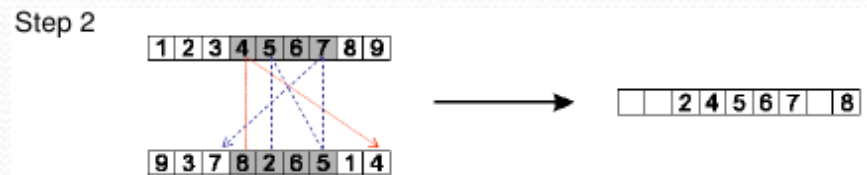
- Analogous for the second child, with parent roles reversed

Crossover for Permutations

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

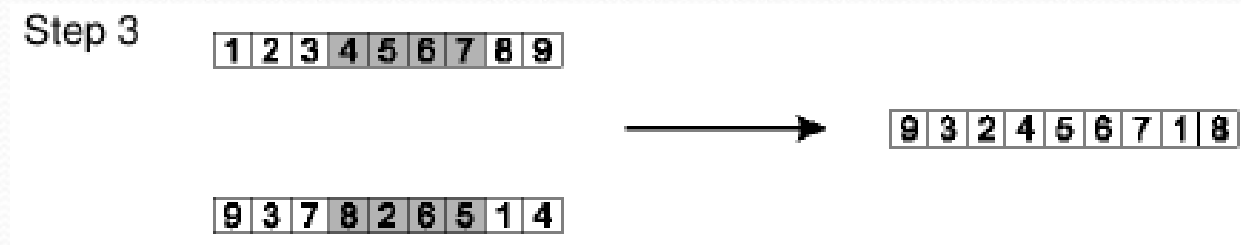


- Starting from the first crossover point look for elements in that segment of P_2 that have not been copied
- For each of these i look in the offspring to see what element j has been copied in its place from P_1
- Place i into the position occupied j in P_2 , since we know that we will not be putting 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.



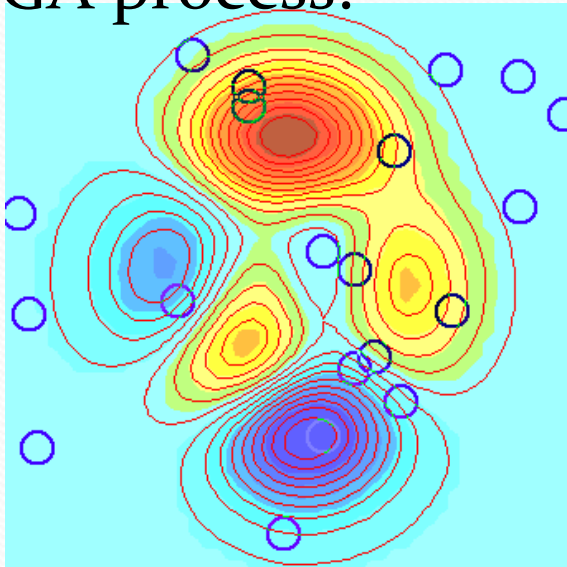
- Idea: maintain position

Order vs Position in Permutations

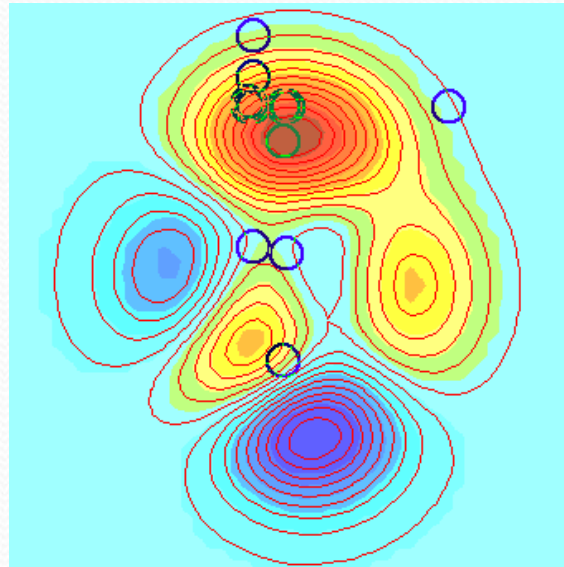
- Order, but not position of numbers is important in problems such as the traveling salesman problem (visiting all cities in a certain order)
- Position, but not order of numbers is important in problems such as allocating visitors in hotels to rooms (visitors have to be allocated once to one room, but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

Example

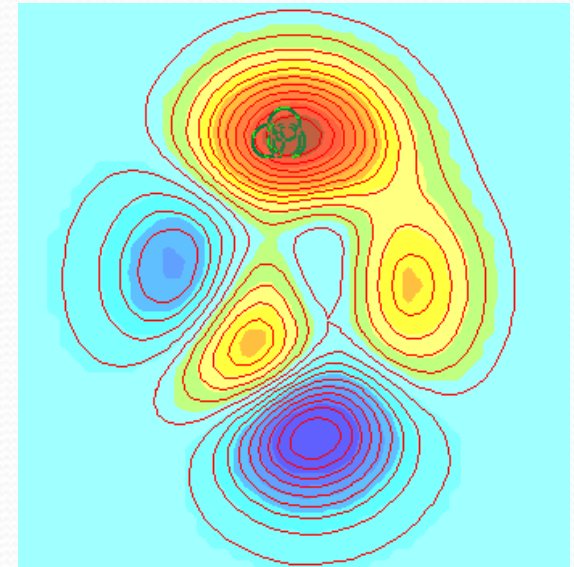
GA process:



Initial population

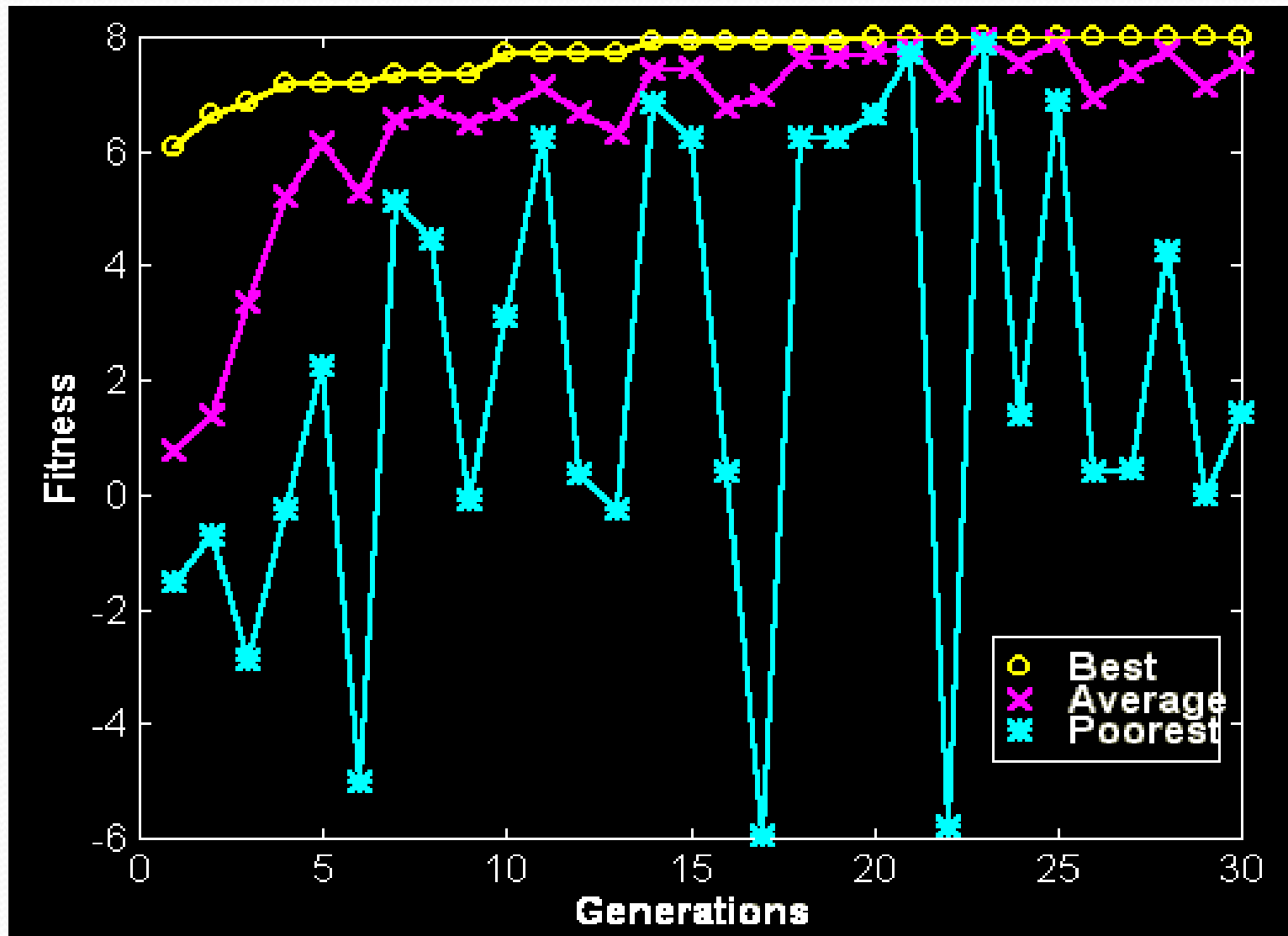


5th generation



10th generation

Performance profile

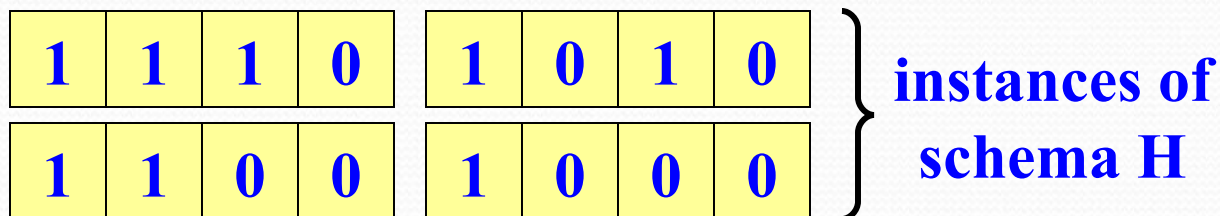
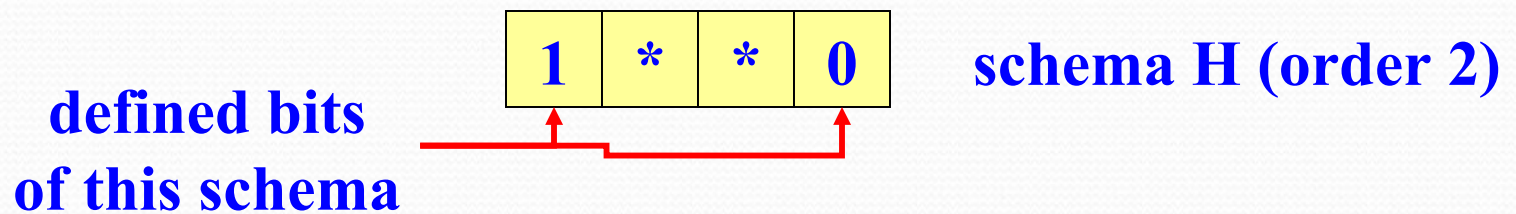


Schema Theory

- Why do genetic algorithms work?
- We will prove that genetic algorithms focus their search on high-quality combinations of high-quality building blocks

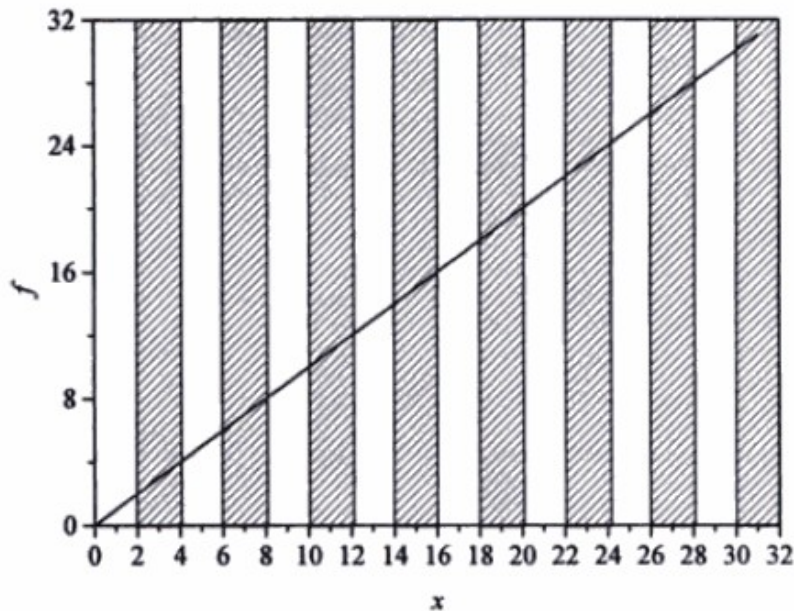
Schema

- A schema is a bit string with 1's, 0's and don't cares (*)
- It represents parts of a chromosome
- Order of schema: number of defined bits in the schema

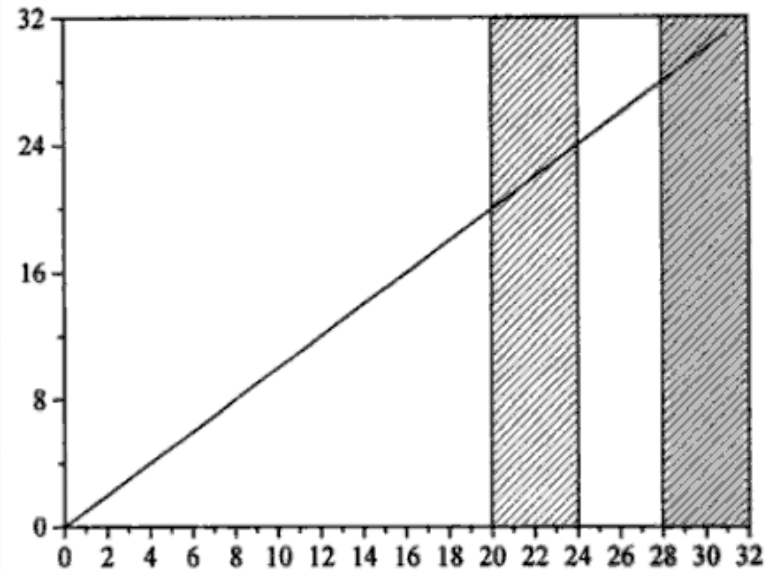


Schema Examples

- $f(x)=x$, “standard” binary encoding



Schema $***1^*$
(Mediocre fitness)



Schema 1^*1^{**}
(Good fitness)

Schema Theory

- Given $\xi(S, t)$, the number of strings in a population at time t matched by schema S
- Estimate $\xi(S, t + 1)$
- Step 1: Selection
 - Let $\bar{f}(P) = \sum_{h \in P} f(h) / |P|$ (Average fitness of population)
 - Let $\bar{f}(S, P) = \sum_{h \in P'} f(h) / |P'|$ (Average fitness schema)
where P' is the set of individuals in P matched by S
 - Then it is expected that $\xi(S, t + 1) = \xi(S, t) \frac{\bar{f}(S, P)}{\bar{f}(P)}$
(after selection only)

Schema Theory

- Step 2: Crossover

- Let $\delta(S)$ be the defining length of the schema, i.e. the distance between the first and last defined elements

$$**1**1* \rightarrow 6 - 3 = 3$$

$$1*****1 \rightarrow 7 - 1 = 6$$

- Let n be the length of the strings in the population
- Then crossover will “destroy” a schema with probability

$$p_c \cdot \frac{\delta(S)}{n - 1}$$

Schema Theory

- Step 3: Mutation

- Let $o(S)$ be the order of schema S
- Then the schema remains unchanged with probability

$$(1 - p_m)^{o(S)}$$

- Overall, after selection, crossover and mutation, the expected number of instances of a schema is

$$\xi(S, t + 1) = \xi(S, t) \frac{\bar{f}(S, P)}{\bar{f}(P)} \left(1 - p_c \frac{\delta(S)}{n - 1}\right) (1 - p_m)^{o(S)}$$

Schema Theory

- **Schema Theorem**

Short, low-order, above average schemata receive exponentially increasing trials in subsequent generations of a genetic algorithm

$$\xi(S, t + 1) = \xi(S, t) \frac{\bar{f}(S, P)}{\bar{f}(P)} \left(1 - p_c \frac{\delta(S)}{n - 1}\right) (1 - p_m)^{o(S)}$$

- **Building Block Hypothesis**

A genetic algorithm seeks near-optimal performance through combination of short, low-order, high-performance schemata, called the building blocks

Implicit Parallelism

- A population of size $|P|$ with individuals of length n represents between 2^n and $2^{|P|n}$ schemata
- Hence, many schemata are implicitly considered at the same time

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

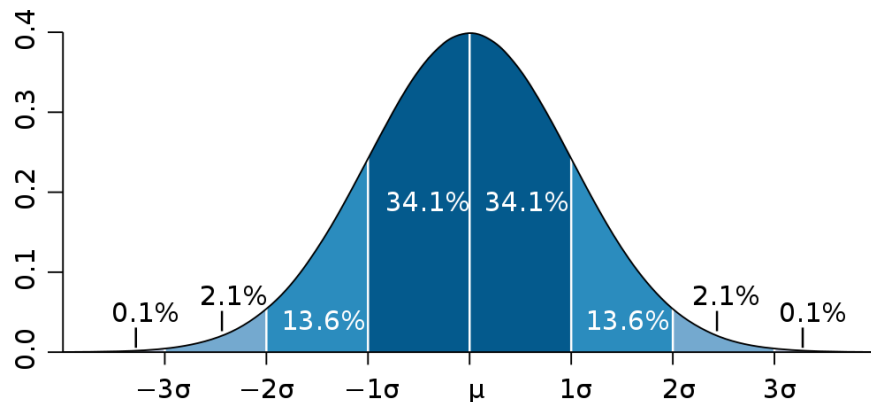
- Not performed *before* mutation and crossover, but after these operations
- It is assumed mutation (& crossover) generate $\lambda > \mu$ individuals (where μ is population size) (typically $\lambda \approx 7\mu$)
- Deterministically eliminate worst individuals from
 - children only: (μ, λ) -ES \rightarrow 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:

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

“sampled from”



Simple modification:
mutation rate for each x_i

Major question:
How to set σ or σ_i ?

MAIN IDEA: 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

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

$c \in [0.8, 1.0]$

Increase mutation rate as we are successful

until termination

“1/5 rule”

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

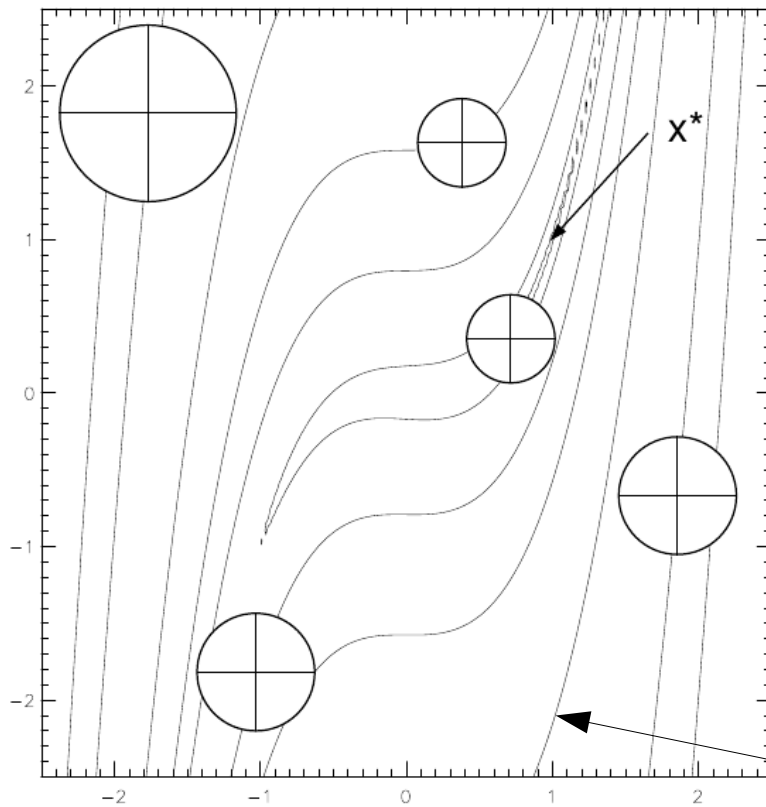
- Here τ_0 is the mutation rate
 - τ_0 bigger: faster but more imprecise
 - τ_0 smaller: slower but more imprecise
- Recommendation for setting τ_0 :

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

*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

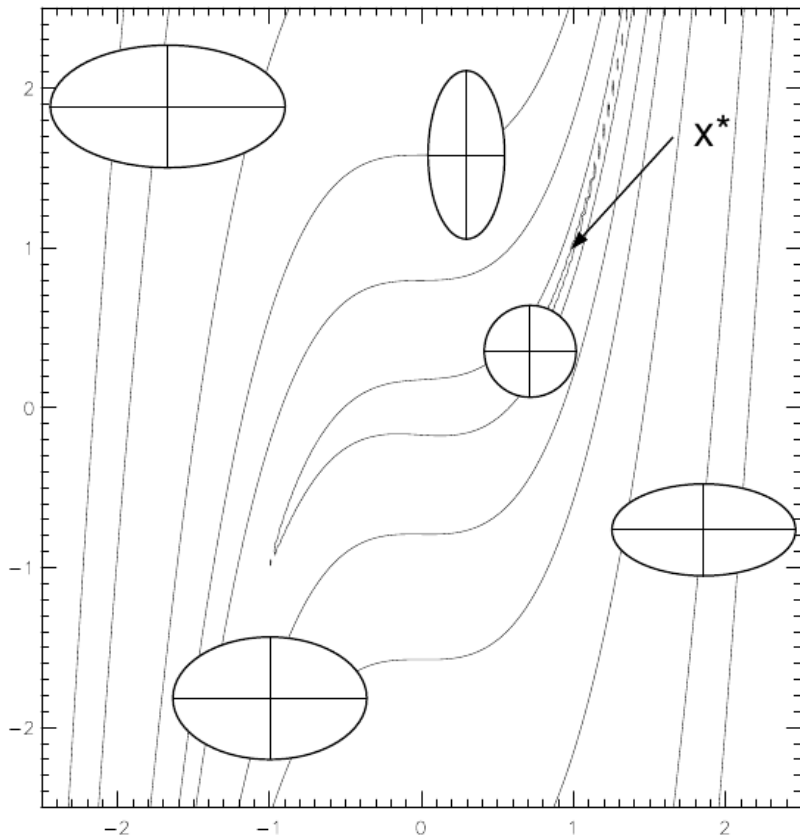
$$\vec{h} = (x_1, x_2, \sigma)$$

- 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

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

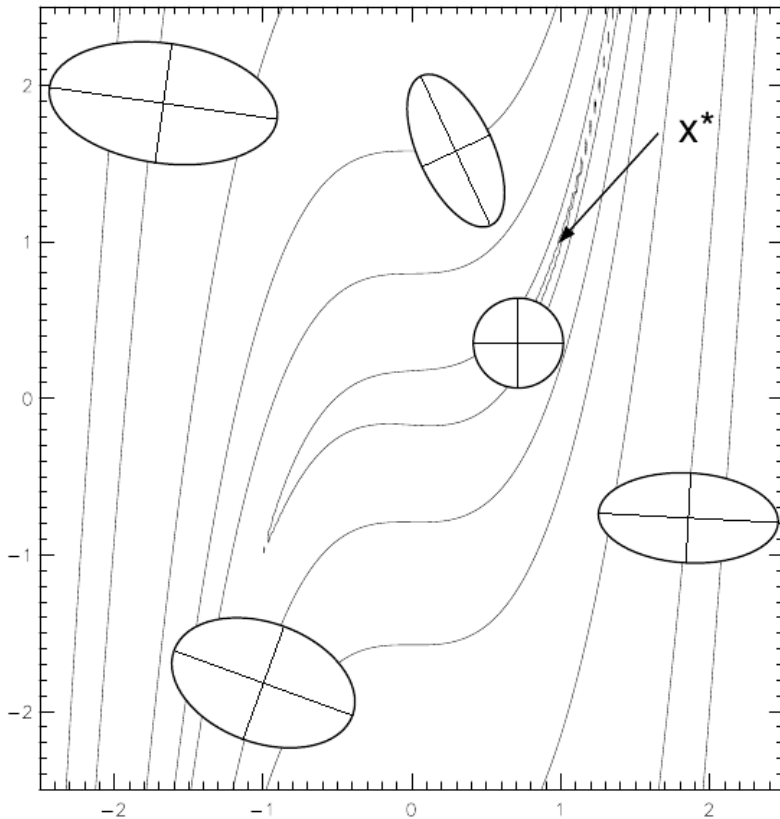
Sample from normal distribution,
the same for all parameters

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

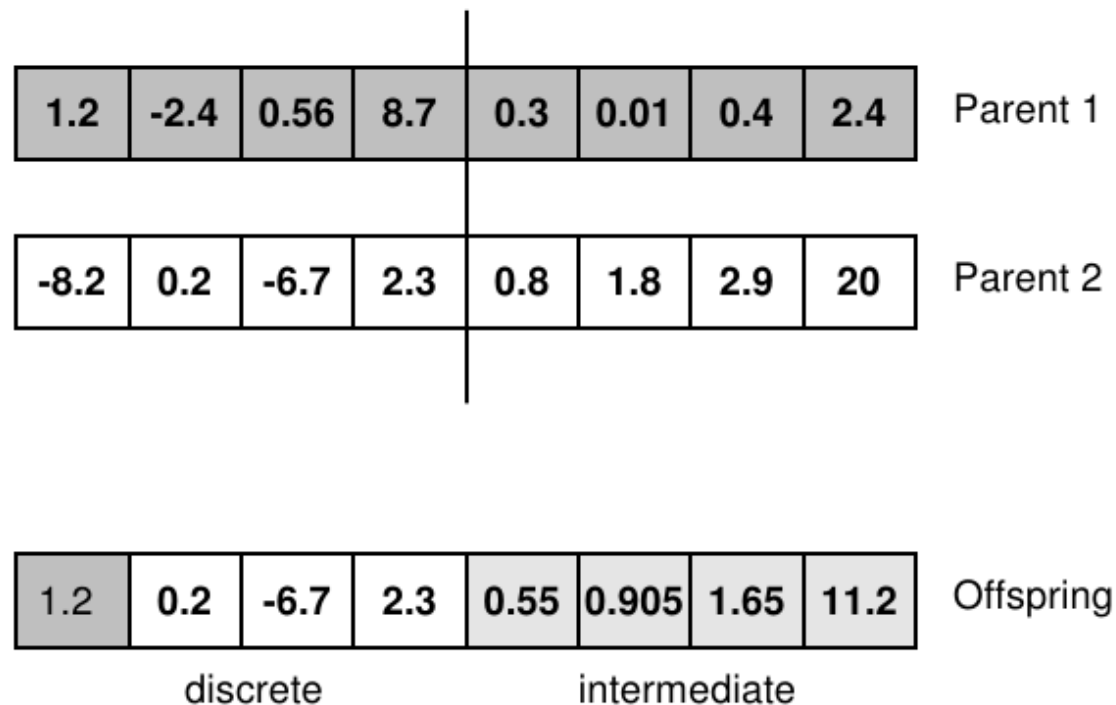
- Also here mutation can be defined
- Mathematical details skipped

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

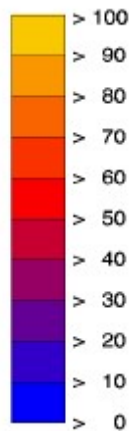
Evolution strategies

- 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

Example

- Casting of blades in gas turbine manufacturing

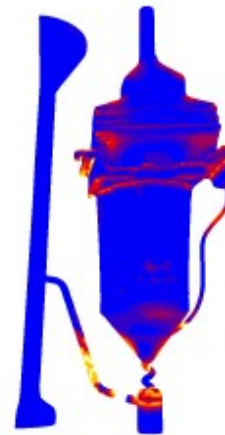
- Objective: Max. homogeneity of workpiece after Casting Process
- Variables: 18 continuous speed variables for Casting Schedule
- Computationally expensive simulation (up to 32h simulation time)



Initial (DoE)



GCM (Commercial
Gradient Based Method)



Evolution Strategy

Turbine Blade
after Casting

Example

- Traffic light control optimization



- **Objective:** Minimization of total delay / number of stops
- **Variables:** Green times for next switching schedule
- **Dynamic optimization**, depending on actual traffic
- Better results (3-5%)
- Higher flexibility than with traditional controllers

Genetic Programming

- Main idea:
individuals are (simple) *computer programs* of arbitrary size
- Redefinitions of
 - selection
 - crossover
 - mutation

Program Trees / Parse Trees

- Representation of
 - Arithmetic formulas

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

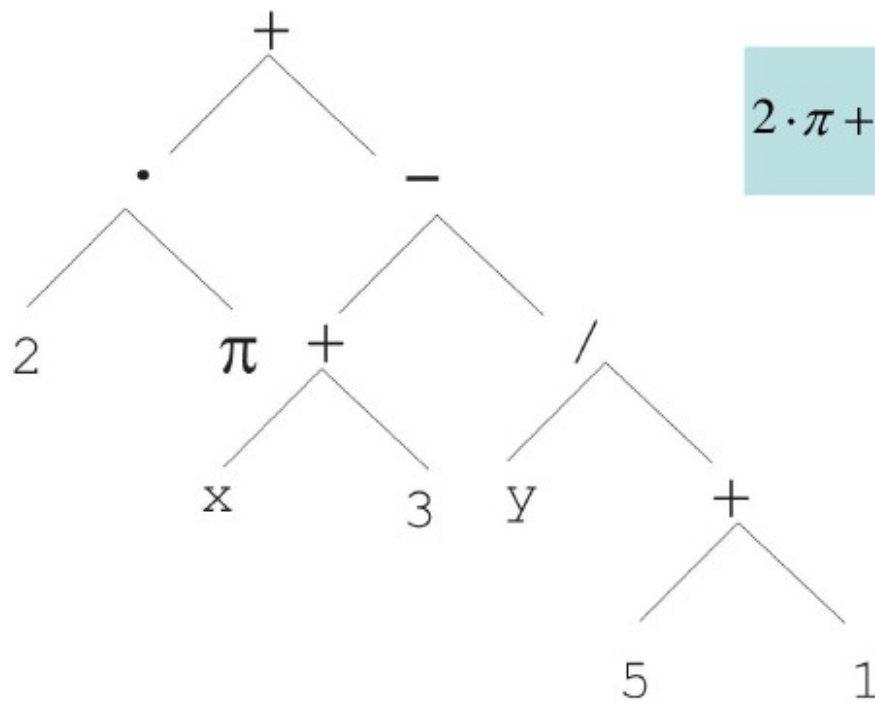
- Logical formulas

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

- Computer programs

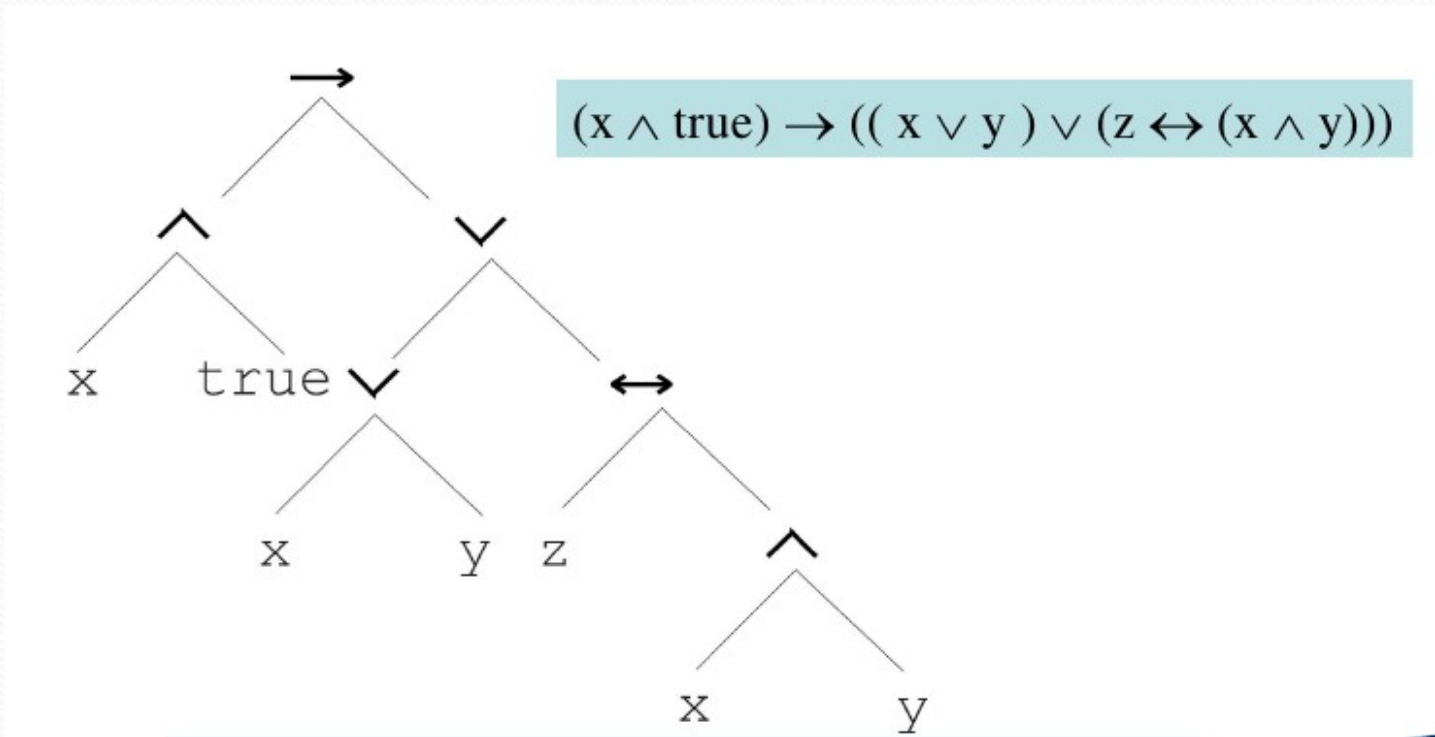
```
i = 1;
while (i < 20)
{
    i = i + 1
}
```

Representation of Arithmetic Formula

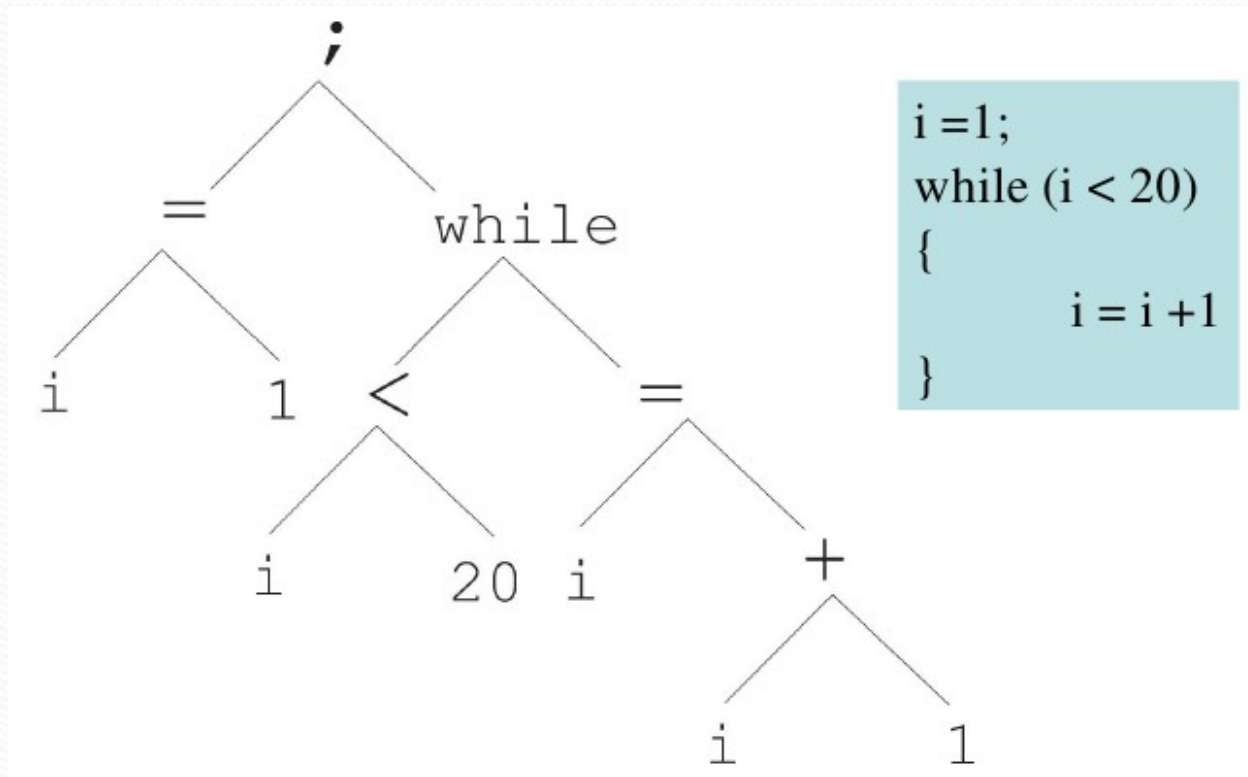


$$2 \cdot \pi + \left((x + 3) - \frac{y}{5 + 1} \right)$$

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)

Representation Issues

- **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

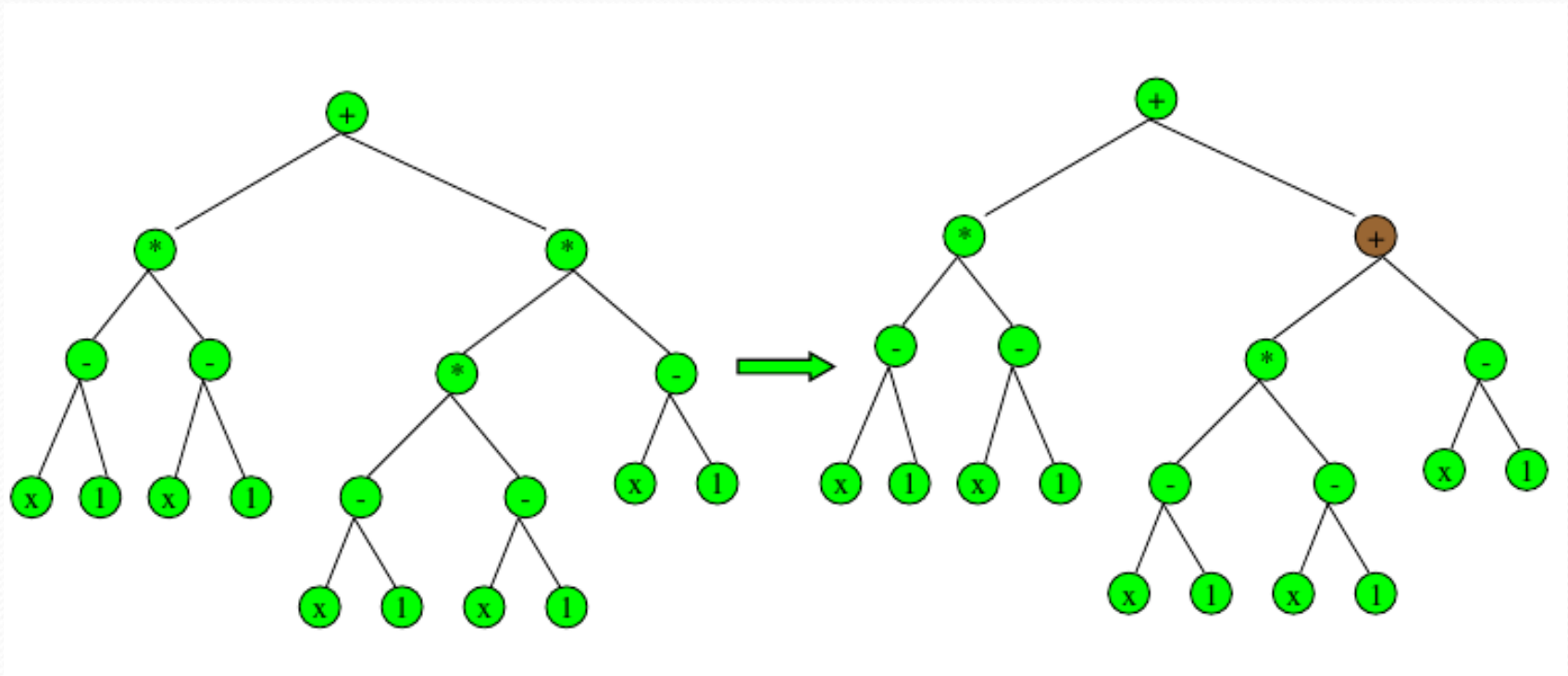
Initialization

- 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

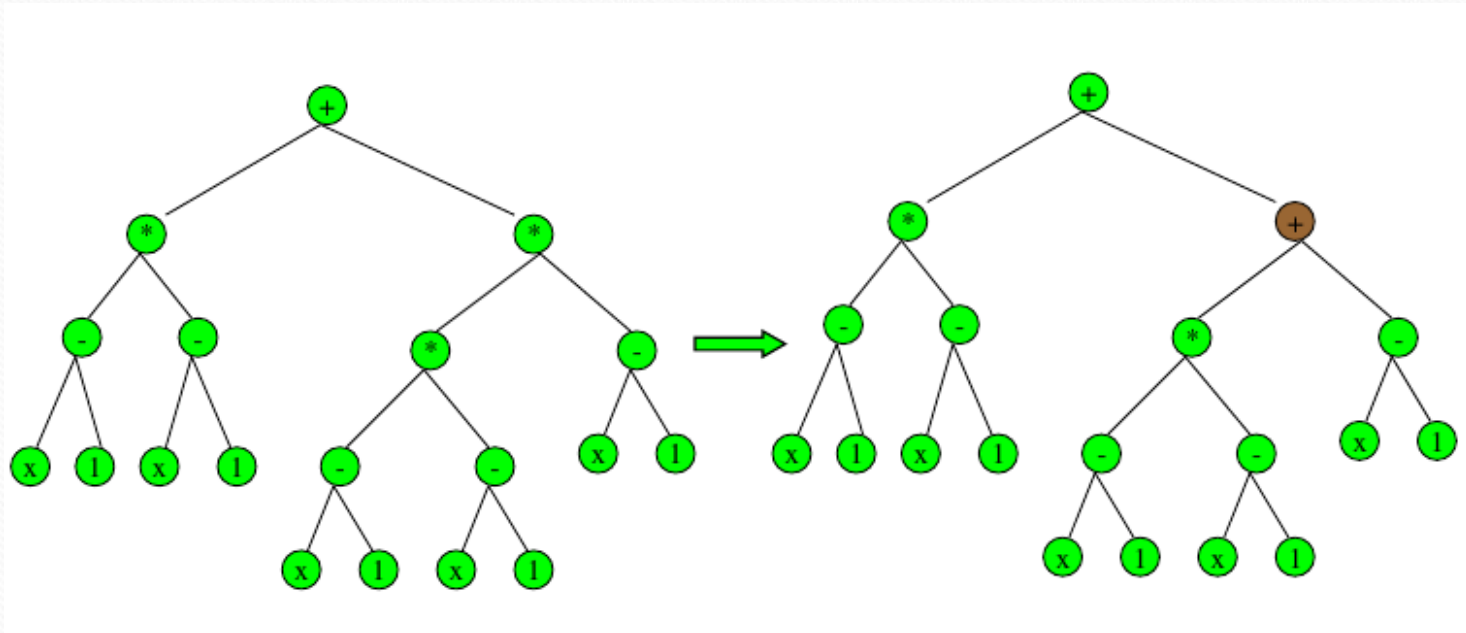
Mutation

| 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 |

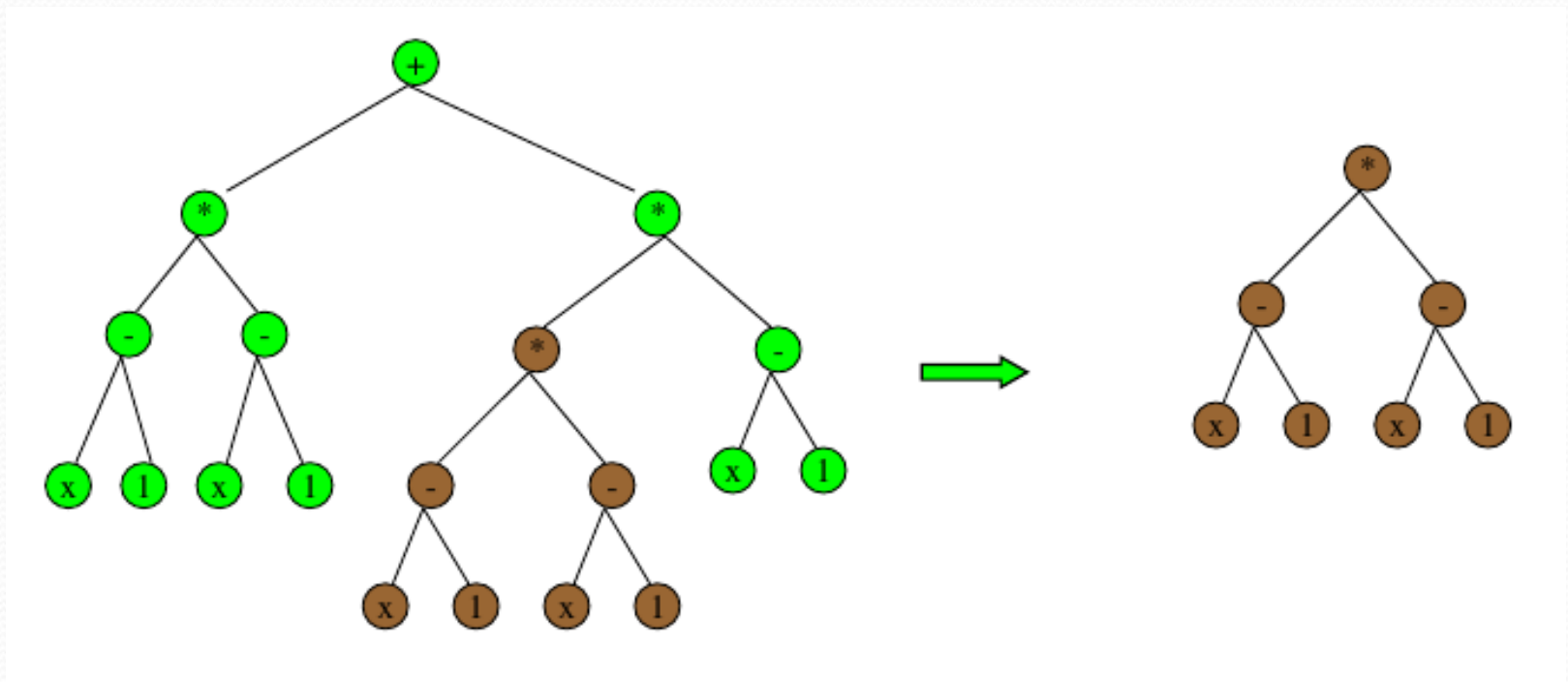
Point Mutation



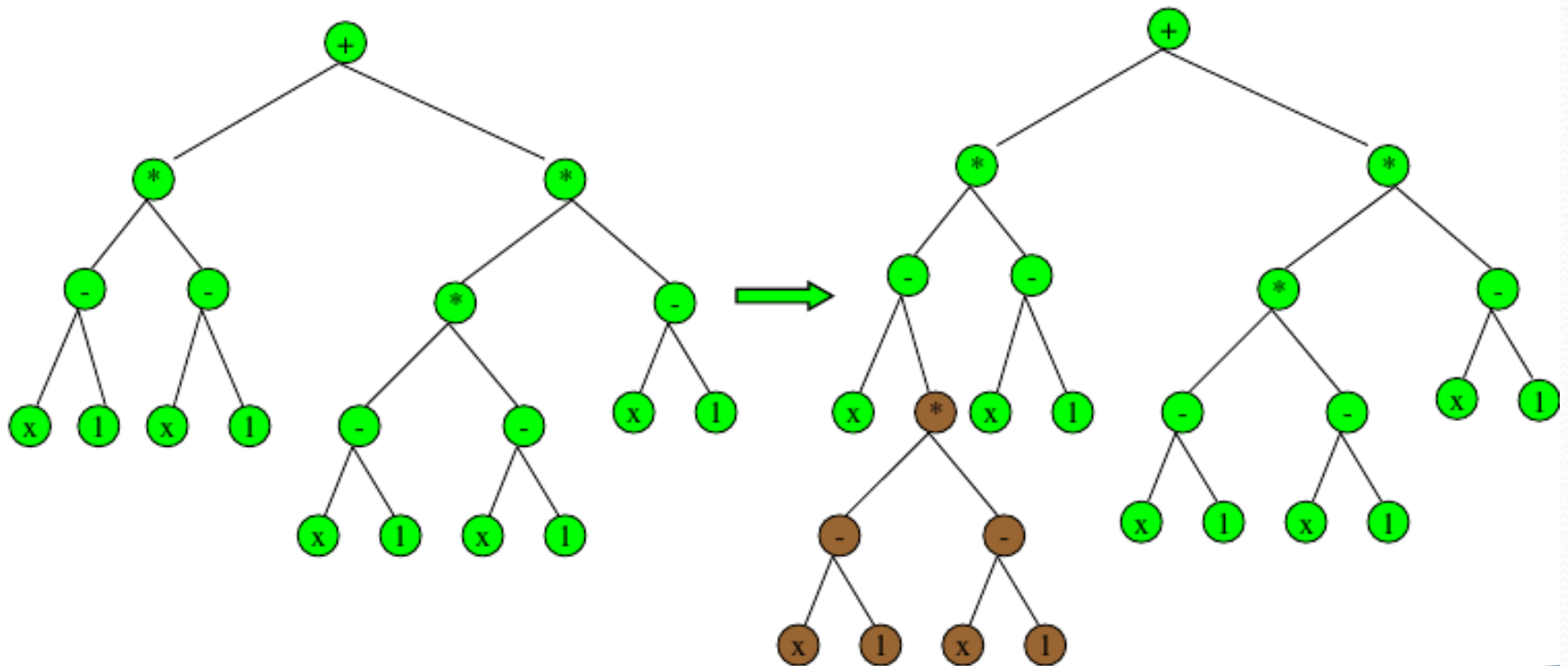
Permutation



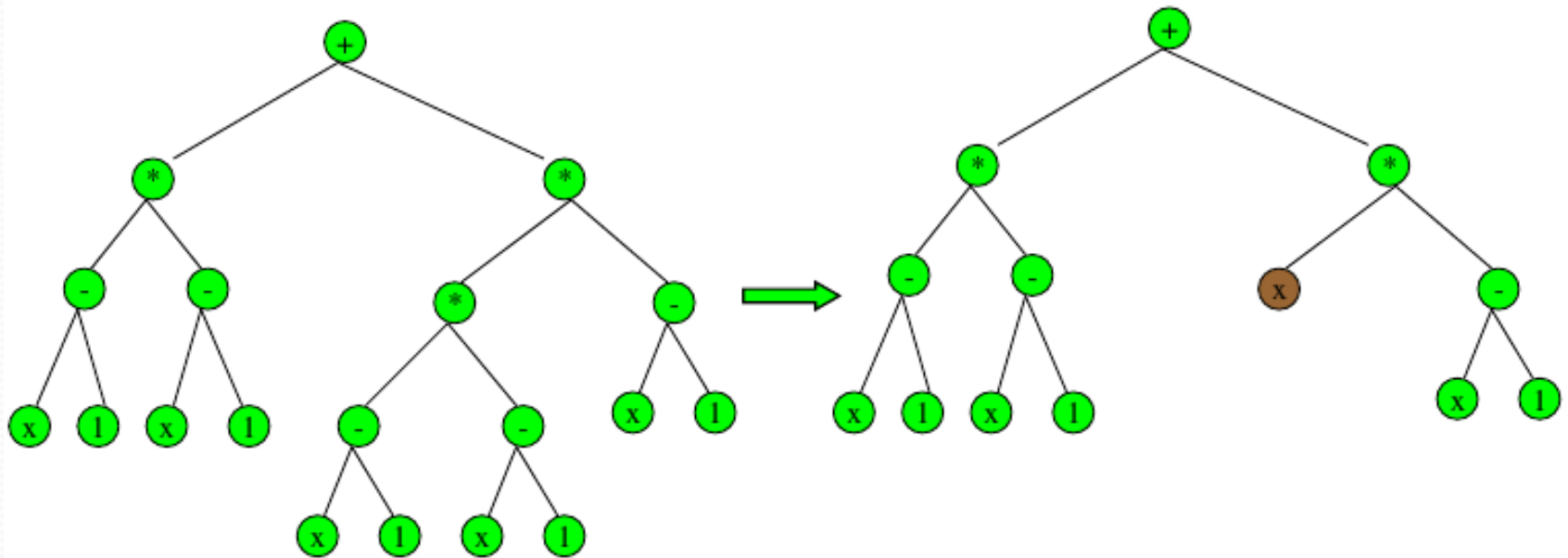
Hoist



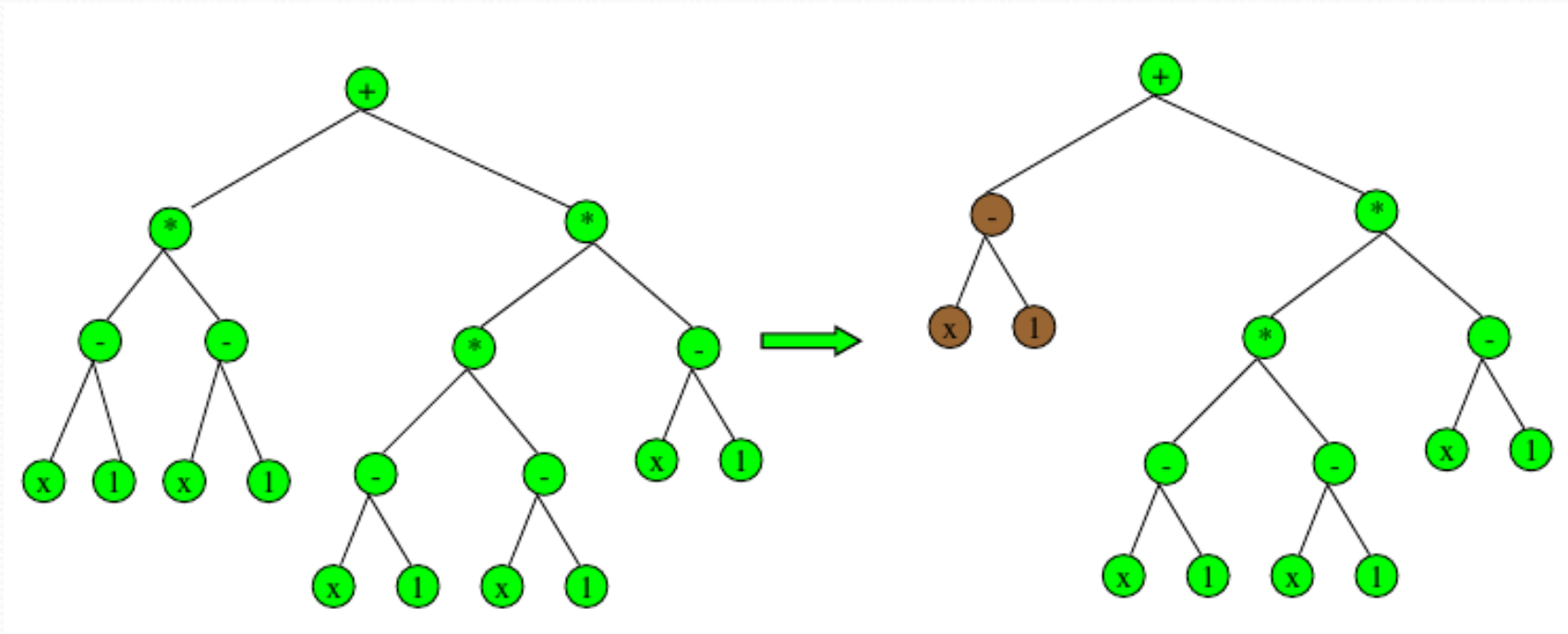
Expansion Mutation



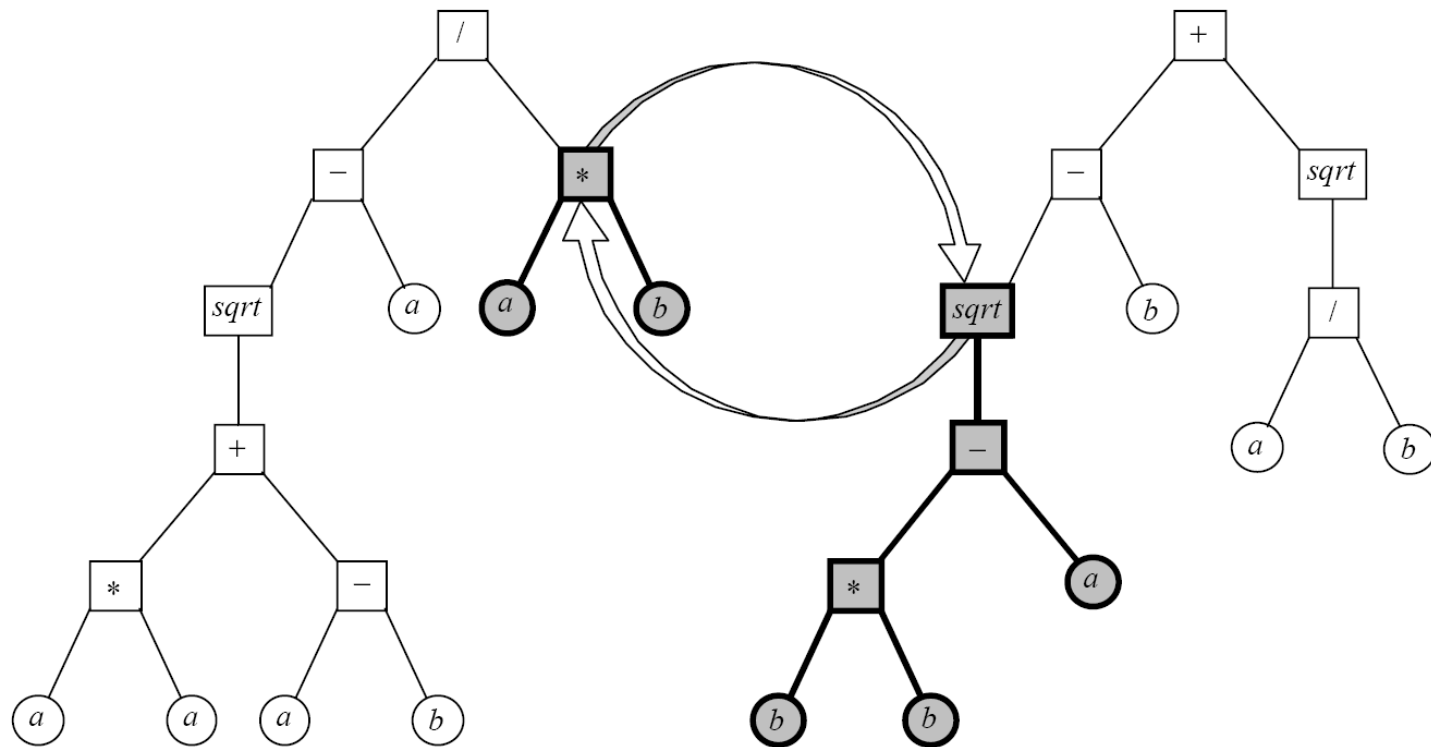
Collapse Subtree Mutation



Subtree Mutation



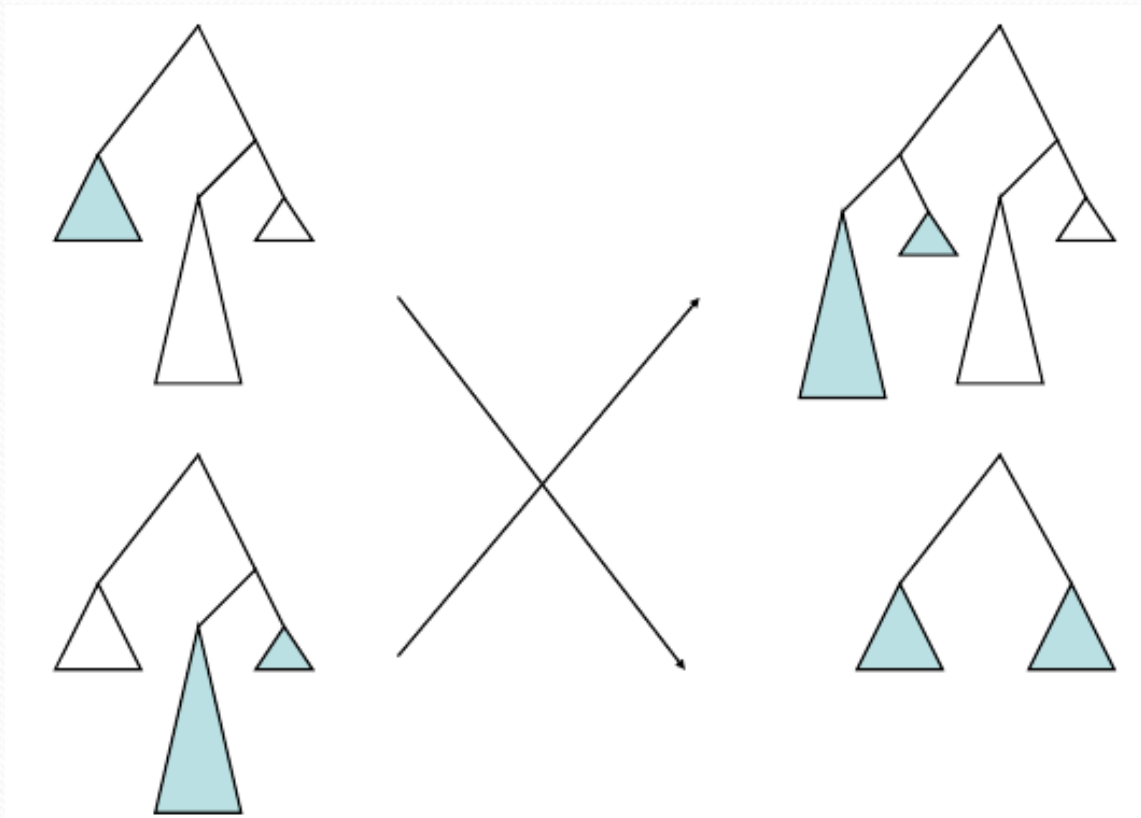
Crossover



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

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

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:
 - $X \text{ AND } X \rightarrow X$
 - $X \text{ OR } X \rightarrow X$
 - $\text{NOT}(\text{NOT}(X)) \rightarrow X$
 - $X + 0 \rightarrow X$
 - $X . 1 \rightarrow X$
 - $X . 0 \rightarrow 0$
 -

Preparatory steps to apply GP

- Determine the set of terminals
 - Atoms of the language (inputs of the program, constants)
- Select the set of primitive functions
 - Basic functions of (programming) language
- Define the fitness function
- Decide on control parameters
 - Similar to GA parameters: population size, stopping condition, crossover rate, etc.
- Choose method to designate the result of a run
 - Usually, best individual encountered

Preparatory steps to apply GP

- Determine the set of terminals
 - Atoms of the language (inputs of the program, constants)
- Select the set of primitive functions
 - Basic functions of (programming) language
- Define the fitness function
- Decide on control parameters
 - Similar to GA parameters: population size, stopping condition, crossover rate, etc.
- Choose method to designate the result of a run
 - Usually, best individual encountered

Example – Symbolic Regression Pythagorean Theorem

Not (necessarily)
linear

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

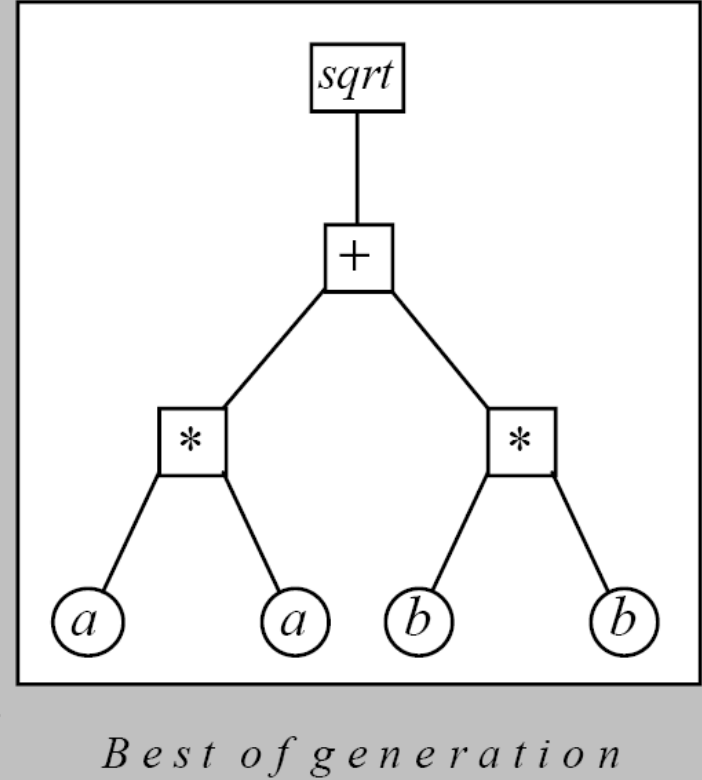
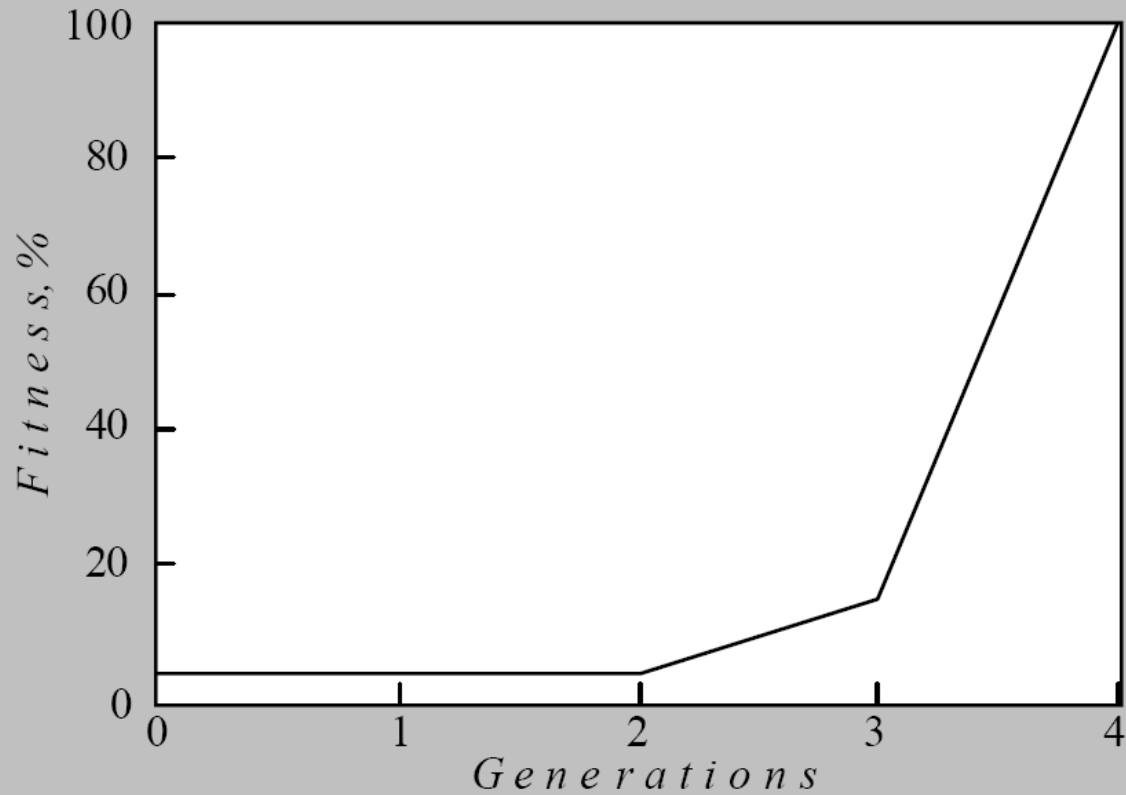
Negnevitsky 2004

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

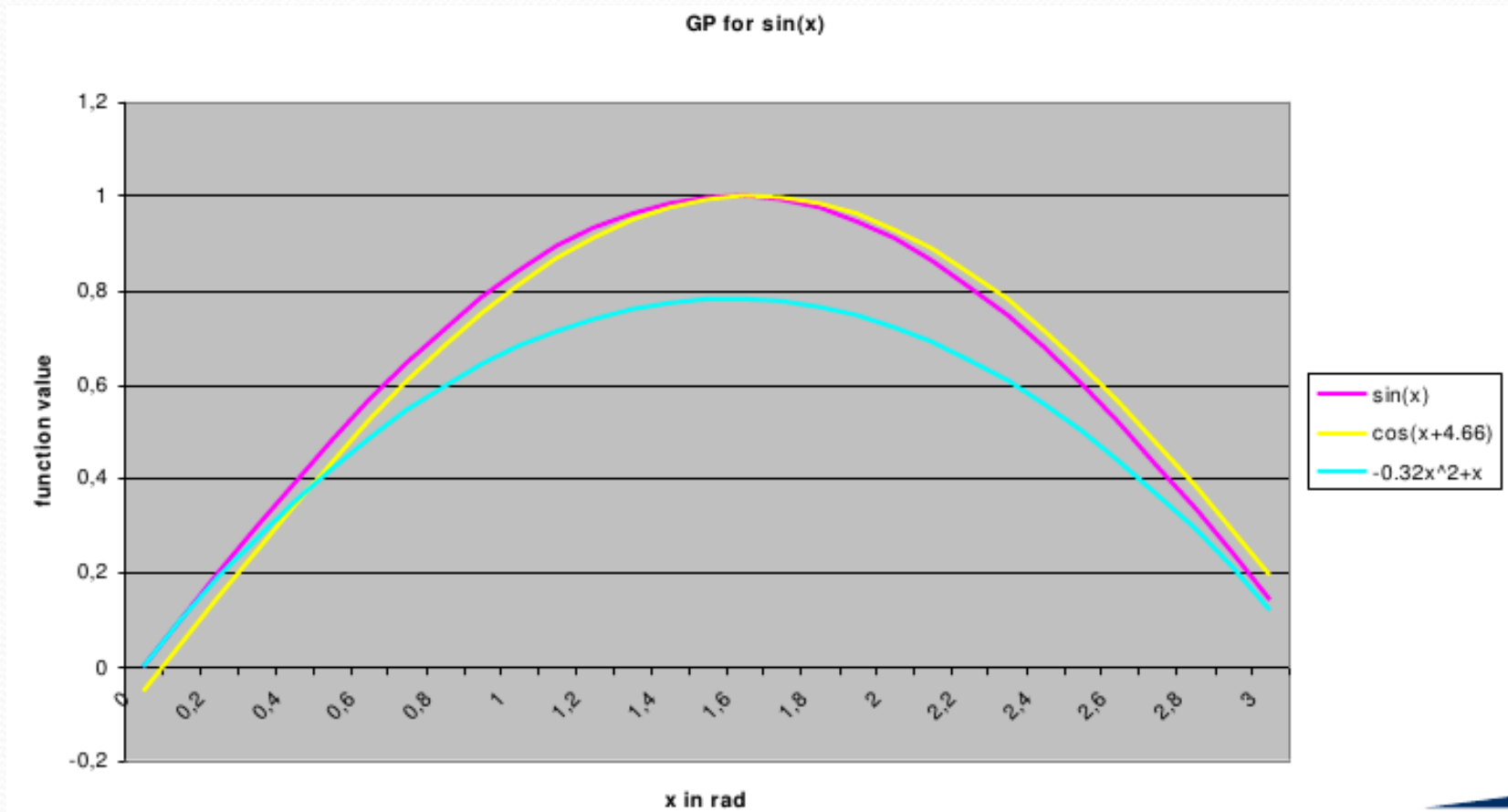
Approximation of $\sin(x)$

- **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