Artificial Intelligence

Lecture 13: Local Search and Evolutionary Computation

Dr. Benjamin Inden

Department of Computing & Technology, Nottingham Trent University

January 28, 2018

Lecture for this week

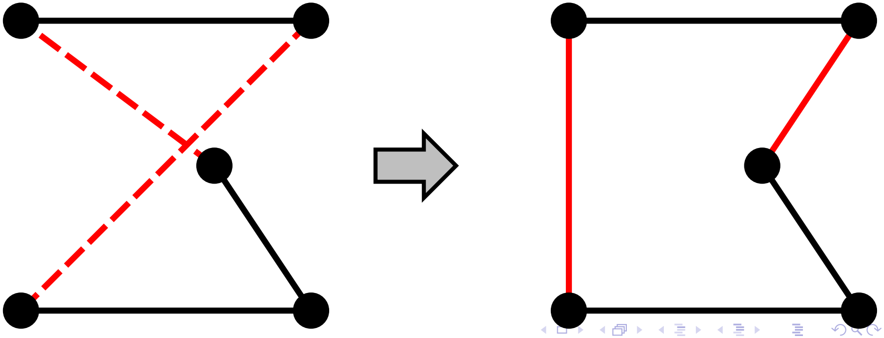
- Based on Lucci & Kopec, Chapter 11; Luke (2013), Essentials of Metaheuristics; Poli et. al (2008), A Field Guide to Genetic Programming; some images and other materials from Wikipedia.
- Please do not post these slides to the internet or use other means of public distribution. They are for your personal use only.

Searching without planning (but with a heuristic)

- For some problems, the particular path taken to the goal is not important. It is just important to be at a goal (or as close as possible to one).
- Consider driving through Romania without a proper map (but with information on the straight line distance).
- For such problems, we typically have a solution condidate and try to iteratively improve it.
- Usually, the new candidates are derived from the old ones by slight modifications. They are their “neighbors” in some sense. Such algorithms are called local search algorithms.

The traveling salesperson problem

- Given are a number of cities (nodes) with roads (edges) between them and their respective lengths/travel costs.
- The task is to find the shortest route that goes trough all cities exactly once, and returns to the starting point.
- The problem is NP complete, but slightly suboptimal solutions can be found quickly by local search
- Neighbors of a candidate can be found by changing the order of a few cities on the route.

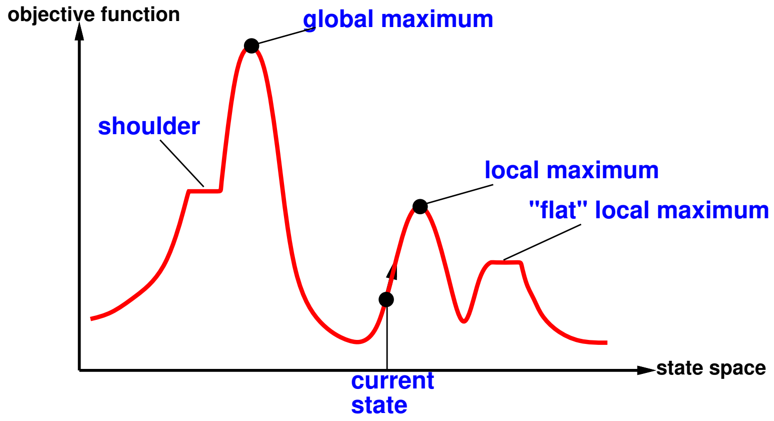


Hill-climbing

```
function HILL-CLIMBING( problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
    end
```

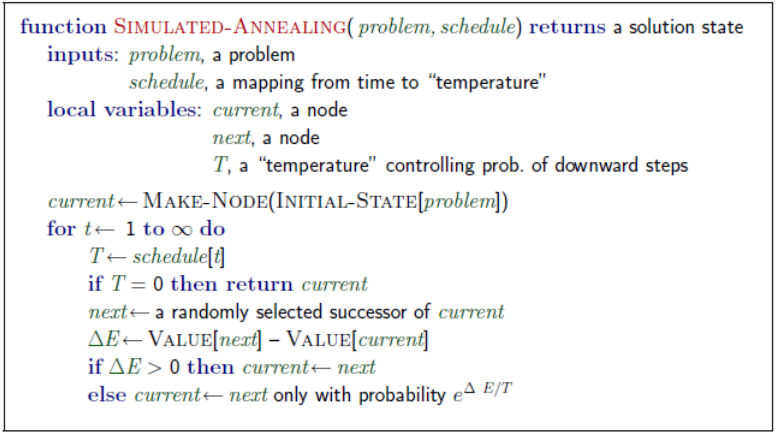
Hill climbing and the state space / fitness landscape



Random-restart hill climbing: overcomes local maxima — trivially complete (but depends on number of local optima)
Random sideways moves: escape from shoulders (but only with the speed of random search)

Simulated annealing

Idea: escape local maxima by allowing some bad moves but gradually decrease their size and frequency



```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
    inputs: problem, a problem
            schedule, a mapping from time to "temperature"
    local variables: current, a node
                     next, a node
                     T, a "temperature" controlling prob. of downward steps

    current ← MAKE-NODE(INITIAL-STATE[problem])
    for t ← 1 to ∞ do
        T ← schedule[t]
        if T = 0 then return current
        next ← a randomly selected successor of current
        $\Delta E$ ← VALUE[next] – VALUE[current]
        if $\Delta E > 0$ then current ← next
        else current ← next only with probability $e^{\Delta E/T}$
```

Properties of simulated annealing

Theoretical results:

- At fixed "temperature" T, state occupation probability reaches Boltzman distribution $p(x) = \alpha e^{\frac{E(x)}{kT}}\}$
- If T decreases slowly enough, simulated annealing always reaches the best state

Devised by Metropolis et al., 1953, for physical process modelling
Widely used in VLSI layout, airline scheduling, etc.

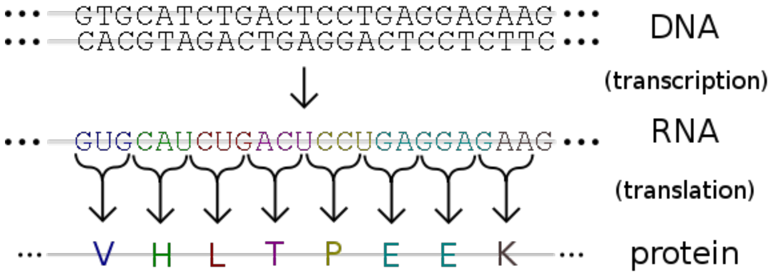
The theory of evolution

- In biology, evolution is a process that changes populations of animals / plants over many generations by mutations (random genetic changes) and selection ("survival of the fittest")
- Charles Darwin and Alfred Russel Wallace (1858, 1859) published a theory stating that these mechanisms have brought about all forms of life starting from a simple common ancestor
- In computing, we are mostly interested in the process of evolution as an inspiration for search algorithms

Local search and evolution

- Neighbors of existing solution candidates are usually generated randomly — analogy with mutations
- The decision whether a solution candidate is kept is analogous to selection
- Differences between local search and evolution:
    - evolution works with populations of “solution candidates”
    - new solution candidates can be generated by combining properties of several previous solution candidates
    - however, strictly speaking, there is no fixed “problem” in an evolutionary process and therefore there cannot be a “solution”

How “search states” are stored and processed in nature



... GTGCATCTGACTCCTGAGGAGAAG ...
... CACGTAGACTGAGGACTCCTCTTC ...
DNA
(transcription)
... GUGCAUCUGACUCCUGAGGAGAAG ...
RNA
(translation)
... V H L T P E E K ...
protein

A protein is made of a string of amino acids, each of which is encoded by a codon (three base pairs of DNA)
There are 4 different bases and 20 different amino acids.

Basic evolutionary mechanisms

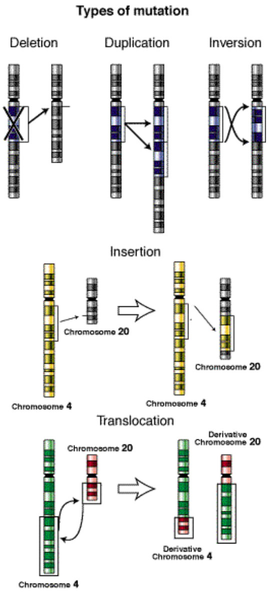
- Mutation
- Recombination
- Selection
- Gene drift
- Gene flow

Mechanisms: Point mutations

- Synonymous Mutations: Change towards another codon for the same amino acid (often third base)
- Non-synonymous Mutations
  - missense mutation: another amino acid
  - nonsense mutation: a stop codon (creates a truncated protein)

Other sequence mutations

- Deletion
- Insertion
- Duplication
- Translocation
- Inversion
- Chromosome fusion
- Chromosome division



Frame shift

CAC|GTA|GAC|TGA|GGA|CTC|CTC|TTC|C.....

V H L T P E E K ...

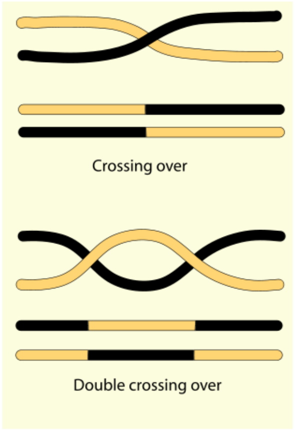
$$\downarrow$$

CAC|~~G~~TAG|ACT|GAG|GAC|TCC|TCT|TCC|.....

V I St L L R R R ...

Mechanisms: Recombination

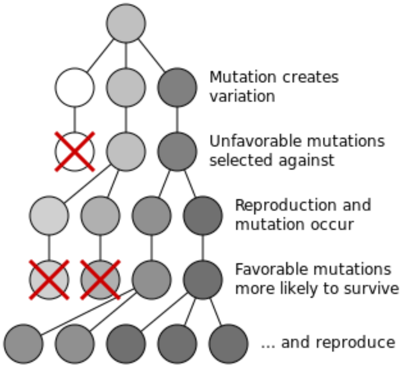
- The cells of many organisms are diploid, i.e. they possess two copies of each chromosome.
- During meiosis (the process of creating haploid daughter cells), homologous chromosomes become aligned. Their DNA strands break up at several positions and rejoin such that parts of the strands are exchanged.



Mechanisms: Isolation and gene flow

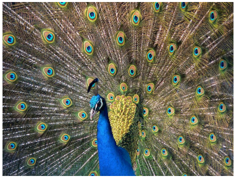
- Subpopulations can become isolated by natural obstacles (e.g., rivers or mountains).
- Individuals can be drifted to islands or other remote places, and become founders of a new population there.
- If individuals from two subpopulations have (almost) no common offspring, gene flow converges to zero, and the subpopulations can evolve into different directions.

Mechanisms: Selection


Mutation creates variation
Unfavorable mutations selected against
Reproduction and mutation occur
Favorable mutations more likely to survive
... and reproduce

Sexual selection

- In most species, one sex invests more into offspring than the other.
- Therefore, individuals of this sex become a rare resource. Their partner choices have a strong influence on the fitness of individuals from the other sex .
- To increase the fitness of their offspring, they should choose partners with high fitness (which is heritable).
- They cannot measure fitness directly, but estimate it based on perceptual features (e.g., the peacock's tail).

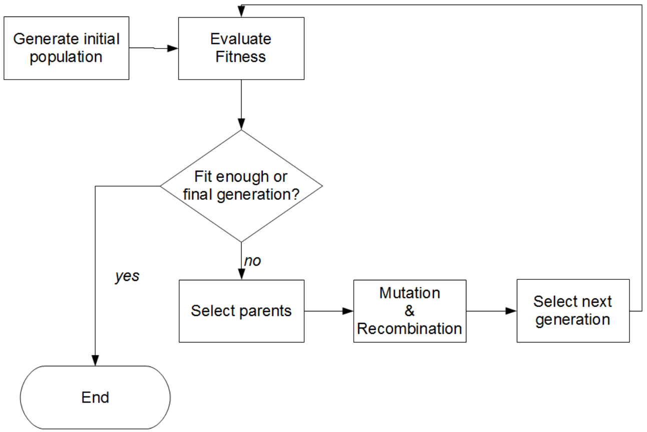


Mechanisms: Drift

- Some individuals do not leave offspring by accident (independently of their fitness); their genes disappear from the population.
- This makes the frequency of other genes increase; eventually only one allele of the gene may be left.
- The probability that a gene undergoes fixation in this way is always proportional to its frequency.
- Drift reduces diversity.

Evolution in the Computer

Evolutionary Algorithms



The three main ingredients (operators)

- Selection
- Mutation
- Recombination

Tournament selection

- n individuals (typically $n = 2$) are taken from the population; the one with the highest fitness is chosen with probability p (typically $p = 1$), otherwise the other(s)
- Increasing n increases the selection pressure, decreasing p decreases the selection pressure

Truncation Selection

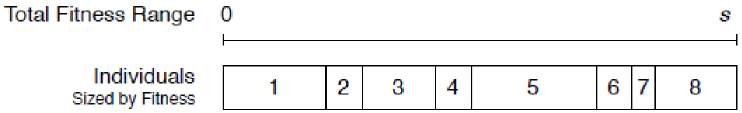
- All Individuals are sorted according to fitness, the best n are selected.
- Selection pressure depending on n as a fraction of the population size
- No Drift.

Uniform Selection

- The probability of being selected is the same for all individuals
- Not useful on its own, but often used as one of two selection steps in an evolutionary loop (selecting parents / selecting children)

Fitness proportionate selection

- The probability of being selected is proportionate to one's fitness.
  - fast convergence with large fitness differences
  - very slow convergence with small fitness differences
- If a random number is drawn for every member of the next generation, there is a high drift



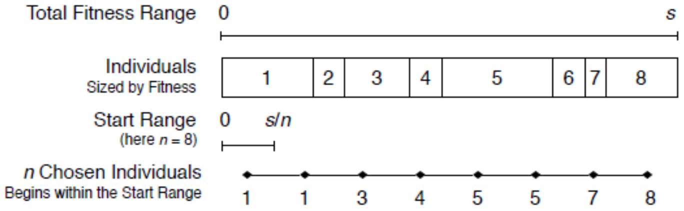
Implementation of fitness proportionate selection

Algorithm 30 *Fitness-Proportionate Selection*

```
1: perform once per generation
2:     global $\vec{p} \leftarrow$ population copied into a vector of individuals $\langle p_1, p_2, ..., p_l \rangle$
3:     global $\vec{f} \leftarrow \langle f_1, f_2, ..., f_l \rangle$ fitnesses of individuals in $\vec{p}$ in the same order as $\vec{p}$ ▷ Must all be $\geq 0$
4:     if $\vec{f}$ is all 0.0s then                                   ▷ Deal with all 0 fitnesses gracefully
5:         Convert $\vec{f}$ to all 1.0s
6:     for $i$ from 2 to $l$ do     ▷ Convert $\vec{f}$ to a CDF. This will also cause $f_l = s$, the sum of fitnesses
7:         $f_i \leftarrow f_i + f_{i-1}$
8: perform each time
9:     $n \leftarrow$ random number from 0 to $f_l$ inclusive
10:    for $i$ from 2 to $l$ do                    ▷ This could be done more efficiently with binary search
11:        if $f_{i-1} < n \leq f_i$ then
12:            return $p_i$
13:    return $p_1$
```

Stochastic universal sampling

Similar to the basic fitness proportinate selection, but less drift



SUS-Implementation

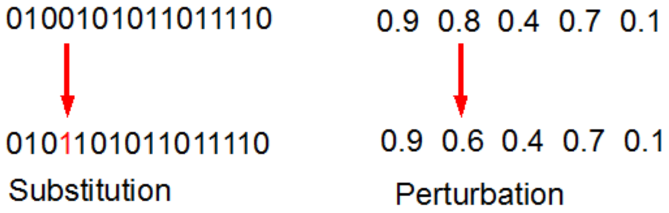
Algorithm 31 *Stochastic Universal Sampling*

```
1:  perform once per $n$ individuals produced    ▷ Usually $n = l$, that is, once per generation
2:      global $\vec{p} \leftarrow$ copy of vector of individuals (our population) $\langle p_1, p_2, ..., p_l \rangle$, shuffled randomly
                                                 ▷ To shuffle a vector randomly, see Algorithm 26
3:      global $\vec{f} \leftarrow \langle f_1, f_2, ..., f_l \rangle$ fitnesses of individuals in $\vec{p}$ in the same order as $\vec{p}$ ▷ Must all be $\geq 0$
4:      global $index \leftarrow 0$
5:      if $\vec{f}$ is all 0.0s then
6:          Convert $\vec{f}$ to all 1.0s
7:      for $i$ from 2 to $l$ do    ▷ Convert $\vec{f}$ to a CDF. This will also cause $f_l = s$, the sum of fitnesses.
8:          $f_i \leftarrow f_i + f_{i-1}$
9:      global $value \leftarrow$ random number from 0 to $f_l / n$ inclusive
10: perform each time
11:     while $f_{index} < value$ do
12:         $index \leftarrow index + 1$
13:     $value \leftarrow value + f_l / n$
14:     return $p_{index}$
```

Elitism

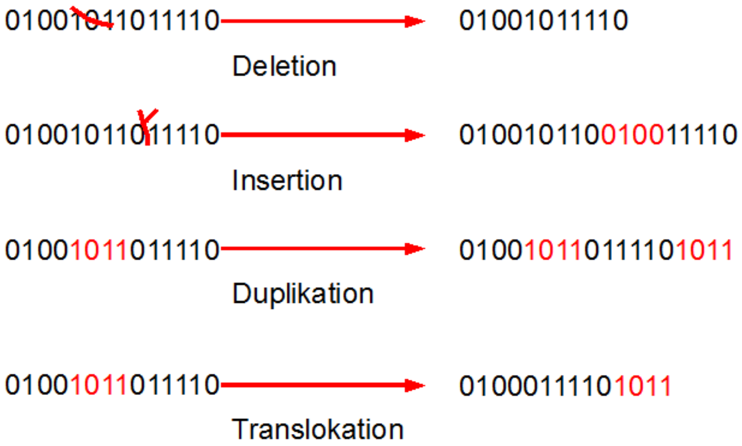
- Means that “the elite” (the best performing individuals) is guaranteed to be preserved
- One implementation: The best n individuals are taken into the next generation unchanged
- Theoretical guarantee: the global optimum will be found
- Makes it difficult to leave a local optimum
- Can be combined with any of the previously discussed selection techniques

Mutations: Substitutions / Perturbations

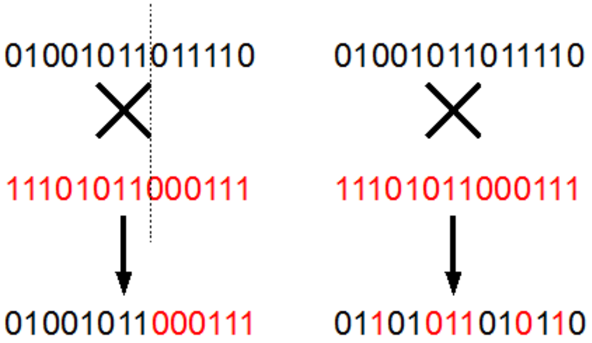


- Perturbation: add a normally distributed random number (mean 0) — often called Gaussian mutation
- If using Gaussian mutation, additional mechanisms may be needed to ensure that values remain in a valid range
- Additionaly, an operator that sets numbers to uniformly distributed random numbers is often used

Structural Mutations



Recombination / crossover



1 point crossover
(extension:
n point crossover)

uniform crossover

Standard families of evolutionary algorithms

- Genetic Algorithms (traditionally working on binary genomes (representations))
- Genetic Programming (working on function trees)
- Evolution Strategies (working on vectors real numbers)
- Evolutionary Programming (working on finite automata)

A typical genetic algorithm

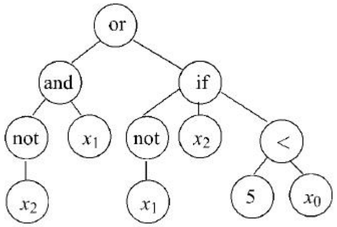
Algorithm 20 *The Genetic Algorithm (GA)*

```
1: $popsize \leftarrow$ desired population size          ▷ This is basically $\lambda$. Make it even.

2: $P \leftarrow \{\}$
3: for $popsize$ times do
4:     $P \leftarrow P \cup \{$new random individual$\}$
5: $Best \leftarrow \square$
6: repeat
7:     for each individual $P_i \in P$ do
8:         AssessFitness($P_i$)
9:         if $Best = \square$ or Fitness($P_i$) $>$ Fitness($Best$) then
10:            $Best \leftarrow P_i$
11:    $Q \leftarrow \{\}$                   ▷ Here's where we begin to deviate from $(\mu, \lambda)$
12:    for $popsize/2$ times do
13:        Parent $P_a \leftarrow$ SelectWithReplacement($P$)
14:        Parent $P_b \leftarrow$ SelectWithReplacement($P$)
15:        Children $C_a, C_b \leftarrow$ Crossover(Copy($P_a$), Copy($P_b$))
16:        $Q \leftarrow Q \cup \{$Mutate($C_a$), Mutate($C_b$)$\}$
17:    $P \leftarrow Q$                                          ▷ End of deviation
18: until $Best$ is the ideal solution or we have run out of time
19: return $Best$
```

Genetic programming

or and not x_2 x_1 if not x_1 x_2 $<$ 5 x_0.

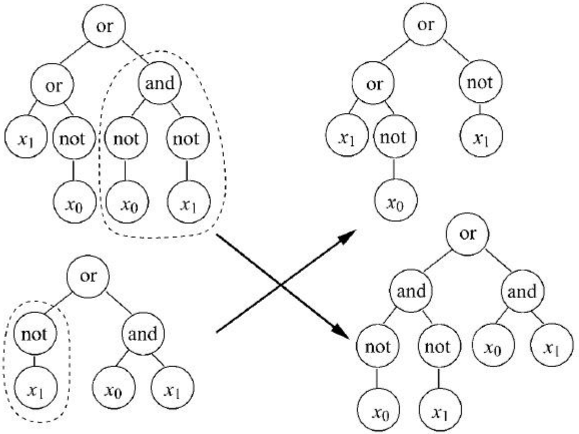


- Function symbols
- Terminal symbols (variables, constants)

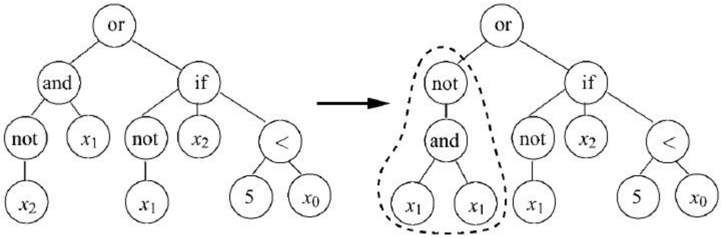
Typical GP settings

- Initialize the population with high diversity (typically with trees of different sizes)
- Selection: often variants of tournament selection
- typically, 80% of the new individuals are created by recombination, 10% by mutation, und 10% by cloning

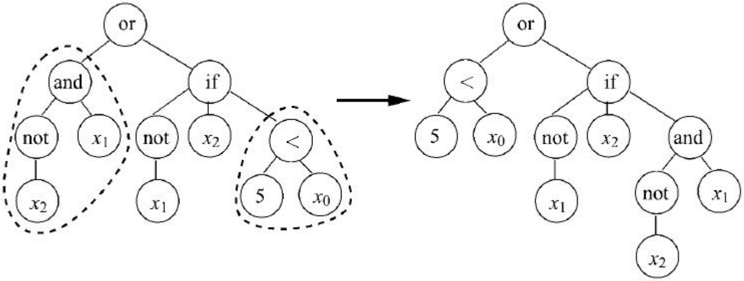
Tree exchange recombination



Random tree mutation



Tree exchange mutation



Typical problems with genetic programming

- Type consistency: Some functions cannot be combined with all sorts of parameters
- Bloat: Trees tend to grow until they need too many system resources, parts of the trees might have no function at all (e.g. “if(1>2) ...” oder “a+(1-1)”)

An application of genetic programming: Symbolic regression

- Task: Approximate an unknown function
- Fitness: Sum of squared errors at sample points
- Typical function symbols used in the tree: $\{+, -, \cdot, /, \exp, \ln, \sin, \cos\}$
- Terminal symbols used in the tree: variables, ephemeral random constants (constants that can be changed by mutations)

Summary

- Hill climbing
- Simulated Annealing
- Genetic Algorithms
- Selection: Tournamen / Truncation / Fitness proportional
- Mutation: Substitution, Perturbation, Structural Mutation
- Recombination: *n*-point, uniform
- Genetic Programming