Metaheuristic algorithms.

Lab 4: Genetic programming.
Bio-inspired metaheuristics: ant colony optimization (ACO), particle swarm optimization (PSO)

---

1. **Genetic programming.**

The aim of genetic programming is to design in an evolutionary manner computational structures (arithmetical/logical expressions, classification/decision rules or even programs). In traditional Genetic Programming applications (as symbolic regression) the elements of the population are hierarchical structures (e.g. syntactic trees). The genetic operators are adjusted to work with such structures. One of the main difficulties in GP is to avoid the proliferation of large structures (the so called bloat problem). A possible solution to this problem is to limit the depth of the trees generated during the evolutionary process.

The most popular application of GP is symbolic regression aiming to evolve an expression which fits well to some data (unlike the numerical regression which aims to estimate the coefficients of a given model, symbolic regression estimates the model itself).

Application 1. Use the "rgp" R package to find an expression which fits a dataset.

Main steps:

- Launch R
- Load package "rgp": Packages ->Load package … or library("rgp") (if the package is not installed then it should be installed by Packages-> Install package(s)…
- Define the set of nonterminals (operators and functions) using functionSet. Example: setNonterminals <- functionSet("+", "*", "-","/")
- Define the set of variables using inputVariableSet. Example: setVariables <- inputVariableSet("x")
- Define the set of constants using constantFactorySet. Example: setConstants <- constantFactorySet(function() rnorm(1)) (random values generated using the standard normal distribution)
- Define the test data: values which will be used to evaluate the approximation accuracy. Example: dateX <- seq(from = -pi, to = pi, by = 0.1)
- Define the fitness function: mean square error (measure of the difference between the values of the test function and the values corresponding to the evolved expressions). Example: fitness <- function(f) rmse(f(dateX), sin(dateX)) (if the reference function is sinus)
- Call the function corresponding to the evolutionary process (geneticProgramming). Example:

```
geneticProgramming(functionSet = setNonterminals,
                   inputVariables = setVariables,
                   constantSet = setConstants,
                   fitnessFunction = fitness,
                   stopCondition = makeStepsStopCondition(10000))
```

Particularities of the genetic programming implemented in "rgp":

- The population elements are R expressions (implemented as tree-like structures)
- The population initialization is based on several construction strategies:
  - "grow" (each branch in the tree will be extended until it reaches the maximal length or until a random event occurs)
  - „full" (all branches in the tree have the maximal length)
  - Combined variant (some elements are generated using the "grow" strategy, others are constructed using the „full" strategy)
- The package implements the traditional crossover and mutation strategies adapted for trees (see slides of lecture 9)
- There are implemented various selection variants using one or several criteria (as in multiobjective optimization). In the multicriterial variant the aim is to optimize the quality of the result, the simplicity of the elements and the population diversity.

Exercise 1: Follow the above steps and test the influence of nonterminals on the quality of the results (by changing the elements of the nonterminals set). Hint: see for instance gp1.r

2. Ant Colony Optimization (ACO)

ACO is a metaheuristic inspired by the behavior of the ant colonies. It is especially used in solving combinatorial optimization problems (e.g. routing, scheduling, assignment) It uses a population of artificial ants (agents) which is changed during an iterative process. At each iteration each ant constructs, component by component, a potential solution. The values for the solution components are chosen randomly based on a probability distribution. The probability distribution is computed by using both local information (what the ant can collect from its neighbourhood) and global information (obtained by using the indirect communication process between ants based on pheromone trails).

Solving TSP using ACO. The input data consists of the graph describing the direct connections between towns and their costs. A population of ants is initially placed on random nodes (or all of them in the first node). At each iteration, each ant visits *n* distinct nodes, constructing a tour. The ants have a local memory where the list of visited nodes is stored in order to avoid visiting twice the same node. The transition of an ant *k* from the node *i* to the node *j* at step *t* is based on the following probability:

$$P_t^k(i,j) = \begin{cases} \dfrac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{l\in N(k)}\tau_{il}^{\alpha}\eta_{il}^{\beta}} & j \text{ was not visited} \\ 0 & j \text{ was visited} \end{cases}$$

The factors appearing in the computation of the probability are:

- τ_{ij} models the pheromone concentration released by the ants on edge *(i,j);* the pheromone concentration is randomly initialized with small positive values. Each ant which visits an edge *(i,j)* can release some pheromone on it contributing to the update of the pheromone concentration:

$$\tau_{ij} = (1-\rho)\tau_{ij} + \rho\sum_k Q_{ij}(k)/\text{cost}(T_k)$$

  ρ is a constant less than *1* which controls the evaporation process, $Q_{ij}(k)$ is *0* if *(i,j)* does

not belong to the tour constructed by ant k. $Cost(T_k)$ denotes the cost of the tour constructed by the ant k.

- η_{ij} models the local information concerning the quality of the edge; the simplest variant is when it is *1/cost(i,j)*.
- α and β are parameters which control the relative importance of those two types of information: the global information provided by the pheromone concentration and the local one provided by the cost of the edge.
- *N(k)* denotes the neighborhood of node *i* and contains the nodes which can be reached from node i and have not been visited yet.

Application 2. Implement an ACO algorithm for TSP.
Hint. See function ACO_TSP.sci

Exercise 2. Change the previous implementation such that when the pheromone matrix elements are updated, the tours visited by all ants are taken into account.
Hint. The updating terms are cumulated after each tour construction.

3. Particle Swarm Optimization (PSO)

PSO is a metaheuristic used for continuous function optimization inspired by the behavior of bird swarms. It uses a population of *m* "particles", each particle *i* being characterized by its position (x_i) and its velocity (v_i). Moreover, each particle memorizes the best position it visited up to the current moment ($xbest_i$). There is also another variable which contains the best position found up to the current iteration by the entire swarm (*xbest*). The evolutionary process consists in the change, at each generation *t*, of the position of all particles in the population according to the following rules:

$$v_i(t+1) = \gamma(v_i(t) + r_1 rand(0,1)(xbest_i - x_i(t)) + r_2 rand(0,1)(xbest - x_i(t)))$$

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

where:

- gamma is a constriction factor (a typical value for gamma is 0.7)
- r_1 and r_2 are two constant values (e.g. $r_1=r_2=2.05$)

Besides this variant, where *xbest* is the global best element from the swarm, there is also another variant where for each particle *i*, *xbest(i)* is selected as the best element from the neighborhood of the particle *i*. The neighborhood of a particle can be defined by various topologies, one of the most used is the ring topology (in this case the neighborhood of size K of particle i is represented by the particles having the indices *{i-K,i-K+1,...,i-1,i,i+1,...,i+K-1,i+K}*).

Application 3. Implement a PSO algorithm (using the above eqs.) and test its behavior for a unimodal function (e.g. sphere) and for a multimodal function (e.g. Griewank).

Hint. See function PSO.m

Exercise 3. Change PSO.m such that it implements the "local best" variant using a ring topology to define the neighbourhood.