Genetic Programming

Instructor: Dr. Michal Bidlo
https://www.fit.vut.cz/person/bidlom/

Brno International Summer School in Information Technology 2021, BUT FIT Course in genetic programming

During this individual practice, we will examine one of the basic applications of genetic programming (GP) -- solving the symbolic regression task. We will concentrate on performing real experiments with various settings, hence we use an implementation of the GP algorithm that already provides all necessary tools which simplify our work substantially. In particular, a custom implementation of the genetic programming algorithm with elitist support is provided called the Elitist Genetic Programming.

Tasks for individual work:

1. Familiarize yourself with the following python cells and run them one after another in order to verify that all worls well. This system implements the genetic programming algorithm with elitism to solve the symbolic regresion task. If you prefer working with a native python script executed locally on your computer, you can download it from https://github.com/bidlom/ElitistGP.
2. When running the main program in the bottom cell, observe the progress of evolution. Notice that after 100 generations the evolved function copies quite closely the reference red points. Imagine that the red points represent some measured data for which we want to find a suitable mathematical prescription. This is the task for genetic programming called the symbolic regression.
3. But maybe you can find a better, more precise solution. In the section User modifiable control parameters below change the value of SEED to 0. This allows initializing the pseudorandom number generator for each independent run of the main program to a random value according to some data from the operating system. We want every run to be unique because the GP is a stochastic process.
4. Run the cell with the user modifiable parameters so that the values take effect.

5. Run the main program repeatedly several times and observe the results. Will you be able to find a better solution than the previous one, such that more precisely approximates the red points?
6. Now, perform your own experiments. Draw on a paper a graph of a function of single variable x and write the coordinates of some points [xi, yi] on the graph according to the instructions specified in the section User modifiable control parameters for the list DATASET below. Replace the DATASET list with the points from your graph. We will try to find a reasonably precise mathematical description for this data by means of genetic programming.
7. Note that after each modification of anything in User modifiable control parameters it is needed to run this cell so that the new values take effect.
8. Again, run the main program and wait for the result. Run it repeatedly, it is natural that some runs provide poor results and some runs provide nice results. Try to modify other user control parameters and run the system again and again. Probably you will notice that for some settings the results are promising. This is the right point when you should run the main program more times and identify the run in which a reasonably precise solution has been obtained. Play with it!
9. If you noticed a good solution, record the value of SEED which is shown below each visualization of the resulting tree. By assigning this value to SEED instead of 0 you will be able to exactly repeat the successful or interesting experiment.
10. As the solution of this homework, download the .py file (not the .ipynb), make a screenshot of the window with your most interesting result and email them both to bidlom@fit.vut.cz. **no later than by the end of this week**.

Please note: **there will be no further demonstration of the solution the next morning**, the code with the initial settings represents a possible one. The main part of your work is to **replace the reference dataset by your own values** for which a reasonable approximation can be found by genetic programming. **Don't hesitate to write me if you had any questions**.

```
!pip install graphviz
!pip install matplotlib
```

```
from statistics import mean
from copy import deepcopy
import matplotlib.pyplot as plt
from IPython.display import Image, display
from graphviz import Digraph, Source
from numpy import arange
from datetime import datetime
import random
import os
import sys
import math
import csv
```

```
print("Import successful.")
```

```
Import successful.
```

```
# Functions for genetic programming. You can specify which of them may really be used
# by choosing a subset in user modifiable control parameters below.
def idx(x): return x
def neg(x): return -x
def inv(x): return 1.0/x if x != 0 else 0.0
def add(x, y): return x + y
def sub(x, y): return x - y
def mul(x, y): return x * y
def pdiv(x, y): return x / y if y != 0 else 0.0
def sin(x): return math.sin(x)
def cos(x): return math.cos(x)
def x2(x): return x ** 2

f_binary = [add, sub, mul, pdiv]    # functions with two operands (binary)
f_unary  = [idx, neg, inv, x2, sin, cos] # functions with one operand  (unary)

# ############### Functions of the graphical interface ###############
def prepare_plots(seed):
    fig, axarr = plt.subplots(2, 2)
    fig.canvas.set_window_title('EVOLUTION FROM SEED = %s' % str(seed))
    fig.subplots_adjust(hspace = 0.6)
    plt.ion() # interactive mode for plot

    axarr[0][0].set_xlabel('generation')
    axarr[0][0].set_ylabel('avg.fit')
    axarr[0][0].set_title('mean fitness evolutionWn+trend from 10 samples')
    axarr[0][0].set_xlim(0, GENERATIONS)

    axarr[1][0].set_xlabel('generation')
    axarr[1][0].set_ylabel('min.fit')
    axarr[1][0].set_title('best fitness evolution')
    axarr[1][0].set_xlim(0, GENERATIONS)

    axarr[0][1].set_xlabel('generation')
    axarr[0][1].set_ylabel('avg.size')
    axarr[0][1].set_title('average tree size evolution')
    axarr[0][1].set_xlim(0, GENERATIONS)

    return axarr

def plot(axarr, line, gen_list, avg_fit_list, avg_fit_trend, min_fit_list, size_list, best_gen, bes
    # plot the average fitness evolution and its trend
    axarr[0][0].set_ylim(0, max(avg_fit_list))
    line[0].set_xdata(gen_list)
    line[0].set_ydata(avg_fit_list)
    line[1].set_xdata(gen_list)
    line[1].set_ydata(avg_fit_trend)
    # plot the best fitness evolution
    axarr[1][0].set_ylim(0, math.ceil(max(min_fit_list)))
    line[2].set_xdata(gen_list)
```

```
        line[2].set_xdata(gen_list)
        line[2].set_ydata(min_fit_list)
        # plot the average tree size evolution
        axarr[0][1].set_ylim(0, max(size_list))
        line[3].set_xdata(gen_list)
        line[3].set_ydata(size_list)
        # the title of the bottom-right plot must be (re)set here...
        axarr[1][1].set_xlabel('x')
        axarr[1][1].set_ylabel('f(x)')
        axarr[1][1].set_title('the best so far solution₩ngen = %s, size = %s' % (best_gen, best_size))
        plt.draw()
        plt.pause(0.01)

    def plot_functions(ax, dataset, individual):
        x = [ds[0] for ds in dataset]
        y = [ds[1] for ds in dataset]
        xds = [ds[0] for ds in dataset]
        ax.cla()
        ax.plot(xds, y, "ro")
        evol_y = [individual.compute_tree(var) for var in x]
        ax.plot(x, evol_y, "b")

    # This function plots the graph of the function of the solution found by GP.
    # Red dots represent the reference dataset, the blue curve is the GP solution.
    # In Google Colaboratory this function is used insted of drawing the statistics
    # graphically during the evolution because the colab does not seem to support
    # matplotlib interactive mode in an easy way. This function is meant to replace
    # the interactive drawing functions defined above (which are normally utilized
    # in the standalone version of the application downloadable from Github).
    def plot_solution(dataset, individual, seed):
        fig, ax = plt.subplots()
        fig.canvas.set_window_title('SOLUTION EVOLVED FROM SEED %s' % str(seed))
        ax.set_xlabel('x')
        ax.set_ylabel('f(x)')
        ax.set_title('the best solution evolved from seed %s' % (str(seed)))
        ax.grid()

        print('the best solution evolved from seed %s' % (str(seed)))
        x = [ds[0] for ds in dataset]
        y = [ds[1] for ds in dataset]
        xds = [ds[0] for ds in dataset]
        ax.cla()
        ax.plot(xds, y, "ro")
        evol_y = [individual.compute_tree(var) for var in x]
        ax.plot(x, evol_y, "b")
        plt.show()

    def stable_mean(fits):
        sm = 0.0
        cnt = 0
        for f in fits:
            if f < 1000.0:
                sm += f
                cnt += 1
        return sm / cnt
```

```
def trend10(lst):
    sm = 0
    cnt = 1
    while len(lst)-cnt >= 0 and cnt <= 10:
        sm += lst[len(lst)-cnt]
        cnt += 1
    return sm/(cnt-1)
# end of the graphical functions

# The genetic programming tree class with all needed operations
class GPTree:
    def __init__(self, data = None, left = None, right = None):
        self.data  = data
        self.left  = left
        self.right = right

    def node_label(self): # return string label
        if (self.data in FUNCTIONS):
            return self.data.__name__
        else:
            return str(self.data)

    def draw(self, dot, count): # dot & count are lists in order to pass "by reference"
        node_name = str(count[0])
        dot[0].node(node_name, self.node_label())
        if self.left != None:
            count[0] += 1
            dot[0].edge(node_name, str(count[0]))
            self.left.draw(dot, count)
        if self.right != None:
            count[0] += 1
            dot[0].edge(node_name, str(count[0]))
            self.right.draw(dot, count)

    def compute_tree(self, x):
        if (self.data in f_binary):
            return self.data(self.left.compute_tree(x), self.right.compute_tree(x))
        elif (self.data in f_unary):
            return self.data(self.left.compute_tree(x))
        elif self.data == 'x': return x
        else: return self.data

    def random_tree_growth(self, depth):
        if depth < MAX_DEPTH:   # to grow the tree we prefer generating functions
            if random.random() < 0.9:
                self.data = FUNCTIONS[random.randint(0, len(FUNCTIONS)-1)]
            else:
                if random.random() < 0.5: # generate terminal node with the variable x
                    self.data = 'x'
                else:   # generate terminal node with a random constant
                    self.data = const()
                self.left = None
                self.right = None
        else:   # to limit the maximal depth we generate a terminal symbol only
            if random.random() < 0.5: # generate terminal node with the variable x
```

```
            if random.random() < 0.5: # generate terminal node with the variable 'x'
                self.data = 'x'
            else:   # generate terminal node with a random constant
                self.data = const()
            self.left = None
            self.right = None

        if self.data in f_binary:
            self.left = GPTree()
            self.left.random_tree_growth(depth = depth + 1)
            self.right = GPTree()
            self.right.random_tree_growth(depth = depth + 1)
        elif self.data in f_unary:
            self.left = GPTree()
            self.left.random_tree_growth(depth = depth + 1)
            self.right = None

    def mutation(self):
        if random.random() < PROB_MUTATION: # mutate at this node
            self.random_tree_growth(depth = random.randint(1, MAX_DEPTH))
        elif self.left  != None: self.left.mutation()
        elif self.right != None: self.right.mutation()

    def size(self): # the number of all nodes of the tree
        if self.data == 'x': return 1
        l = self.left.size()  if self.left  != None else 0
        r = self.right.size() if self.right != None else 0
        return 1 + l + r

    def build_subtree(self): # count is list in order to pass "by reference"
        t = GPTree()
        t.data = self.data
        if self.left  != None: t.left  = self.left.build_subtree()
        if self.right != None: t.right = self.right.build_subtree()
        return t

    def scan_tree(self, count, second): # note: count is list, so it's passed "by reference"
        count[0] -= 1
        if count[0] <= 1:
            if not second: # return subtree rooted here
                return self.build_subtree()
            else: # glue subtree here
                self.data  = second.data
                self.left  = second.left
                self.right = second.right
        else:
            ret = None
            if self.left  != None  and count[0] > 1: ret = self.left.scan_tree(count, second)
            if self.right != None and count[0] > 1: ret = self.right.scan_tree(count, second)
            return ret

    def crossover(self, other): # crossover of 2 trees at random nodes
        if random.random() < XO_RATE:
            second = other.scan_tree([random.randint(1, other.size())], None) # 2nd random subtree
            self.scan_tree([random.randint(1, self.size())], second) # 2nd subtree "glued" inside 1
```

```
    def draw_tree(self, fname, footer = ''): # needs the graphviz library to work
        dot = [Digraph()]
        dot[0].attr(kw='graph', label = footer)
        count = [0]
        self.draw(dot, count)
        Source(dot[0], filename = fname + ".gv", format="png").render()
        display(Image(filename = fname + ".gv.png"))
# end of the GPTree class

# ############### Functions of the genetic programming algorithm ###############
def init_population(): # Random initialization of the GP population
    pop = []
    for i in range(POP_SIZE):
        t = GPTree()
        t.random_tree_growth(depth = random.randint(1, MAX_DEPTH+1))
        pop.append(t)
    return pop

# Evaluation of candidate solution (the goal is to minimize the sum of absolute differences
# of the evolved function from the target function for given values of the input variable x.
def fitness(individual, dataset):
    fit = sum([abs(individual.compute_tree(ds[0]) - ds[1]) for ds in dataset]) + 0.01*individual.si
    return fit

# Select one individual using tournament selection
def selection(population, fitnesses):
    tournament = [random.randint(0, len(population)-1) for i in range(TOURNAMENT_SIZE)] # select to
    tournament_fitnesses = [fitnesses[tournament[i]] for i in range(TOURNAMENT_SIZE)]
    return deepcopy(population[tournament[tournament_fitnesses.index(min(tournament_fitnesses))]])

# A random constan generator for terminal nodes
def const(): return random.random()*random.randint(-CONST_RANGE, CONST_RANGE)

# The elitist genetic programming algorithm
def GP_run(dataset, seed, plot = False):
    index_of_the_best = 0
    best_fitness = 1e60 # to be minimized
    best_generation = 0

    # initialize drawing interface and structures
    if plot:
        axarr = prepare_plots(seed)
        gen_list = []
        avg_fit_list = []
        avg_fit_trend = []
        min_fit_list = []
        size_list = []
        line = [None, None, None, None]
        # the line list will represent the following curves:
        # line[0] - average fitness of current population
        # line[1] - average of the last 10 samples (the trend) from line[0]
        # line[2] - fitness of the best individual
        # line[3] - average tree size of current population
        line[0], = axarr[0][0].plot(gen_list, avg_fit_list, 'b-') # 'b-' = blue line
        line[1], = axarr[0][0].plot(gen_list, avg_fit_trend, 'k-') # 'k-' = black line
```

```
            line[2], = axarr[1][0].plot(gen_list, min_fit_list, 'g-') # 'g-' = green line
            line[3], = axarr[0][1].plot(gen_list, size_list,    'r-') # 'r-' = red line

        # the GP algorithm
        population = init_population() # generate and evaluate random initial population
        fitnesses = [fitness(ind, dataset) for ind in population]
        index_of_the_best = fitnesses.index(min(fitnesses)) # identify the best individual
        for gen in range(GENERATIONS):
            nextgen_population=[]
            # elitism: add the best individual and his mutant to the new population
            nextgen_population.append(population[index_of_the_best])
            for i in range(1, POP_SIZE): # create new population by selection, crossover and mutation
                parent1 = selection(population, fitnesses)
                parent2 = selection(population, fitnesses)
                parent1.crossover(parent2)
                parent1.mutation()
                nextgen_population.append(parent1)
            population = nextgen_population

            # evaluate the new population, identify the best individual
            fitnesses  = [fitness(ind, dataset) for ind in population]
            index_of_the_best = fitnesses.index(min(fitnesses))
            sizes      = [ind.size() for ind in population] # calculate the size of each individual

            # generate statistical data for real-time visualization of the evolution
            if plot:
                gen_list.append(gen)
                avg_fit_list.append(stable_mean(fitnesses))
                avg_fit_trend.append(trend10(avg_fit_list))
                min_fit_list.append(min(fitnesses))
                size_list.append(mean(sizes))

            # found a better solution than the best-so-far? remember it and plot its function
            if  fitnesses[index_of_the_best] < best_fitness:
                best_fitness = fitnesses[index_of_the_best]
                best_generation = gen
                if plot:
                    plot_functions(axarr[1][1], dataset, population[index_of_the_best])
                else:
                    print('In generation %s improvement occured!' % str(gen))

            # plot the statistical data after each generation
            if plot:
                plot(axarr, line, gen_list, avg_fit_list, avg_fit_trend, min_fit_list, size_list,W
                                           best_generation, population[index_of_the_best].size())
            else:
                print('Generation %s;WtWtbest fitness %s;WtWtmean size %s' % (str(gen),
                                                                              str(fitnesses[index_of_the_be
                                                                              str(mean(sizes))))

        # return the best individual after performing the maximal number of generations
        return population[index_of_the_best]
    # end of the GP functions

    print('Everything looks well.')
```

```
Everything looks well.


################################################################################################
# User modifiable control parameters of Genetic Programming (GP) with recommended values:
# For a given settings, it is needed to run more independent GP experiments with SEED=0 !!
# If the settings does not provide acceptable solutions for your data, it is needed to
# perform more experimentations. Play with the values and the function set, observe the
# behavior of the GP algorithm, improvements of the best solution during evolution,
# modify the parameters and/or your data and repeat the experiments with new values.
POP_SIZE        = 100   # population size (20...100)
MAX_DEPTH       = 8     # maximal initial random tree depth (2...8)
GENERATIONS     = 300   # maximal number of generations (100...300)
TOURNAMENT_SIZE = 5     # tournament selection base (2...5)
XO_RATE         = 0.4   # crossover rate
PROB_MUTATION   = 0.1   # per-node mutation probability
CONST_RANGE     = 5     # will be -CONST_RANGE....0....CONST_RANGE
# Initialization value of random number generator; for SEED = 0,
# a random initialization value will be generated (0 IS NEEDED FOR
# PERFORMING VARIOUS INDEPENDENT EXPERIMENTS). The actual seed value
# is printed after finishing each evolutionary run. You may specify
# non-zero value for repeating successful experiments.
SEED = 1626412026

# Basic GP functions
# idx - identity function (returns the input value x)
# neg - negation of x (returns -x)
# inv - inversion of x (returns 1/x or 0 for x=0)
# add - addition (returns x + y)
# sub - subtraction (returns x - y)
# mul - multiplication (returns x * y)
# pdiv - division (returns x / y or 0 for y=0)
# sin - sin(x) with x in radians
# cos - cos(x) with x in radians
# x2 - the second power of x
# You may choose a subset of the functions here for tuning the GP system
# for your input data:
FUNCTIONS = [idx, add, sub, inv, pdiv]

# The dataset for the symbolic regression may be specified here (if you do not utilize the
# read_csv function described above. If you are going to use the read_csv function, set the
# DATASET list below to []. Otherwise fill in your own values in the form of pairs [x, f(x)]
# as shown by the following example (the dataset in this example is primarily used for
# the first demonstration of symbolic regression by means of genetic programming).
#
# It is recommended to specify approximately 20 lines of data. In order to make the task,
# more feasible, it is HIGHLY RECOMMENDED to consider the interval of the x-values of size
# from 1.0 to 2.0 (i.e. the difference of the x-value from the last and from the first
# data line to be between 1.0 and 2.0) and the function values to be between -20.0 and +20.0.
# Note that the x-values are divided by a suitable constant (here 30) to fit the difference
# max_x - min_x in the recommended interval <1.0; 2.0>.
DATASET = [₩
[1.0/30,15],₩
[2.0/30,16],₩
[3.0/30,17],₩
```

```
[3.0/30,17],₩
[4.0/30,18],₩
[5.0/30,19],₩
[6.0/30,20],₩
[7.0/30,19],₩
[8.0/30,18],₩
[9.0/30,17],₩
[10.0/30,16],₩
[11.0/30,15],₩
[12.0/30,14],₩
[13.0/30,13],₩
[14.0/30,12],₩
[15.0/30,11],₩
[16.0/30,10],₩
[17.0/30,11],₩
[18.0/30,12],₩
[19.0/30,13],₩
[20.0/30,14],₩
[21.0/30,15],₩
[22.0/30,16],₩
[23.0/30,17],₩
[24.0/30,18],₩
[25.0/30,19],₩
[26.0/30,20],₩
[27.0/30,19],₩
[28.0/30,18],₩
[29.0/30,17],₩
]

print('Values updated.')
# end of user parameters
```

```
Values updated.
```

```
# Set use initialization value of the pseudorandom generator (default is 0)
def init_random_seed(s = 0):
    if s != 0:  # if a specific non-zero value given, use it...
        rseed = s
    else:   # ...otherwise set a ``random'' initialization value (default)
        rseed = int(datetime.now().timestamp())
#        rseed = int(random.randrange(1<<32)+os.getpid())
    random.seed(rseed)
    # finally, return the initialization value (seed) which later allows us
    # to repeat promising runs
    return rseed

# The main program
def main():
    global SEED, DATASET

    if DATASET == []:
        DATASET = read_csv()
    rs = SEED # save the user defined seed value
    SEED = init_random_seed(SEED)
```

```
        solution = GP_run(DATASET, SEED)

        plot_solution(DATASET, solution, SEED)
        solution.draw_tree(str(SEED), 'SEED = %s' % str(SEED))

        # Restore the saved SEED value; if it was set to 0 in the user defined
        # parameters above, then this allows by just repeated running of the main
        # function to execute various independent evolutionary GP runs. If it was
        # non-zero, restarting main will allow reproducing a specific experiment
        # with this seed value.
        SEED = rs

    if __name__== "__main__":
        main()
```

```
In generation 0 improvement occured!
Generation 0;          best fitness 257.39743541522995;              mean size 10.33
In generation 1 improvement occured!
Generation 1;          best fitness 195.7903608615509;         mean size 14.95
In generation 2 improvement occured!
Generation 2;          best fitness 158.67102576046003;              mean size 23.95
In generation 3 improvement occured!
Generation 3;          best fitness 148.47616137358867;              mean size 30.58
In generation 4 improvement occured!
Generation 4;          best fitness 117.63455614371723;              mean size 34.11
In generation 5 improvement occured!
Generation 5;          best fitness 98.08143005651203;         mean size 41.45
In generation 6 improvement occured!
Generation 6;          best fitness 81.80128938981488;         mean size 41.82
Generation 7;          best fitness 81.80128938981488;         mean size 30.16
In generation 8 improvement occured!
Generation 8;          best fitness 76.68934151913592;         mean size 29.33
In generation 9 improvement occured!
Generation 9;          best fitness 74.89418702225686;         mean size 34.61
In generation 10 improvement occured!
Generation 10;         best fitness 73.95010143082472;         mean size 35.05
Generation 11;         best fitness 73.95010143082472;         mean size 29.57
In generation 12 improvement occured!
Generation 12;         best fitness 71.04902945091919;         mean size 36.36
Generation 13;         best fitness 71.04902945091919;         mean size 35.94
Generation 14;         best fitness 71.04902945091919;         mean size 38.38
In generation 15 improvement occured!
Generation 15;         best fitness 70.41509074196296;         mean size 36.28
Generation 16;         best fitness 70.41509074196296;         mean size 35.74
Generation 17;         best fitness 70.41509074196296;         mean size 38.09
In generation 18 improvement occured!
Generation 18;         best fitness 70.40509074196295;         mean size 34.55
Generation 19;         best fitness 70.40509074196295;         mean size 36.4
Generation 20;         best fitness 70.40509074196295;         mean size 41.62
Generation 21;         best fitness 70.40509074196295;         mean size 37.31
Generation 22;         best fitness 70.40509074196295;         mean size 41.58
In generation 23 improvement occured!
Generation 23;         best fitness 70.32091195547225;         mean size 38.18
Generation 24;         best fitness 70.32091195547225;         mean size 40.94
Generation 25;         best fitness 70.32091195547225;         mean size 39.12
In generation 26 improvement occured!
Generation 26;         best fitness 68.55401643573117;         mean size 38.72
Generation 27;         best fitness 68.55401643573117;         mean size 46.24
Generation 28;         best fitness 68.55401643573117;         mean size 45.02
Generation 29;         best fitness 68.55401643573117;         mean size 45.78
In generation 30 improvement occured!
Generation 30;         best fitness 67.04648892023827;         mean size 44.27
Generation 31;         best fitness 67.04648892023827;         mean size 46.64
Generation 32;         best fitness 67.04648892023827;         mean size 40.82
Generation 33;         best fitness 67.04648892023827;         mean size 50.41
Generation 34;         best fitness 67.04648892023827;         mean size 52.69
Generation 35;         best fitness 67.04648892023827;         mean size 50.68
In generation 36 improvement occured!
Generation 36;         best fitness 66.41978392410927;         mean size 50.02
Generation 37;         best fitness 66.41978392410927;         mean size 54.46
Generation 38;         best fitness 66.41978392410927;         mean size 57.25
Generation 39;         best fitness 66.41978392410927;         mean size 56.62
Generation 40;         best fitness 66.41978392410927;         mean size 58.38
```

```
Generation 41;          best fitness 66.41978392410927;         mean size 51.09
Generation 42;          best fitness 66.41978392410927;         mean size 52.01
Generation 43;          best fitness 66.41978392410927;         mean size 52.43
In generation 44 improvement occured!
Generation 44;          best fitness 66.0461507380949;          mean size 56.06
Generation 45;          best fitness 66.0461507380949;          mean size 52.16
Generation 46;          best fitness 66.0461507380949;          mean size 54.41
Generation 47;          best fitness 66.0461507380949;          mean size 52.98
Generation 48;          best fitness 66.0461507380949;          mean size 53.25
Generation 49;          best fitness 66.0461507380949;          mean size 49.67
Generation 50;          best fitness 66.0461507380949;          mean size 54.06
Generation 51;          best fitness 66.0461507380949;          mean size 48.86
Generation 52;          best fitness 66.0461507380949;          mean size 59.08
Generation 53;          best fitness 66.0461507380949;          mean size 58.54
Generation 54;          best fitness 66.0461507380949;          mean size 53.29
Generation 55;          best fitness 66.0461507380949;          mean size 52.2
In generation 56 improvement occured!
Generation 56;          best fitness 65.24751162670776;         mean size 47.82
Generation 57;          best fitness 65.24751162670776;         mean size 58.77
Generation 58;          best fitness 65.24751162670776;         mean size 57.9
In generation 59 improvement occured!
Generation 59;          best fitness 65.22751162670775;         mean size 60.15
Generation 60;          best fitness 65.22751162670775;         mean size 53.12
In generation 61 improvement occured!
Generation 61;          best fitness 63.46379221385115;         mean size 54.8
Generation 62;          best fitness 63.46379221385115;         mean size 51.06
Generation 63;          best fitness 63.46379221385115;         mean size 56.92
Generation 64;          best fitness 63.46379221385115;         mean size 65.33
Generation 65;          best fitness 63.46379221385115;         mean size 60.04
Generation 66;          best fitness 63.46379221385115;         mean size 59.73
In generation 67 improvement occured!
Generation 67;          best fitness 59.53751595461892;         mean size 61.01
In generation 68 improvement occured!
Generation 68;          best fitness 55.87223237953423;         mean size 57.4
In generation 69 improvement occured!
Generation 69;          best fitness 54.18897387922127;         mean size 61.49
Generation 70;          best fitness 54.18897387922127;         mean size 64.89
Generation 71;          best fitness 54.18897387922127;         mean size 73.69
Generation 72;          best fitness 54.18897387922127;         mean size 80.25
Generation 73;          best fitness 54.18897387922127;         mean size 71.86
Generation 74;          best fitness 54.18897387922127;         mean size 79.67
Generation 75;          best fitness 54.18897387922127;         mean size 88.9
Generation 76;          best fitness 54.18897387922127;         mean size 98.51
Generation 77;          best fitness 54.18897387922127;         mean size 96.93
Generation 78;          best fitness 54.18897387922127;         mean size 92.27
Generation 79;          best fitness 54.18897387922127;         mean size 92.85
Generation 80;          best fitness 54.18897387922127;         mean size 89.75
Generation 81;          best fitness 54.18897387922127;         mean size 88.96
Generation 82;          best fitness 54.18897387922127;         mean size 97.44
Generation 83;          best fitness 54.18897387922127;         mean size 89.44
Generation 84;          best fitness 54.18897387922127;         mean size 90.43
Generation 85;          best fitness 54.18897387922127;         mean size 95.89
Generation 86;          best fitness 54.18897387922127;         mean size 94.99
In generation 87 improvement occured!
Generation 87;          best fitness 51.69568077468901;         mean size 102.18
Generation 88;          best fitness 51.69568077468901;         mean size 108.39
Generation 89;          best fitness 51.69568077468901;         mean size 104.2
In generation 90 improvement occured!
```

```
Generation 90;          best fitness 50.45251190204531;         mean size 107.76
Generation 91;          best fitness 50.45251190204531;         mean size 118.5
Generation 92;          best fitness 50.45251190204531;         mean size 102.65
Generation 93;          best fitness 50.45251190204531;         mean size 93.85
Generation 94;          best fitness 50.45251190204531;         mean size 112.61
Generation 95;          best fitness 50.45251190204531;         mean size 108.88
Generation 96;          best fitness 50.45251190204531;         mean size 102.61
In generation 97 improvement occured!
Generation 97;          best fitness 49.984896541257804;                mean size 119.46
Generation 98;          best fitness 49.984896541257804;                mean size 117.35
Generation 99;          best fitness 49.984896541257804;                mean size 129.56
Generation 100;         best fitness 49.984896541257804;                mean size 123.42
In generation 101 improvement occured!
Generation 101;         best fitness 44.83978873716435;         mean size 144.29
Generation 102;         best fitness 44.83978873716435;         mean size 162.69
In generation 103 improvement occured!
Generation 103;         best fitness 44.65603089477527;         mean size 150.12
In generation 104 improvement occured!
Generation 104;         best fitness 41.54286840671069;         mean size 182.04
Generation 105;         best fitness 41.54286840671069;         mean size 189.84
In generation 106 improvement occured!
Generation 106;         best fitness 39.94350791705231;         mean size 205.35
Generation 107;         best fitness 39.94350791705231;         mean size 219.93
Generation 108;         best fitness 39.94350791705231;         mean size 245.78
Generation 109;         best fitness 39.94350791705231;         mean size 229.36
In generation 110 improvement occured!
Generation 110;         best fitness 39.85830914887674;         mean size 232.56
In generation 111 improvement occured!
Generation 111;         best fitness 38.96074417947903;         mean size 218.83
Generation 112;         best fitness 38.96074417947903;         mean size 202.61
Generation 113;         best fitness 38.96074417947903;         mean size 208.29
In generation 114 improvement occured!
Generation 114;         best fitness 37.304377348657084;                mean size 208.29
Generation 115;         best fitness 37.304377348657084;                mean size 250.72
Generation 116;         best fitness 37.304377348657084;                mean size 286.68
In generation 117 improvement occured!
Generation 117;         best fitness 37.237665672032236;                mean size 247.01
Generation 118;         best fitness 37.237665672032236;                mean size 266.54
Generation 119;         best fitness 37.237665672032236;                mean size 250.8
In generation 120 improvement occured!
Generation 120;         best fitness 37.181618948260194;                mean size 269.16
In generation 121 improvement occured!
Generation 121;         best fitness 37.14439334343079;         mean size 239.86
In generation 122 improvement occured!
Generation 122;         best fitness 37.11822745903905;         mean size 246.14
In generation 123 improvement occured!
Generation 123;         best fitness 36.97274355545995;         mean size 291.95
In generation 124 improvement occured!
Generation 124;         best fitness 36.74065679805288;         mean size 248.85
Generation 125;         best fitness 36.74065679805288;         mean size 278.04
In generation 126 improvement occured!
Generation 126;         best fitness 36.664838312398665;                mean size 285.18
In generation 127 improvement occured!
Generation 127;         best fitness 36.65485474433342;         mean size 275.88
Generation 128;         best fitness 36.65485474433342;         mean size 282.33
Generation 129;         best fitness 36.65485474433342;         mean size 274.52
In generation 130 improvement occured!
Generation 130;         best fitness 36.63260485654646;         mean size 251.8
```

```
Generation 131;         best fitness 36.63260485654646;         mean size 288.87
Generation 132;         best fitness 36.63260485654646;         mean size 297.58
In generation 133 improvement occured!
Generation 133;         best fitness 34.97031427312292;         mean size 292.09
Generation 134;         best fitness 34.97031427312292;         mean size 293.12
Generation 135;         best fitness 34.97031427312292;         mean size 295.42
In generation 136 improvement occured!
Generation 136;         best fitness 34.3203715632738;          mean size 305.96
In generation 137 improvement occured!
Generation 137;         best fitness 31.33244488711156;         mean size 305.4
Generation 138;         best fitness 31.33244488711156;         mean size 298.74
Generation 139;         best fitness 31.33244488711156;         mean size 303.63
Generation 140;         best fitness 31.33244488711156;         mean size 287.71
Generation 141;         best fitness 31.33244488711156;         mean size 295
In generation 142 improvement occured!
Generation 142;         best fitness 31.247224621012176;                mean size 288.44
Generation 143;         best fitness 31.247224621012176;                mean size 306.33
In generation 144 improvement occured!
Generation 144;         best fitness 31.035042032368846;                mean size 309.27
Generation 145;         best fitness 31.035042032368846;                mean size 270.65
Generation 146;         best fitness 31.035042032368846;                mean size 287.91
In generation 147 improvement occured!
Generation 147;         best fitness 30.98969895048679;         mean size 289.41
In generation 148 improvement occured!
Generation 148;         best fitness 30.939650656778984;                mean size 290.47
In generation 149 improvement occured!
Generation 149;         best fitness 30.855280829654905;                mean size 289.5
Generation 150;         best fitness 30.855280829654905;                mean size 287.32
Generation 151;         best fitness 30.855280829654905;                mean size 273.33
In generation 152 improvement occured!
Generation 152;         best fitness 30.845280829654907;                mean size 280.52
In generation 153 improvement occured!
Generation 153;         best fitness 30.205307126703534;                mean size 292.89
Generation 154;         best fitness 30.205307126703534;                mean size 299.45
Generation 155;         best fitness 30.205307126703534;                mean size 277.21
Generation 156;         best fitness 30.205307126703534;                mean size 289.67
Generation 157;         best fitness 30.205307126703534;                mean size 298.5
In generation 158 improvement occured!
Generation 158;         best fitness 30.13597917117244;         mean size 281.86
In generation 159 improvement occured!
Generation 159;         best fitness 29.870637750844082;                mean size 299.54
Generation 160;         best fitness 29.870637750844082;                mean size 312.84
Generation 161;         best fitness 29.870637750844082;                mean size 301.36
Generation 162;         best fitness 29.870637750844082;                mean size 312.33
In generation 163 improvement occured!
Generation 163;         best fitness 29.771474026788187;                mean size 299.44
Generation 164;         best fitness 29.771474026788187;                mean size 293.35
Generation 165;         best fitness 29.771474026788187;                mean size 296.02
In generation 166 improvement occured!
Generation 166;         best fitness 29.704940131123042;                mean size 285.9
Generation 167;         best fitness 29.704940131123042;                mean size 278.42
Generation 168;         best fitness 29.704940131123042;                mean size 266.02
In generation 169 improvement occured!
Generation 169;         best fitness 29.640448822173973;                mean size 244.34
Generation 170;         best fitness 29.640448822173973;                mean size 268.36
Generation 171;         best fitness 29.640448822173973;                mean size 247.66
Generation 172;         best fitness 29.640448822173973;                mean size 256.64
In generation 173 improvement occured!
```

```
Generation 173;         best fitness 29.457128844863288;                mean size 234.45
Generation 174;         best fitness 29.457128844863288;                mean size 244.31
In generation 175 improvement occured!
Generation 175;         best fitness 29.2899346237195;          mean size 222.9
In generation 176 improvement occured!
Generation 176;         best fitness 29.25769278856618;         mean size 227.7
Generation 177;         best fitness 29.25769278856618;         mean size 227.27
In generation 178 improvement occured!
Generation 178;         best fitness 29.22927265129067;         mean size 217.49
Generation 179;         best fitness 29.22927265129067;         mean size 238.26
Generation 180;         best fitness 29.22927265129067;         mean size 229.57
In generation 181 improvement occured!
Generation 181;         best fitness 27.44385045097991;         mean size 224.61
Generation 182;         best fitness 27.44385045097991;         mean size 247.5
Generation 183;         best fitness 27.44385045097991;         mean size 237.94
Generation 184;         best fitness 27.44385045097991;         mean size 218.61
Generation 185;         best fitness 27.44385045097991;         mean size 226.25
In generation 186 improvement occured!
Generation 186;         best fitness 27.140996818498024;                mean size 231.84
Generation 187;         best fitness 27.140996818498024;                mean size 242.19
Generation 188;         best fitness 27.140996818498024;                mean size 247.55
Generation 189;         best fitness 27.140996818498024;                mean size 229.56
Generation 190;         best fitness 27.140996818498024;                mean size 226.08
In generation 191 improvement occured!
Generation 191;         best fitness 26.89457469734793;         mean size 223.6
In generation 192 improvement occured!
Generation 192;         best fitness 26.693550390191465;                mean size 254.32
Generation 193;         best fitness 26.693550390191465;                mean size 239.21
Generation 194;         best fitness 26.693550390191465;                mean size 263.46
Generation 195;         best fitness 26.693550390191465;                mean size 251.14
In generation 196 improvement occured!
Generation 196;         best fitness 26.461096255975537;                mean size 251.07
Generation 197;         best fitness 26.461096255975537;                mean size 273.95
Generation 198;         best fitness 26.461096255975537;                mean size 255.2
Generation 199;         best fitness 26.461096255975537;                mean size 261.81
Generation 200;         best fitness 26.461096255975537;                mean size 259.65
Generation 201;         best fitness 26.461096255975537;                mean size 232.42
In generation 202 improvement occured!
Generation 202;         best fitness 26.327864331328065;                mean size 266.73
Generation 203;         best fitness 26.327864331328065;                mean size 254.92
Generation 204;         best fitness 26.327864331328065;                mean size 253.59
Generation 205;         best fitness 26.327864331328065;                mean size 250.4
In generation 206 improvement occured!
Generation 206;         best fitness 26.245458082093716;                mean size 253.25
Generation 207;         best fitness 26.245458082093716;                mean size 250.85
Generation 208;         best fitness 26.245458082093716;                mean size 227.11
Generation 209;         best fitness 26.245458082093716;                mean size 253.69
Generation 210;         best fitness 26.245458082093716;                mean size 246.31
Generation 211;         best fitness 26.245458082093716;                mean size 250.21
In generation 212 improvement occured!
Generation 212;         best fitness 26.22675748479523;         mean size 245.24
Generation 213;         best fitness 26.22675748479523;         mean size 242.3
In generation 214 improvement occured!
Generation 214;         best fitness 25.574393943029577;                mean size 253.77
Generation 215;         best fitness 25.574393943029577;                mean size 232.34
Generation 216;         best fitness 25.574393943029577;                mean size 236.46
Generation 217;         best fitness 25.574393943029577;                mean size 260.87
Generation 218;         best fitness 25.574393943029577;                mean size 244.38
```

```
Generation 219;         best fitness 25.574393943029577;                mean size 262.54
In generation 220 improvement occured!
Generation 220;         best fitness 25.19345858650797;         mean size 265.82
Generation 221;         best fitness 25.19345858650797;         mean size 270.23
Generation 222;         best fitness 25.19345858650797;         mean size 264.36
Generation 223;         best fitness 25.19345858650797;         mean size 253.97
Generation 224;         best fitness 25.19345858650797;         mean size 270.77
In generation 225 improvement occured!
Generation 225;         best fitness 25.12230444299838;         mean size 247.25
Generation 226;         best fitness 25.12230444299838;         mean size 265.71
In generation 227 improvement occured!
Generation 227;         best fitness 25.096650358057893;                mean size 228.6
Generation 228;         best fitness 25.096650358057893;                mean size 271.77
Generation 229;         best fitness 25.096650358057893;                mean size 242.64
Generation 230;         best fitness 25.096650358057893;                mean size 261.64
Generation 231;         best fitness 25.096650358057893;                mean size 268.99
In generation 232 improvement occured!
Generation 232;         best fitness 25.092158581857856;                mean size 276.71
Generation 233;         best fitness 25.092158581857856;                mean size 250.7
Generation 234;         best fitness 25.092158581857856;                mean size 260.38
Generation 235;         best fitness 25.092158581857856;                mean size 264.65
Generation 236;         best fitness 25.092158581857856;                mean size 258.46
Generation 237;         best fitness 25.092158581857856;                mean size 254.93
Generation 238;         best fitness 25.092158581857856;                mean size 233.35
In generation 239 improvement occured!
Generation 239;         best fitness 25.067765941036114;                mean size 263.37
Generation 240;         best fitness 25.067765941036114;                mean size 256.32
Generation 241;         best fitness 25.067765941036114;                mean size 260.38
Generation 242;         best fitness 25.067765941036114;                mean size 297.16
In generation 243 improvement occured!
Generation 243;         best fitness 24.974095437505984;                mean size 265.89
Generation 244;         best fitness 24.974095437505984;                mean size 255.81
Generation 245;         best fitness 24.974095437505984;                mean size 270.33
Generation 246;         best fitness 24.974095437505984;                mean size 237.29
Generation 247;         best fitness 24.974095437505984;                mean size 279.25
Generation 248;         best fitness 24.974095437505984;                mean size 239.79
Generation 249;         best fitness 24.974095437505984;                mean size 244.16
Generation 250;         best fitness 24.974095437505984;                mean size 260.36
Generation 251;         best fitness 24.974095437505984;                mean size 279.68
In generation 252 improvement occured!
Generation 252;         best fitness 24.583736142724653;                mean size 283.68
Generation 253;         best fitness 24.583736142724653;                mean size 298.27
Generation 254;         best fitness 24.583736142724653;                mean size 245.04
Generation 255;         best fitness 24.583736142724653;                mean size 251.65
Generation 256;         best fitness 24.583736142724653;                mean size 272.96
Generation 257;         best fitness 24.583736142724653;                mean size 269.22
In generation 258 improvement occured!
Generation 258;         best fitness 24.485098205606125;                mean size 282.28
Generation 259;         best fitness 24.485098205606125;                mean size 245.74
Generation 260;         best fitness 24.485098205606125;                mean size 258.77
Generation 261;         best fitness 24.485098205606125;                mean size 247.11
Generation 262;         best fitness 24.485098205606125;                mean size 243.33
Generation 263;         best fitness 24.485098205606125;                mean size 262.83
Generation 264;         best fitness 24.485098205606125;                mean size 269.15
Generation 265;         best fitness 24.485098205606125;                mean size 261.88
Generation 266;         best fitness 24.485098205606125;                mean size 249.14
Generation 267;         best fitness 24.485098205606125;                mean size 250.52
In generation 268 improvement occured!
```

```
Generation 268;         best fitness 24.428909007436825;                mean size 237.19
Generation 269;         best fitness 24.428909007436825;                mean size 263.27
In generation 270 improvement occured!
Generation 270;         best fitness 24.370072677235033;                mean size 253.3
Generation 271;         best fitness 24.370072677235033;                mean size 242.17
Generation 272;         best fitness 24.370072677235033;                mean size 252.22
Generation 273;         best fitness 24.370072677235033;                mean size 263.48
In generation 274 improvement occured!
Generation 274;         best fitness 24.12521552538716;         mean size 251.94
Generation 275;         best fitness 24.12521552538716;         mean size 260.03
Generation 276;         best fitness 24.12521552538716;         mean size 233.41
Generation 277;         best fitness 24.12521552538716;         mean size 232.55
Generation 278;         best fitness 24.12521552538716;         mean size 238.1
Generation 279;         best fitness 24.12521552538716;         mean size 218.86
Generation 280;         best fitness 24.12521552538716;         mean size 236.43
In generation 281 improvement occured!
Generation 281;         best fitness 24.111064183874884;                mean size 230.47
Generation 282;         best fitness 24.111064183874884;                mean size 230.41
Generation 283;         best fitness 24.111064183874884;                mean size 229.86
In generation 284 improvement occured!
Generation 284;         best fitness 24.044794427789505;                mean size 221.89
In generation 285 improvement occured!
Generation 285;         best fitness 24.009579954129002;                mean size 227.32
Generation 286;         best fitness 24.009579954129002;                mean size 213.36
Generation 287;         best fitness 24.009579954129002;                mean size 225.28
Generation 288;         best fitness 24.009579954129002;                mean size 202.04
Generation 289;         best fitness 24.009579954129002;                mean size 225.8
In generation 290 improvement occured!
Generation 290;         best fitness 23.927672997375055;                mean size 192.29
Generation 291;         best fitness 23.927672997375055;                mean size 215.51
Generation 292;         best fitness 23.927672997375055;                mean size 205.81
Generation 293;         best fitness 23.927672997375055;                mean size 221.39
Generation 294;         best fitness 23.927672997375055;                mean size 224.92
Generation 295;         best fitness 23.927672997375055;                mean size 215.29
Generation 296;         best fitness 23.927672997375055;                mean size 194.52
Generation 297;         best fitness 23.927672997375055;                mean size 200.97
Generation 298;         best fitness 23.927672997375055;                mean size 229.93
Generation 299;         best fitness 23.927672997375055;                mean size 221.08
the best solution evolved from seed 1626412026
```