|  |  |
| --- | --- |
| **Оюутны код: B190900003** | **Лаборатори №: 4** |
| **Оюутны нэр: Д. Батсүх** |  |

# Даалгавар 1: Travelling sells person бодлогыг Genetic algorithm ашиглан бодно уу.

# • Хавтгай дээр дурын давхцаагүй 6 цэгийг тодорхойлно

# • Тухайн 6 цэгийг А, В, С, Д, Г, Е хотууд гэж үзэн тэдгээрийн хооронд зорчих хамгийн бага зайг Genetic algorithm ашиглаж тооцолно

# • Дараах үр дүнг хэвлэж харуулна

# • 6 хотын байрлалыг

# • 6 хотын хоорондын замыг Genetic algorithm ашиглан зурсан үр дүн

Main code:

import math

import random

import matplotlib.pyplot as plt

*def* plotTSP(*paths*, *points*, *num\_iters*=1):

   x = []; y = []

   for i in paths[0]:

      x.append(points[i][0])

      y.append(points[i][1])

   plt.plot(x, y, 'co')

   # Set a scale for the arrow heads (there should be a reasonable default for this, WTF?)

   a\_scale = *float*(max(x))/*float*(100)

   # Draw the older paths, if provided

   if num\_iters > 1:

      for i in range(1, num\_iters):

         # Transform the old paths into a list of coordinates

         xi = []; yi = []

         for j in paths[i]:

               xi.append(points[j][0])

               yi.append(points[j][1])

         plt.arrow(xi[-1], yi[-1], (xi[0] - xi[-1]), (yi[0] - yi[-1]),

*head\_width* = a\_scale, *color* = 'r',

*length\_includes\_head* = True, *ls* = 'dashed',

*width* = 0.001/*float*(num\_iters))

         for i in range(0, len(x) - 1):

               plt.arrow(xi[i], yi[i], (xi[i+1] - xi[i]), (yi[i+1] - yi[i]),

*head\_width* = a\_scale, *color* = 'r', *length\_includes\_head* = True,

*ls* = 'dashed', *width* = 0.001/*float*(num\_iters))

   # Draw the primary path for the TSP problem

   plt.arrow(x[-1], y[-1], (x[0] - x[-1]), (y[0] - y[-1]), *head\_width* = a\_scale,

*color* ='g', *length\_includes\_head*=True)

   for i in range(0,len(x)-1):

      plt.arrow(x[i], y[i], (x[i+1] - x[i]), (y[i+1] - y[i]), *head\_width* = a\_scale,

*color* = 'g', *length\_includes\_head* = True)

   #Set axis too slitghtly larger than the set of x and y

   plt.show()

*class* City:

*def* \_\_init\_\_(*self*, *x*=None, *y*=None):

      self.x = None

      self.y = None

      if x is not None:

         self.x = x

      else:

         self.x = *int*(random.random() \* 200)

      if y is not None:

         self.y = y

      else:

         self.y = *int*(random.random() \* 200)

*def* getX(*self*):

      return self.x

*def* getY(*self*):

      return self.y

*def* distanceTo(*self*, *city*):

      xDistance = abs(self.getX() - city.getX())

      yDistance = abs(self.getY() - city.getY())

      distance = math.sqrt( (xDistance\*xDistance) + (yDistance\*yDistance) )

      return distance

*def* \_\_repr\_\_(*self*):

      return *str*(self.getX()) + ", " + *str*(self.getY())

*class* TourManager:

   destinationCities = []

*def* addCity(*self*, *city*):

      self.destinationCities.append(city)

*def* getCity(*self*, *index*):

      return self.destinationCities[index]

*def* numberOfCities(*self*):

      return len(self.destinationCities)

*def* getCities(*self*):

      coords = []

      for i in range(len(self.destinationCities)):

         coords.append([self.getCity(i).getX(), self.getCity(i).getY()])

      return coords

*class* Tour:

*def* \_\_init\_\_(*self*, *tourmanager*, *tour*=None):

      self.tourmanager = tourmanager

      self.tour = []

      self.fitness = 0.0

      self.distance = 0

      if tour is not None:

         self.tour = tour

      else:

         for \_i in range(0, self.tourmanager.numberOfCities()):

            self.tour.append(None)

*def* \_\_len\_\_(*self*):

      return len(self.tour)

*def* \_\_getitem\_\_(*self*, *index*):

      return self.tour[index]

*def* \_\_setitem\_\_(*self*, *key*, *value*):

      self.tour[key] = value

*def* \_\_repr\_\_(*self*):

      geneString = "|"

      for i in range(0, self.tourSize()):

         geneString += *str*(self.getCity(i)) + "|"

      return geneString

*def* getIndexList(*self*):

      indexList = []

      for i in range(self.tourSize()):

         for j in range(tourmanager.numberOfCities()):

            if self.tour[i] == tourmanager.getCity(j):

               indexList.append(j)

      return indexList

*def* generateIndividual(*self*):

      for cityIndex in range(0, self.tourmanager.numberOfCities()):

         self.setCity(cityIndex, self.tourmanager.getCity(cityIndex))

      random.shuffle(self.tour)

*def* getCity(*self*, *tourPosition*):

      return self.tour[tourPosition]

*def* setCity(*self*, *tourPosition*, *city*):

      self.tour[tourPosition] = city

      self.fitness = 0.0

      self.distance = 0

*def* getFitness(*self*):

      if self.fitness == 0:

         self.fitness = 1/*float*(self.getDistance())

      return self.fitness

*def* getDistance(*self*):

      if self.distance == 0:

         tourDistance = 0

         for cityIndex in range(0, self.tourSize()):

            fromCity = self.getCity(cityIndex)

            destinationCity = None

            if cityIndex+1 < self.tourSize():

               destinationCity = self.getCity(cityIndex+1)

            else:

               destinationCity = self.getCity(0)

            tourDistance += fromCity.distanceTo(destinationCity)

         self.distance = tourDistance

      return self.distance

*def* tourSize(*self*):

      return len(self.tour)

*def* containsCity(*self*, *city*):

      return city in self.tour

*class* Population:

*def* \_\_init\_\_(*self*, *tourmanager*, *populationSize*, *initialise*):

      self.tours = []

      for i in range(0, populationSize):

         self.tours.append(None)

      if initialise:

         for i in range(0, populationSize):

            newTour = Tour(tourmanager)

            newTour.generateIndividual()

            self.saveTour(i, newTour)

*def* \_\_setitem\_\_(*self*, *key*, *value*):

      self.tours[key] = value

*def* \_\_getitem\_\_(*self*, *index*):

      return self.tours[index]

*def* saveTour(*self*, *index*, *tour*):

      self.tours[index] = tour

*def* getTour(*self*, *index*):

      return self.tours[index]

*def* getFittest(*self*):

      fittest = self.tours[0]

      for i in range(0, self.populationSize()):

         if fittest.getFitness() <= self.getTour(i).getFitness():

            fittest = self.getTour(i)

      return fittest

*def* populationSize(*self*):

      return len(self.tours)

*class* GA:

*def* \_\_init\_\_(*self*, *tourmanager*):

      self.tourmanager = tourmanager

      self.mutationRate = 0.015

      self.tournamentSize = 5

      self.elitism = True

*def* evolvePopulation(*self*, *pop*):

      newPopulation = Population(self.tourmanager, pop.populationSize(), False)

      elitismOffset = 0

      if self.elitism:

         newPopulation.saveTour(0, pop.getFittest())

         elitismOffset = 1

      for i in range(elitismOffset, newPopulation.populationSize()):

         parent1 = self.tournamentSelection(pop)

         parent2 = self.tournamentSelection(pop)

         child = self.crossover(parent1, parent2)

         newPopulation.saveTour(i, child)

      for i in range(elitismOffset, newPopulation.populationSize()):

         self.mutate(newPopulation.getTour(i))

      return newPopulation

*def* crossover(*self*, *parent1*, *parent2*):

      child = Tour(self.tourmanager)

      startPos = *int*(random.random() \* parent1.tourSize())

      endPos = *int*(random.random() \* parent1.tourSize())

      for i in range(0, child.tourSize()):

         if startPos < endPos and startPos < i and i < endPos:

            child.setCity(i, parent1.getCity(i))

         elif startPos > endPos:

            if not (i < startPos and i > endPos):

               child.setCity(i, parent1.getCity(i))

      for i in range(0, parent2.tourSize()):

         if not child.containsCity(parent2.getCity(i)):

            for j in range(0, child.tourSize()):

               if child.getCity(j) == None:

                  child.setCity(j, parent2.getCity(i))

                  break

      return child

*def* mutate(*self*, *tour*):

      for tourPos1 in range(0, tour.tourSize()):

         if random.random() < self.mutationRate:

            tourPos2 = *int*(tour.tourSize() \* random.random())

            city1 = tour.getCity(tourPos1)

            city2 = tour.getCity(tourPos2)

            tour.setCity(tourPos2, city1)

            tour.setCity(tourPos1, city2)

*def* tournamentSelection(*self*, *pop*):

      tournament = Population(self.tourmanager, self.tournamentSize, False)

      for i in range(0, self.tournamentSize):

         randomId = *int*(random.random() \* pop.populationSize())

         tournament.saveTour(i, pop.getTour(randomId))

      fittest = tournament.getFittest()

      return fittest

if \_\_name\_\_ == '\_\_main\_\_':

   tourmanager = TourManager()

   # Create and add our cities

   city = City(60, 200)

   tourmanager.addCity(city)

   city2 = City(180, 200)

   tourmanager.addCity(city2)

   city3 = City(80, 180)

   tourmanager.addCity(city3)

   city4 = City(140, 180)

   tourmanager.addCity(city4)

   city5 = City(20, 160)

   tourmanager.addCity(city5)

   city6 = City(100, 160)

   tourmanager.addCity(city6)

   # Initialize population

   pop = Population(tourmanager, 50, True)

   print ("Initial distance: " + *str*(pop.getFittest().getDistance()))

   # Evolve population for 50 generations

   ga = GA(tourmanager)

   pop = ga.evolvePopulation(pop)

   for i in range(0, 100):

      pop = ga.evolvePopulation(pop)

   # Print final results

   print ("Finished")

   print ("Final distance: " + *str*(pop.getFittest().getDistance()))

   print (pop.getFittest().getIndexList())

   print (tourmanager.getCities())

   plotTSP([pop.getFittest().getIndexList()], tourmanager.getCities())

Оролт:

A. 60, 200

B. 180, 200

C. 80, 180

D. 140, 180

G. 20, 160

E. 100, 160

Гаралт:  
 356.27592311423297

Visualize:

