CSC-20043 Computational and Artificial Intelligence I
Evolutionary Algorithms Practical 2

Steady-State GA for the Travelling Salesman Problem

The code outline below is available as file adcTSP-outline.java. The missing parts are for you to complete, to check your understanding. (This is not assessed, and you do not need to submit anything.)

Instructions:

A) If you do not know what the Travelling Salesman Problem is then look it up
B) Read through the code outline and understand what each part should do
C) Write code to replace the CCCC, to complete the code that calculates fitnesses
D) Write code to replace the DDDDs, to complete the code that selects parents for reproduction
E) Write code to replace the EEEEs, to produce the child genotype using mutation and crossover. Your mutation function should swap two towns around in a route. Your crossover function should copy the towns before a cut point from parent 1 and take the remaining towns in the order they appear in parent 2.
F) Run the completed program several times. Has it evolved routes of length approx. 424?

```
import java.util.Random;
class adcTSP { // Travelling Salesman Problem

final int numTowns                      = 30;
final int numGenotypes                  = 1000;
final int genotypeLength                = numTowns;
final int numReproductions              = 1000000;
final int numMutationsPerReproduction = 1;

final int towns[/*numTowns*/][/*2*/]=
{ {82, 7} , {91,38} , {62,32} , {71,44} , {83,69} , {68,58} ,
  {54,67} , {87,76} , {13,40} , {71,71} , {44,35} , {18,54} ,
  {64,60} , {37,84} , {41,94} , { 2,99} , { 7,64} , {22,60} ,
  {25,62} , {54,62} , { 4,50} , {74,78} , {18,40} , {24,42} ,
  {25,38} , {41,26} , {45,21} , {58,69} , {58,35} , {83,46} };
double distances[][] = new double[numTowns][numTowns];

int genotypes[][] = new int[numGenotypes][genotypeLength];
double fitness[] = new double[numGenotypes]; // fitness = -1 * route length
int fittestIndividual = 0;

Random rng = new Random();
int random(int n) { return rng.nextInt(n); } // random integer between 0 and n-1

void calculateFitness(int individual) // fitness = -1 * route length
{
  CCCC
}

void initialise()
{ /* calculate distances between towns */
  for (int i=0;i<numTowns;i++) for (int j=0;j<=i;j++)
  { int dx=towns[j][0]-towns[i][0], dy=towns[j][1]-towns[i][1];
    distances[i][j] = distances[j][i] = Math.sqrt((double)(dx*dx+dy*dy));
  }

  /* generate initial (random) population */
  for (int individual=0;individual<numGenotypes;individual++)
  { int remainingTowns[] = new int[numTowns];
    for (int i=0;i<numTowns;i++) remainingTowns[i] = i;
    for (int g=0;g<genotypeLength;g++)
    { int index = random(numTowns-g);
      genotypes[individual][g] = remainingTowns[index];
      remainingTowns[index] = remainingTowns[numTowns-g-1];
    }
  }
  /* calculate fitness array for initial population */
  for (int individual=0;individual<numGenotypes;individual++)
    calculateFitness(individual);
}
void mutate(int individual) // swap two towns in individual's route
{
  EEEE
}

void crossover(int parentA, int parentB, int child)
{
  EEEE
}

void steadyStateGaMainStep()
{ /* pick three individuals a,b,c at random */
  DDDD

  /* reorder such that c is the least fit */
  DDDD

  /* but don't loose fittestIndividual */
  if (c==fittestIndividual)  {int temp=b;b=c;c=temp;}

  /* crossover a,b (in random order) to create child that replaces c */
  if (random(2)==1)          {int temp=a;a=b;b=temp;}
  crossover(a,b,c);
  /* mutate child */
  for (int m=0;m<numMutationsPerReproduction;m++) mutate(c);
  /* calculate fitness of child */
  calculateFitness(c);
}

void outputStatistics(int numRreproductionsSoFar)
{ if (numRreproductionsSoFar==0) System.out.println("#repros\tbest\t: route");
  System.out.print(""+numRreproductionsSoFar+"\t"+(-fitness[fittestIndividual])+"\t:");
  for (int g=0;g<genotypeLength;g++)
    System.out.print(" "+(genotypes[fittestIndividual][g]+1));
  System.out.println();
}

void run()
{ initialise();
  outputStatistics(0);

  for (int reproduction=1;reproduction<=numReproductions;reproduction++)
  { steadyStateGaMainStep();
    if (reproduction%20000==0) outputStatistics(reproduction);
  }
}

public static void main(String args[])
{ new adcTSP().run();
}

}
```