**Description for Genetic algorithm for Travelling salesman problem**

Unzip each\_with\_each\_2.zip to one new clear folder. Then open in matlab tsp\_ga.m It will be opened in matlab editor. Push F5. You will be asked to change folder, choose "change folder". Then after few seconds figure must appear.

in gui\_version subfolder there are gui version. Run tsp\_ga\_gui.m

Gui version is just same code as tsp\_ga.m but add GUI, you can change here probabilities and randomize nodes even during generations.

We have a set of cities (points) in 2d plane. Each city has road to each city. We need to find loop-path that will be in each city only one time and path length is minimal.

The genetic algorithm is sequence of following operations that repeated in generations loop:

1) find length of paths

2) find probabilities

3) prepare to crossover according probabilities

4) crossover, parents replaced with children.

5) mutations

So we have a lot of paths with different lengths. A path can have crossover with another path and mutate. After crossover it will be replaced by its children. All paths recorded in matrix G :



here we have population with 6 paths (population size ps=6), and it is for 4 cities. Each row of matrix G is describe one path, for example path No.3 is:

4 3 2 1, this means that city No. 4 connected to city No. 3, city No. 3 connected to city No. 3, city No. 3 connected to city No. 2, city No.2 connected to city No. 1, AND city No. 1 connected to city No. 4, I.e last city in the list always connected to first one. Also each city here can be repeated only one time. So we always consider only closed paths that was in each city one time.

In the code to generate initial population of random paths randperm matlab function was used. It return randomly permutated numbers, for example:

>> randperm(4)

ans =

3 4 2 1

To have fast calculation of paths length before start iteration it is calculated matrix of distancies dsm, it is nn x nn matrix where nn is number of cities, dsm(i1,i2) is distance between city i1 and city i2.

there is a inverse distance is used, that is 1/distance. If we want to maximize distance then we want to minimize inverse distance.

The feature of this TSP task for GA is that after crossover and mutation we must have again closed paths that was in each city one time. So we need to use specific crossover and mutation that preserve this property. The idea for such crossover was get here:

<http://www.ceng.metu.edu.tr/~ucoluk/research/publications/tspnew.pdf>

We have 2 parents and want to replace them with 2 childrens:



first we chose random point of cross-section, say between 2st and 3rd positions:



Now we want to have children that have first part like in parent 2. We start to copy cities from parent 2 to parent 1, we go from beginning to cross-section, also together with coping we make swap:



after that we have:



So each time we need to find where is city number of parent 2 in parent 1 to make swap.

now go next (second) position, we do nothing here because it already 2. So we get 1st child:



The same for child 2 but for child 2 we copy cities of parent 1 to parent 2 cells.

We chose parents to pot in crossover. Then more big inverse distance of some path then it have bigger probability to be parent in crossover.

For this case roulette\_wheel\_indexes.m function was made. First calculate probabilities as inverse distances divided by sum of inverse distances:

where is probability of path to be put to crossover, is path length.

for example we have 6 paths with numbers

1 2 3 4 5 6

and let it have probabilities

0.05 0.5 0.05 0.05 0.05 0.3

then by using roulette\_wheel\_indexes.m function we will put for the crossover for example

1 2 6 2 6 2

you can see that 2 most frequent and 6 also frequent, rest a rare or even not presented. But all this is random.

So after that we have pair wise crossover:

1 and 2 (gives 2 children)

6 and 2 (gives 2 children)

6 and 2 (gives 2 children)

roulette\_wheel\_indexes.m function works in following way:

we have matlab function rand that give random number form [0;1] range. So we split this range in to 6 (as in current example) pieces and length of each piece is equal to probability. Then we see to what pieces it come. The probability of choise of a piece is proportional it's length, because rand gives random number from uniform distribution.



and this will repeat 6 times to chose 6 parents. Then 6 parent will be replaced with 6 childrens.

after crossover there are mutations. There are 3 types of mutations used:

1) mutation of exchange 2 random cities in the path

Chose 2 random cities



and swap them, result:



all mutations has adjusted probabilities that simulated with rand function condition

rand<pm is true with probability equal to pm because rand return random number with uniform distribution in [0;1] range.

2) mutation of exchange 2 pieces of path

first randomly chose point of spit, for example between 2 and 3 positions:



now swap two pieces:



3) mutation of flip random piece of path

first chose some random piece of path:



now flip it left-right:



Elitism means that we save best path and put it manually to next generation without changes.

G(1,:)=Gb; % elitism

It was put on first place but order is not important. The elitism makes improving of best path at each iteration or keep same path.