ECTA Homework 2
Genetic Algorithms
and
TSP

Erick Kramer, Mihir Patil
erick.romero@smail.inf.h-brs.de , mihir.patil@smail.inf.h-brs.de

May 17, 2018

1 Assignment Description

1. Write a Genetic Algorithm to solve the Traveling Salesman Problem (TSP)
   - All cities visited once, coming back to the start city
   - 100 largest cities in Germany (data files on LEA)
   - Minimize the distance traveled
2. This time, to give you further insight into the inner workings of genetic algorithms we would like you to compare different mutation and crossover rates. Please use the plotting templates from the last weeks homework for your comparision. Use the parameters listed below for you comparision and explain the observed effects. Therefore please test and compare the effects of 4 different mutation rates and 4 different crossover rates.
   - Mutation rates:
   - 1/nCities%
   - 1%
   - 10%
   - 99%
   - Crossover rates:

- 1%
- 10%
- 80%
- 99%

3. Ensure a large enough sample for reliable results by repeating the experiments at least 30 times and reporting the median

2 The Assignment

2.1 Coding GA for TSP

1. **Tournament Selection**

```
%------------- BEGIN CODE --------------

%% Tournament Selection
%Number of iterations to have as many parents for desired children
%2 would assure that we have an array of 200 parents

parentIds = randi(p.popSize, [p.popSize 2]);

iterations = 2;

% Array of random values from 1 till popSize
rand_fit = randi(p.popSize, [p.sp p.popSize*2]);

%Using the max over the fitness give us the [max_value (which ...
    we do not
%care), and the index of the max_value] of the pairs randomly ...
    instantiated.
[¬, fit_idx] = min(fitness(rand_fit));

%Create the parentIds array with the index of the maximum ...
    values of the
%rand_fit array.
for i=1:length(rand_fit)
    parentIds(i) = rand_fit(fit_idx(i),i);
end
%------------- END OF CODE --------------
```

2. **Crossover**

```
%------------- BEGIN CODE --------------

%%

%Initialization of the children array
children = zeros(size(pop));

nCities = size(pop,2);

%Calculation of the number of parents
num_parents = size(parentIds,1);

%Array containing a flag for the parents over which crossover ...
    is going to
%be performed.
%If the random number between 0 and 1 is less than the ...
    probability of
%crossover True is stored, False otherwise.
cross_flag = rand(2,length(parentIds)) < p.crossProb;

%Iterate over the parents array in steps of 2
for i = 1 : num_parents

    %If both parents are active for crossover
    if cross_flag(1,i) && cross_flag(2,i)

        % -- Using set theory to find missing and common values
        parentA = parentIds(i,1);
        parentB = parentIds(i,2);

        if p.cross_flag == 'one_point'
            %One point crossover functions
            children(i,:) = one_point_crossover(pop, parentA, ...
                                                parentB,nCities);
        elseif p.cross_flag == 'two_point'
            %Two point crossover function
            children(i,:) = two_point_crossover(pop, parentA, ...
                                                parentB,nCities);
        else
            disp('Invalid crossover statement, keeping first ...
                parent')
            children(i,:) = pop(parentIds(i,1),:);
        end

    %If one or both parents are not active for crossover
    else
        %Create the genome for the children by passing the ...
            first parent
        children(i,:) = pop(parentIds(i,1),:);
    end
end
%------------- END OF CODE -------------
```

3. **Mutation**

```
%------------- BEGIN CODE --------------

%% No mutation happening, can you do better?
children = children;

%Logic array of children to be mutated
mutPaths = rand([p.popSize,1])<p.mutProb;

%Array of indices of children to be mutated
pathNo = find(mutPaths);

%Two random cities to be swap
swapCities = randi([1 p.nGenes], [1 2]);

if ¬(isempty(pathNo)) && swapCities(1) ≠ swapCities(2)
   for i = 1:length(pathNo)
       individual = children(pathNo(i),:);
       individual([swapCities(1) swapCities(2)]) = ...
           individual([swapCities(2) swapCities(1)]);
       children(pathNo(i),:) = individual;
   end
end
%------------- END OF CODE --------------
```

4. **Elitism**

```
%------------- BEGIN CODE --------------

%% Here we just keep the first individual as an elite, can you ...
    do better?

%Number of individuals to be kept
num_elites = round(p.popSize * p.elitePerc);

%Get the indeces of the sorted individuals
[¬, eliteIds] = sort(fitness, 'ascend');

%Maintain only the top individuals
eliteIds = eliteIds(1:num_elites);
%------------- END OF CODE -----
```

2.1.1 Comparing Algorithms

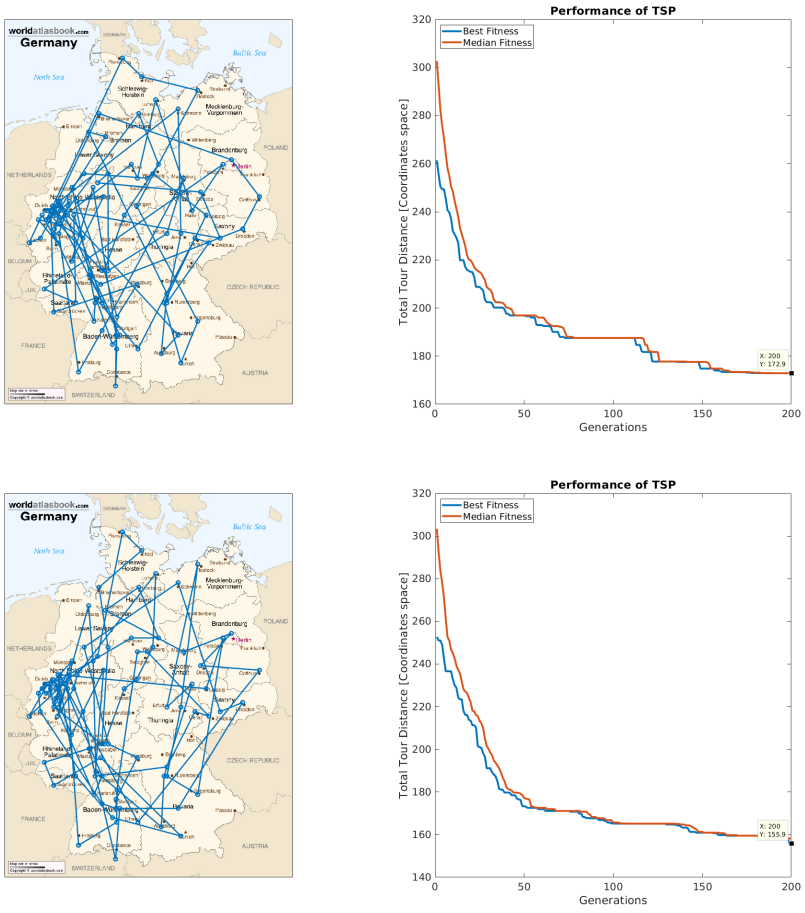

Figure 1: **One point vs two point crossover with 30 runs**
Top Left: One point crossover best individual, *Top Right:* One point crossover fitness and median, *Bottom Left:* Two point crossover best individual, *Bottom Right:* Two point crossover fitness and median

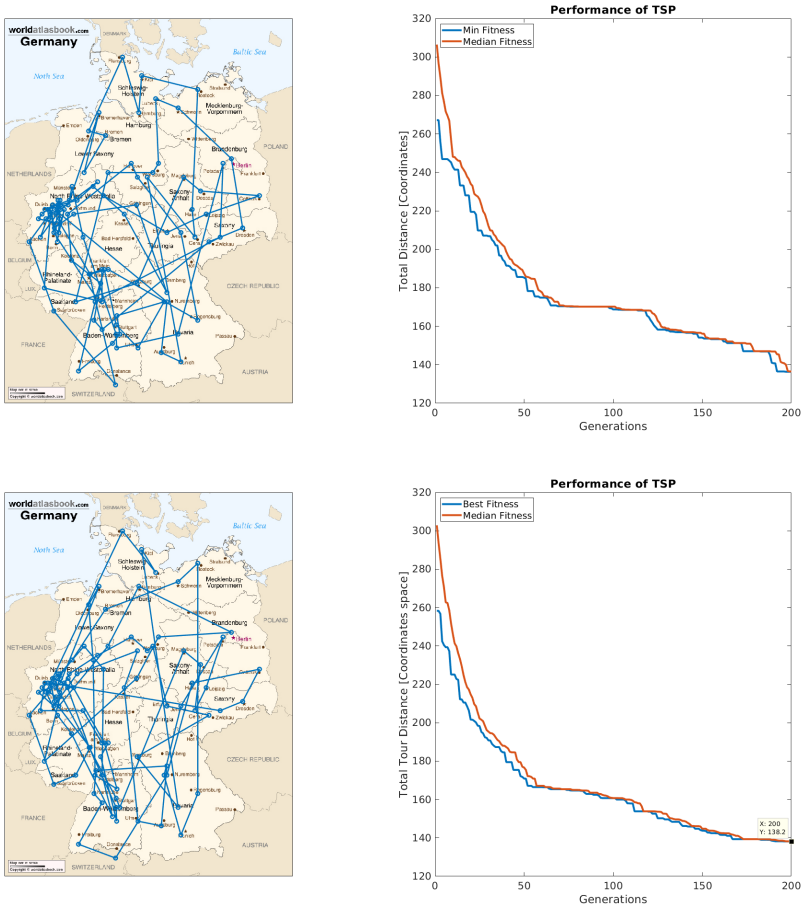

Figure 2: **Best configuration mutation/crossover (80% crossover, 10% mutation) rate combination of single run vs 30 runs**
Top Left: Single run best individual, *Top Right:* Single run fitness and median, *Bottom Left:* 30 runs best individual, *Bottom Right:* 30 runs fitness and median
MIHIR: Insert explanation behind the different rates of mutation and crossover rates and why one works better than the other one.