Evolutionary Computation Theory and Application - Assignment 2: Traveling Salesman Problem -

Debaraj Barua (9030412), Md Zahiduzzaman (9030432)

1 General Remarks

Please Follow those remarks. Deviating will lead to a reduced score

- Lable your axis
- Include a descriptive, not covering legend in your plots
- Caption you images with a clear description
- Remember to name the file correctly
- Make sure that both team members submit the same file, with the same name
- Please make sure that all figures and lines are clearly readable

2 Solution

Parameter	**Value**
Population size	50
Crossover Rates	1%, 10%, 99%, 100%
Mutation Rates	1%, 10%, 99%, 100%
Repetitions	30
Generations	1000
Average best fitness	108.1341
Minimum best fitness	69.8973

Table 1: 99% mutation rate used for various crossover rate and 99% crossover rate used for various mutation rate. For 30 repetition, both crossover and mutation rate was set to 99%

2.1 Details of Approach

1. **Tournament Selection**

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

%% TOURNAMENT SELECTION SOLUTION
parentIds = NaN(p.popSize, 2);

for child_count = 1:p.popSize
    group = randi(p.popSize, [p.sp,2]); % Get two sets of random ...
       individuals
    [¬, winner_index] = min(fitness(group)); % Get a parent from ...
       each set based on highest fitness

    first_parent_index = group(winner_index(1,1), 1);    % Get the ...
       first parent
    second_parent_index = group(winner_index(1,2), 2);  % Get the ...
       second parent

    parentIds(child_count, :) = [first_parent_index ...
       second_parent_index];
end
%------------- END OF CODE --------------
```

- We select two groups of random individuals.
- We select the best performing individual from each group and choose these as the two parents.

2. **Crossover**

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

%% ONE POINT CROSSOVER SOLUTION
children = NaN(p.popSize, p.nGenes);
for index=1:p.popSize
    % Determine whether to do cross over or not
    doCrossOver = rand(1) < p.crossProb;

    if(doCrossOver)
        % Define cross over point
        crossOverPoint = randi(p.nGenes);

        first_parent_index = parentIds(index,1);
         % Take first part from the first parent
        first_part = pop(first_parent_index,1:crossOverPoint);

        second_parent_index = parentIds(index,2);
        % Find genes not present in first part
        missing_genes = setdiff(1:p.nGenes,first_part);
        % Get the remaining genes from second parent
        second_part = intersect( pop(second_parent_index,:) ...
           ,missing_genes,'stable');
        % combine the two parts
        children(index,:) = [first_part second_part];
    else
        % Take first parent in case of no cross over
        children(index, :) = pop(parentIds(index,1), :);
    end
end
%------------- END OF CODE -------------
```

- We iterate over each individual in the population and check if we should use crossover using the crossover probability.
- If False, we use the first parent as the child.
- If True, we select one random point in the gene and split it into two parts.
- We then create a new individual by selecting the first part from the first parent and the second part from the second parent.

3. **Mutation**

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

%% POINT MUTATION SOLUTION
for childIndex=1:p.popSize
    % Determines whether do mutation or not
    doMutation = rand(1) < p.mutProb;
    if doMutation
        genes = children(childIndex, :);

        index_1 = randi(p.nGenes);
        index_2 = randi(p.nGenes);

        start_index = min(index_1, index_2);
        end_index = max(index_1, index_2);

        first_part = genes(1 : start_index);
        second_part = genes(start_index+1 :  end_index);
        third_part = genes(end_index+1 :  end);

        genes = [third_part first_part second_part];

        children(childIndex, :) = genes;
    end
end
%------------- END OF CODE --------------
```

- We iterate over each individual in the population and check if we should mutate it using the mutation probability.
- If True, we select two random points in the gene and split it into three parts.

- We then create a new individual by reorganizing these three parts, as shown above.

4. **Elitism**

```
%------------- BEGIN CODE --------------
[¬,sorted_population_indices]= sort(fitness,'ascend');
eliteIds = sorted_population_indices(1:ceil(p.popSize * ...
   p.elitePerc));   % Take population with highest fitness
%------------- END OF CODE ---------------
```

- From each generation, we save some best performing individuals, and forward them to the next generation.
- This ensures that we do not loose all the best individuals to mutation and/or crossover.

3 Results

3.1 Different mutation rates

Describe and explain the different mutation rates and how they influence the learning behavior. Please remember to also focus on why, not only on what. Also elaborate on the mutation rate you have chosen as best mutation rate.

- From the graph above we see that the fitness improves as we increase the mutation rate.
- The complexity of problem with brute force is of the order of $O(n!)$. That is, we have a huge search space and thus to solve within a reasonable execution time, we need higher randomization or change.
- Thus, we can conclude that using a higher mutation rate will avoid getting stuck in local minima and enable us to converge to a solution faster.
- Thus we select a mutation rate of 100% as our best estimate of the mutation parameter.

3.2 Different crossover rates

Describe and explain the different crossover rates and how they influence the learning behavior. Please remember to also focus on why, not only on what. Also elaborate on the crossover rate you have chosen as best mutation rate.

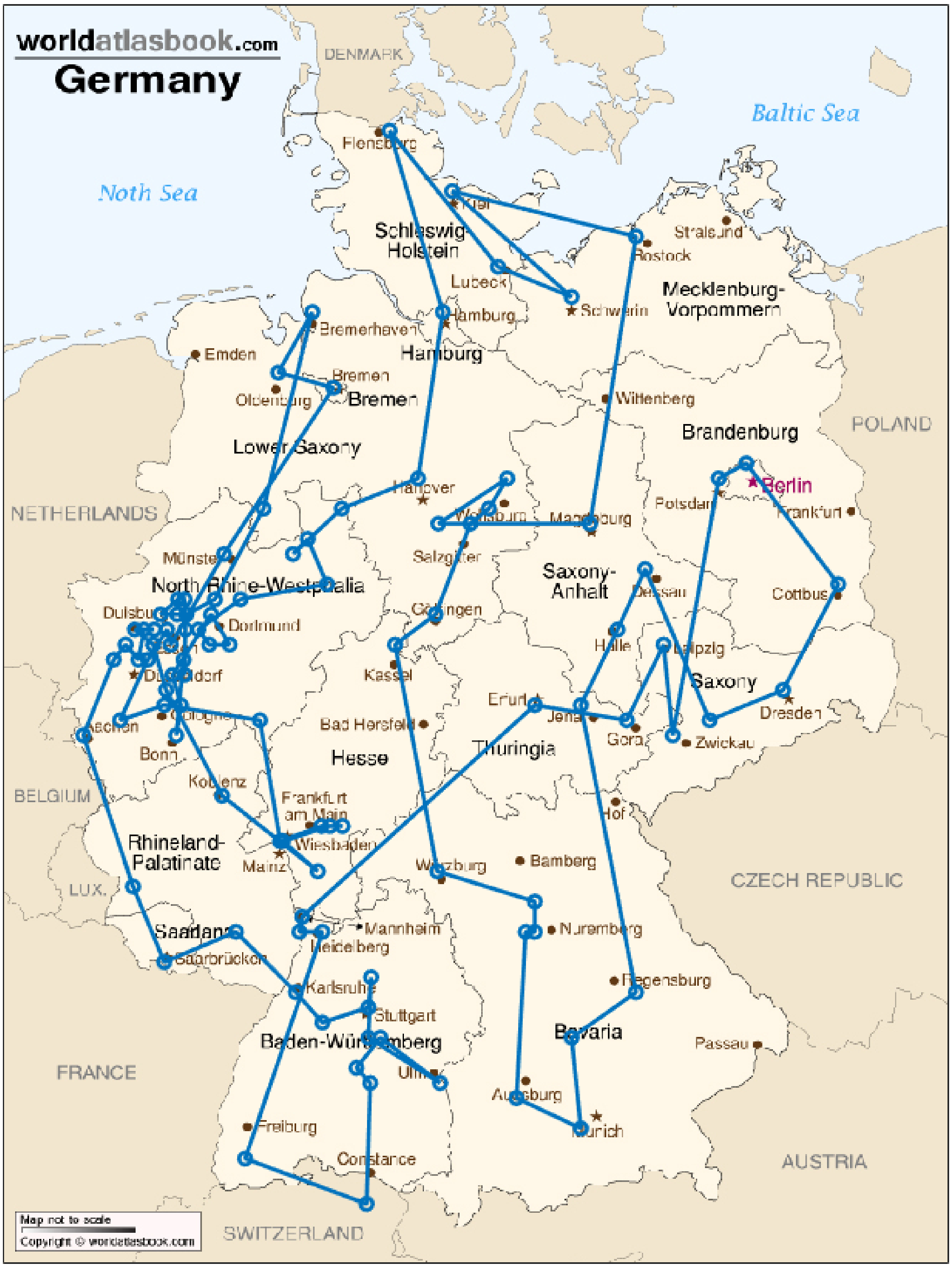

Figure 1: Map using 100% crossover and 100% mutation

- As elaborated for the mutation rate, we observe that on increasing the crossover rate, we get better fitness values.
- Following a similar logic, we choose 100% crossover rate as our best estimate.

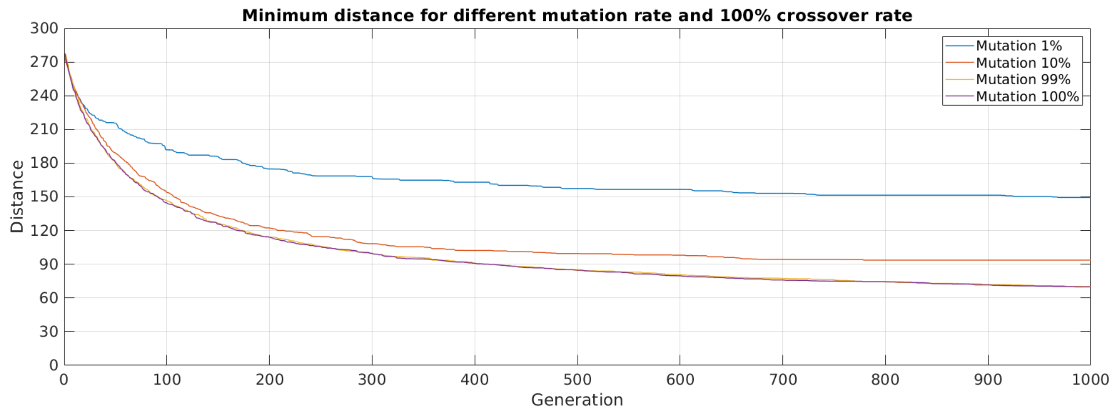

Figure 2: Different mutation rate for 100% crossover

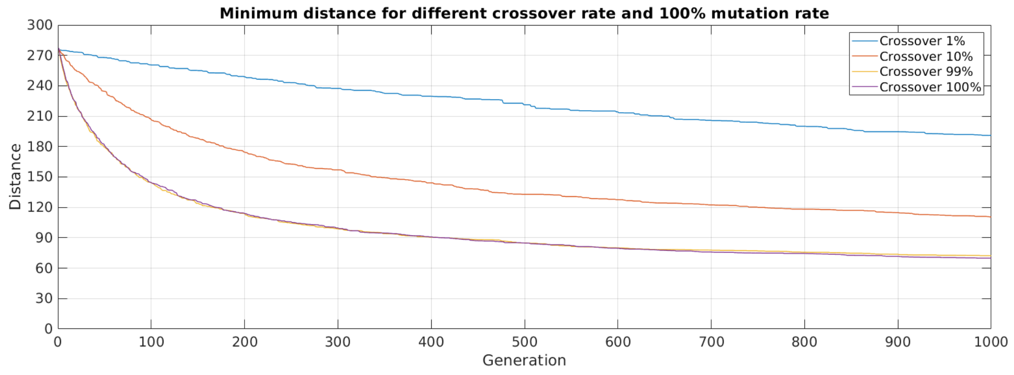

Figure 3: Different crossover rate for 100% mutation

- However, on comparing with the previous graph in Figure 2, we can observe that using a higher crossover rate results in converging to a better solution faster.
- In Figure 3, we can also observe that for lower values of crossover, the fitness value does not saturate completely by 1000 generations.
- This can be reasoned as below:
  - Crossover is preceded by the selection of parents.
  - During this step, two groups of parents are selected and the best performing individuals from these groups are used as parents.
  - Thus, it stands to reason that when our crossover rate is high, we are using better performing parents to potentially create better performing child.
  - When our crossover is low, we forgo the possibility of using the fitness of previous generation, and must always rely on mutation. Thus leading to slower convergence.