

I. The Best Result Computed by my GA

```

Shift GA: Fitness: 2728 Generation: 1340 Individuals: 669642
 14 -> 16 -> 18 -> 19 -> 20 -> 21 -> 22 -> 23 -> 60 -> 32 -> 33 -> 35
 34 -> 69 -> 41 -> 37 -> 38 -> 36 -> 63 -> 40 -> 42 -> 68 -> 46 -> 52
 53 -> 44 -> 43 -> 17 -> 64 -> 66 -> 12 -> 15 -> 28 -> 24 -> 26 -> 25
 27 -> 6 -> 7 -> 9 -> 11 -> 10 -> 58 -> 61 -> 56 -> 70 -> 3 -> 4
 5 -> 2 -> 0 -> 31 -> 30 -> 29 -> 65 -> 1 -> 45 -> 13 -> 54 -> 55
 62 -> 39 -> 47 -> 51 -> 50 -> 67 -> 49 -> 48 -> 57 -> 59 -> 8
Greedy GA: Fitness: 1944 Generation: 1551 Individuals: 775349
 24 -> 29 -> 30 -> 31 -> 45 -> 46 -> 49 -> 48 -> 51 -> 50 -> 67 -> 47
 39 -> 40 -> 68 -> 62 -> 54 -> 55 -> 52 -> 53 -> 44 -> 43 -> 42 -> 35
 34 -> 69 -> 41 -> 37 -> 63 -> 38 -> 36 -> 60 -> 33 -> 32 -> 26 -> 22
 17 -> 64 -> 28 -> 0 -> 65 -> 1 -> 2 -> 5 -> 25 -> 27 -> 6 -> 7
 9 -> 11 -> 10 -> 58 -> 61 -> 57 -> 56 -> 70 -> 3 -> 4 -> 8 -> 59
 13 -> 66 -> 14 -> 12 -> 15 -> 16 -> 18 -> 19 -> 20 -> 21 -> 23

```

II. Description of Used Techniques

Simple Genetic Algorithm, e. g. the genetic algorithm using only selection, simple crossover, elitism and simple mutation as is described in the lecture notes. For this GA and all its successors except the greedy algorithm, the main problem was how to encode the permutation of the cities so that crossover operator could be easily applied /without destroying the permutation/. I have solved this problem by using the encoding that codes only the offsets of the cities /combined with swapping/. This means that on the i -th position from the left, only $(i-1)$ numbers are allowed and thus crossover will always produce valid offsprings. For example, the permutation EDBAC will be coded like 4,2,1,1,0. The last zero is redundant but used for more clear code.

Middle Genetic Algorithm is the simple genetic algorithm with double crossover and progressive mutation, e. g. genetically stronger individuals are likely not to suffer mutation while genetically weak individuals may mutate with higher probability. In this case the mutation could be also applied more than once to certain individual.

At this point, I was not able to increase the performance of the GA and so I tried a new way. I have realized that although many possible solutions are very similar /they exactly vary only in one edge/ such as are the EDBAC and BACED permutations, these two permutations has the ability to produce new offsprings using the simple crossover operator. So I introduced my new operator, shift /denoted as H in logfile/. In the *Shift GA*, this operator is simply used to shift the permutation, if the offspring is the same as one if its parents.

Thinking forward I have realized that this shift could be also very useful when applied randomly as mutation on any individuals as well as on the best individual copied from previous generation to the next one, now in many shifted copies. The *Advanced Shift GA* represents these changes. This is also the best result I was able to generate without looking for any other solutions.

However I have found the information about the *Greedy Crossover*¹ and tried to implement it as well. Although I am not sure if I may use this because the way how crossover works is not found in evolution /as far as I know/ and may be similar to other search methods, I was surprised by the enormous speedup.

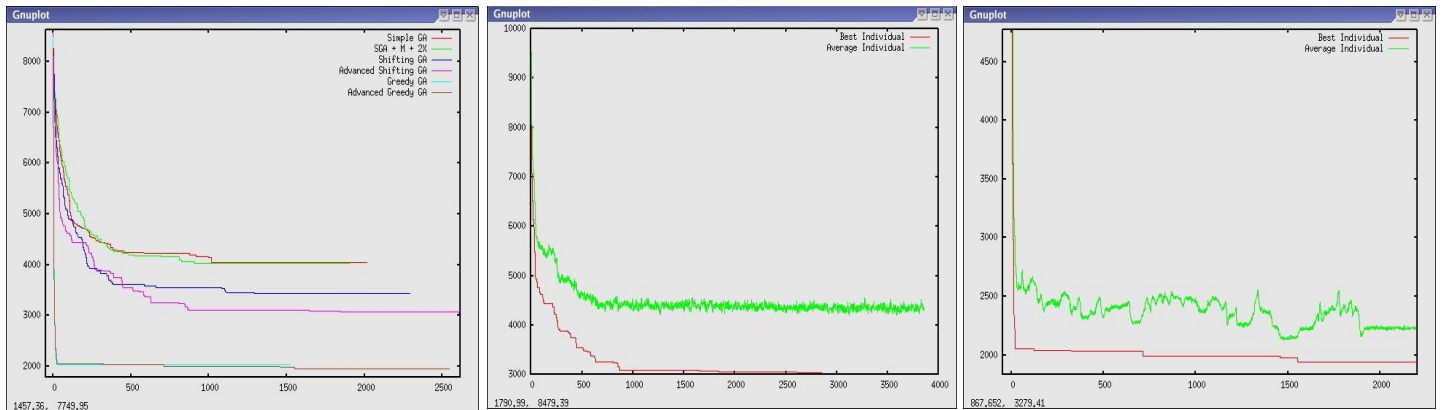
Thinking about the Greedy Crossover I was trying to make it even better and I have made some changes to the code to improve the divergence of the population, such as the random beginning. Later I have found this methods described on Internet² as well.

III. Performance Analysis

According to the previous information, I have put the graph showing the average performance of all five used methods on the next page. The first image shows the best individual found so far where fitness is on the y-axis and number of generations is on the x-axis. The second figure shows the best and average fitness of population for the Advanced Shift GA and the last image shows the same for the Advanced Greedy Crossover Algorithm.

1 <http://www.tsp.gatech.edu/>

2 <http://www.msu.edu/~miagkikh/web/5.ASC>



The high overlap of the shift algorithm is due to the shifted individuals, which are considered to be different ones because of just basic function measuring diversity. Interesting is the graph of greedy crossover where the nearly optimal solution for the given initial population is achieved in extremely small number of generations and so the premature convergence of the population is still a problem. Another interesting thing happens when the diversity drops to the constant level – this is the time when certain permutation completely took over the population.

The population size in all cases is 500 individuals. The application is terminated if better solution has not been found in the previous 200 generations.

IV. Fitness evaluation and Roulette Wheel Implementation

In the algorithm, I do not use the fitness described as the cost of the way. Instead I compute the average, best and worst fitness of the population and between these three points I plot a function with defined minimum, maximum and average. Then I assign as a "fitness" to each individual the value of this function and remember the sum of all these values in the population. Roulette wheel is then implemented as selection of random number less than sum of new fitness values and then from the beginning I decrease the random number by the fitness of i-th individual. When I reach zero, I perform selection.

The function mapping old fitness to the new fitness could be linear, cubic, or inversed cubic. The cubic function has the fastest convergence. In all examples the linear function has been used because it gives the best results.

V. Summary of Logfile Operators and program usage

The following operators could be found in the log file Selection /S/, Single Crossover /X/, Double Crossover /Y/, Greedy Crossover /G/, Mutation /M/ and Shift /H/. Please note that the fitness printed in the logfile is not the fitness used in the GA /because this fitness does not show the progress of the population/ but just the path cost and thus it is decreasing in time.

The application is highly customizable by the macros all to be found in the ga_tools.h file. Their names and values are self explanatory. Application is now in the state of producing the greedy crossover algorithm. Please note that the ftv70.atstp file must be in the same directory as the executable.

Application is the KDevelop project, but could be run on any computer with the GNU autotools by using this commands in the application root folder:

```
./configure
make all
```

VI. Conclusion

All the presented subalgorithms are successful compared to random permutation generation. However the Advanced Shift GA is the most powerful technique without using any other comparison, just genetic operators and fitness comparison of all individuals. The greedy crossover is far more powerful, but includes the comparison of the fitness of single pieces of genetic information.