Self Organizing Systems 2022W
Assignment 1: Christmas market with GA

Group 12:
Gawor Adrian Jan 11905152
Braunsperger Martin 11909911
Blohm Peter 11905150

1 Problem Description

The Time-constrained "Travelling Christmas market visitor" problem is one of the custom problems described in the given PDF.

As a quick summary:

Given the locations and opening times of various Christmas markets all around Vienna, one wants to find a route to visit as many of them in a single day as possible.

- Each market may only be visited at most once and only during open hours.
- The time spent at a market is exactly 30 minutes.
- Travelling time between markets must be accounted for.

2 Representation of the Problem Domain

While explicitly stated in the problem description, that this problem is not technically a version of the TSP, we still chose to represent the solution as such, a permutation which dictates the order in which the markets are visited.

This encodes some properties of the problem quite nicely, and using a different target function, all constraints can be expressed.

One disadvantage this approach could cause is that changes to later parts of a solution will not affect the fitness, since markets later in the list will not be visited at all, regardless of their relative order.

Changes to these parts of the solution, however, could be viewed as silent mutations, and higher mutation rates will counteract these non-visible changes.

Before solving the actual problem, however, some amount of preprocessing is needed to get a usable instance akin to TSP.

2.1 Preprocessing

Since the location of the markets was only given as links to their Google-Maps entry, the preprocessing was performed in two steps.

First, the coordinates of the markets were obtained. This was done by sending requests to Google for those resources. Since, depending on the link, either the page or the URL of the entry will contain the latitude and longitude of the location, they can be grabbed from the responses via regex.

Afterwards, using the Matrix-API of Openroute Service, which uses the OSRM, was used to get a matrix of *walking* durations between all locations.

The reason the walking durations were used was that there were no free API's for public transport in Vienna found and driving after a few Glühweins could impact the travel times significantly.

To gauge performance of the algorithm depending on the instance size, a few other instances were generated from random locations around Vienna, but using the real walking times between those. Since the time-constraint limits the quality of the solution, these instances use closing times *after 24:00*, which is of course unreasonable but causes no problems in the chosen implementation.

Alternatively, walking and staying durations could be shortened until satisfactorily large solutions are possible.

2.2 Target Function

As mentioned above, the problem was essentially encoded as a TSP variation with following target function: Starting at the first opening time of an instance, the walking times throughout the day are accumulated, keeping the time of arrival for each market in mind.

If a market is visited before it is opened, the person is assumed to *wait until it is open* and only then continue the journey.

When a market is visited that closes either before the arrival or during the stay, the journey is over.

Since the total time frame as well as the duration of stay per market is fixed, minimizing the walking times then has the same effect as maximising the time spend on markets which is the same as maximising the amount of visited markets.

An interesting question to ask is whether to count markets which close during the stay and are not visited for the full 30 minutes.

While this does not influence the actual target function, we decided to include closing markets into the solution, which is important when looking at the resulting numbers for the visited markets.

3 GA implementation

Our approach to solve this problem was using genetic algorithms. An existing implementation for genetic algorithms was used from the scikit-optimize library, which had a specific preset for the TSP.

This preset is defined by the default set of operators which will be described below:

- Mutate-Reverse aka 2-Opt

  A random section of the solution will be reversed, which for the conventional TSP corresponds to a 2-Opt-like mutation.

- Tournament selection with groups of 3 Based on the ranking by the target function as described above.

- PMX-Crossover

  This modified version of crossover was already described in the lecture as an adaptation for TSP and starts by copying a chunk of one parent solution and filling the other values in a way as to still produce a valid permutation.

4 Results

To analyse the behaviour of the chosen approach, results were collected for different population sizes and different iteration limits. It is to note that for each iteration limit, the algorithm as completely restarted, and did not just continue from a previously measured state.

The results were according to expectations: population size as well as iteration limits increased the individual run-time by a linear factor respectively.

A resulting plot for this experiment can be seen in figure 1, clearly showing the linear increases of runtime but on the contrary a slight average increase of the visited markets. An example of a found route is shown via a screenshot of google maps in figure 2.

Plots of different instances, also included in the submission, showed that the amount of iterations until good solutions are found, also as expected, increases very fast with the amount of possible stops.

Choosing stopping criteria by amounts of iterations without improvement seems to be a good idea as in most cases more or less linear improvement can be seen until a sudden stop.

As a second experiment, the different settings for the selection operator in figure 3 and for the mutation operator in figure 4 are shown. The defaults proved to be fine, but another mutation operator, the swap-mutation, where two different stops are, as implied, swapped, seems to produce more consistent results.

5 Discussion

In conclusion the chosen approach via genetic algorithms seemed to work well. While gauging performance with only one approach and without any optimal results for the instances is not that trivial, the original instance could be used to produce solutions of up to 19 visited Christmas markets, where the last one is not visited for the full 30 minutes. This route, also depicted in figure 2 is included in a text file in the submission package.

While some experiments with different operators have been made, proper gridsearch over the amount of hyper-parameters possible takes a long time and there are many ways to further tweak the performance of the algorithm.

The default settings for the hyperparameters preselected for the TSP proved fine for this specific problem as well.

One of the important lessons learned was not to blindly apply operators which are pre-implemented, as especially for this representation, the wrong crossover or mutation operator easily breaks the result.

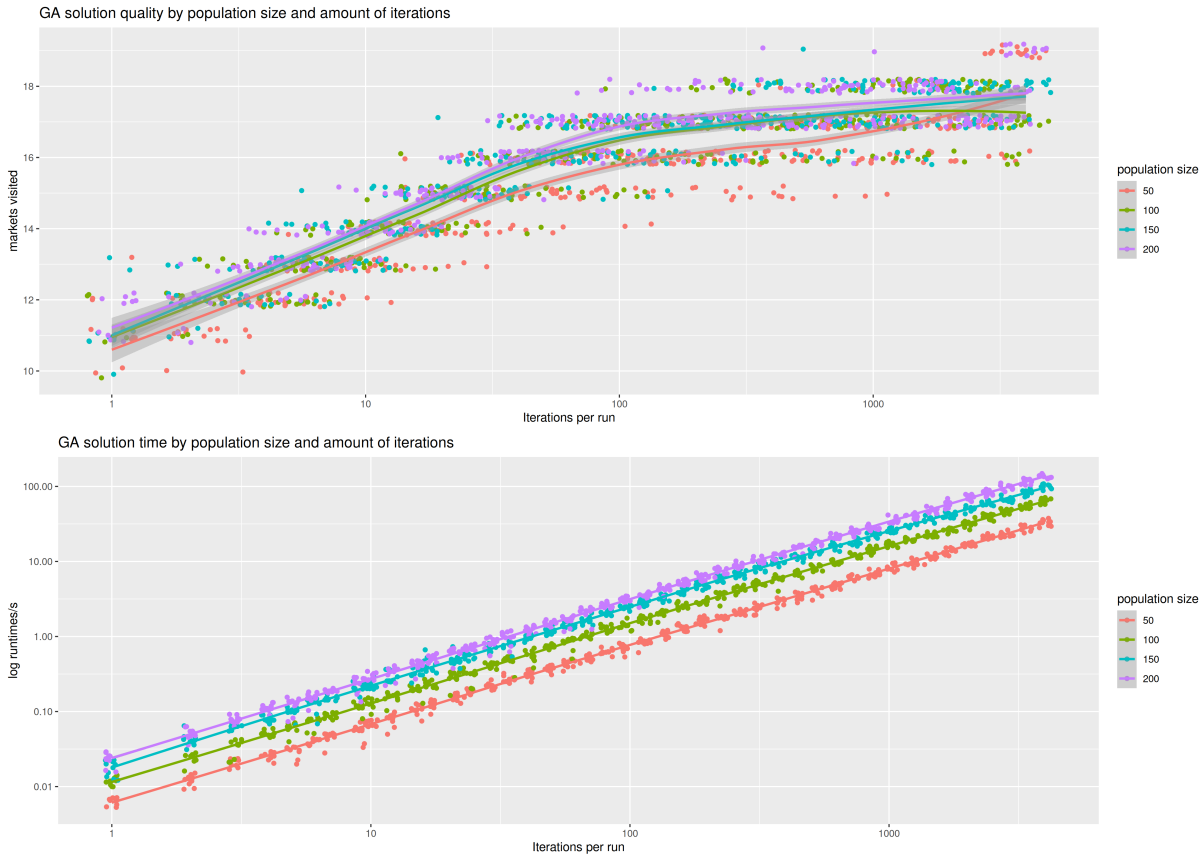

Figure 1: Results for the original TCMT Instance for different population sizes and iterations

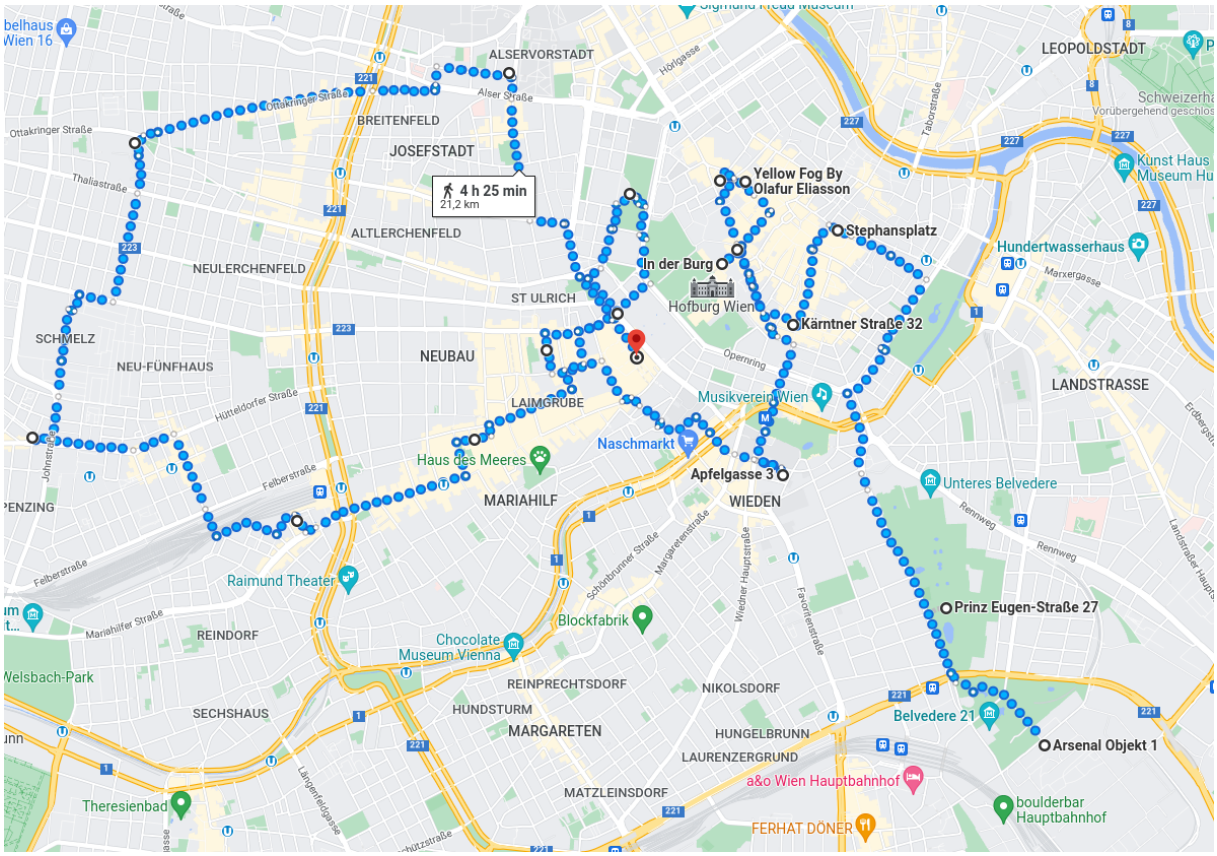

Figure 2: One Example of the best found route with nearly 19 full stops. The interactive version can be found here

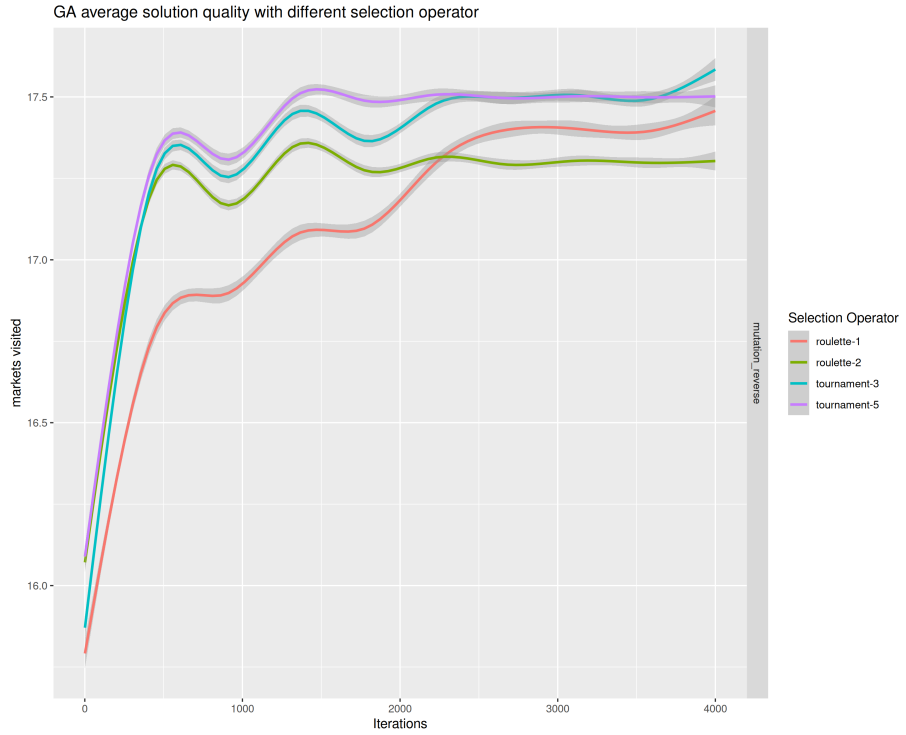

Figure 3: Experiment for comparing different selection operators. Tournament selection performs best.

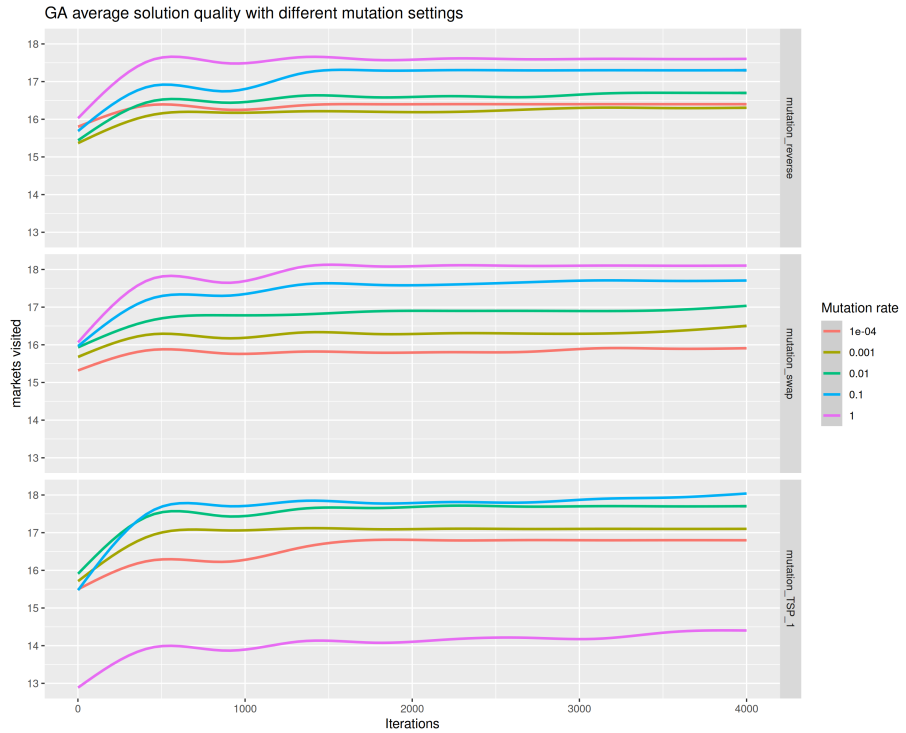

Figure 4: Experiment for comparing different mutation settings. Swap Mutation shows better expected results than reversing mutation. TSP-Mutation showed good results with smaller learning rates.