Seidel Rally Vienna

Self Organizing Systems

Martin Wustinger (12122402)
Jonas Unruh (12331457)

November 2024

1 Problem Description

For our problem implementation we chose the custom topic "Seidel Rally Vienna", were the objective is to drink a Seidel (0.3l beer) in a bar in every single district of Vienna. This is a variant of the Travelling Salesperson Problem (TSP), where we need to find the shortest path through every district of Vienna, while visiting a bar in each. The districts contain a list of multiple bar options, the number was varied in the experiments, but for the base implementation it is a list of three randomly chosen bars.
It was specified that it is possible to define a sequence for visiting the districts, so we decided on following the reverse sequence ($23 \rightarrow 1$). To get the travel times between bars Google Maps Directions API was used.

2 Creating the Graph

2.1 Graph Generator

To streamline the creation of graphs for this exercise and to enable the generation of multiple graphs with minimal additional effort, we developed a step-by-step graph generator. The Python package *osmnx* provides easy access to street and geographic information, and we used it in our context to filter all locations in Vienna labeled as pubs or bars, based on the assumption that these would be places where one could also drink a beer. In the next step, we filtered and sorted these locations by postal code, resulting in a substantial dataset from which we could extract subsets to create graphs for our problem. One of these subsets is illustrated in Figure **??**.
The subsequent step involved calculating travel distances between the pubs and bars, for which we used the Google Directions API in conjunction with the coordinates obtained from OpenStreetMap. One minor challenge we encountered was that the names of pubs and bars on Google Maps did not always match their names on OpenStreetMap, making the coordinates a more reliable and consistent metric for comparison.

This subset selection process was easily repeatable, allowing us to test our algorithms on a total of 20 graphs in two different configurations. In the first configuration, we adhered to the specifications provided in the exercise description and selected 3 pubs or bars per district. In the second configuration, we aimed to challenge our algorithms further by selecting 10 locations per district, thereby increasing the overall graph size.

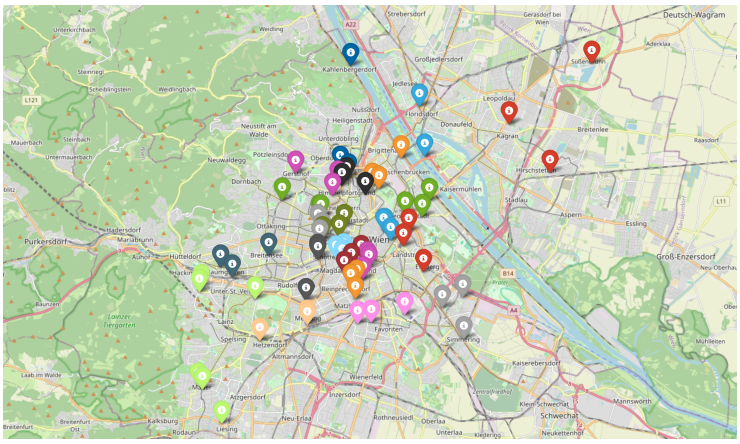

Figure 1: Subsets of pubs for Graph Creation

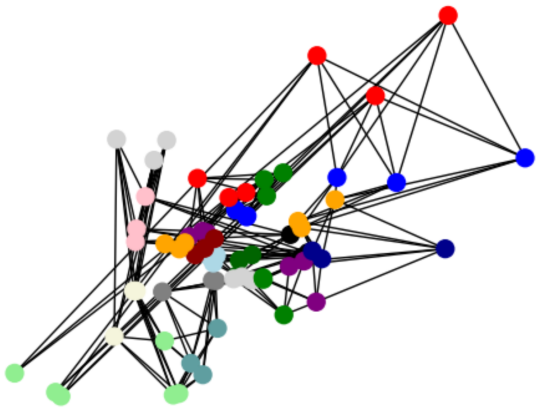

Figure 2: Optimized Graph

2.2 Problem specific Optimization

The following optimization was particularly relevant for the Ant Colony System, one of the algorithms we utilized. For the Genetic Algorithm, the underlying

graph structure was less crucial, as it primarily operated on permutations of the pubs and bars rather than on the graph itself. However, for the Ant Colony System, the problem could be simplified to a basic shortest path computation by introducing two artificial nodes: one start node, connected to the pubs and bars in the 23rd district, and one end node, connected to the pubs and bars in the 1st district. Additionally, we removed all edges that connected pubs and bars between non-consecutive districts in our predefined sequence, as our ants were restricted from traveling along those paths. This significantly reduced the time required for graph generation and simplified the problem instances.

3 Chosen Algorithms

3.1 Ant Colony System

After conducting research on existing Ant Colony Optimization frameworks, we did not find any that were sufficiently appealing, so we decided to implement the version explained in our lecture. Following several tests, we ultimately settled on the Ant Colony System described in the slides and proposed by Dorigo and Gambardella.
The key components of this implementation are as follows:

Pheromone Update Rule: The pheromone update process can be divided into two main categories. The first category deals with local pheromone evaporation, which is triggered each time an ant uses an edge in its path. This mechanism makes the edges less attractive the more frequently they are used, encouraging ants to explore alternative routes. The second category is the global pheromone update, which in our implementation only reinforces the best path found so far, thereby directing the search more intensively toward the area around this optimal route.

Transition Rule: Unfortunately, the exact formulas were not detailed in the lecture slides, which posed some challenges in defining the transition rules for the ants. In our final version the hyperparameter Q, dictates the probability of an ant taking a random next step, as opposed to always choosing the step with the highest probability p_{ij}^k. This randomness is again weighted by p_{ij}^k for each possible step. Overall, this method increases the likelihood of an ant selecting the current optimal step compared to a random one, and it makes the decision-making process adjustable through the hyperparameter. The following formulas are employed:

$$p_{ij}^k = \frac{\tau_{ij}^\alpha \cdot \eta_{ij}^\beta}{\sum_{l \in J_i^k} \tau_{ij}^\alpha \cdot \eta_{ij}^\beta} \tag{1}$$

$$j = \begin{cases} \arg\max_{j \in J_i^k} p_{ij}^k & q < Q \\ \text{select } j \text{ with probability } p_{ij}^k, \forall j \in J_i^k & q \geq Q \end{cases} \tag{2}$$

$$\tau_{ij} \ldots \text{ amount of pheromone deposit on edge } e_{ij} \text{ initially 1} \tag{3}$$

$$\eta_{ij} \ldots \text{ desirability of edge } e_{ij} \text{ in our case } \frac{1}{w_{ij}} \tag{4}$$

$$w_{ij} \ldots \text{ travel time in seconds from location } i \text{ to } j \tag{5}$$

$$q \ldots \text{ random number between 0 and 1} \tag{6}$$

Edge Weight Normalization: One test we conducted involved normalizing the edge weights using a Min/Max Scaler. The underlying idea was that, because the edge weights were quite high—since travel times are given in seconds—they might have a different influence compared to the set pheromones. To, so to speak, level the playing field, we normalized the edge weights to ensure that both contributing probabilities were within a similar numerical range. Ultimately, the results of this test were only marginal, which is why we included them only in the appendix.

3.2 Genetic Algorithm

The second algorithm approach we choose to implement was a genetic algorithm. The information we found online, as well as the lecture slides seemed to indicate that this approach would be very well suited to the task.
To create an appropriate version for our task we implemented the algorithm from scratch. The algorithm consist of the following parts:

Initial Population: The first step is to generate an initial random population. In this case it will consist of N random routes that follow the predefined sequence of districts. These routes are generated via a random choice of bar from each district.

Fitness Function: The fitness of each route is calculated by getting the inverse of the time needed to complete each route.

Selection Process: We followed the roulette wheel approach for the selection process. This wheel takes the ratio of each routes fitness compared to the population fitness. This makes it possible for worse routes to also get selected and thereby ensures diversity in the population which will in turn protect against getting stuck in local minima. Each route maybe selected multiple times and will be subsequently used as a parent for the next generation.

Crossover: The crossover defines the mixing of the two parents genes in two new children. There is a 50% chance for each bar to be swapped in the children. Meaning we follow a uniform crossover variation.

Mutation: The mutation was implemented by generating a random mask across all children that would mutate each route if a certain probability was reached. The mutation was then executed by selecting a random district and randomly choosing a new bar within it.
All of these parts are combined and looped over for several generations creating the final genetic algorithm, that should be quite effective in solving the TSP.

4 Results

To compare the results for both algorithms we defined two metrics to look at:

- Epochs needed to find solution
- Relative runtimes compared to problem size

4.1 Setup

We defined different hyper parameters for both algorithms which might have some influence on the final results, however this will be mostly ignored in the analysis, as we cannot guarantee any optimal selection of parameters.
For the ACO we used the values defined in table 1 and for our GA the values defined in table 2.

Params	**Values**
Epochs	100
Ants	100
Pheromone Deposit	1
Evaporation Rate	0.01
Alpha	1
Beta	1
Q	0.5

Table 1: Hyper parameters for ACO

To thoroughly test each implementation 20 different graphs were created with randomly selected bars in each district. Half of these graphs contain the specified three bars per district and half contain ten. Both implementations will be used for each metric and can be seen as a comparison between both algorithms in a varied problem space. To remove luck based results each graph was run by each algorithm using 100 different random seeds. This should result in an accurate depiction of results.

Params	Values
Epochs	100
Population	500
Crossover	0.8
Mutation Percentage	0.05

Table 2: Hyper parameters for GA

4.2 Epochs needed to find a good solution

Figure 3 shows the results for the ACO running on graph 8. This graph contains 3 bars per district. The most obvious results that are directly visible from this graph are the extreme increase in performance over the first ca. 5 epochs, which lead to an initial good result, about 3% above the actual minimum. From there however the results start heavily varying and the improvement becomes much slower, with one random seed taking until epoch 80 to find the optimal solution. In contrast to this figure 4 shows the same graph for the GA. The values here take about double the amount of epochs to converge to the initial good solution the ACO achieved, however from there on the results do not start to spread out and stay relatively close together. The improve also does not stagnate nearly also heavily as for the ACO and the optimal route is found between epochs 20 and 30, as compared to the 15-50 for the ACO.
These initial results lead us to believe that in a larger search space the GA might have a significant advantage when it comes to finding the optimal solution but could also take a lot longer to find a good solution. It also seems plausible that randomness has a much higher influence on the ACO, as shown by the large spread of evolutions in our results.

Looking at the results for the larger datasets should help confirm or deny the suspicions, as these behaviors would be exacerbated.
Figure 5 shows the results the ACO produced on graph 18 and it seems quite obvious that our thoughts on the ACO are confirmed. Most of the single runs converge to a very good solution extremely quickly, taking only about 5 to 10 epochs. However no run manages to find the optimal solution and the final spread of different routes that is captured is quite large.
Figure 6 shows the same for the GA. These runs all stick quite close together and all mange to find the optimal solution by about epoch 40, except for one outlier. But to get to an initial good solution where ACO took between 5-10 epochs the GA needs about 20.

4.3 Relative runtimes compared to problem size

To compare the relative runtimes we measured how long each epoch took to run and by adding these up and averaging them for each graph we can get a pretty good comparison. This will also give us a good reference when looking at the results for the larger dataset.

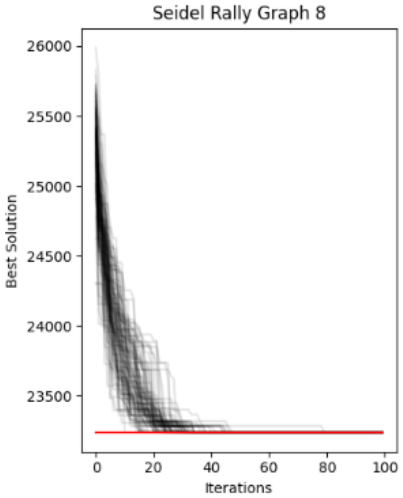

Figure 3: Results on base for Ant Colony Optimization

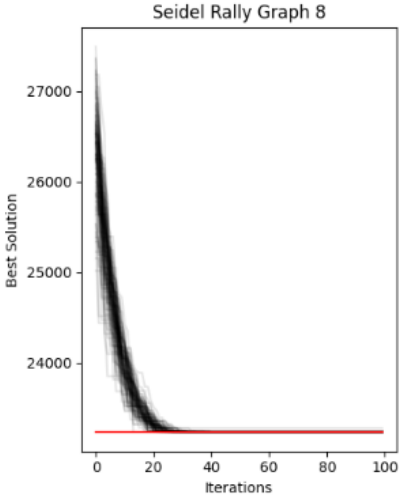

Figure 4: Results on base for Genetic Algorithm

Table 3 shows the average training time of a single epoch in seconds for each graph, as well as the overall average for the small dataset. Here we see that the GA on average trains a single epoch about three times faster than the ACO. This could have a few reasons, but the main one we can think of is that the amount of data that is changed within the GA is much smaller than the ACO, which is also why the spread generated in training is a lot smaller.
We would assume that this trend could be slightly increased when enlarging the search space.

	Ant Colony Optimization	**Genetic Algorithm**
Graph 1	8.96	3.22
Graph 2	8.95	2.83
Graph 3	8.91	2.85
Graph 4	9.82	2.85
Graph 5	10	2.79
Graph 6	8.82	2.79
Graph 7	8.76	2.79
Graph 8	8.81	2.82
Graph 9	8.75	2.79
Graph 10	8.71	2.77
Average	**9.05**	**2.85**

Table 3: Timed results for base dataset

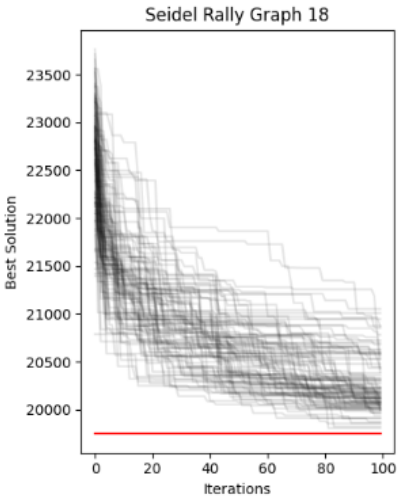

Figure 5: Results on enlarged dataset for Ant Colony Optimization

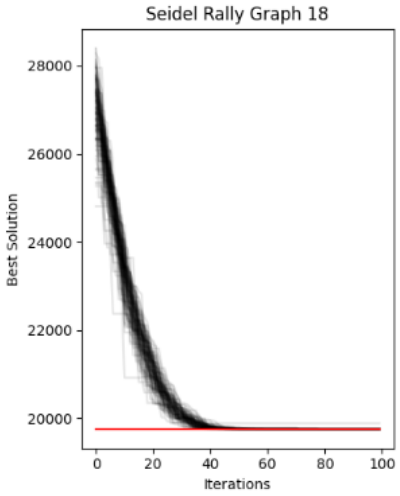

Figure 6: Results on enlarged dataset for Genetic Algorithm

Table 4 shows the same results as before, for the larger search space. We can see an increase of about 1.5 seconds per epoch for the ACO and an increase of 0.31 seconds for the GA. These values correspond to an increase of 16.24% and 10.88% respectively. Which means that our suspicions are confirmed, but when seeing that the search space increased from 3 pubs per district to 10 the increase in time seems quite decent for both algorithms.

	Ant Colony Optimization	**Genetic Algorithm**
Graph 11	10.46	3.03
Graph 12	10.51	3.02
Graph 13	10.56	3.03
Graph 14	10.54	3.06
Graph 15	10.46	3.03
Graph 16	10.47	3.09
Graph 17	10.48	3.73
Graph 18	10.51	3.34
Graph 19	10.46	3.15
Graph 20	10.52	3.12
Average	**10.5**	**3.16**

Table 4: Timed results for enlarged dataset

5 Conclusion

Overall both algorithms seem to do their job quite well. Both achieve their respective goal in different ways and with different efficiencies however. The ACO algorithm is quite a bit slower in calculation than the GA but finds a good solution very fast. The GA takes about double the amount of epochs for this but manages to find the optimal solution almost every time which ACO struggles with. In theory we could harness the benefits of both these algorithms and use an ACO to generate a good initial population for our GA, but seeing as the time per epoch is so much higher for the ACO overall this seems not that useful.
Overall we think it is fair to say that both algorithms have value and should be chosen based on the task at hand.

6 Appendix

6.1 Supplementary Results

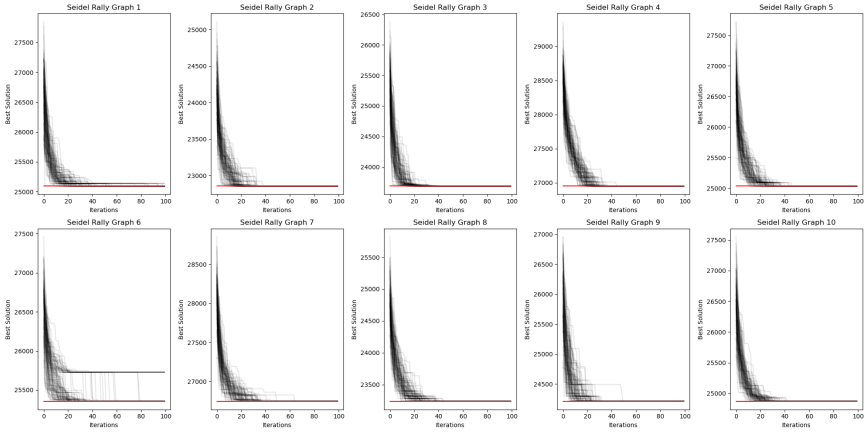

Figure 7: All Runs of the Ant Colony System on Graphs 1-10

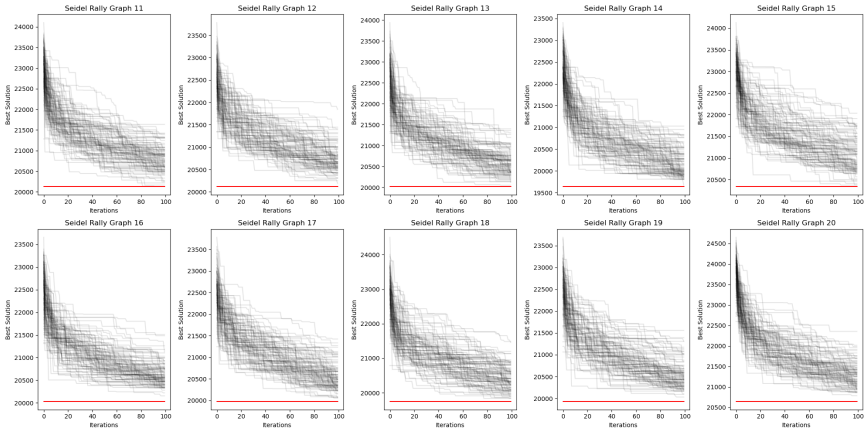

Figure 8: All Runs of the Ant Colony System on Graphs 11-20

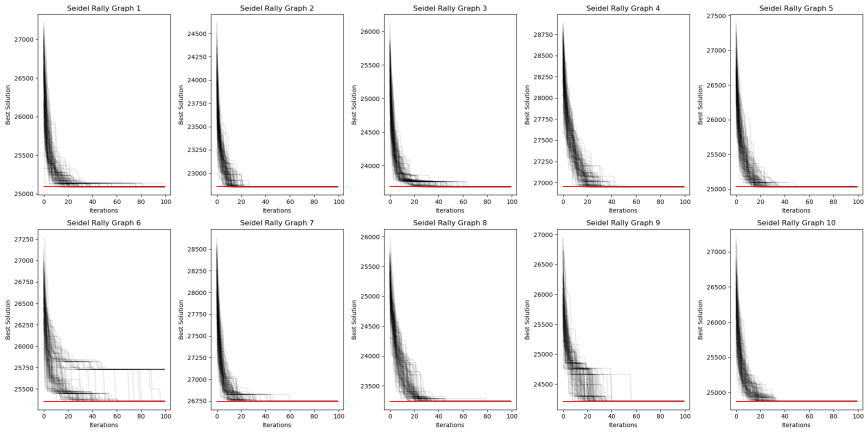

Figure 9: All Runs of the Ant Colony System on Graphs 1-10 with Edge Weight Normalization

Visit our GitHub: github.com/yourusername/yourrepository

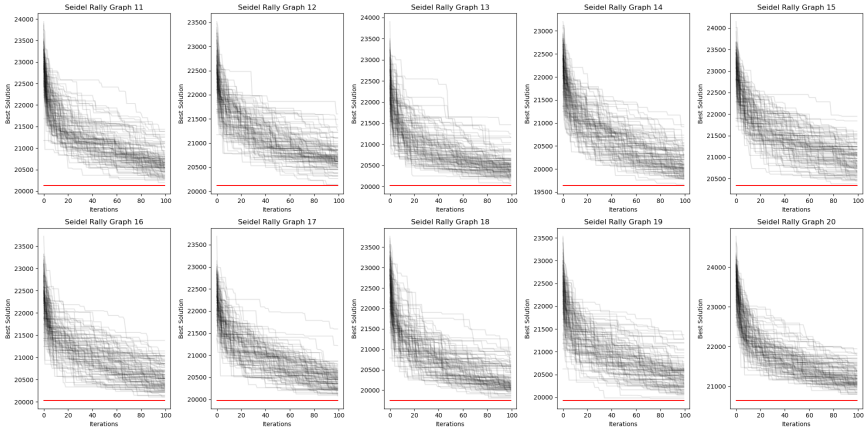

Figure 10: All Runs of the Ant Colony System on Graphs 11-20 with Edge Weight Normalization

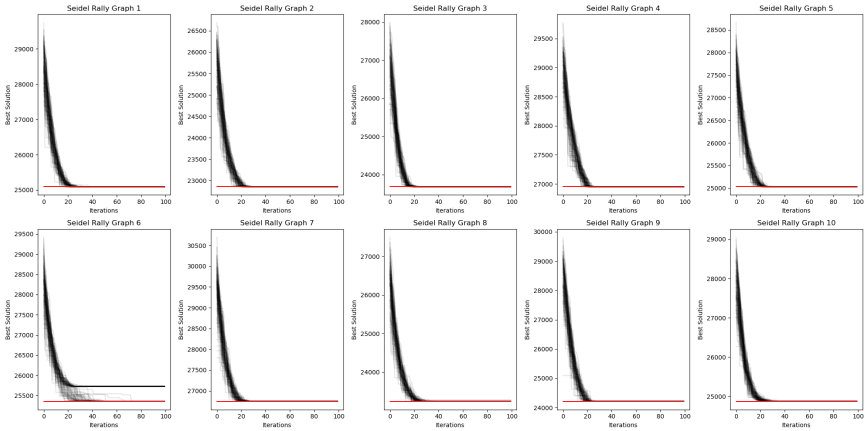

Figure 11: All Runs of the Genetic Algorithm on Graphs 1-10

Visit our GitHub: github.com/yourusername/yourrepository

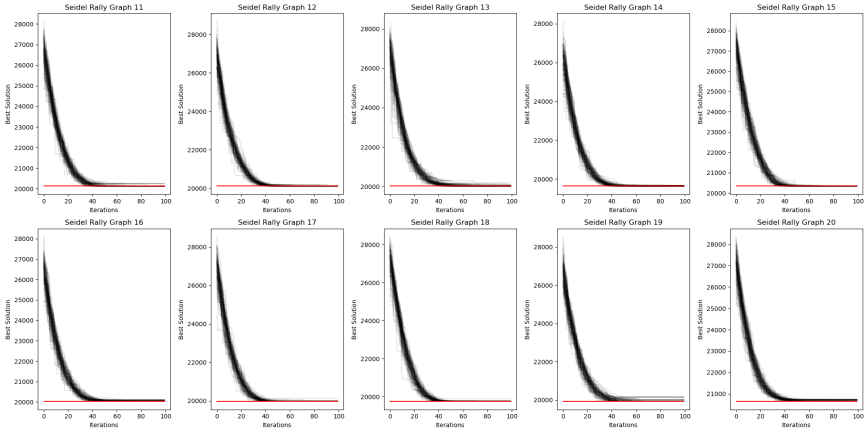

Figure 12: All Runs of the Genetic Algorithm on Graphs 11-20