technische universität
dortmund

Masterthesis

Customizable Roundtrips with Tour4Me

Metaheuristic Approaches for Personalized Running and Cycling Routes

Lisa Salewsky

17.07.2024

Supervisors:
Prof. Dr. Kevin Buchin
Mart Hagedoorn, M. Sc.

Technische Universität Dortmund
Fakultät für Informatik
Algorithm Engineering (LS-11)
http://ls11-www.cs.tu-dortmund.de

Abstract

The ability to create customizable roundtrips is crucial for many people who enjoy outdoor activities. In this work, the `Touring Problem` is used to find good routes that adhere to user-preferences. Finding and creating good and customizable routes for various outdoor sports that are presented in a user-friendly application which works in real time is the key problem this thesis addresses.

Much research has been done for shortest path problems and finding the quickest or fastest route, however, previous work lacked to address roundtrip paths. For these problems, a shortest or quickest route is not enough. Instead, a multitude of constraints has to be considered while trying to find a relatively good solution.

In order to generate a user-friendly app that can compute roundtrips in real time for outdoor activities, an interface has been created. The web interface was structured to be easy to understand and use while simultaneously offering a good overview of all parameters for customization as well as all calculated tours. Additionally, different metaheuristics, i.e. Ant Colony and Simulated Annealing, have been implemented to solve the `Touring Problem`.

The resulting application is easy to use and offers a selection of algorithms to calculate the desired tours. Several tests were conducted, which demonstrated that the metaheuristics return results that match the user's preferences. All algorithms can be configured to run in real time, yet an option to allow for longer run times exists. Thus, the app is user friendly and generates solution tours for the `Touring Problem` in real time.

Contents

Chapter 1

Introduction

Whether exploring peaceful forest trails on a weekend hike, cruising scenic routes on a bike or embarking on an enjoyable run along riverside paths, many people typically opt for tours tailored to their personal preferences. Outdoor activities are pursued for leisure and often not meant as a means of transportation from one location to another. On the contrary, in most cases, roundtrip routes that start and end at the same point are preferred. For some, these roundtrips mean beginning a run or cycling tour at home and concluding back there. For others, returning to the point of departure, where their vehicle is parked, after a hike is necessary.

Outdoor activities cannot only be fun but also offer many inherent benefits: For overall health [36, 41, 47], the cardiovascular system [34], as a measure against many different diseases [36], and for social [33, 35, 49] and psychological benefits [6, 10, 45, 49]. Furthermore, touristic cycling can prove beneficial through increasing tourism for a city by offering attractive roundtrips for outdoor activities to tour the surroundings [7]. Another important benefit involves reducing the influence of traffic on the environment [24], when more people start to enjoy outdoor activities and thus consider taking the bike, a skateboard or other means instead of a car for commuting more often.

Creating a user-friendly web app for generating customizable roundtrips could encourage more people to engage in outdoor activities. Such an app not only simplifies route planning but also suggests scenic paths, thereby promoting participation in outdoor sports. Whether planning holiday trips, several-day tours or discovering new routes for exercise routines, the option to generate personalized tours underlines the versatile usefulness the app can provide. Especially when considering that the option to fully customize the generated tours allows the app to assist in route planning for all kinds of activities and trips.

While algorithms for shortest paths are an important and well-studied part of computer science, the topic of generating customizable roundtrips remains relatively unexplored. Many different approaches to optimize for short travel times, fast and easy combination of public transportation options or simply the shortest path in terms of tour length, many

algorithmic options are available: Improved routing algorithms from A to B can help reduce travel times by car, bicycle or even on foot and thus, there are many different solution strategies for the shortest path problem [11, 17, 21, 43, 50]. Furthermore, considerable work on optimizing public transportation [5] and managing traffic jams has been done [14, 15]. Examples include Dijkstra (uni- and bidirectional) [43, 50], A* Search (also uni- and bidirectional) [43, 50], Greedy algorithms [50], Branch-and-Bound algorithms [30], the Bellman-Ford-Moore algorithm [11], and many more [16, 21, 43].

All of these approaches have in common that they always try to find the shortest or quickest path between two different points. However, for planning round trips to train towards a specific fitness goal none of these algorithms are applicable. For these cases, creating routes of precise lengths is essential for effectively monitoring progress. Even without athletic objectives in mind, controlling the route length ensures individuals avoid overexertion and returning too soon. For these scenarios, shortest path algorithms are not suitable as the shortest path from a starting point back to the same point would always be to never leave. Therefore, a different approach is needed to create these types of routes: A modified version of the `Arc Orienteering Problem` (AOP) [44], detailed later in this thesis (see Section 2.1), which will be referred to as the `Touring Problem`, in accordance with Tour4Me [9] is required and forms the basis for this thesis (see 1.1.1).

Finally, the computational complexity is an interesting part of this problem. Since the calculation of a roundtrip with additional customizable parameters is a version of the AOP, the computational complexity will be at least as hard. Thus, the customizable AOP will be at least NP-hard [2]. Additionally, Gemsa et al. [22] present a proof of the computational complexity of their Simple and Relaxed Jogging Problems, which solve a similar question as this thesis. The authors show NP-hardness by reduction of Hamiltonian Cycle to the optimization problem corresponding to their original problems.

1.1 Related Work

Much research has been done for shortest path algorithms and their optimization (e.g. [11, 17, 21, 43, 50]). However, comparatively little work has been done on the more complex problem of finding a round trip with several additional conditions [22]. While there are a few tools that can be used to calculate round trips, most of them only focus on cycling or create a very limited set of trips that do not satisfy the needs of most people, or both. Some examples for these tools are RouteLoops[1] and RouteYou [2] which do not address the full range of potential trips that individuals may require.

Adding new options for user inputs that enable a higher degree of customization can vastly improve the usability of a tool. The usefulness is not only determined by the

[1] https://www.routeloops.com/, last accessed: 22.03.2024
[2] https://www.routeyou.com, last accessed: 22.03.2024

implemented algorithms, but also by the interface, the data used, and the selection options presented to the user.

As both RouteLoops and RouteYou are commercial programs, obtaining any details about the used algorithms, heuristics, metaheuristics or even the language they used for programming these solutions was not possible. All gathered information is collected from exploring the functionality of the two tools through manually testing them and reading both the general information, and the FAQ pages provided by the websites (for further reading see 1.1.2).

1.1.1 Tour4Me

The tool which this thesis will be based off, Tour4Me[3] [9], incorporates some of the mentioned customization options in the web interface. The app offers the option to choose the favored surface (asphalt, gravel, pebblestone, etc.) as well as make selections about preferred path types (cycleways, footways, tracks, etc.). Furthermore, users can also mark certain types of both the surface and the path types as undesirable rather than just keeping them neutral or marking them as preferable. This feature allows for much more customization. What the tool does not incorporate yet, is the option to make selections about the preferred elevation and steepness or route complexity. However, the tour can be optimized for a circular route by maximizing the *covered area* of the tour. Covered area describes how much area is surrounded by the calculated tour. A trip with many crossing parts is less desirable than one that is more round. Therefore, maximizing the surrounded area accounts for overlapping parts as well as smaller circles within the tour or generally less round shapes. An example is illustrated in Figure 1.1.

Figure 1.1 shows the outlines of two possible tours, where the black outline is more round than the red one. Here, the covered area of the black tour is larger than the one of the red tour, since in the calculation, all smaller areas are accounted for regarding their overlap as well as the direction of the turns (clockwise or counterclockwise). Thus, the many overlaps of areas and the crossings of path parts reduce the covered area value. Using this calculation, the red tour has a lower covered area, making the black one more desirable.

Tour4Me implements a solution for the `Touring Problem`, which is used to describe the task of finding appealing and ideally interesting roundtrips. To achieve a relatively good solution, two factors are taken into consideration. First is the total profit, that can be collected within the given length restriction for the tour. The second factor is an additional quality function that assures for a relatively round tour by maximizing the area that is surrounded by the created roundtrip. Tour4Me presents a selection of four different algorithms to calculate the tour as well as some additional customization options.

[3] http://tour4me.cs.tu-dortmund.de/, last accessed: 18.04.2024

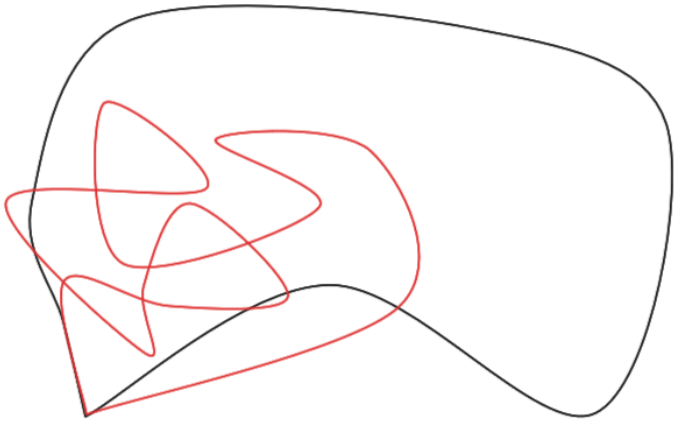


Figure 1.1: An example sketch showing the outlines of two different tours.

The offered choices include a Greedy Selection approach, Integer Linear Programming, MinCost with Waypoints, which are intermediate points used to calculate shortest paths between them (see 1.1.2 for more details), and Iterative Local Search (ILS) [9].

The Greedy Selection [9, 50] is the simplest algorithm out of the four and only ensures that the chosen route is a roundtrip. The resulting path is built by iterating over the valid edges and picking the most profitable of these until the cycle is finished or no candidate is left. A valid edge is determined by checking whether the start- and endpoint s can still be reached if that selected edge is picked next. Edges that yield the highest profit and are valid are chosen at every node [9, 50].

For Integer Linear Programming [9, 25], the `Touring Problem` must be stated in an appropriate form. To do so, a single instance can be encoded as $\mathcal{I}(G, w, \pi, B, v_0)$, containing the Graph G, edge costs w, the profit function π, the budget (length restrictions) B, and the starting (and end-) point v_0. Given this encoding, cycles $P = (v_0, ..., v_i, ..., v_0)$, which are always at most of length L, can be built. For the current definition, a few additional variables can be introduced to encode whether or not an edge is part of a solution (and how many times this edge occurs), whether or not an edge is the k-th edge of the solution, and whether or not a vertex is the k-th vertex of a solution. Using these variables, constraints can be built to describe the desired behavior of the algorithm.

The MinCost algorithm [9, 22] requires *Waypoints* since the basic algorithm is designed to solve shortest path problems. Without these added Waypoints, the underlying computations would always result in never leaving the starting position. Even though this algorithm is not originally meant to solve roundtrip problems, the implementation takes into account the cost and profits of edges to create a solution tour, which makes this

method more suited to the task than a simple Greedy search. To create an optimized tour, the inefficiency of paths must be measured. This optimization is done by calculating the quotient of the edge costs and the profit the edge yields. Using this inefficiency, a ring of candidate points R_s surrounding the starting point s can be calculated. From these candidates, new rings R_r with the same requirements can be calculated. The solution path is then obtained by selecting all those circles that intersect with R_s. In order to guarantee the best solution, all possible combinations are calculated and the most favorable of these solutions is returned [9]. Further details can be found in the original paper by Gemsa et al. [22] which offers a Greedy Faces approach as well as two variants for the Partial Shortest Paths algorithm, of which the 2-via-routes option was implemented in Tour4Me.

Building on the solution that was constructed using MinCost, Iterative Local Search [9, 32, 46] can be applied to improve the found tours. ILS is split into two main phases: the removal step and the improvement step. The algorithm starts with a full roundtrip and then removes partial paths P from the current best solution S (the removal step). After that, the previous tour now has a gap, which must be closed by iteratively adding new vertices and edges that improve the solution profit while always staying within the given budget (the improvement step). When adding new edges to the solution that close the path, both the profit as well as the length of these paths must be considered. Since the roundtrip has a given maximum length, the profit must be maximized while keeping track of and never exceeding this length constraint. The best solution is constantly improved until the user-selected time limit is reached.

Searching for viable edges is performed using a depth first approach. Thus, bounding the maximum depth of this step can drastically speed up the algorithm. This speedup then results in more iterations of removal and improvement being possible within the given time limit.

1.1.2 Roundtrip Paths

There are a few tools and some research that deal with the construction of roundtrips. Some of the papers specifically focus on running routes ([22, 37]) or tours tailored to cycling ([20, 46]). RouteLoops and RouteYou are two commercial tools that offer an interface to calculate tours, but do not have many customization options. Neither has there been much research on roundtrip generation, nor are there any tools that offer customizability. Aside from these few approaches for roundtrip calculation, some ideas to improve the experience and training effect of running, how to assist various different sports with technology, and even approaches on how to determine pleasant surroundings of paths have been developed, but these approaches have not been combined into a single application yet.

Computing Running Routes

The problem of calculating good running roundtrips is not new. In addition to the commercial applications, there are research papers on this subject as well. One of these papers by Gemsa et al. [22] presents two approaches to handle the new routing problem labeled `Jogging Problem` by the authors, which is split into two variants: one being the simple version that only aims to build a cycle containing the starting point s and with the desired length. The other variant is a more complex version that allows for some flexibility regarding the length of the final tour during optimization, which is named `Relaxed Jogging Problem` [22]. This relaxation allows for considering more factors to also optimize for the resulting shape, the area surrounding the tour, and/or the simplicity of the path.

The second problem is chosen as the one to optimize, since the relaxation enables the addition of conditions other than just the length of a tour. For solving this selected problem, two different ideas are proposed. The first approach – *Greedy Faces* – is based on the idea of extending previous cycles. The algorithm starts with a cycle containing the starting point s that can be selected by the user. This roundtrip can then be extended to gradually approach the length specified by the user. The second algorithm is named *Partial Shortest Paths* and uses Waypoints or *Via-Vertices*. These Waypoints are a number of new points that can be connected using shortest paths. When the via-vertices are connected with each other and the start, they form a roundtrip.

For both algorithms, the authors measure the badness of paths, the number of edges that are shared, as well as the number of turns. The badness is used to take the additional constraints into account. To reduce the possibility of having a roundtrip which turns at the end and uses all paths twice, the number of shared edges has to be minimized. Otherwise, this construction would effectively form a simple U-turn tour (turning by 180 degrees). The number of turns corresponds to the complexity of the tour and is measured by a percentage of doing a full U-turn. Gemsa et al. define the point between two edges as a *turn* if the angle is larger than or equal to 15 degrees and less than 180 degrees. These turns can then be used to determine the complexity of a tour: More turns meaning a more complex tour.

Computing Cycling Routes

For cycling, there are some papers ([20, 46]) discussing ideas for generating cycling tours. Ehrgott et al. [20] discuss a bi-objective model that takes travel time and *suitability for cycling* into account. Suitability is defined as a combined measure of objective factors, containing but not limited to the volume and speed of traffic on the roads, which can impact the safety of these path segments. However, subjective values like the individual fitness level are not considered for this implementation. All objective values that are regarded significant for the suitability of the tour are accumulated into this single measure, so that there are only two values to optimize at the same time.

The authors offer a solution for the issue that many of the values can have a different level of importance for different people. While some people might not want hilly routes at all, others could enjoy the challenge a certain steepness presents. Because of this difference in basic preferences, Ehrgott et al. chose to offer a choice set of several alternative routes, from which the user can pick the one that works best with their personal preferences.

This approach takes several different factors into account, but does not offer any means to influence the importance specific factors have on the generated routes beforehand. Furthermore, the presented ideas are focused on shortest path applications, not on roundtrips.

Verbeeck et al. [46] concentrate on cycle trips in their paper, which also builds a foundation for the ILS algorithm used in Tour4Me (see Section 1.1.1). The authors build a `cycle trip planning problem (CTPP)` as an alternative version of the Arc Orienteering Problem (see Section 2.1 for further details). The initial idea is to use a metaheuristic approach of ILS to build roundtrips that optimize the profit of the trip. Since the AOP is already NP-hard and the CTPP has an even higher complexity, attempting to solve this problem with an analytic, exact algorithm will not be feasible in terms of time constraints. Because of this complexity, the authors developed two approaches – a branch-and-cut algorithm and a metaheuristic method – to try and solve their CTPP quickly. The branch-and-cut approach turned out to return results on smaller sets within a reasonable time. However, the algorithm is too slow for larger problem space instances.

Therefore, Verbeeck et al. developed the ILS approach which can be split up into three phases: initialization, improvement, and selection. During the initialization phase the implemented algorithm gathers a first set of possible solutions by using the insert move that aims to find a path with the highest score. The insertion phase starts with every arc that leaves the starting point and builds a maximum-profit path until a feasible solution is obtained. This step is done for all possible starting points, so several solutions are created. Then, the results from the previous step are optimized in the improvement phase. To do so, a part of the solution is removed during every iteration. Next, the newly constructed gap between the two nodes where the path was removed is closed, using the same insert move from the initialization, thus improving the previous solution. This process then iterates over the whole tour until the removal encounters the end vertex (which is equivalent to the start vertex) again.

Using this approach, the authors were able to create a path within the given time constraints, build a roundtrip, ensure that its length lies between a maximum and minimum value and optimize its profit. They also stated that their ideas can be used as "building blocks" [46] for further development. The fact that Verbeeck et al. stress, is that vertices can be visited multiple times (except the start vertex), however arcs and their complements (if existent) cannot. Thus, they enforce trips to not take the same paths twice.

They conducted several benchmark tests for their implementations, but the code is not publicly available. Furthermore, there does not seem to be any way to try out the existing

implementation and assess how many parameters are used, which of them can be changed and how much customization is overall possible. The fact that the authors do not allow passing an arc twice also limits the options to select a preferred tour shape that might include those that simply run one way and have a U-turn at the end.

RouteLoops & RouteYou

RouteLoops and RouteYou are two commercial programs that can be found and used online. However, neither the code itself nor information about the implementation are accessible.

RouteLoops has two text fields for entering the starting point and the length of the trip. Aside from that, no customization is possible. The interface has a few features to show more information about the route. After the calculation, distance markers or elevation can be displayed. However, these outputs cannot be used as inputs to get a route with as little elevation as possible. The page claims that the difficulty of a route can be shown for tours that are placed within the United States. However, even when creating a route in the United States, a display of the difficulty could not be replicated. RouteLoops also does not actually create loops but rather picks a route that has a high value (for example with a river in a park) and lets the user run along that path, turn around at the end and run back the same way.

To create a roundtrip, some Waypoints are used. These points can be removed or more can be added when editing the tour. Between the Waypoints, the shortest path is created to connect them.

RouteYou offers several different user input options that will return varying results, however, picking the same option again will also give different results every time. Here, the roundtrips are more round than with RouteLoops, but again, elevation or difficulty are not taken into account. Even though both do offer the possibility to edit the returned roundtrip, this editing changes the length of the route arbitrarily. Furthermore, the user cannot specify directly what type of surface, surroundings, etc. are preferred.

1.1.3 Other Running Related Research

Aside from the few directly related papers and applications, some general research regarding running with technology has been done. Jensen and Mueller [27] focus on the utilization of interactive technologies that can be used to monitor or enhance the performance of athletes. They are particularly interested in improving these gadgets and apps to enhance their usability. In their paper, they discuss the current state of different technologies and propose the following three key questions as ideas of what aspects to focus on for further improvement: The first question “How to interact” focuses mainly on the question how interaction with any app or gadget can be designed so it will not hinder the actual

activity of running. The second question "What information" aims to enhance the types of information presented to the user while running (for example to change the running style mid-run) While the third question "When to assist" addresses the timing aspect of any kind of assistance during a workout. They strive to find suggestions on what to focus on when trying to produce apps or gear for runners.

1.1.4 Apps That Assist with Sports

Aside from Tour4Me, RouteLoops and RouteYou, another prototype for running route recommendations has been developed. Other papers, such as one by Loepp and Ziegler [31], used the Partial Shortest Paths algorithm from the idea Gemsa et al. presented [22] to build a recommendation-based app. However, they tried to incorporate more criteria, such as elevation levels or information about the surroundings, allowing users to pick from a variety of options when generating personalized tours. Furthermore, the authors added a feature to use routes by other users, but their subsequent survey revealed that none of their users were interested in that particular feature.

The customized tours that could be generated were well-received, perceived as high-quality, and the difficulty was considered low by the users. This app is only a prototype and was never fully expanded into a full-fledged product. Currently, the app runs only on smartphones that use Android.

In the corresponding paper [31], Loepp and Ziegler also express several issues with and shortcomings of existing apps. Some of these problems have already been identified in the introduction of the two websites RouteLoops and RouteYou (see 1.1.2). They also stress that most research either concentrates on shortest paths or on the assistance with the training itself rather than finding a good route.

Apps like *Runtastic*[4], *Sportractive*[5] or *Strava*[6] are designed to help runners track the tours they have already run. These apps measure pace, position, altitude, and several other stats during a run, to then provide feedback to the user. Planning a route is not one of the features these apps offer. Even apps designed to assist with the training and creating a plan such as *Trainingpeaks*[7] or *SportTracks*[8] do not offer a feature to create routes or roundtrips based on a set of preferences [31].

A German app that is meant to provide suitable routes for a variety of different outdoor sports *Komoot*[9] does offer a route selection. However, only tours other users have planned manually and added can be selected. No customization or automated route creation is

[4] https://www.runtastic.com/, last accessed: 22.03.2024
[5] http://sportractive.com/, last accessed: 22.03.2024
[6] https://www.strava.com, last accessed: 22.03.2024
[7] https://www.trainingpeaks.com/, last accessed: 22.03.2024
[8] https://sporttracks.mobi/, last accessed: 22.03.2024
[9] https://www.komoot.de/, last accessed: 22.03.2024

offered. As Loepp and Ziegler pointed out in their user study [31], user feedback, their preferences, and the option to customize were crucial features for a route generation app.

1.1.5 Other tour optimization ideas

Aside from approaches to calculate good roundtrips and the various sports-assisting apps and technology, there is another point that is related to generating desirable tours. Some papers (e.g. [3]) discuss the question of how to find scenic routes, which aspects impact how appealing a route is, and how the availability of more panoramic routes can influence the users' decisions. To gain some understanding of what is considered scenic and what features can lead users to take longer tours into account, Alivand et al. [3] created a route choice model. Their application calculated and displayed the shortest path from a source to a destination and additionally a set of routes that were longer (considering their length or the travel-time or both) but had more panoramic views along the path. The points they used to decide what can be considered a scenic view were gathered from a set of geo-tagged photos and travel blogs.

From their testings, users were happy to take detours that were on average 90% longer than the fastest tour. The paper focused on touristic trips from a start to a specific destination, but their findings can easily be translated to other modes of travel, including roundtrips.

1.2 Goal and Methodology

As stated before, existing tools that can calculate roundtrips leave out certain data like the elevation of nodes or path types of edges. The absence of these information impacts the quality of the created routes for individual users or even whole user groups. For example, people who prefer running with little to no elevation might end up with a route that takes them uphill for half of the distance. While this path may be a suitable choice for other users – such as joggers who prefer more challenging routes or those who want to hike and enjoy ascending – the constant elevation gain can be undesirable for beginners. Some people might prefer to choose whether they are running through the city or in a park depending on the elevation level. For these users, the created route would be highly unfavorable, even though the result matches other constraints for what is considered a nice roundtrip. Therefore, giving the user as many customization options as possible without becoming overwhelming can be a crucial point.

The goal of this thesis is to develop a usable application for computing running or cycling roundtrips of (almost) arbitrary length. In this context *usable* means an app that operates in real-time, produces routes of the desired length, and prioritizes paths based on the user's input. To achieve this goal, this thesis is an extension of the already existing

prototype Tour4Me [9] and adds metaheuristic approaches that have been considered the most fitting for this purpose.

First, an interface for testing the new approaches was built. To add user options like the length of the desired roundtrip, as well as other preference inputs, an additional overlay was needed. Based on this interface, different algorithms were added and compared with each other to identify the ones that produced promising outputs.[10] However, the definitions of a high quality result can vary significantly based on the criteria used. An ideal algorithm would be fast, consistently generate a route, and incorporate all user preferences. Achieving all these objectives with a single algorithm is not possible. Therefore, various approaches have been implemented and analyzed according to how well they meet the mentioned criteria.

The implemented approaches include Ant Colony [4, 18, 23, 48] and Simulated Annealing [1, 13, 19, 51]. Both of these metaheuristics have been implemented as a standalone solution as well as in combination with the previous Greedy algorithm, the MinCost implementation, and with each other. After implementing the algorithms, the most suitable algorithms as well as the most fitting parameter configurations for Ant Colony, Simulated Annealing and the other variations had to be determined. The aim for the app was to calculate a high-quality tour for any typical roundtrip request for running and cycling.

In addition to finding suitable algorithms that allow for fast and reliable computation of all typical roundtrips, working on the interface and data used also enhanced the usefulness of the app. Improving the interface and adding more options like elevation data, including more information (for example surrounding tags) have been equally important changes to increase the usability. There are several opportunities to improve the app not only by changing the used algorithms but also through adding user selection options and upgrading the GUI. Enhancing the interface through extending available inputs and sliders to better specify tour parameters, and overall refining the usability constitutes another pillar of improving the app aside from adding more or faster algorithms.

Overall, the goal of this thesis is to build a user-friendly application that can compute roundtrips for various outdoor activities. This thesis focuses primarily on adding customization options and the respective front-end changes as well as the implementation of Ant Colony and Simulated Annealing as two metaheuristic approaches for solving the problem. The resulting web application is able to calculate tours for (almost) any length, take user preferences into account, and operate in real time.

[10] https://github.com/LisaSalewsky/MA-Customizable-Roundtrips-with-Tour4Me, last accessed: 13.07.2024

1.3 Structure

The second chapter covers the essential background information and fundamentals. First, a brief overview of shortest path algorithms and the existing options is presented. Afterwards, the Arc Orienteering Problem, which builds the basis for calculating paths that optimize for other values than their respective length, is presented and described in detail. Then, the fundamentals of the two implemented algorithms Ant Colony and Simulated Annealing are explained.

Chapter three outlines the implemented changes, beginning with the overall architecture of the software and a list of the technologies and languages used. Then the added database and data acquisition, as well as the needed processing are described. Afterwards, the front-end changes, and new parameters are explained. The final section of the third chapter gives of about the implemented algorithms and specifies where changes to the algorithms presented int the given literature were needed. Furthermore, the implemented combinations are described.

In the fourth chapter, several test cases are conducted, and their results presented and analyzed. Several parameter configurations for both implemented base algorithms had to be determined. The effect of the customization options were also examined and displayed. Lastly, the chapter compares all implemented algorithms, noting their quality change over time.

The last chapter summarizes all results, detailing how the set goals were met. The chapter concludes with several ideas for future work, highlighting potential areas for further development and optimization.

Chapter 2

Fundamentals and Background

As stated in the introduction, most routing algorithms focus on finding shortest paths between two or more points. Many of those approaches have been reviewed in surveys (see for example [50]). Additionally, there have been many heuristic approaches, like local search variants [8, 26, 40] or different neighborhood-based ideas [8, 26, 40] that offer faster results in exchange for not necessarily finding the single best solution, but only close approximations. Much research has been done and is still ongoing for these kinds of problems, stemming from the fact that many graph routing problems (for example the `Traveling Salesman Problem (TSP)` [23], the `Vehicle Routing Problem (VRP)` [8, 26] or the `Arc Orienteering Problem (AOP)` [2, 9]) are NP-hard [39].

Finding the shortest path is important in various parts of daily life. Whether the problem is finding the best (shortest, quickest, most convenient) way to commute to work or reach a supermarket by car or bike, or minimize travel time by bus or any other means of transportation. The examples for shortest path problems are numerous. Additionally, shortest paths are not limited to real-world networks but can also prove useful for social networks or any form of digital network [38]. Here, different algorithms can help calculate friend networks or support the routing of data through virtual networks.

2.1 Arc Orienteering Problem

The `Arc Orienteering Problem` is a variant of the `Orienteering Problem (OP)` and forms the central concept for roundtrip generation. Souffiau et al. [44] describe the problem and underlying ideas, as well as basic definitions of the AOP in detail. While many other variants of the OP mainly use the nodes of a graph to determine the benefit, the AOP focuses specifically on arcs – the edges – of the graph. Here, the underlying question is to generate a path that maximizes the profit of the contained edges while staying within the bounds of a maximum length (C_{max}). Each edge has a cost c_{ij} and a profit p_{ij}, which contribute to the overall length and overall profit of the generated path. The maximum

length has to be defined. Furthermore, a start- and endpoint have to be determined. For roundtrips, these two points are defined by the same vertex, so only the starting point s needs to be selected.

In a graph $G = (V, E)$, vertex positions are denoted by v_{ij}. Whether or not an edge is part of a path is given by χ_{ij}. Based on Souffriau et al. [44], the objective is to maximize the overall profit, while adhering to several constraints:

First, both the startpoint and the endpoint have to be part of the resulting path, ensuring that the specified start will always be used. For roundtrips, this constraint can be simplified to only using the first or the last point of a tour. Secondly, the resulting tour has to be fully connected. Otherwise, several independent path fragments could be created without ever resulting in a closed tour. Additionally, all edges and vertices that are part of the result have to be visited. Leaving partial paths out and still adding these to the result is not possible. Furthermore, the resulting length is constrained by a maximum value C_{max}. Lastly, ensuring that no sub-tours are created is another important factor, preventing inner loops from occurring and thus resulting in a single outer loop.

Using these constraints and the base formula (the sum of all profits) to be maximized, the `Arc Orienteering Problem` can be solved by various different algorithms.

2.2 Shortest Path Algorithms

Despite not being directly usable for solving roundtrip problems, shortest paths still form an important background for the rest of the thesis. Shortest path algorithms have been studied extensively for many years. In 1994, Deo and Pang [17] created an overview tree for different types of shortest path sub-classes to give a better understanding of how to systematically classify a certain question into one of these categories. The tree is visualized in Figure 2.1. This visualization gives a first idea of how complex the shortest path problem can be and how many different types of questions arise in different networks and with different goals.

Since the shortest path problem has been well-studied and continues to advance in terms of the quality of the returned paths as well as in optimizing the run time of algorithms, the number of approaches to solve this problem is enormous. The presented tree offers a systematic approach to classify problems of which most fall into one of two categories: they are either single-source shortest paths (SSSP, on the leftmost branch the two bottom-left items) or all-pairs shortest paths (APSP, on the leftmost branch the two bottom-right items).

SSSP The first approach – SSSP – uses a single starting point and aims to find the shortest path between this point and one or all other vertices. These problems are commonly encountered in many everyday routing scenarios. One-to-one paths, which involve

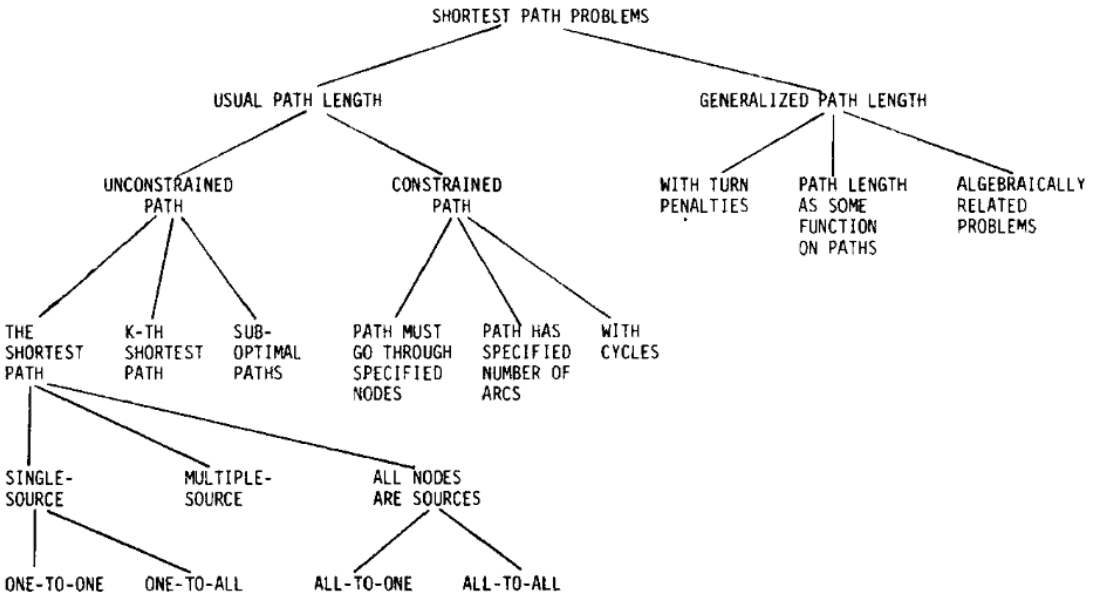


Figure 2.1: This image shows a conceptual tree of different variations of shortest path algorithms, taken from *Shortest-path algorithms: Taxonomy and annotation* [17]

finding the shortest path to a specified destination, have already been discussed in this chapter's introduction (see Section 1) [42]. One-to-all paths can be useful for cases like fire departments or the police, which may need a map of the quickest routes to every place in their jurisdiction [42].

APSP The second category aims to find the shortest paths between all vertices in a graph – either from a specified vertex or from every vertex to every other vertex. This calculation of all paths is essential for transportation networks and similar use cases. All-to-one path calculations can be useful in scenarios where, following an accident, the nearest of all available emergency vehicles – with the shortest path – needs to be determined [28]. For all-to-all paths, many traffic-load calculation problems arise, such as distributing trains along the rail network [12].

Which of these shortest path algorithms performs best typically depends on the type of graph the implementation is used on, the graphs structure, and the specific problem to be solved. A graph can be categorized in different ways: planar or non-planar, directed or undirected, weighted or unweighted, and containing only non-negative weights or allowing negative ones as well. They also can contain cycles or be acyclic, among other categorizations. These different types determine which approaches can be applied and which will yield better results. Some algorithms like Dijkstra can only be used on a specific type of graph without modifications. In this case, the graph needs to contain only non-negative

edges. Others are modified versions, specifically designed to handle issues like graphs with negative edges.

2.3 Heuristic Approaches

In contrast to exact approaches, heuristics do not guarantee for optimal outcomes but can instead improve the runtime of an algorithm. A heuristic is a technique that is based on experience or statistical insights, specifically focusing on the structure of individual problems [23]. The downside of using such an approach is that there is no longer a guarantee to calculate the global optimum, as heuristics mostly find only partial or approximate solutions to a given problem. In many cases where exact algorithms would take too much time or space to find the optimal solution, heuristics can be used to find the best possible solution within the given constraints. For heuristics, several different ideas have been developed, which can be categorized into construction heuristics, improvement heuristics, and metaheuristics [29, 40]. However, differentiating between these types is not always trivial and the distinctions are often blurred.

Construction heuristics build their solution from a starting point until a certain boundary is reached. They typically do not have a separate improvement phase. Improvement heuristics aim to enhance an already existing solution. They perform improvement steps repeatedly until a specified boundary is met, such as a time limit or reaching the threshold for a good enough approximation. (Iterative) Local Search and Neighborhoods are examples of improvement heuristics that can be used to reach a more optimized solution.

Metaheuristics are a type of heuristic approaches that also seeks to find an approximate solution to a problem that is as optimal as possible. The distinction between classical heuristics and metaheuristics is that the latter are less tailored to specific problems. Metaheuristics typically treat a problem as a black box and do not use distinct structures to solve a problem. On the contrary, a metaheuristic is typically more generalized and can be used to solve several types of problems. These strategies are used to enable the metaheuristics to not only produce solutions that are locally optimal, but to broaden the search space they can use for finding optima.

Classical heuristics often carry the inherent risk of finding only a local optimum, which may be far from the global one. To mitigate this risk, higher-level approaches are necessary. These methods can include using multiple neighborhood structures to broaden the search space, like Simulated Annealing does, or using entirely new concepts like the Ant Colony approach or Genetic Algorithms. The metaheuristic ideas used in this thesis will be explained in the following subsections.

2.3.1 Ant Colony

Ant Colony is a metaheuristic approach that is inspired by the behavior of biological ants, their colonies, and how they search for food. Real ants start by leaving their nests and walking around randomly. When they discover a food source, they pick up the food and return to their nest. On their way back, they distribute a substance called *pheromone*, which can then be detected by other ants, indicating that a path leads to a potentially good food source. Other ants are then more likely to follow a path with more pheromone placed on the respective edges and will, in turn, lay down their pheromone as well, leading to an accumulation of the pheromone on good paths. Over time the pheromone dissipates and, if not renewed, will eventually evaporate completely, reducing the attractiveness of the corresponding path [18, 23].

Furthermore, pheromone distribution inherently leads to using shorter paths. When several ants have to choose between paths, they will first select at random. However, once an ant discovers the food source, returns to the nest and distributes pheromone along the way back, this process automatically increases the likelihood of the respective path being taken by other ants. Shorter paths will receive more pheromone as the returning ants will be quicker, leading to a faster distribution of pheromone. Consequently, more ants will choose this shorter path and thus deposit even more pheromone on it, leading to a self-reinforcing loop that converges when all ants choose the best path only. Eventually, the pheromone on all paths that are not as good (i.e. longer) will evaporate over time, leaving the best route as the only remaining option [18, 23].

To illustrate pheromone distribution, Figure 2.2 shows how real ants find food and establish a good path to the source. In part **(a)** on the left side, there are many ants moving between two points, A and E, which could represent the nest and a food source. In part **(b)** in the middle, an obstacle has been added. This construction now gives the ants a choice of which path to follow. Initially, the likelihood of choosing either path is around 50%. While taking the path, the ants distribute pheromone on it. On the shorter route, the ants will reach the food source earlier, thus returning quicker than those who took the long path and distribute more pheromone on the shorter path. For the first few ants, there will be almost no change in the attractiveness of either path. However, as more ants take the shorter route and return quicker, more pheromone will accumulate on that path. This increased pheromone concentration leads to a shift in attractiveness, making the shorter path more attractive and more likely to be chosen by later ants, as shown in **(c)** on the right side. These ants will, in turn, again increase the amount of pheromone placed, making the path even more attractive. Thus, the ants create a self-reinforcing loop of positive feedback through their pheromone, which eventually leads to a state where all ants constantly choose the shorter path.

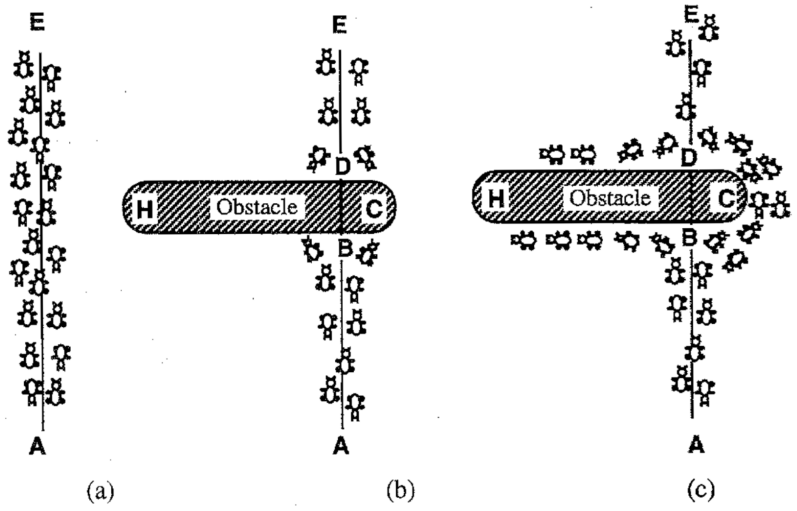


Figure 2.2: This figure shows an example of pheromone distribution with real ants. Taken from *Ant System: An Optimization by a Colony of Cooperating Ants* [18]

This behavior can be replicated in virtual graphs for various routing problems. Ant System has been first introduced in 1990 by Dorigo et al. [18] and the following explanations about calculations are based on their findings. In their paper, the authors describe how to use ants for solving the TSP. This problem is different from the question of finding a roundtrip with a certain length (plus additional user preferences). However, in the paper, they stress the adaptability of Ant System approaches, showing both versatility and robustness on different example problems.

Calculations

To transform the analogy of real ants into an algorithm, some formulas and calculations are needed. Ants are very simple agents. They can only do two things: pick the next node to move to and place pheromone on a path. They communicate with other ant agents through the pheromone trails, making this process a decentralized way of communication without the need for a central agent. For the algorithm, a set number of m ants move through the graph, try to find a good route, and place pheromone on edges. Every ant has a defined amount of pheromone to place. How much of the pheromone will be laid on a path ($\Delta\tau_{ij}^k$) can be calculated in several different ways. Here, $\Delta\tau$ describes the pheromone distribution, while the indices ij indicate the edge where the pheromone is to be placed and k indexes the respective ant. The authors propose the following three ideas:

$$\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{L_k} & \text{if } (i,j) \in \text{tour described by } \texttt{tabu}_k(1) \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

$$\Delta\tau_{ij}^{k} = \begin{cases} Q & \text{if the k-th ant goes from i to j between time} \\ & \text{t to t+1} \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{c_{ij}} & \text{if the k-th ant goes from i to j between time} \\ & \text{to t+1} \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

Equation 2.1 is the default formula used by the authors for solving the TSP. The constant Q must be chosen according to the problem in question. The variable L_k describes the length of the entire tour created by the k-th ant. This property is useful for the TSP setting, but is relatively meaningless for tours with a fixed length, as it remains the same value for every ant and every run made. In cases, where the user defines the length of the tour, L_k will only scale the values chosen for Q. The `tabuList` contains all nodes that are not allowed to be visited.

Equation 2.2 uses only the constant Q to describe pheromone placement, without considering the full tour length or individual edge costs. Pheromone is placed evenly on all edges, making this method equivalent to an unscaled version of Equation 2.1 for a fixed length tour.

The last Equation 2.3 divides the constant by the length, i.e. the cost of each edge (c_{ij}). This division reduces the amount of pheromone placed on longer edges compared to shorter edges. Although this equation is not influenced directly by the fixed length, this property may still be less useful for tours with a specified length than for TSP. Since tours that are meant to cover a fixed distance differ from the TSP, where the goal is to find a shortest path that visits all selected cities, Equation 2.3 seems the least promising candidate for effective pheromone distribution.

The calculation and placement of pheromone can be defined with different formulas depending on the problem and the optimization goal. A suitable option will be detailed in the implementation Chapter (3).

Applying a suitable formula to calculate the pheromone distribution, this value can then be used to calculate the overall distributed pheromone for each edge (i, j) that was placed by all ants during one iteration. This value is described by $\Delta\tau_{ij}$ as follows:

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{2.4}$$

Trail Intensity Calculation This overall value can then be employed to calculate the so called *intensity* of the placed pheromone trail. Since pheromone evaporates over time, this property has to be modeled as well, using a new parameter ρ, which describes how much

of the pheromone stays on the trail between two time steps. Thus, the overall pheromone intensity can be described by

$$\tau_{ij}(t+n) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}^{k} \tag{2.5}$$

where $\tau_{ij}(t)$ is the previous pheromone intensity and $t+n$ describes the next time step after one full tour was created in n steps.

Using these calculations, the pheromone intensity on all paths can be represented. The next step is to determine the probability with which ants will choose a certain edge over other options. For this calculation, two more properties are needed: the visibility of an edge and the tabuList, containing not-allowed nodes.

Visibility and Transition Probability Calculation The tabuList collects all nodes that have been visited before, ensuring that no node is visited more than once, which is necessary for round trips to remain circular rather than repeating the same path in both directions. In Chapter 4, various configurations are tested to represent different shapes and allow for more user-defined options. Thus, for other shapes, this list is not needed. For the TSP-case, the visibility ν_{ij}^{k} is calculated using the cost c_{ij} of the edge (i,j) as follows:

$$\nu_{ij}^{k} = \frac{1}{c_{ij}} \tag{2.6}$$

and the transition probability is given by

$$p_{ij}^{k} = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\nu_{ij}]^{\beta}}{\sum_{k \in \texttt{allowed}_k} [\tau_{ij}(t)]^{\alpha} \cdot [\nu_{ij}]^{\beta}} & \text{if } j \in \texttt{allowed}_k \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

using all previously defined values to calculate visibility ν_{ij}^{k}, trail intensity τ_{ij}, pheromone distribution per ant $\Delta\tau_{ij}^{k}$ and overall $\Delta\tau_{ij}$. Here, α and β are parameters that influence the weight of trail intensity and visibility respectively. Higher values of α increase the significance of the pheromone on the trail (setting α to 0 would lead to completely ignoring the pheromone placed), and higher values of β increase the importance of the visibility of an edge (making longer edges less attractive when using the given visibility calculation for TSP). These parameters will be experimented with and their influence will be evaluated in Chapter 4. For the visibility, a calculation that is suitable to the AOP is determined in Section 3.2.1.

In their paper, Dorigo et al. suggest picking middling values for α and β in the range of $[0.5, 5]$. They further stated that the best tour was achieved using $\rho = 0.5$ and $Q = 100$. Overall, the results of experimenting with different parameter configurations showed that for very high or very low values of α, no good results could be generated.

Some of these equations must be adjusted and fitted to the given `Arc Orienteering Problem` (see Section 2.1). The adjusted calculations and explanations of the changes made can be found in Section 3.2.1.

Algorithm details

A possible structure for an Ant Colony algorithm is displayed in listing B.1, based on the original framework provided in [18]. Therefore, all formulas are centered around the shortest path that connects all cities in the graph. Here, the tabuList is used to track all visited cities, the probability is calculated using Equation 2.7. The variables `NC` and $\texttt{NC}_{MAX}$ are used for the loop count and the maximum number of loops allowed respectively.

In the initialization, the base values are set. Differences $\Delta\tau_{ij}(t)$ as well as t and `NC` are set to 0 while the trail intensity $\tau_{ij}(t)$ is assigned an initial value of c. The ants are placed on a set of cities. Then, the main loop is started in which the ants are moved according to the probability. All visited towns are appended to the tabuList, which is full once all cities are added. This step completes one tour for all ants.

The next steps are necessary to reset and recalculate the needed values for the next run. In the loop, the ants are moved back to their starting positions, the length of their respective tours is calculated and with it the pheromone that is to be placed on the edge ($\Delta\tau_{ij}^{k}$) for the current ant k. Using this pheromone, the final pheromone distribution for all ants ($\Delta\tau_{ij}$) is updated. Furthermore, the new shortest tour, if a better one was found, is updated. After the individual pheromone values are updated, the trail intensity $\tau_{ij}(t+n)$ has to be recalculated. For this step, the evaporation rate ρ scales the current trail intensity and the calculated resulting pheromone for all ants ($\Delta\tau_{ij}$) are added.

The rest of the loop is updating the loop count `NC`, the time step t, and resetting the pheromone for each edge to the initial 0. The loop is continued with emptied `tabuList` if the maximum count was not reached and is ended otherwise, returning the shortest tour found.

2.3.2 Simulated Annealing

Simulated Annealing (SA) builds upon the statistical physics concept of annealing. The physical process of annealing describes a thermal procedure to improve solids. Improving in this case means "obtaining low energy states of a solid in a heat bath" [1]. Low energy is needed to achieve the stable solid state of a crystal [13] (see Figure 2.3 for a visualization of the different states).

The physical process is split into two general steps:

- Heating the solid until the atoms can move more freely
- Cooling the material to a specified temperature

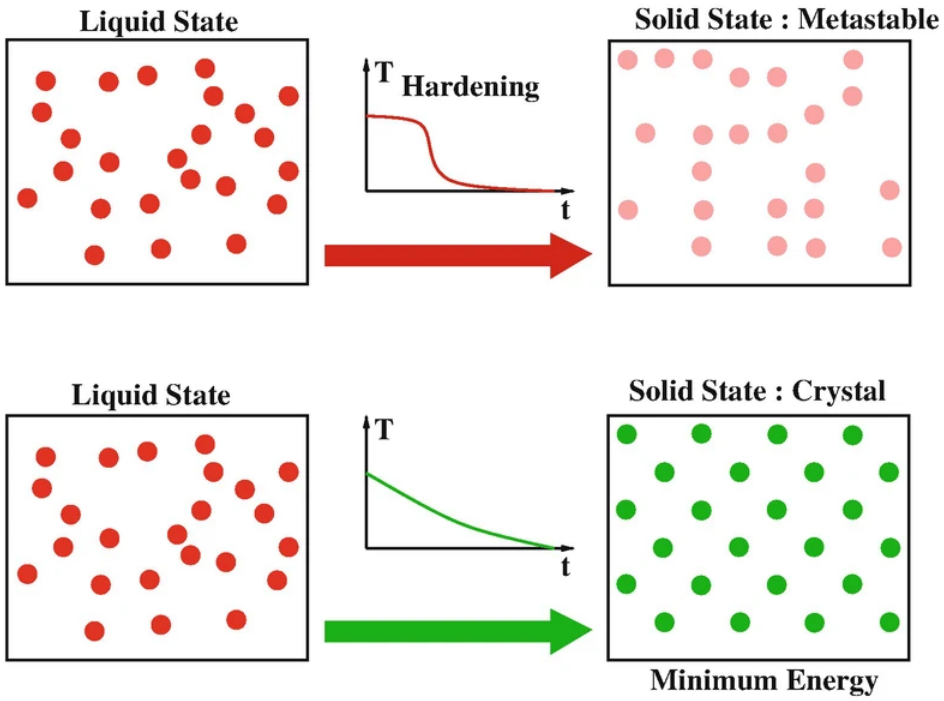


Figure 2.3: The illustration shows two different cooling procedures for a material. The upper one depicts the effects of sudden cooling, resulting in a meta-stable state, while the lower one portrays a slower cooling schedule, which results in a stable solid state. Taken from *Simulated Annealing: From Basics to Applications* [13].

The basic concept is that, given a sufficiently high starting temperature to which the material is heated and a long enough cooling time, the material can reach a stable solid state.

Figure 2.3 shows the effects of different cooling schedules on a material and the type of solid state this material will obtain as a result. The top-left image shows a liquid state, in which molecules are free and can move around. Cooling down quickly hinders the atoms from moving into proper positions, resulting in an unordered, non-symmetrical state, which is meta-stable (top-right). In contrast, the lower part depicts a slower cooling process, where the atoms are given enough time to move into symmetrical form (bottom-right). This state is equivalent to a minimum energy state, resulting in a stable crystal structure and a highly durable solid material.

Based on this analogy, SA was developed as a concept to solve NP-hard computational problems by improving local search variants [1, 19]. Local search approaches are good at finding locally optimal solutions, but can easily fail to find a global optimum. SA addresses this problem with the underlying idea based on the two steps of heating and slow cooling. The distinguishing feature of SA is that the algorithm permits the acceptance of solutions with a certain probability (temperature) that may be inferior to the optimal solution.

Basic Structure of SA First, a basic solution for the problem is needed. For the use case of this thesis, a basic solution is a roundtrip. Furthermore, a function that determines the quality of the solution is needed, so that comparing different solutions becomes possible. Starting from the initial roundtrip, neighboring solutions need to be generated and their quality will be assessed using the quality function. Any neighboring roundtrip that yields a better result will be accepted as the new solution that has to be improved. Worse roundtrips can also be accepted based on a cooling schedule.

The schedule contains an initial temperature and a function to calculate the cooling of this value, gradually reducing the temperature over time. In the beginning, the solution is likely to be further away from a global optimum. Therefore, a higher temperature allows for an increased likelihood of accepting solutions with a lower quality, which in turn allows for escaping local optima. As SA progresses, the temperature decreases, reducing the likelihood of accepting inferior solutions as the algorithm approaches the global optimum. Achieving this global optimum requires several nested runs: an outer loop updates the temperature with lower values while an inner loop tests several possible neighborhoods for each temperature state. Solutions that are better than the previous one are always chosen, while solutions that are worse than the previous one are selected based on the temperature.

For SA to yield a good result, several parameters need to be established. First, a starting roundtrip is calculated to create the baseline. Then, the initial temperature and the cooling formula are determined. Furthermore, a neighboring roundtrip has to be identified, along with a formula to assess the quality of all solutions found. The number of runs is also decided, while both the runtime and the quality of the resulting tour are taken into account.

For some parameters, such as the initial temperature and the cooling formula, thorough testing of different configurations is necessary. For others, only a limited set of options is relevant. Determining a neighboring roundtrip as well as an initial solution are parameters with a wide array of calculation possibilities: Any viable algorithm can be chosen to calculate the initial solution. For this thesis, the choice is detailed in Section 3.2.2. To determine a neighboring solution, there are a few options that can yield good results. One approach is to randomly select two vertices and switch their position in different ways within the roundtrip path or variations of this idea [51]. Again, the operation chosen for this thesis is described in Section 3.2.2.

Calculations

For SA, several parameters need to be calculated. First, the initial temperature and the respective cooling schedule or a function to decrease the value have to be determined. Various functions can be applied, provided that they decrease the temperature and are suited to the problem. For example, a simple function that calculates the temperature by

using the difference between the quality of the neighboring solution $f(j)$ and the quality of the current solution $f(i)$, divided by the natural logarithm of a random number r_i [51]:

$$T_{i+1} = \frac{-|f(j) - f(i)|}{ln(r_i)} \tag{2.8}$$

In this equation, r_i is smaller than a previously calculated probability p_i. Only when r_i was smaller, the respective solution with a worse quality is accepted. The probability p_i is calculated using the difference between the quality of the neighboring solution $f(j)$ and the quality of the current solution $f(i)$, divided by the maximum temperature as exponents of the euclidean function: $p_i = e^{\frac{-|f(j)-f(i)|}{t_{max}}}$ [51].

Another option is to multiply the current temperature by a factor α smaller than 1 [19]:

$$T_{i+1} = \alpha \cdot T_i \tag{2.9}$$

Or scaling by using a constant b to form a fraction [19]:

$$T_{i+1} = \frac{T_i}{1 + b \cdot T_i} \tag{2.10}$$

Several other options exist, as listed in the paper by R. W. Eglese [19], to calculate the temperature, where the key criterion is that the temperature decreases over time.

The quality function f is highly dependent on the problem at hand and thus can vary significantly. For the TSP, the quality of a tour is determined by the respective length. For the AOP, the quality is determined by the values that are to be collected when generating a roundtrip. How the quality is calculated for the specific case in this thesis is described in detail in Section 3.2.2.

Another important calculation is the probability of accepting a worse solution. This probability can be determined by using the temperature and the quality values of the two solutions [19]. The difference between the quality of the neighboring solution $f(j)$ and the current solution $f(i)$ is always negative, because a positive difference implies $f(j) > f(i)$, resulting in the neighboring solution being chosen automatically. Since the probability is calculated only when the difference is negative, the exponent will always be less than 1, resulting in a valid probability function that has a value closer to 1 when the difference is small, and closer to 0 when this difference is large.

$$p = e^{\frac{f(j)-f(i)}{T}} \tag{2.11}$$

Algorithm details

The SA algorithm has a basic structure that remains consistent across different problems. This structure is outlined in the pseudocode in B.2, based on Eglese [19] and Zhan et al. [51].

First, an initial roundtrip has to be determined. This calculation can be performed using any algorithm that produces a valid solution, such as a Greedy approach or the Min-Cost algorithm, which is also used for solving the AOP in the original Tour4Me application (see Section 1.1.1). Furthermore, a starting temperature must be set, which will be used for the initial probability calculations and then decreased after the iterations of the inner loop.

Next, the two loops of the main SA step are executed. The outer loop is bound by a selectable number of runs, while the inner loop is bound by a selectable number of repetitions that should be performed with a fixed temperature. The outer loop controls how many new temperatures are tested – in the physics analogy, this property equates to how long the material is left to cool – with decreasing temperatures. The inner loop controls the number of attempts made before the temperature is adjusted. Alternatively, the outer loop can use reaching a certain quality value or a maximum run time as the stopping condition. Performing several calculations in the inner loop with the same temperature can increase the probability of finding a better solution for the set temperature, as well as the likelihood of choosing a solution with a worse quality.

Chapter 3

Implemented Changes

This work extends Tour4Me[1], an application originally written in C++[2], HTML[3], and JavaScript[4]. The implemented interface of this extension uses C#[5] as programming language to enable easy porting of the web application to a desktop or mobile application. To improve query times and to allow for comprehensive global coverage (see Sections 3.1.2 and 5.2), a database that offers features of a spatial database was added. The reasons for and positive effects of this decision are described in the following section.

Furthermore, several modifications were made beyond the programming language and data access. New options and parameters to improve the customizability of preferences for a generated tour were added as well. These changes required updating front-end design (see Section 3.1.3) as well as the back-end and all solvers (see Section 3.2).

3.1 Application

To incorporate these various changes, the entire application was changed – including the front-end representation, back-end implementation, and data retrieval process. The Open Street Map (OSM)[6][7] data are downloaded and stored in a database from which the graph for calculating roundtrips is built. The front-end design was changed to improve the overview and general user experience as well as to accommodate new customization options. Finally, the algorithm selection was extended by two additional metaheuristic approaches.

[1] http://tour4me.cs.tu-dortmund.de/, last accessed: 18.04.2024
[2] https://devdocs.io/cpp/, last accessed: 22.03.2024
[3] https://devdocs.io/html/, last accessed: 22.03.2024
[4] https://devdocs.io/javascript/, last accessed: 22.03.2024
[5] https://learn.microsoft.com/en-us/dotnet/csharp/, last accessed: 22.03.2024
[6] https://wiki.openstreetmap.org, last accessed: 22.03.2024
[7] https://wiki.openstreetmap.org/wiki/Main_Page, last accessed: 19.04.2024

3.1.1 New Architecture

For the new application, the architecture had to be restructured. An illustration of the new design is shown in Figure 3.1. Instead of reading the data for the graph from a static *.txt* file, containing all nodes and edges for Dortmund, a database is used to manage nodes, edges, additional information, and the relationships between them. The database used is Microsoft SQL Server Management Studio[8], which can handle spatial data, supports spatial queries, and works well in combination with the C# implementation. This database can be populated using an import python script that creates an osmnx-graph[9][10] for a user-specified location. From this graph, the nodes and edges are extracted along with their additional information. For the current use case, nodes are stored with their OSM-ID, which is converted into a UUID, their latitude and longitude coordinates, their elevation profile, and tags of the surface, path type, and the surroundings they are placed in. Since OSM does not provide height profiles, elevation data is sourced from Open-Elevation.[11]

Given that most open-source providers have a limited bandwidth to supply users with data based on their API-calls, the opportunity to use a locally hosted version that Open-Elevation offered was crucial to assure usability. The python script used to create and fill the database and the respective tables requires access to the Open-Elevation data. A local docker container[12] with the respective data can be used to access the needed information without being bound to the servers and their throughput boundaries. Even though the accuracy of the locally hosted version is not ideal, the provided docker container forms the most practical method that did not rely on querying a website and being limited by their API-call restrictions. Section 5.2 discusses other data source options for future work.

Back-end The back-end is implemented in C#, allowing for the future creation of a mobile- or desktop application in addition to the existing web application (see Section 5.2). C# supports SQL queries and filtering using LINQ[13] for easy runtime database querying.

The Tour4Me application was automatically translated from C++ to C# using ChatGPT[14]. This translation is not part of the thesis and only allows for easier extension since the basic structures used in Tour4Me did not need to be reimplemented from scratch.

[8] https://learn.microsoft.com/en-us/sql/sql-server/sql-docs-navigation-guide?view=sql-server-ver16, last accessed: 22.03.2024

[9] https://osmnx.readthedocs.io/en/stable/, last accessed: 15.04.2024

[10] https://networkx.org/, last accessed: 22.03.2024

[11] https://open-elevation.com/, last accessed: 20.03.2024

[12] https://www.docker.com/, last accessed: 08.07.2024

[13] https://docs.telerik.com/devtools/aspnet-ajax/controls/grid/asp.net-3.5-features/linq-to-sql---binding-and-automatic-crud-operations, last accessed: 22.03.2024

[14] https://chatgpt.com/, last accessed 14.06.2024

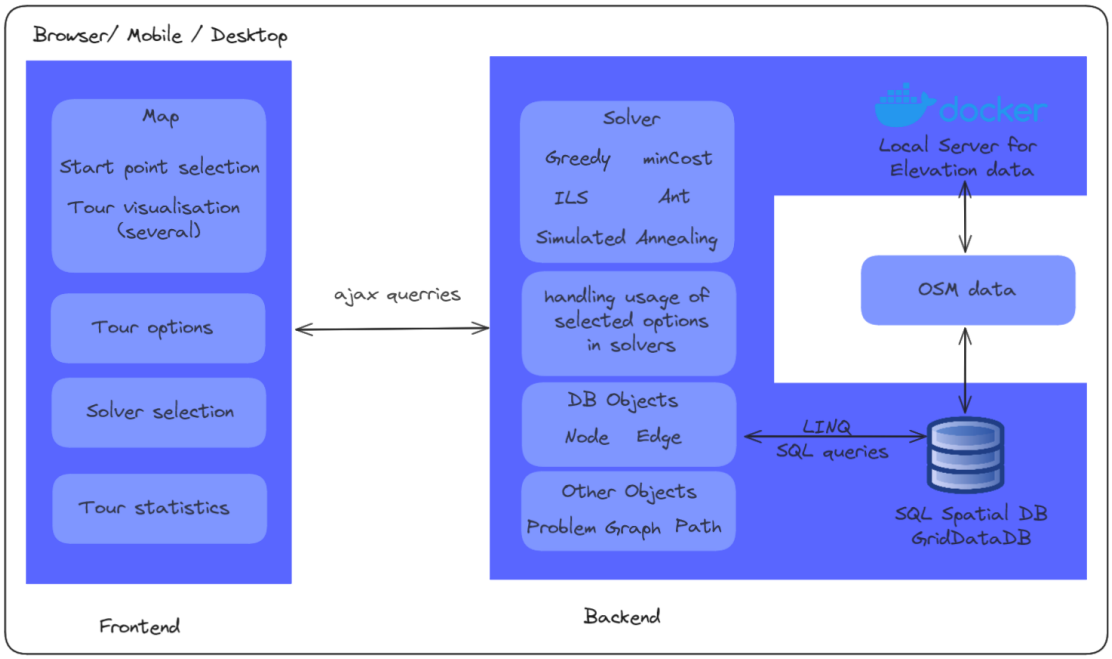


Figure 3.1: Visualization of the used architecture

Front-end The front-end is implemented using HTML, CSS[15], JavaScript, and C# code behind. The base-styling is done using bootstrap[16], but additional custom CSS is added to create a nature-based color palette as well as several custom effects, and transitions for the side and bottom menus. The communication between the front-end and back-end is handled using Ajax-queries[17].

The map is visualized using leaflet[18], which displays Open Street Map data. The leaflet map allows for setting markers, adding a search bar, creating polygons to illustrate the generated routes, and offers an open-source map view.

Application Structure Visualization 3.1 illustrates the structure of the whole application. The distinct parts and features as well as communications between them are presented. The front-end runs as a web application in the browser but the visualization can also be customized to be executable as a mobile or desktop application (see Section 5.2). On the visualized map, a marker can be set to the current location if the user grants permission to access their location data. Searching for a specific address, drag-and-dropping the marker on the map, or scrolling the map and selecting a position by clicking on the place to mark are also possible. The visualization of the calculated tours is realized using the map and polygons built from the respective points. These tour information can be tog-

[15] https://devdocs.io/css/, last accessed: 22.03.2024
[16] https://getbootstrap.com/docs/4.3/getting-started/introduction/, last accessed: 22.03.2024
[17] https://api.jquery.com/category/ajax/. last accessed: 22.03.2024
[18] https://leafletjs.com/, last accessed: 20.03.2024

gled on or off to allow for an easier overview after generating several tours. For debugging purposes, there is a feature to show the entire graph that is being used for the calculation with the currently selected maximum length.

In addition to the map, the front-end contains two menus: One holding the parameters the user can use to customize the tours according to their preferences and the information menu containing a report of the core data of the calculated path that is being visualized. A more detailed description of the front-end design, concept sketches, and the final implementation are outlined in Subsection 3.1.3.

The back-end manages all solvers that have been implemented, the database objects, and intermediate objects like the graph that is created from the nodes, edges, and their connections. The back-end also handles the parameters, selecting the correct solver, and generally managing the ajax requests received from the front-end. The database objects are generated when building the graph by accessing the database and using the stored values.

Finally, the database is filled using a python script that queries OSM-data to fetch all nodes and edges for a user-selected place. For this example, data for Dortmund were retrieved. Additionally, elevation data were obtained from the docker server using the latitude and longitude coordinates.

3.1.2 Database

The database is a relational database using Microsoft SQL server, administered in Microsoft SQL Server Management Studio. This database supports spatial data, which is a crucial feature for storing and processing the nodes and edges. Utilizing spatial features allows for filtering nodes within a specific radius, retrieving only a relevant subset of data points. This filtering option within the database significantly enhances the data retrieval speed and the graph creation efficiency. Compared to the previous method of generating a fixed graph for the city of Dortmund, the database offers further important advantages. For example, far more nodes than just points within Dortmund can be used. Although the database creation and the adding of points is relatively slow, this process is a one-time operation performed before the application is used and does not impact the tour calculation. Once the data has been added, all points can be accessed without needing to retrieve the entire database.

Gathering OSM Data To create the database, a python script is used. This script first creates an osmnx graph from OSM data using the `graph_from_place` function

```
ox.graph_from_place(place_name, network_type='all', custom_filter=
    ↪ custom_filter)
```

Highway type	Description
motorway	A restricted access major divided highway, normally with 2 or more running lanes plus emergency hard shoulder. Equivalent to the Freeway, Autobahn, etc.
trunk	The most important roads in a country's system that aren't motorways. (Need not necessarily be a divided highway.)
proposed	For planned roads.
construction	For roads under construction.
motorway_link	The link roads (sliproads/ramps) leading to/from a motorway from/to a motorway or lower class highway. Normally with the same motorway restrictions.
trunk_link	The link roads (sliproads/ramps) leading to/from a trunk road from/to a trunk road or lower class highway.

Table 3.1: This table shows a listing of different OSM highway types and their definition taken from the Wiki page[20]

In this script, the `place_name` is the name of the city, state, country or region for which osmnx should gather the data points. For a city, the city's name, state, and country need to be added. If a whole state should be selected, the state name and country are required. The `network_type` has six values to choose from[19]: `all_private`, `all`, `bike`, `drive`, `drive_service`, and `walk`. Of these options, `all` was used for the query to allow fetching all network types before filtering.

The `custom_filter` is defined to select only those edges, where walking, running, and cycling is possible by specifically de-selecting respective highway types. The excluded types describe the street types detailed in Table 3.1 according to the OSM Wiki.

```
custom_filter = '["highway"]["highway"!~"motorway|trunk|proposed|
    ↪ construction|motorway_link|trunk_link"]'
```

Adding Elevation Data Next, the elevation data has to be added for the nodes of this queried graph. These information are not part of osmnx and need to be retrieved from a different source. For this thesis, Open-Elevation[21] was used. The data were downloaded and hosted in a local docker container that could be queried instead of the API to avoid being limited by their throughput boundaries.

[19] https://osmnx.readthedocs.io/en/stable/user-reference.html#module-osmnx.settings, last accessed: 19.04.2024

[20] https://wiki.openstreetmap.org/wiki/Key:highway, last accessed: 19.04.2024

[21] https://open-elevation.com/, last accessed: 20.03.2024

Then, the surroundings information need to be obtained. These information are provided by OSM, however, they are not saved for nodes but for *areas*, which are clusters of nodes and edges. A different query that retrieves these areas and matches the respective tags to the saved nodes by matching the IDs has to be executed.

After gathering all the neccessary information, the script first checks if the database already contains matching tables for the respective objects (nodes, edges, incident edges) and if not, generates them. Then the nodes and edges that have been retrieved from osmnx can be iterated and inserted into the database using basic SQL. During the iteration over the edges, the references between nodes and edges can be created, adding references to the source- and target node in the edge table. Additionally, references are inserted into the *IncidentEdges* table that manages nodes and all their incident edges to allow querying the relationship both ways. Using these references later enables the back-end code to easily access the endpoints of a graph or gather the incident edges for a node.

3.1.3 Interface and Front-end Changes

The front-end was updated from the existing version of Tour4Me to allow for a better overview after adding several additional customization options for the user to select from. First, a conceptual idea (see Figures 3.2 and A.1 to A.3) was built, mapping out the general structure of the new interface. The main changes focused on replacing the previous pop-ups through permanent menus. On the side, a burger-menu button was added that allows for a side menu to be folded and unfolded to show the customization options (see Figure A.2). The result view was moved from an overlay on the map to a foldable footer menu (see Figure A.3). Furthermore, the displayed tours and the respective information can now be unfolded additionally in this footer menu. Previously, the tour information (including the length and other relevant data) was only accessible through a pop-up.

Displaying all the resulting data, while giving the option to fold and unfold the two menus, allows users an easier access and enables straightforward comparison between different tours. Previously, only the data for a single tour at a time could be displayed in the pop-up. Now, the information of all tours are visible simultaneously.

Front-end Concept The new interface has a more botanic color scheme, using mainly dark greens, browns, and blue while the text is off-white. The side menu *options* displays a wider selection of preference settings. In Figure 3.3, the side menu is visualized. The uppermost drop-down (see Figure A.4) can be used to set a pre-selection that fills in the following fields with suggested default values. These default values can always be customized afterwards but offering a default option can generate a heightened usability for many users. The next selection drop-down shows all currently implemented algorithms (see Figure A.5). The resulting tours will turn out differently – having higher importance on the covered area (roundness), or elevation and edge profit (see also Chapter 4) – depending on

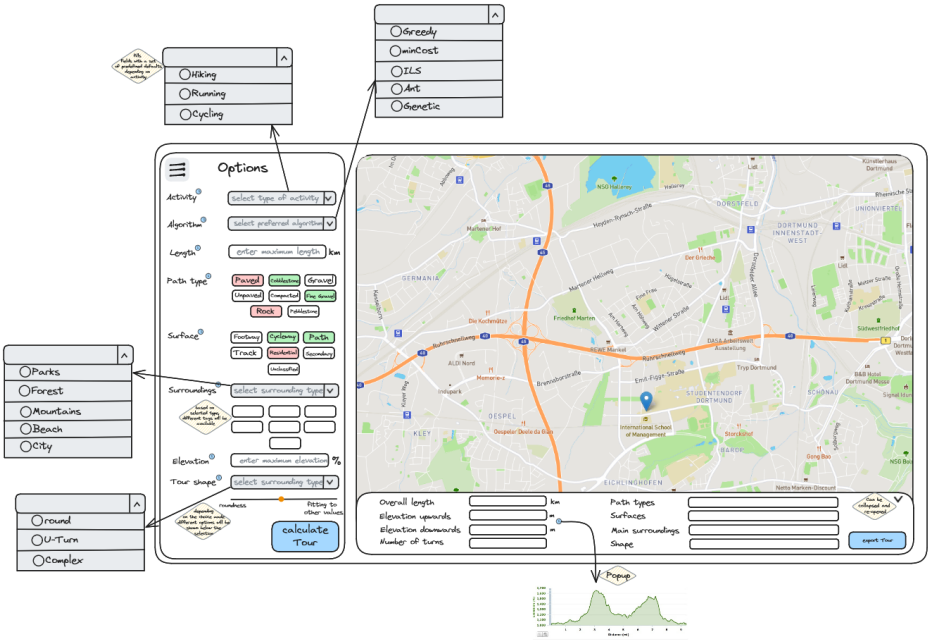


Figure 3.2: Design concept for the front-end view, including descriptions for drop-downs and pop-ups

the choice. While Greedy always prioritizes the maximum edge profit only and ignores the roundness or elevation of the tour, MinCost does the exact opposite and prioritizes covered area. Ant colony is more focused on edge profit, but also takes elevation and roundness into account. Finally, SA produces results that mostly focus on roundness while still taking edge profits and elevation into consideration. The two offered base SA tours start with an empty solution with either a given probability distribution or a fully random selection and then strive to improve them. The respective combinations of both Ant Colony and SA start with Greedy and MinCost for both algorithms. For SA specifically, a combination with Ant Colony as the base tour was implemented as well.

Implemented Changes for the Option Menu All following options are direct tour parameters and describe user preferences. The length is an estimate of the final tour length and is never used as a hard limit, but allows for tours to be within a range of a few hundred meters longer or shorter than the selected length. However, tours are never more than a kilometer longer or shorter than the selected length.

The options for choosing the surface and the path type are implemented using selection buttons. These buttons can be neutral (when they only show a white border), positive (colored in green) or negative (colored in red). A neutral button describes properties that are neither preferable nor undesirable while green marks preferred values and red marks undesired values. Surfaces describe properties of the ground, path types characterize the

type of street. In OSM, there are many other options, however, some are already filtered (for example highways) and others are not of significant interest and would only result in an overwhelmingly large selection. To choose desired or undesired surroundings, a drop-down menu has been implemented. This menu allows the user to select a general type – forest, grasslands, or other options – that will then display the respective tags (see Figure A.6). This approach was used to minimize the number of tags and allow for a clearer overview for the user.

The tour shape offers the options to choose a round tour, a U-turn, a complex tour, or do a custom selection (see A.7). Round tours have the highest importance (80 %) on the roundness of a tour and split the remaining 20% between elevation and edge profits. This setting makes rounder tours more likely to be calculated. U-turns can have a special implementation that ensures for a slightly different path back to the beginning (see Section 5.2). Currently, to calculate U-turns, the importance of edge profits and elevation are simply set to 50% each, while ignoring the covered area completely through setting the respective importance to 0%. Complex tours are configured similarly to the settings for U-turns, but do not fully remove the importance of the covered area. For the custom selection, three sliders are displayed. The slider inputs are linked so that they influence each other, ensuring that the three probabilities always sum to 100% (see Figure 3.3c).

Maximum values for elevation and steepness can be entered below the tour shape drop-down. The elevation describes the maximum difference in elevation for the whole tour. This property does not differentiate between ascending or descending parts but sums up all differences and divides the overall result by two, to accommodate for the roundtrip. The steepness is used to ensure, as far as possible, for that no part of the tour is steeper than the chosen limit. With the inaccuracy of the elevation data, satisfying this constraint can sometimes be impossible. In these cases, creating any tour, even if it exceeds the limit is seen as more desirable than having no result at all. Thus, neither the maximum steepness nor the maximum elevation are implemented as hard limits.

At the bottom of the side menu, the maximum runtime can be chosen. With the current settings, no algorithm runs as long as 30 seconds, however adding an option to allow for calculations that use the upper run time bound is also possible. Clicking the “Compute Path”-button will send a query to the back-end, where all the selected information are processed, a graph of suitable size is created, the matching solver is selected, and a tour is calculated. The resulting roundtrip is then parsed into the latitude and longitude values of the nodes that form the tour and returned to the front-end. Here, the latitude and longitude values can be displayed and connected to form a polygon that visualizes the resulting tour.

Route Information Overview All interesting tour information are displayed in the *Route information* footer menu. The menu as well as the calculated tours can be folded

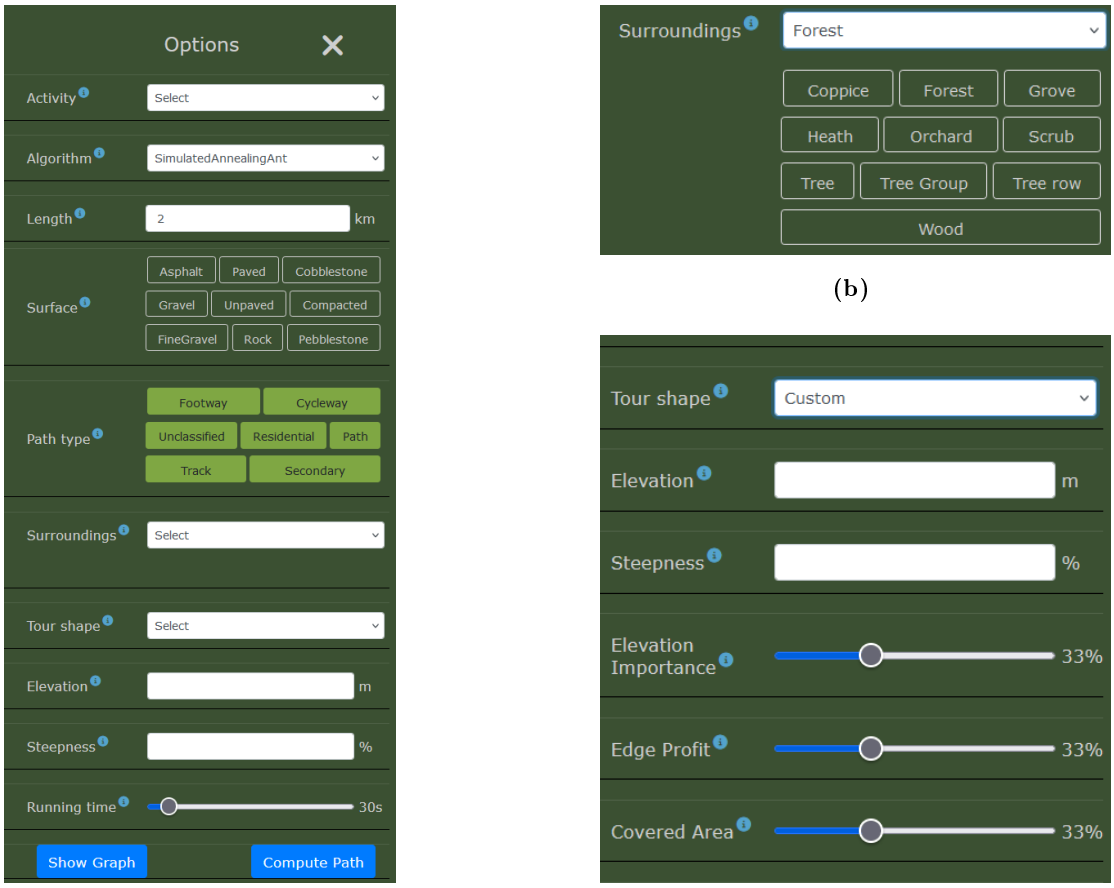


Figure 3.3: This figure shows the newly implemented side menu. (a) displays the full side menu (b) is a closeup of the surroundings when *Forest* is selected and (c) shows the three importance sliders when the selected tour shape is *custom*.

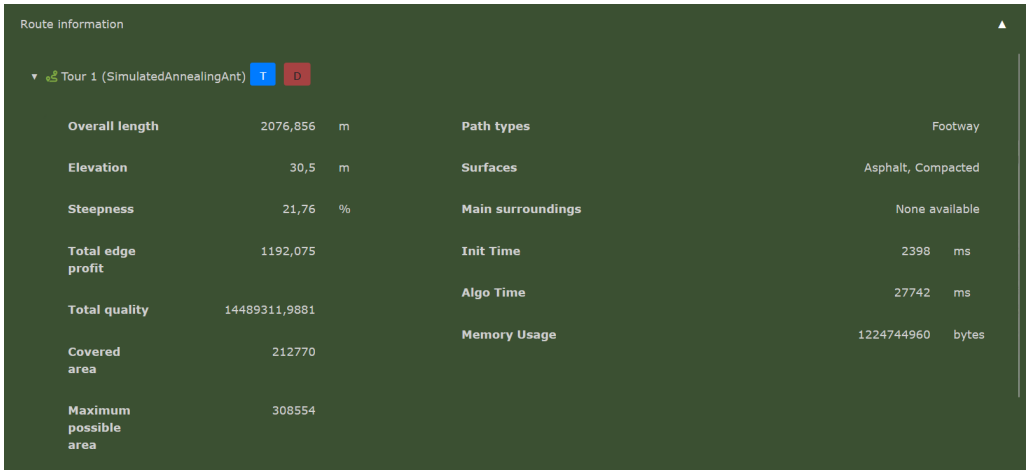


Figure 3.4: This figure shows the Route information menu with one tour and all the related information unfolded and displayed

and unfolded. This option does not decrease the size of the map significantly, but rather results in an added scroll-option, so the additional information can be displayed below the map. The shown tours are colored in the same color as the respective displayed tour. A selection of ten colors is implemented, allowing for ten different tours to be calculated and displayed before a color is repeated. All of these colors are kept within the same botanical color scheme. Next to the tour name, the used algorithm is written. Furthermore, there is one button to toggle the visibility (T) of the polyline in the leaflet map and one to delete (D) the generated tour entirely (see Figures 3.4 and 3.5).

Clicking on the tour name will unfold (see Figure 3.4) and fold (see Figure 3.5) the respective information. In this view, the final overall length, elevation, maximum steepness, collected edge profits, total quality, covered area and for reference the maximum possible covered area independent of the calculated path are shown. On the right side, the collected path types and surfaces, surroundings, and the time it took to initialize the graph and to calculate the tour as well as the memory consumption are displayed. The last three values are mostly not of interest to the typical user but allow for a deeper insight into the calculation. If needed, these outputs can be easily removed for an even more user-friendly application.

3.1.4 Parameter Changes

To include the parameters that have been added in the front-end, a few minor changes had to be made to the existing algorithms. However, since MinCost already uses the quality calculation to determine the tour's quality, nothing else had to be changed. In the quality calculation the elevation difference, steepness difference, and the respective importance were added and the scaling was changed (see Equation 3.8). Aside from these changes, more tags had to be added when creating the problem class. This addition was

Figure 3.5: This figure shows the *Route information* menu with several folded tours displayed

done similarly to the existing tags (surface and path type) and did not need any further changes. The Greedy algorithm only takes the profit an edge can add into account and thus did not need any change.

3.2 Algorithmic Changes

For the tour calculation, two new metaheuristics were implemented. The first one is Ant Colony, which was modified to match the AOP instead of TSP like the original. The second metaheuristic is a version of SA. Here, the calculation of neighboring solutions had to be matched to return roundtrips. The next sections will explain the changes needed to create implementations suitable for the AOP. For both metaheuristics, the pseudocode and changed formulas will be explained. The base idea and the overall working of the algorithms stays identical to the explanations in Sections 2.3.1 and 2.3.2.

3.2.1 Ant Colony

For Ant Colony, several parts of the original algorithm had to be adjusted. First, for solving TSP, finding a shortest path and visiting all specified cities are the main objectives. Whereas for AOP, these objectives differ by a lot: Here, finding a *matching* tour length that maximizes the profit is the important goal. Thus, the pheromone calculation, the trail intensity calculations, and the edge visibility had to be changed.

Pheromone Calculation In the original paper, the pheromone amount to be placed on the collected edges is calculated based on the length of the tour each ant builds and the pheromone each ant can place on an edge. This scaling results in higher pheromone density for shorter tours and lower pheromone density for longer ones. However, this constraint does not apply to the AOP. Here, ideally all resulting tours should be approximately of the same length, close to the user-selected one. Thus, the pheromone amount to be placed is based on the `edgeProfit` $p(i,j)$ the edge (i,j) with the cost $c(i,j)$ will grant the tour:

$$\Delta\tau_{ij}^{k} = c(i,j) \cdot p(i,j) \cdot Q \text{ if } (i,j) \in \text{tour collected by the ant} \tag{3.1}$$

The cost of the edge from node i to node j is multiplied by the profits the same edge can collect based on the assigned tags, which is multiplied with the pheromone amount, a single ant can place on the trail (Q). The latter is a means to scale the amount of pheromone placed and also enables the use of different ants for further combinations of Ant Colony, for example with genetic algorithms (see Section 5.2). If the edge has a tag specified as desirable, the `edgeProfit` p is increased by one. If the edge has a tag that is specified as undesirable, the profit will be decreased by one. The resulting profit value will then be set to $+1$, if the summed profit was larger than or equal to 1, to -1, if the summed profit was smaller than 0, and to 0.0001 otherwise. The first case describes the situation where more desirable tags were attached to the edge, while the second case arises when more undesirable tags were attached to the edge. The third case describes either an equal number of desirable and undesirable tags or no tags that match either group. When using this configuration, edges with more desirable tags will gain as much profit as they are long. Edges with more undesirable tags will lower the profit by their length and edges without tags or an equal number of desirable and undesirable tags, will impact the profit only by a fraction of their length. In Equation 3.2, $\texttt{tags}(i,j)$ describes the list of tags attached to each edge, `desirable` forms the list of all user-selected desirable tags, and similarly, `undesirable` build the collection of all user-selected undesirable tags.

$$p(i,j) = \begin{cases} 1 & \text{number of tags} \in \texttt{tags}(i,j) \& \in \texttt{desirable} \; > \\ & \text{number of tags} \in \texttt{tags}(i,j) \& \in \texttt{undesirable} \\ -1 & \text{number of tags} \in \texttt{tags}(i,j) \& \; \texttt{tag} \in \texttt{undesirable} \; > \\ & \text{number of tags} \in \texttt{tags}(i,j) \& \in \texttt{desirable} \\ 0.0001 & \text{otherwise} \end{cases} \tag{3.2}$$

Trail Intensity Calculation The trail intensity itself is calculated the same way as in the original paper (see 2.5). However, for further calculations in the Ant Colony algorithm, it is scaled by dividing by the desired length L, based on whether or not an edge is visited twice:

$$\tau'_{i,j} = \begin{cases} \frac{\tau_{i,j}}{L} & \text{edge already visited} \\ \tau_{i,j} & \text{otherwise} \end{cases} \tag{3.3}$$

These updated values are not saved globally or used for other ants but saved in a separate set, so the penalty will only be applied for the respective ant that visits the edge twice.

Visibility Calculation Furthermore, the visibility cannot be based on the length as presented in the original paper, since all tours should result in a length close to the desired one. Thus, to calculate the visibility, the edge quality has to be determined, for which all selectable values have to be taken into account: The `coveredArea` A, that maps the roundness of the tour, and the respective selected importance i_A, the `edgeProfit` p, and the respective selected importance i_p as well as the `elevation` difference e from the maximum e_{max} summed with the `steepness` difference s from the maximum s_{max} and the respective importance i_e:

$$\nu_{ij}^k = i_A \cdot 100 \cdot \frac{\sqrt{|A| \cdot \pi} \cdot 2}{L} + i_p \cdot 100 \cdot p + i_e \cdot \left(\frac{e_{max} - e}{e_{max}} + \frac{s_{max} - s}{s_{max}} \right) \tag{3.4}$$

The respective importance values are multiplied by 100 to scale them from the decimal values to percentages bigger than 1. For the `covereArea`, the square root is taken in order to scale the size and account for the quadratic increase. The whole area is then scaled by the desired length of the tour L to account for the larger value the area will have compared to the edge profits. The full derivation of the calculation for scaling the area is shown in equations 3.5 and 3.6, where A is the area and C the circumference of a circle.

$$\begin{aligned} C &= 2 \cdot \pi \cdot r \\ \frac{C}{\pi \cdot 2} &= r \end{aligned} \tag{3.5}$$

$$\begin{aligned} A &= \pi r^2 \\ A &= \pi \cdot \left(\frac{C}{\pi \cdot 2} \right)^2 \\ A &= \pi \cdot \frac{C^2}{\pi^2 \cdot 2^2} \\ A &= \frac{C^2}{\pi \cdot 4} \\ A \cdot \pi \cdot 4 &= C^2 \\ \sqrt{A \cdot \pi} \cdot 2 &= C \end{aligned} \tag{3.6}$$

Probability Calculation As in the original paper by Dorigo et al. [18], the probabilities for the edges are calculated using the visibility and the trail intensity as well as the two parameters α and β (see also Equation 2.7), however, here, the scaled value τ'_{ij} is used:

$$p_{ij}^{k} = \begin{cases} \frac{[\tau'_{ij}(t)]^{\alpha} \cdot [\nu_{ij}]^{\beta}}{\sum_{k \in \texttt{allowed}_k} [\tau'_{ij}(t)]^{\alpha} \cdot [\nu_{ij}]^{\beta}} & \text{if } j \in \texttt{allowed}_k \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

Other Changes Additionally, there is no tabuList and thus, this list cannot be used as a stopping criterion. Instead, a set number of tours is calculated. Furthermore, the ants do not need to save the length of their current tour, as that value is not of any importance for the AOP version. Instead, the full quality of a tour is calculated and saved, so the best found tour for all ants can be determined. The ants all have to start at the same position (the selected starting point) rather than in n different towns. The full updated pseudocode is shown in Listing 3.1

Algorithm 3.1 AntColonyAOP

```
1: initialize graph and problem (starting point, graph with nodes max 2/3 · L distance)
2: init trailIntensity τ_ij ← 0.0001 for all edges(i,j)
3: set starting pheromone values pheromone(i, j) ← 0 for all edges(i,j)
4: generate ants
5: set current node ← starting point
6: for # runs do
7:    for # ants do
8:       get all incident edges for current Node
9:       while edge that doesn't exceed maximum length can be found do
10:         for edge (i,j) in incident edges do
11:            scale edge if already visited (τ'_{i,j})
12:            calculate quality which resembles the edge visibility using 3.4
13:         end for
14:         calculate probability to pick the edge based on 3.7 and scale them to sum to 1
15:         pick edge based on probability
16:         calculate new pheromone values for the picked edge according to 3.1
17:         move ant from current node to neighbor according to picked edge
18:         add edge and node to solution lists
19:      end while
20:   end for
21:   update trail intensities according to 2.5
22:   pick tour with best quality to use for next run
23: end for
24: pick tour with best quality as result
```

To set up the new Ant Colony implementation, the graph and a helper class *problem* have to be initialized first. The graph is created from the database, using the selected

starting point and $\frac{2}{3} \cdot L$ of the user-selected tour length L to only fetch the needed number of nodes and edges. The problem saves all other user inputs:

- desirable tags
- undesirable tags
- max tour length
- quality (initially 0)
- max elevation
- max steepness
- elevationImportance
- edgeProfitImportance
- coveredAreaImportance
- generated solution path

The edges hold their length, the profit according to the tags, their pheromone and trail intensity. Here, the trail intensity is initialized with 0.0001 as a starting value, so all edges are equally as likely to be picked, however the value has to be bigger than 0 for the probability calculation. As in the original paper, the pheromone are initialized with 0.

The ants can then be created and initialized. All ants use the same α and β as well as the same base amount of pheromone they can distribute. However, the values could be changed throughout the runs, which can be of interest for future combinations (see Section 5.2).

For a set number of runs, all ants calculate a tour in parallel. To achieve this parallelization of the ants, they all start at the selected starting point and then iteratively check the incident edges. Only edges that can fully be traversed while not exceeding the selected maximum length will be used here. If no edge is short enough to still fit into the tour, the roundtrip will be closed by calculating the shortest path from the last picked node back to the beginning. While viable edges are still available, they are first scaled. Then, their visibility and the probability of choosing each edge are calculated. Based on these probabilities, one edge is picked, the pheromone values of the edge are updated and the ant is moved. Finally, the picked edge and the new node can be added to the current solution.

After all ants have calculated their full tours, the overall trail intensity is updated, using the evaporation rate and the gathered pheromone updates from all ants. Then, the tour with the highest quality value is picked for the next run. After the last run the tour with the highest quality is chosen as the final result.

3.2.2 Simulated Annealing

For SA, the needed changes have to be performed for the variables that are meant to be determined based on the problem which is to be solved with SA:

- the quality of a solution
- how to find a neighboring solution

- the initial temperature and the cooling schedule

Thus, the pseudocode does not change by much, however, the exact calculations can be specified. The only additions to the pseudocode are building the Waypoint list, calculating distances and probabilities (see Listing B.3)

Quality Calculation The quality $f(j)$ of the solution j is – like for Ant Colony – calculated based on the `coveredArea` A and the respective `coveredAreaImportance` i_A, the `edgeProfit` p and the respective `edgeProfitImportance` i_p, and the `elevation` difference e from the maximum e_{max} summed with the `steepness` difference s from the maximum s_{max} and the respective `elevationImportance` i_e. The result of this calculated value is then scaled by the difference between the actual length L_{calc} of the tour from the target length L to form the final quality of the tour.

$$f(j) = \frac{i_A \cdot 100 \cdot \frac{\sqrt{|A| \cdot \pi} \cdot 2}{L} + i_p \cdot 100 \cdot p + i_e \cdot \left(\frac{e_{max} - e}{e_{max}} + \frac{s_{max} - s}{s_{max}} \right)}{|L - L_{calc}|} \tag{3.8}$$

The respective importance values are again multiplied by 100 and the `coveredArea` is scaled the same as for the Ant Colony (see Equations 2.6, 3.5 and 3.6)

Determining a Neighboring Solution The neighboring solution is created based on Waypoints, which are used similarly to how they work in the MinCost tour creation (see Section 1.1.1): The Waypoints form base points of the tour and are connected using a weighted version of a shortest path calculation. For this calculation, the shortest path of any point in the created graph to the starting point is determined by dividing the edge length by the profit that can be gained when choosing the edge.

For SA, several different versions have been implemented. The first two that are described here start with an empty tour and build the solution from only the starting point. In the next section, other combinations and alterations are discussed as well.

The two versions that start with an empty solution are a weighted and a fully random version of the same base idea: A random point (with or without a probability distribution) is picked as a new Waypoint. Until at least five Waypoints have been found, adding or moving a Waypoint are the only options available, however when only one Waypoint exists, a second one *must* be added first. After that, adding, moving, and removing of a Waypoint are possible until 15 points are reached. After that, a Waypoint can only be moved or removed. When the number of Waypoints is between 6 and 14, all three options (adding, moving and removing) are possible. Using these boundaries ensures that the resulting tour will not end up too short while also assuring that not too many distance calculations are needed, since the number of Waypoints determines how many distance calculations have to be done. For moving and adding, the closest existing Waypoint has

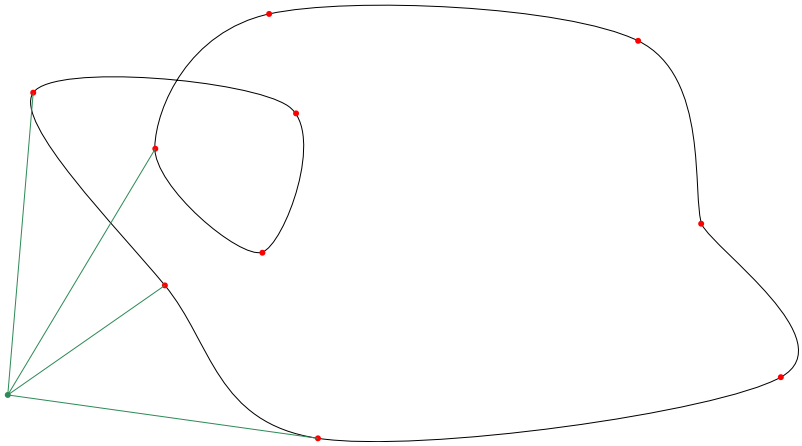


Figure 3.6: This figure shows the path, which is separated in ten Waypoints, a new random point and four of the distances to the closest Waypoints.

to be known, thus all distances from the new Waypoint to all existing ones have to be calculated.

Using Waypoints Whether a Waypoint is moved, removed, or a new one is added is picked randomly within the previously given constraints. The starting point can neither be chosen for moving nor for removing.

First, a random point is selected from the set of available nodes. This set is a smaller version of all available nodes, allowing only nodes that have a shortest distance of at most $\frac{L}{4}$ from the starting point. For the version using a probability distribution, the probabilities are calculated using a scaled version of the distance of every point j from the starting point s: $dist(j,s)^2$. For the fully random version, a distance of 1 is used for every edge. Every point that is too far from the starting point is assigned 0 and will be removed from the returned result. The respective calculated distances are then scaled by the sum of all distances to sum to 1.

The final selected point is the new Waypoint candidate for adding and moving. For the case of removing a Waypoint, a new Waypoint is picked as well, even though in this case, the selected point will not be added into the tour (see figures 3.6-3.9b for a visualization of all options and how these operations alter the tour). To pick the Waypoint that will be moved or removed or to find the two Waypoints between which the new point should be added, all distances between the new Waypoint and all existing ones have to be calculated. The closest Waypoint of the current list will then be picked as visualized in the following Figures (3.6 to 3.9b):

The two illustrations show a path (black outline) with ten Waypoints that are evenly distributed. In the figure, a random point that has been picked for the following steps is highlighted in green. Additionally, four distances to the nearest Waypoints from the path are highlighted using the four green lines from the new point to the respective four closest

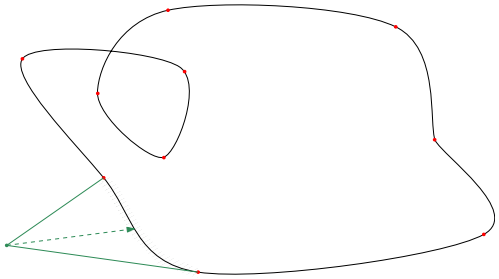
Waypoints. The third connection from the top is the shortest one, indicating the closest Waypoint.

After picking a Waypoint, three options are available. These moves are listed and explained in the following. Illustrations for each move are given below.

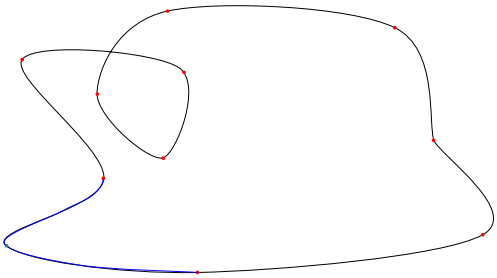
- **Adding:** from the two neighbors of the selected closest Waypoint on the existing tour, the closest one is picked and the new point is added in between, updating the path between the overall closest point to be the shortest path between the Waypoint and the new point and updating the path between the closer one of the neighboring Waypoints to be the shortest path between the new point and that Waypoint (see B.4)
- **Moving:** the selected closest Waypoint on the existing is deleted, a new shortest path from the predecessor of the deleted Waypoint to the new point and a shortest path from the new point to the successor of the deleted Waypoint is calculated and the new point is added as the moved version of the selected Waypoint (see B.5)
- **Removing:** the selected closest Waypoint is deleted and a new shortest path between the predecessor of the deleted Waypoint and the successor of the deleted Waypoint is calculated (see B.6)

Adding a Waypoint This move is visualized in Figures 3.7a and 3.7b. The first illustration (3.7a) highlights the distances to the closest Waypoint and the closer one of the respective neighbors. As shown in Figure 3.6, the second connection from the top would be shorter than the fourth one, which was chosen in this figure. This choice stems from the fact that only the single closest Waypoint is of interest for the initial selection. For every following move, only the respective neighbors can be used for any operation afterwards. The process of adding a Waypoint changes the path between the closest Waypoint on the existing line and the closer one of the respective neighbors. This path fragment is removed, which is indicated by crossing out the part of the previous path in Figure 3.7a. In illustration 3.7b, the new, changed path is visualized. All previous Waypoints are still on the path, but the changed fragment (highlighted in blue) now includes the additional Waypoint.

Moving a Waypoint The process of moving a Waypoint includes in principle removing one existing Waypoint and adding in the new random point. Figure 3.8a visualizes the closest Waypoint on the current path. The dashed arrow indicates how this chosen Waypoint will be moved towards the new point. For this move, both adjacent path fragments have to be updated. The new path with the moved Waypoint is illustrated in Figure 3.8b. The previous Waypoint is still visualized, but not part of the path anymore. The new Waypoint is included on the path and both adjacent path fragments are updated.

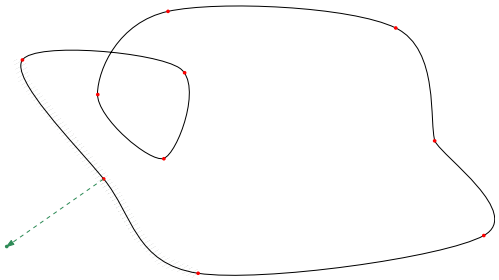


(a) This figure shows the path, the randomly picked point and the path to the closest Waypoint. The random point will be added between the closest Waypoint and the respective closest neighbor. The path fragment that will be changed through adding is crossed out.

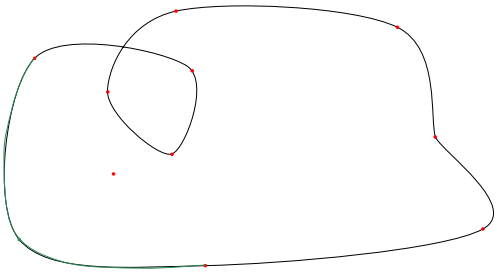


(b) This figure shows the path after the Waypoint was added.

Figure 3.7: These two figures display how a Waypoint is added into the tour

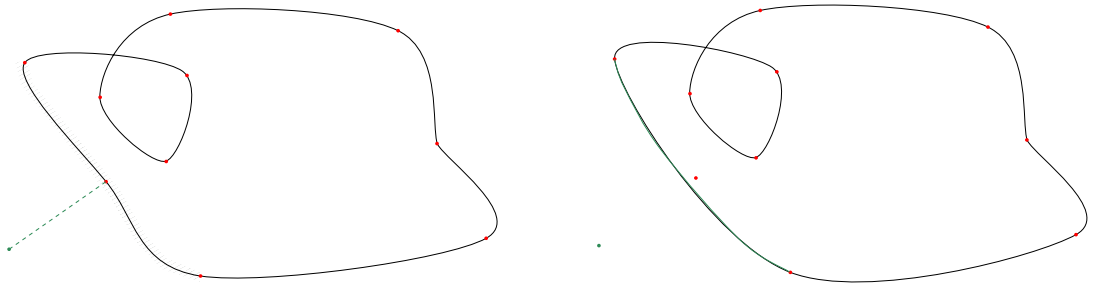


(a) This figure shows the path, the randomly picked point and the path to the closest Waypoint, which will be moved following the arrow to the random point. The path fragment that will be changed through moving is crossed out.



(b) This figure shows the path after the Waypoint was moved.

Figure 3.8: These two figures display how a Waypoint is moved in the tour



(a) This figure shows the path, the randomly picked point and the path to the closest Waypoint. The path fragment that will be changed through removing is crossed out.

(b) This figure shows the path after the Waypoint was removed.

Figure 3.9: These two figures display how a Waypoint is removed from the tour

Removing a Waypoint To remove a Waypoint, technically no new Waypoint is needed. However, choosing a random point and determining the closest of the existing Waypoints is using the same mechanism as for adding and moving, which keeps the applied methods consistent. While picking and removing one of the Waypoints without finding and using a random point would be possible, this approach would be an entirely different concept, ignoring the probability distributions used to determine a new random Waypoint. Thus, illustration 3.9a is very similar to Figure 3.8a. Only the arrowhead is missing since the closest Waypoint is not moved but removed from the path. For this move, again both neighboring path fragments have to be updated. The resulting path – containing neither the random point nor the previous Waypoint – is visulaized in illustration 3.9b.

For the new tour, the overall quality is calculated using Equation 3.8. Then, if the quality is better than the one of the previous tour, the new tour will always be accepted. If the quality is worse, the probability with which the tour is accepted anyways is calculated based on Equation 2.11. This step is done for a set number of inner runs, before the temperature is adjusted based on Equation 2.8.

To better illustrate the described added steps to choosing and building a solution, the pseudocode for the inner loop is given in Listing 3.2:

Algorithm 3.2 Generate neighborhood roundtrip j

```
1: pick random Waypoint based on probability distribution
2: calculate all distances to current Waypoints and pick closest
3: if # Waypoints < 3 then
4:     add new Waypoint between closest and that Waypoint's closest neighbor
5: else
6:     if # Waypoints ≤ 5 then
7:         randomly decide if to add or move
8:     else
```

```
9:        if 5 < # Waypoints < 15 then
10:         randomly decide if to add, remove or move
11:       else
12:         randomly decide if to remove or move
13:       end if
14:     end if
15: end if
16: calculate new quality
17: return new solution
```

3.2.3 Combinations

In addition to the pure versions of Ant Colony and SA, a few combinations with the two existing algorithms (Greedy and MinCost) as well as a combination of first using Ant Colony and then applying SA have been implemented. For this combination, only small changes had to be made to the existing code base, which will be shortly illustrated in the following paragraphs.

Ant Colony + Greedy and Ant Colony + MinCost To realize the combination of ant with the two already implemented algorithms, the changes are minor. First, the respective algorithm has to be run to create a base tour to build the ant algorithm from. Then, the initialization has to place trail intensity and pheromone on the edges of the existing tour. Lastly, the importance of the trail intensity α has to be higher than in the pure ant algorithm to make it more likely that the ants will follow the previous tour (see Equation 3.4).

SA + Greedy, SA + MinCost, SA + Ant Colony To implement the combinations with SA, similarly to the Ant Colony combination, the base tours have to be calculated first. Then, the Waypoint list can be created from the respective base tour that has been built, splitting the tour into 10 Waypoints and the paths connecting them. Based on this list of Waypoints, the normal SA algorithm as described previously can be performed. Other than that, no changes are needed.

Chapter 4

Evaluation

This chapter presents a comprehensive analysis of the overall performance of Ant Colony, SA, and their implemented combinations. In this analysis, several different parameter combinations are tested to achieve results of high quality. The performance of Ant Colony is influenced by various parameters that could be changed and tested. However, analyzing all possible options would exceed the scope of this thesis. Thus, the most important parameters with the highest impact on the results have been chosen. For SA, most variables could be tested in different combinations, however, the way a neighborhood was constructed has not been varied at all, since only two different options have been implemented (using a probability distribution or picking Waypoints randomly). Furthermore, the implemented combinations of Ant Colony with Greedy and MinCost, as well as the combinations of SA with Greedy, MinCost, and Ant Colony have been analyzed in comparison to each other.

For both Ant Colony and SA, determining the number of outer loops as well as the number of ants per outer run for Ant Colony and the number of repetitions per outer run for SA are crucial parameters. Furthermore, using the variables that can be changed by the user (importances of covered area, elevation, and edge profit) and analyzing their impact on the respective value as well as the overall quality are interesting factors to examine. Thus, these test cases are performed for both algorithms.

First, the number of outer runs that are needed to achieve relatively good results for both algorithms (see pseudocode 3.1 and B.3 respectively) is determined. Next, the inner loops and how the number of ants or the number of inner repetitions affects the quality are analyzed. Additionally, the impact of each quality feature (covered area, edge profit and elevation) depending on the respective importance is visualized. Lastly, the influence of each of these features on the quality is analyzed as well.

To achieve the following graphs, 30 test runs were conducted per varied variable unless stated otherwise. The number of test runs is relatively small since conducting more tests per value would have been too time consuming. Due to this small number of trials, the median results are presented and described. However, for most cases, the average results

are additionally presented in the appendix. For both algorithms, to determine the number of outer runs, 100 test cases were conducted, making the average results more reliable. Thus for these two cases, the average graphs were plotted.

For every case, the used base parameters that have not been varied are described in the corresponding paragraphs. Some parameters remained constant across all test runs, specifically:

- The starting point in front of Otto-Hahn-Straße 14
- The tour length of 4000m
- The desired tags for path type, surface, and surroundings
  - Path type: asphalt
  - Surface: paved, cobblestone, gravel, unpaved, compacted, fine gravel, rock, and pebblestone
  - Surroundings: forest: tree
- A maximum elevation of 100m
- A maximum steepness of 100%
- For Ant Colony, additional parameters include:
  - The evaporation rate of 0.4
  - The scaling value for penalties, in case a penalty was needed (e.g. using an edge twice) of 100
  - The initial trail intensity of 0.0001
  - The pheromone amount every ant can place per edge of 10
- For SA, additional parameters include:
  - The initial temperature value of 0.9
  - The number of Waypoints used to split up initial solutions of 10

The choice for the steepness was made due to the coarse grained elevation values. The inaccuracy of this data resulted in many parts with a calculated steepness exceeding 100% which did not represent the reality. Therefore, a high percentage was chosen to minimize the influence of the inaccurate data.

In all graphs, the x-axis represents the varied parameter and the y-axis shows the resulting values that will be analyzed. Only for Ant Colony, test runs with two varied parameters (namely α and β) were conducted. In these cases, the x- and y-axes show the varied parameters and the z-axis shows the resulting values that will be analyzed. To provide a clearer overview of the results, several different views for these cases have been created.

4.1 Ant Colony

For Ant Colony, the initial test cases focused on finding good values for α and β, as these two parameters significantly influenced all following results. After determining suitable values for α and β, the test cases outlined in the introduction, i.e. the number of outer runs, ants per run, and impact of the importances on the respective value and the overall quality, were performed.

The visibility for Ant Colony is calculated based on the respective importances for covered area (i_A), edge profit (i_p), and elevation and steepness (i_e). These parameters are varied in increments of 0.1 for one optimization option in all cases where the importance is plotted. When plotting the covered area importance, the importances of the edge profit and elevation are set to half of the remaining importance. For example, if covered area importance of 0.2 is selected, the remaining importance is 0.8 and thus, the importance for edge profit and elevation are each set to 0.4.

α and β Before running any test cases on user-defined parameters, fitting values for α and β need to be determined. These parameters influence the probability of choosing an edge and impact the entire algorithm. All test cases were performed with 100 ants per run and 25 outer iterations, with the importances set to 0.33 each. According to Dorigo et al. [18], the values of α and β should be in the range of $[0.5, 5]$. To identify good values, five scenarios were created:

- β set to 1 and α varying in $[0.5, 5]$
- α set to 1 and β varying in $[0.5, 5]$
- Both α and β varying in $[0.5, 5]$
- β set to 1.5 and α varying in $[0.5, 5]$
- α set to 0.8 and β varying in $[0.5, 5]$

For the five listed cases, the corresponding graphs are visualized in Figures 4.1 to 4.5 and additional visualizations can be found in the appendix (Figures A.8 to A.10). The first Figure (4.1) shows the median quality results of 30 test runs for $\beta = 1$ per α, where the steps started in 0.1 increments between 0.5 and 1.0 and then continued in 0.5 increments from 1.0 to 5.0.

The results as shown in Graph 4.1 indicate that for a constant $\beta = 1$, the best outcomes can be achieved with a low α between 0.5 and 2.0, and with $\alpha = 3.5$.

Similarly, the second figure displays results for $\alpha = 1$ and a varying β. Again, between 0.5 and 1.0, the steps on the x-axis increased by 0.1, and by 0.5 from 1.0 to 5.0.

Graph 4.2 indicates that a low β of $0.5, 1.5$, or $\beta = 3.5$ could yield good results. However, since both graphs have a constant second variable, the results could be dependent

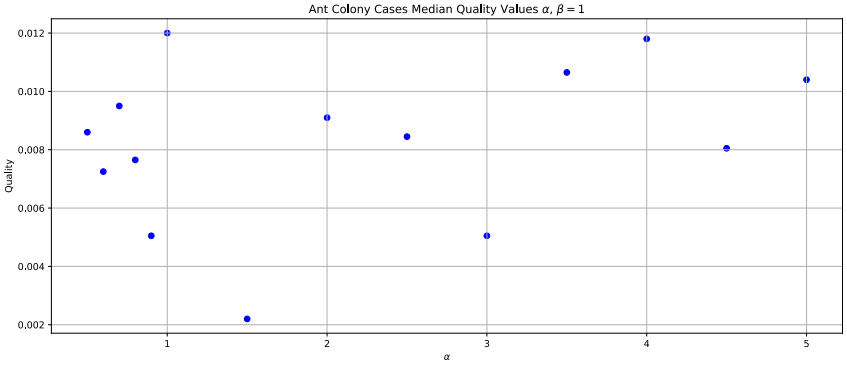


Figure 4.1: Ant colony median quality values over varied α, $\beta = 1$

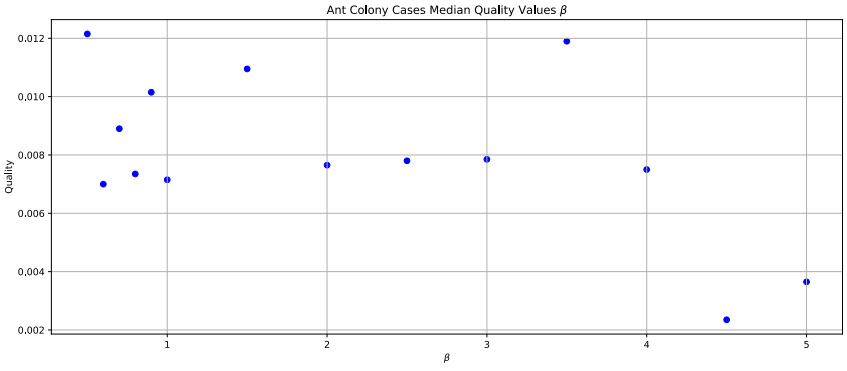
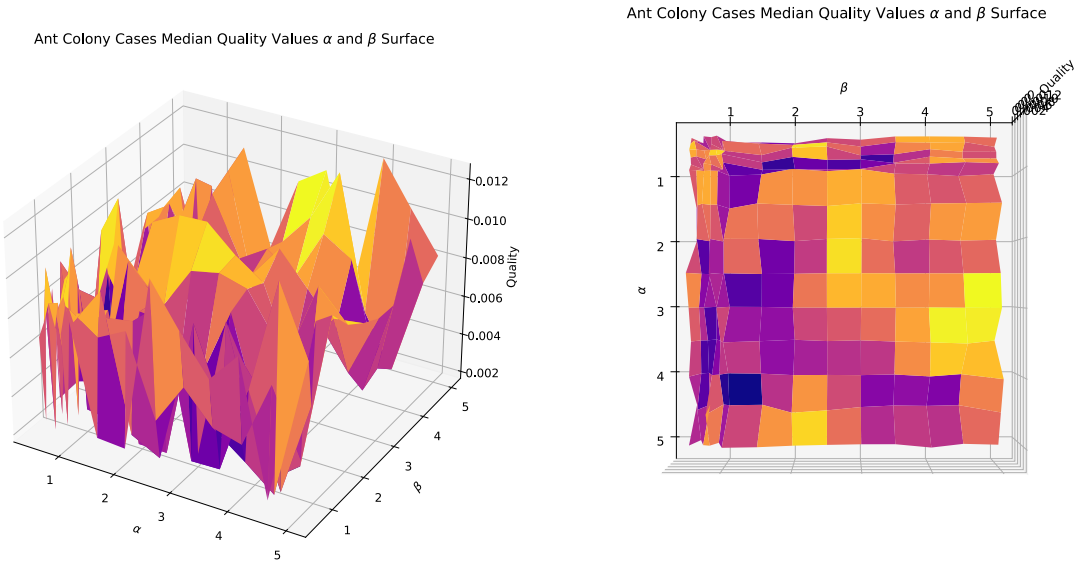


Figure 4.2: Ant colony median quality values over varied β, $\alpha = 1$



(a) Ant colony median quality values over varied α and β surface plot

(b) Ant colony median quality values over varied α and β surface plot top-down view

Figure 4.3: Plot of varied α and β in different views

on the respective value. To get a better overview over the effects of both α and β, additional test cases were run where both variables varied in the interval $[0.5, 5]$. Again, the step size between 0.5 and 1.0 was 0.1 and the step size between 1.0 and 5.0 was 0.5.

The applied color map highlights low values in blue and purple, middling values in magenta and pink, and high values in orange and yellow. In the top subfigure, the different points of the median of all test cases can be seen. The two figures below show a surface plot of the same test case, Subfigure 4.3a in a side view, and Subfigure 4.3b in a top-down view. All three figures show yellow points for a combination of very low α and low β values with $\alpha < 1.0$ and $\beta \leq 3.0$, for very high values of β ($\beta \geq 4.0$) and α values in a range of $2 \leq \alpha \leq 4.5$, and one high value for $\alpha = 5$ and $\beta = 2.5$.

The median quality plots in Graph 4.3 provide several potential options for α and β. However, the plots of the average qualities (see Figure A.10) show a peak at $\alpha = 0.8$ and $\beta = 1.5$, aligning with one of the options shown in the median quality plots in Figure 4.3.

Furthermore, tests with α set to 0.8 and a varying β as well as with β set to 1.5 and a varying α have been conducted. The results are shown in Figures 4.4 and 4.5.

The plots in Graphs 4.4 and 4.5 show that for a constant β of 1.5, higher quality values can be found for $0.6 \leq \alpha \leq 0.8$ and $\alpha = 4$. Of these possible options, the peak for $\alpha = 0.8$ matches the results of Figures 4.3. More high values present in the second plot (see Figure 4.5) where α was set to 0.8: For $\beta = 0.6$, $0.9 \leq \beta \leq 1.5$ and $3.5 \leq \beta \leq 4$, high quality values are returned. Of these possible options, the peak for $\beta = 1.5$ matches the results of Figure 4.3, highlighting that these choices result in a higher quality of the returned tour.

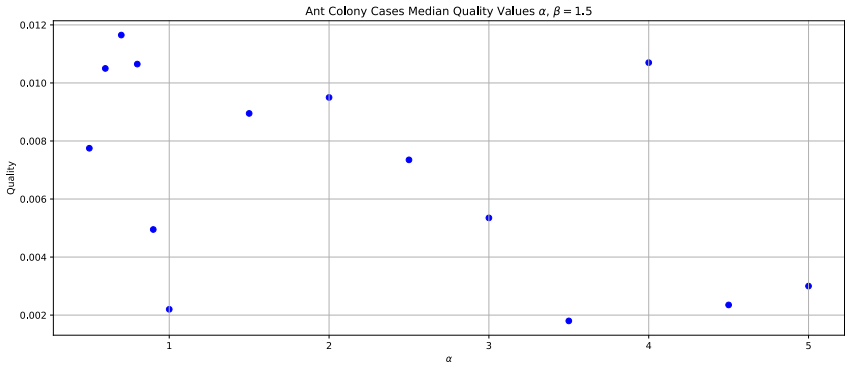


Figure 4.4: Ant colony median quality values over varied α, $\beta = 1.5$

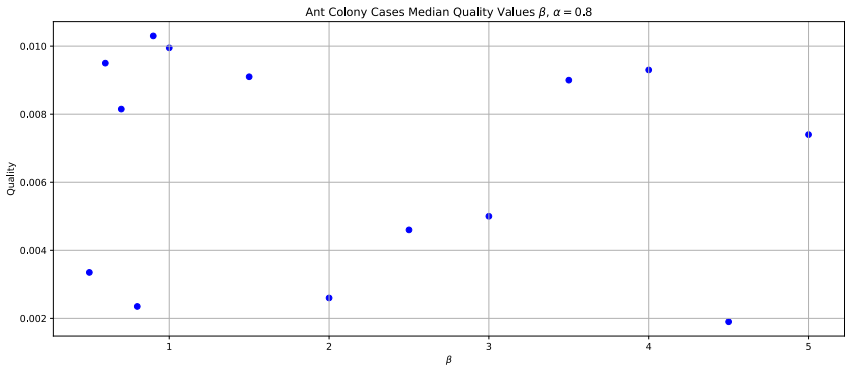


Figure 4.5: Ant colony median quality values over varied β, $\alpha = 0.8$

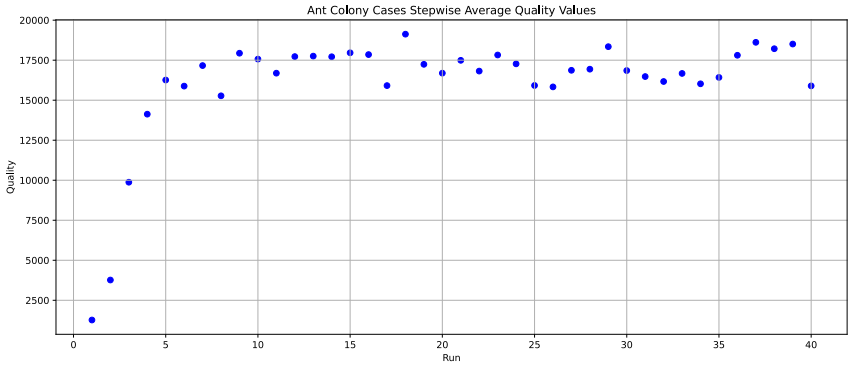


Figure 4.6: Ant colony average quality values over runs

Based on these results, α and β were chosen to be set to $\alpha = 0.8$ and $\beta = 1.5$ for the following test cases.

Quality Over Runs The next conducted test (see Figure 4.6) visualizes how many runs of the outer loop of Ant Colony are needed before the quality increase slows to a plateau. The respective figure shows the results of the average quality values from 100 tests per outer loop run, each executed with 100 ants. The "Run"-axis displays the number of runs in the outer loop, while the "Quality"-axis indicates the resulting quality value.

Figure 4.6 shows a steep increase in quality for 1 to 5 runs. After this initial phase, the increase slows to a near plateau and remains relatively even for the subsequent runs. The graph indicates that after 10-15 runs, no significant increase in quality is achieved, even with a larger number of runs. Thus, 20 iterations were deemed to be sufficient for generating good results in the following test cases.

Next, the number of ants and how they impact the quality of a given solution was examined. Graph 4.7 displays the median quality values of the results of 30 test runs per selected number of ants. Notably, some outliers of exceptionally high qualities are evident with fewer ants. These outliers are more apparent in the plot that shows all runs (see Figure A.11 in the appendix). However, the increase in median quality values from 100 to 500 ants demonstrates that, overall, more ants result in better outcomes.

For the following use cases, 100 ants were selected, since the case with 100 ants was the first that did not show a single outlier, but several higher quality values and a relatively high median value. Although using more ants improves the results, the run time, which is already relatively long for just 100 ants and the selected 20 iterations, also increases.

Covered Area To analyze the impact of the covered area importance on both the resulting covered area and general quality, several tests were conducted. The first Figure (4.8) shows the median covered area results for 30 test runs per importance value and

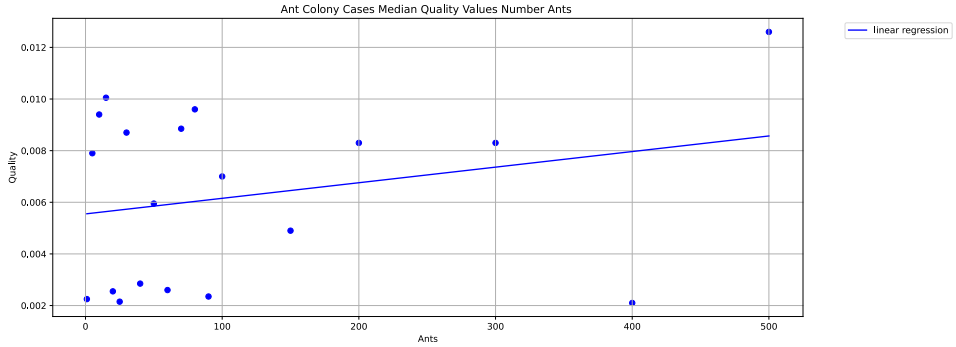


Figure 4.7: Ant colony median quality values over number ants

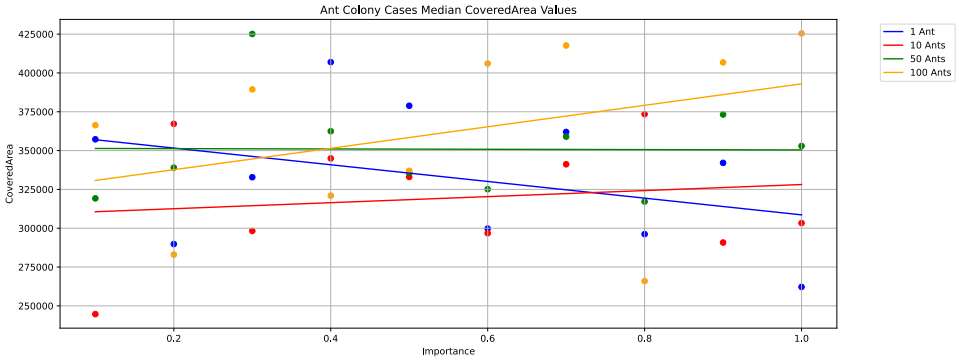


Figure 4.8: Ant colony median covered area values over covered area importance

the respective lines of best fit for the covered area importance values. The second Figure (4.9) shows the results of the median quality values with the same configuration. Graphs displaying the respective average values can be found in the appendix (see A.13 and A.14).

Graph 4.8 highlights that increasing the importance of the covered area, correlates with higher returned area values. Only in the case of a single ant does the graph show a decrease, indicating that with one ant, the randomness is too great to achieve the expected results. This observation is underscored by the plotted points (blue), which oscillate significantly between high and low values. For all other cases (10, 50, and 100 ants), an increase in covered area is observed. With the steepest increase for 100 ants, resulting in the highest returned area values for an importance of ≥ 0.4.

Graph 4.9 illustrates how the quality changes with varying importance for the covered area. In this visualization, a substantial increase in quality can be observed for 1 and 10 ants, a slight increase for 50 ants, and a decrease for 100 ants. This development suggests that the covered area is not the most impactful value in determining the quality. Since overall quality is used to calculate the probability of choosing the next point during tour

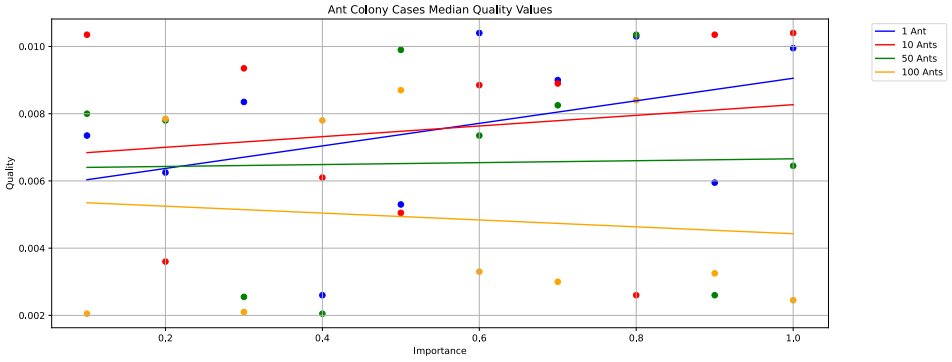


Figure 4.9: Ant colony median quality values over covered area importance

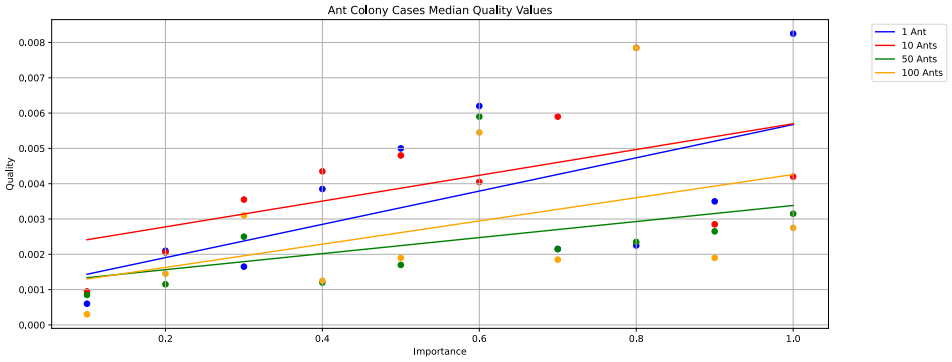


Figure 4.10: Ant colony median quality values over covered area importance, elevation importance and edge profit importance both set to 0

creation, this development indicates, that Ant Colony is negatively affected by a high importance of the covered area.

This result changes, when only the covered area is considered and both elevation importance and edge profit importance are set to 0 (see Figure 4.10). In the respective graph, with increasing covered area importance, the overall quality increases, indicating that neither the edge profit nor the elevation change impacts the quality, as was expected.

Overall, these results show that while the covered area does impact the quality, edge profit and elevation have a much more significant impact on the overall result. This larger impact is expected, as ants typically choose the next edge with the highest probability. The probability is calculated using covered area, elevation, and edge profit, but the current quality increase is valued higher than the overall quality increase.

All of the graphs show that quality values are lower when more ants are used. However, the covered area is generally larger with more ants – especially when considering extreme points. The graph analyzing the influence of the number of ants on the quality (Figure

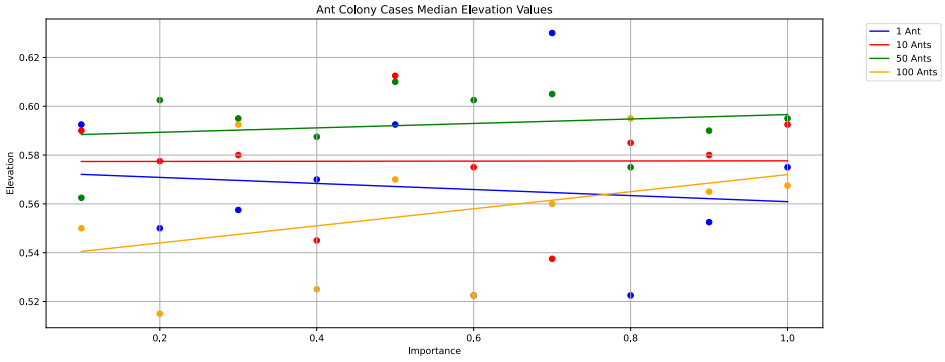


Figure 4.11: Ant colony median elevation values over elevation importance

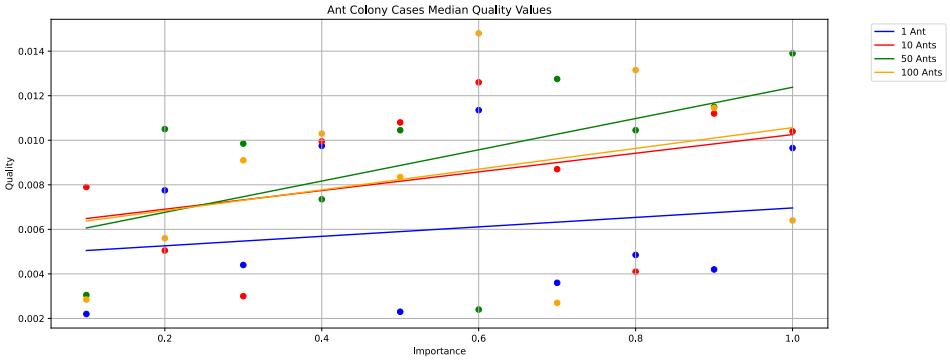


Figure 4.12: Ant colony median quality values over elevation importance

4.7) shows, that there are many fluctuations with fewer ants. Thus, although the quality seems mostly lower, using 100 ants yields more reliably good results.

Elevation To analyze the impact of the elevation importance on both the resulting elevation and on the general quality, several tests were conducted. The first Figure (4.11) shows the median elevation results for 30 test runs per importance value along with the respective lines of best fit for the elevation importance values. The second figure (4.12) shows the results of the median general quality values with the same configuration. Graphs displaying the respective average values can be found in the appendix (see figures A.15 and A.16).

Graph 4.11 highlights that as the importance of the elevation increases, the returned elevation value also rises. For this case, the returned values are not height meters but describe the elevation measure used in the quality calculation (see Equation 2.6). Similarly to the covered area, using fewer ants results in a slightly decreasing value when the importance is increased. This graph shows a downward slope for 1 and 10 ants and only a very slight increase for 50 ants. The only case that shows a definitive increase with higher

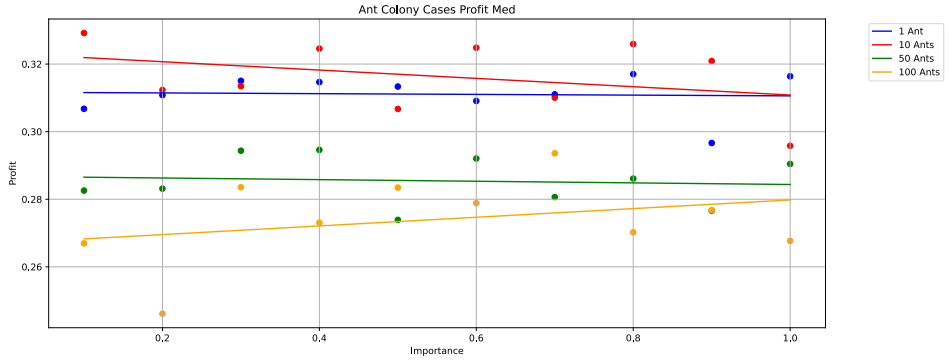


Figure 4.13: Ant colony median edge profit values over edge profit importance

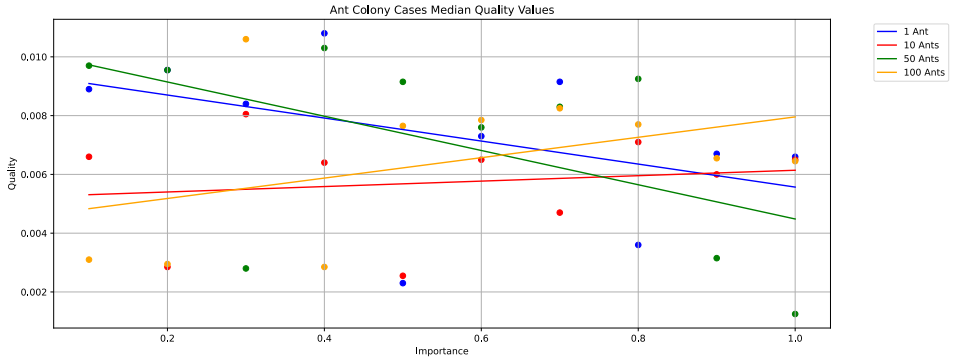


Figure 4.14: Ant colony median quality values over edge profit importance

importance is the one where 100 ants are used. Again, this behavior as well as the fact that the overall elevation measure is the lowest for the 100 ants can be explained by a less varying result which does not have as many outliers.

Graph 4.12 shows that for higher elevation importance the quality also increases. In this visualization, all qualities increase with the importance. Furthermore, the result for 100 ants has the second highest quality results. With rising elevation importance values, the quality increases, indicating that elevation has a larger impact on the quality than the covered area when using the Ant Colony algorithm.

Edge Profit To analyze the impact of the edge profit importance on both the resulting edge profit and on the general quality, several tests were conducted. The first Figure (4.13) shows the median edge profit results for 30 test runs per importance value, along with the respective lines of best fit for the edge profit importance values. The second Figure (4.14) shows the results of the median quality values with the same configuration. Graphs displaying the respective average values can be found in the appendix (see figures A.17 and A.18).

Graph 4.13 highlights that higher importance of the edge profit, results in increased edge profit values. Again, this increase is only evident when 100 ants are used. With fewer ants, the importance decreases the resulting edge profit values. These results, as well as the fact that the resulting profit is again lower than for fewer ants, are due to the higher variability when fewer ants are used. For 100 ants, increasing the edge profit importance results in higher edge profits.

Graph 4.14 shows that for 100 ants, the quality increases significantly with the importance of the edge profit. For fewer ants, the results are more mixed. For 1 and 50 ants, a significant decrease can be seen. For 10 ants, the quality increases slightly. These results highlight how variable the returned quality is when only a few ants are used. The result for the 100 ants fits into the results for the covered area, showing that the edge profit is more important for the resulting quality than the covered area. Both the elevation and the edge profit increase the quality when their importances are increased. However, the edge profit graph shows that the overall quality is mostly lower when more ants are used. This behavior is equivalent to the graph for elevation values.

Overall, all graphs show that all three user-defined values contributing to the quality rise with an increase in their respective importance. The quality only decreases for high covered area importances. The overall resulting tour does take the selected user preferences, which correspond to the importances into account. All in all, the graphs show that with fewer ants, high variability can cause results that do not match expectations. However, for the selected 100 ants, all cases display the expected behavior. Furthermore, the test cases have shown that for Ant Colony to achieve good results, the covered area is less important than the elevation and edge profits. This result can be attributed to the fact that for every single ant, the decision of which edge to choose is local. Thus, both the elevation and the edge profit have a larger weight than the covered area, which is a more global variable.

4.2 Simulated Annealing

For SA, the initial test cases focused on determining a good temperature function. The rate at which the temperature decreases is crucial as this development significantly influences the probability of accepting a worse solution, making this parameter the most vital variable to configure. Four different functions were selected based on the options presented in Section 2.3.2:

- $T_{i+1} = \frac{-|f(j)-f(i)|}{ln(r_i)}$
- $T_{i+1} = 0.5 \cdot T_i$
- $T_{i+1} = e^{-2} \cdot T_i$
- $T_{i+1} = \frac{T_i}{1+T_i}$

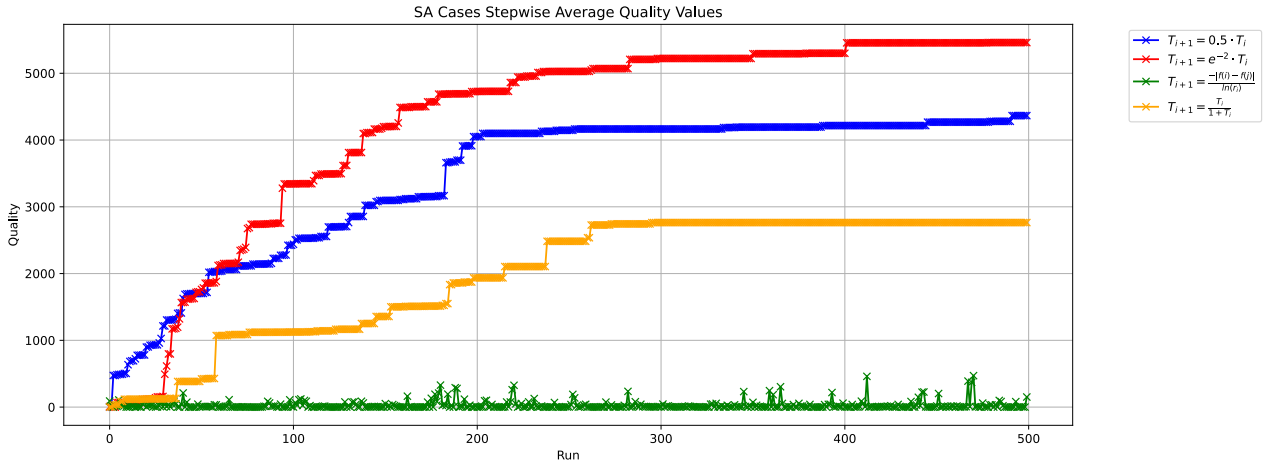


Figure 4.15: SA average quality values over runs

The test cases were executed according to the parameters outlined in this chapter's introduction, including the number of outer iterations, inner repetitions, and the influence of the importances on the respective variables and the overall quality. For SA, the user-defined importances are incorporated into the quality calculation and consistently impact the returned results.

The tests for the needed number of runs and the determination of a suitable temperature function have been executed in the same test case. For the importances, three cases have been constructed and executed, the same way the tests have been done for Ant Colony. The evaluation for the necessary number of runs and the suitable temperature function as well as for the importances were concluded and tested following the same methodology used for Ant Colony tests.

Quality Over Runs and Repetitions To determine a good number of runs and a fitting temperature function for achieving relatively good results, the median quality over number of runs performed has been plotted for the four different temperature functions in Figure 4.15. The respective figure presents the average quality values from 100 test runs per outer loop run, each using 10 inner repetitions with importances of 0.33 each. The "Run"-axis displays the number of runs in the outer loop, while the "Quality"-axis shows the resulting quality value.

Figure 4.15 presents several important findings: First, calculating the temperature using a multiplying factor – in this case e^{-2} – yields the best resulting quality. Furthermore, for most functions, a plateau is reached after 200 runs, except for the function denoted as $T_{i+1} = \frac{-|f(j)-f(i)|}{ln(r_i)}$ (green). This function does not improve the quality at all, but rather remains relatively stable with some outliers where the quality increases temporarily. In all other cases, where quality improvements occur, there are noticeable "steps" where the

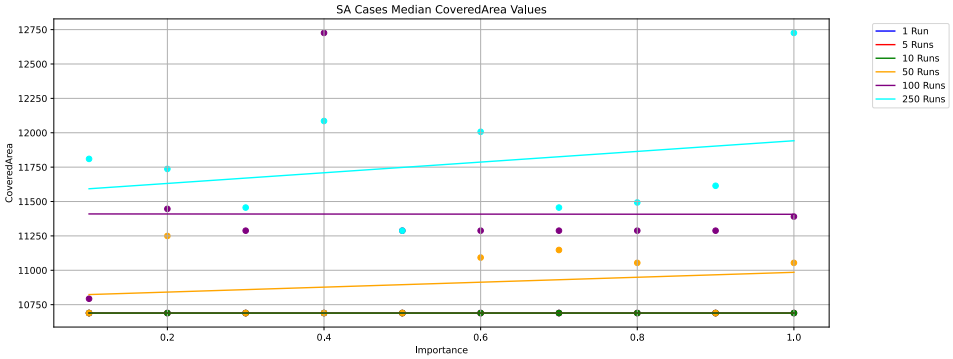


Figure 4.16: Simulated Annealing median covered area values over covered area importance

quality remains consistent for a few runs before increasing again. These steps are most noticeable in the yellow curve but are also evident in the blue and red graphs. This stepping behavior arises because the SA tends to find local optima. The algorithm can eventually escape these local optima with more runs. The obvious steps indicate where these local optima occur. Furthermore, these steps often overlap across the different temperature functions.

Using more internal repetitions increases the quality (see Figure A.20), but also inevitably significantly increases the run time. Given that the first figure shows a plateau at 250 runs and using 10 repetitions already yielded a median runtime of 22 seconds (see Figure A.19), using more internal repetitions would have increased the runtime excessively. Based on these results, the following tests have been performed with the temperature function that yielded the best results ($T_{i+1} = e^{-2} \cdot T_i$), 10 internal repetitions, and 250 runs.

Covered Area To analyze the impact of the covered area importance on both the resulting covered area and on the general quality, several tests were conducted. The first Figure (4.16) shows the median covered area results for 30 test runs per importance value, along with the respective lines of best fit for the covered area importance values. The second Figure 4.17 presents the median results for overall quality with the same configuration. Graphs displaying the respective average values can be found in the appendix (see A.21 and A.22).

Graph 4.16 highlights that the higher the importance of the covered area, the higher the returned area values. For 1, 5, and 10 runs, the covered area is significantly lower than for 50, 100, and 250 runs causing the three resulting lines of best fit for fewer runs to overlap. In the three cases with larger resulting covered area values, an increase can be seen as the importance value rises. Most importantly, for SA, the more runs that are performed, the higher the returned area values generally are.

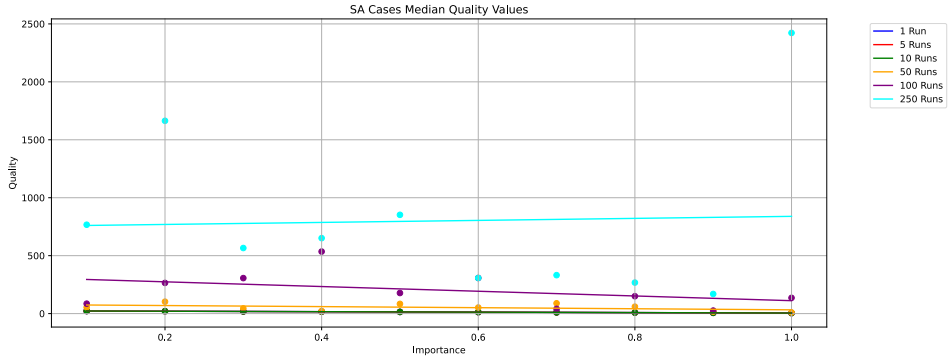


Figure 4.17: Simulated Annealing median quality values over covered area importance

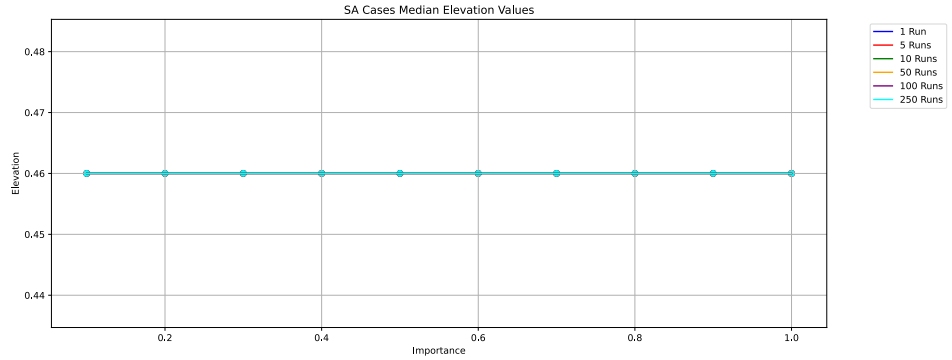


Figure 4.18: Simulated Annealing median elevation values over elevation importance

Graph 4.17 shows how the quality changes with a varying importance for the covered area. Here, the results for 1, 5, and 10 runs all overlap because the quality values are too low to be distinctly visible, similarly to the figure displaying the results for covered area. For 250 runs, a slight increase in the resulting quality can be seen, however, with fewer runs, the quality mostly decreases as the importance of the covered area nears 100%. This behavior indicates that with fewer runs, the area can negatively impact the resulting quality. However, for 250 runs, the impact is distinctly positive, showing that the covered area is relatively important for the quality of SA.

Elevation To analyze the impact of the elevation importance on both the resulting elevation and on the general quality, several tests were conducted. The first Figure 4.18 shows the median elevation results for 30 test runs per importance value, along with the respective lines of best fit for the elevation importance values. The second Figure 4.19 presents the results of the median general quality values with the same configuration. Graphs displaying the respective average values can be found in the appendix (see figures A.23 and A.24).

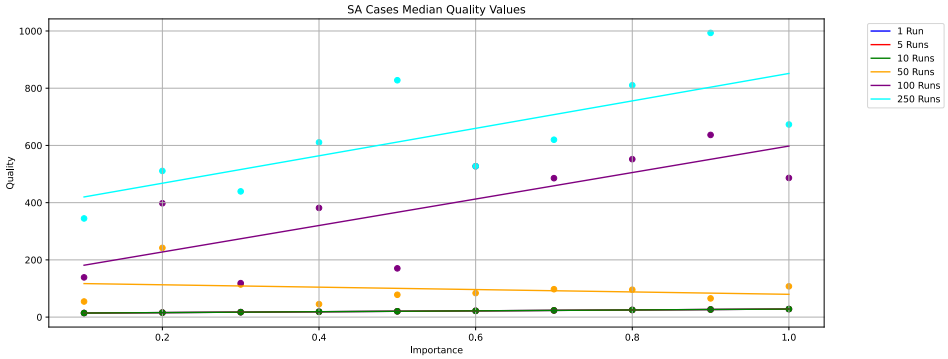


Figure 4.19: Simulated Annealing median quality values over elevation importance

Graph 4.18 shows that the elevation importance does not change the elevation resulting values at all. Instead, all results converge to the same value when considering the median. In the average graph, this development does look slightly different, however, this result shows, that the elevation does not have a significant impact. This smaller impact could be due to the selected starting point. Given that the SA with an empty starting solution and a probability function was used for all test runs, the possibility arises, that the probability functions did not offer many different options within the selected area, resulting in most returned paths having the same elevation values regardless of the number of runs or of the importance level chosen. In the average graph (see Figure A.23), there is a slight increase in the elevation value for most runs. The results for 10 and 50 runs are slightly decreasing, which could be due to not performing enough runs to reach a solution that better fits the chosen importances. However, the overall area where the tests were performed limits the available options, leading to minimal variability in the elevation values. All results are between 0.43 and 0.49, indicating very similar elevation values overall.

Graph 4.19 shows that for the elevation importance, the quality distinctly increases, especially for 100 and 250 runs. The results for 1, 10, and 50 runs overlap for this case as well. The increase in quality can be attributed to the lower importance of the edge profits and covered area, which increase the influence of the elevation result, indicating that the returned elevation result has already been very good.

Edge Profit To analyze the impact of the edge profit importance on both the resulting edge profit as well as on the general quality, several tests were conducted. The first Figure (4.20) shows the median edge profit results for 30 test runs per importance value and the respective lines of best fit for the edge profit importance values. The second Figure (4.21) shows the results of the general quality values with the same configuration. Graphs displaying the respective average values can be found in the appendix (see figures A.25 and A.26).

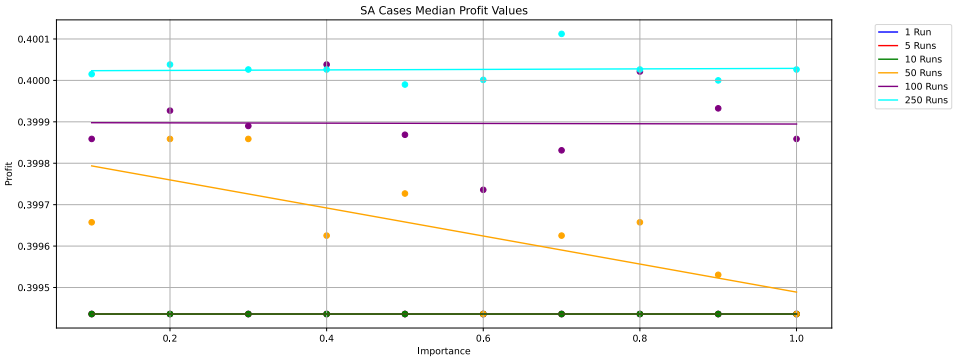


Figure 4.20: Simulated Annealing median edge profit values over edge profit importance

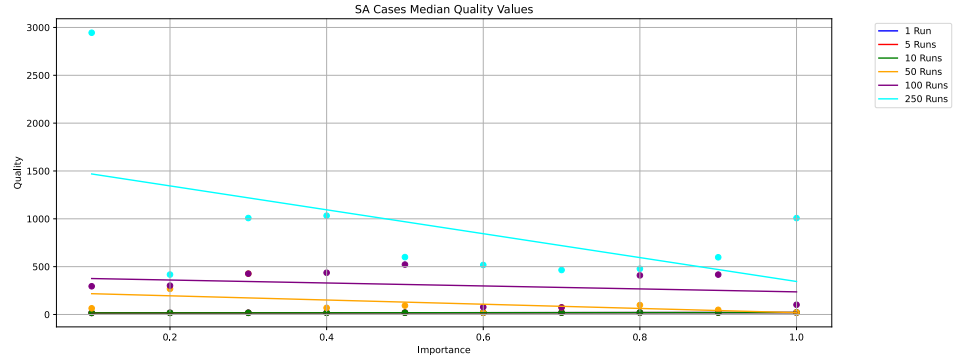


Figure 4.21: Simulated Annealing median quality values over edge profit importance

Graph 4.20 highlights that increasing the importance of the edge profit rises the returned edge profit values by a tiny amount. This increase is only prevalent for 250 runs, while for fewer runs, a slight, and for 50 runs even a steep decrease can be observed. As with the covered area, this development is most likely because not enough runs could be executed to reach a solution that results in an increase in the edge profits and the quality. Overall, the changes are within a very small spectrum of possible results, with most graphs varying by less than 0.0001 and only the graph for 50 runs varying by 0.0003. This highlights the results of the elevation graphs, that not much variability was possible for the specific configuration used for testing.

Graph 4.21 shows that for 250 repetitions, the quality decreases with the importance of the edge profit. For fewer repetitions, the resulting qualities stay on a relatively even level. In the respective average plot (see Figure A.26), this decrease and the difference in total value between the quality result for an edge profit importance of 0.1 and an edge profit importance of 1.0 is not as large as in the median plot. The overall quality is, however, still decreasing. This result highlights that for Simulated Annealing, optimizing for the edge profit negatively impacts the overall quality. Especially when viewed in combination with the results for the covered area, this finding indicates that the edge profits are less relevant for achieving high quality outcomes using SA.

Overall, all graphs demonstrate that the three user defined importances contribute to a respective rise of the returned value in the resulting tour. The covered area increases the quality slightly, while the edge profits cause a slight decrease. This behavior implies that the more global variable covered area is more important for the quality of Simulated Annealing, which overall focuses on optimizing a global result. The elevation had a significant impact for the given test case, likely because the elevation result was generally very good for all calculated solutions, thus having a more positive impact as the elevation importance was valued higher.

4.3 All Algorithms

Lastly, all implemented algorithms, including the combinations, were evaluated across varying maximum run times. Each algorithm was configured as previously specified: For Ant Colony and the respective combinations, 100 ants were used, for SA and all respective combinations, 10 inner repetitions were selected. All test cases were executed with equal importances of 0.33 for all the quality values. The maximum run times tested were 1, 2, 5, 10, 15, 20, and 30 seconds, with each maximum time being tested over 30 test runs.

Figure 4.22 shows the median quality values of the 30 test runs for all algorithms per allowed maximum run time. The ant algorithms, including both of their combinations did all produce low quality results compared to the Simulated Annealing ones and did not show

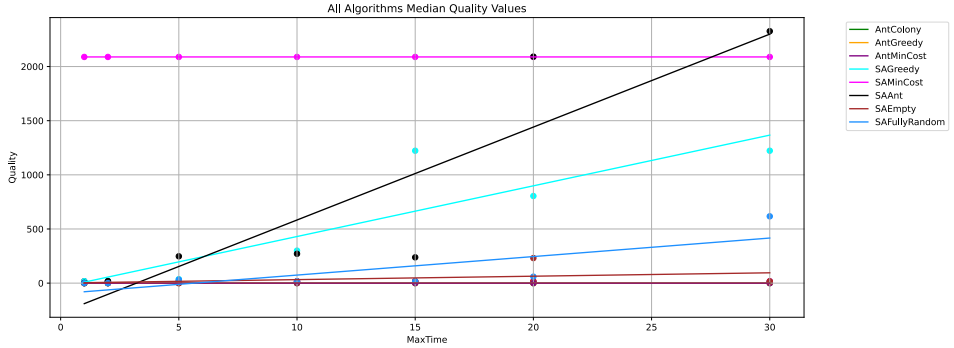


Figure 4.22: All implemented algorithms' median quality values over time

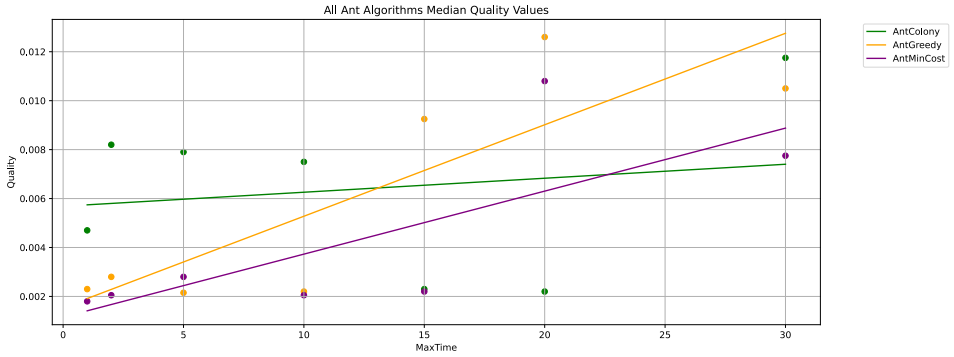


Figure 4.23: All ant variants' median quality values over time

a significant increase in quality. To better illustrate these changes, two separate graphs were plotted.

Figure 4.23 displays the performance of the three Ant Colony versions, using only Ant Colony or a combination with either the Greedy or the MinCost tour as a base. Both combinations show a large improvement in quality over time, indicating that the base tours are optimized considerably by the ants. However, the basic Ant Colony algorithm, only shows a slight quality increase. Overall, all quality values are much lower than the SA results, even though an increase can be seen for all cases.

The results indicate that Ant Colony is not the most suited for finding a good tour. Even though the algorithm does take all user configurations into account, the local decisions every ant made, result in a relatively low overall quality.

The final plot (Figure 4.24) provides a more detailed comparison of the various SA implementations. In this graph, the `SimulatedAnnealingEmpty` is the standard SA algorithm used in all previous test cases. This implementation started with an empty base tour and a probability configuration. The fully random version also started with an empty

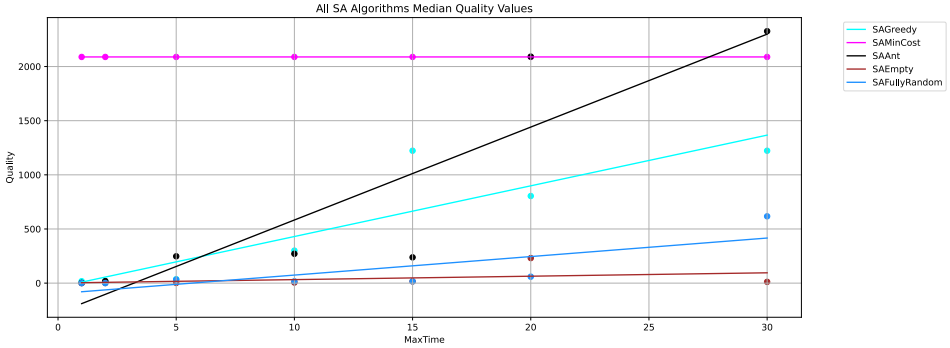


Figure 4.24: All SA variants' median quality values over time

tour but lacks a probability distribution for node selection. The other three combinations used their respective referenced algorithm as base tours to optimize.

This graph indicates that the combination of MinCost and the SA algorithm did not significantly improve. However, using the shortest allowed run times, the result is still of much higher quality than the Ant Colony optimized one. Furthermore, the empty solution with a probability distribution had the lowest overall quality, with only minor improvements over time. The fully random version showed a slightly steeper increase but both algorithms that start with an empty configuration remained at the lowest quality level. However, the combination with the Greedy base tour as well as with the Ant Colony have a very steep increase.

Overall, the graphs show, that the combination of MinCost and Simulated Annealing results in a very good solution even for short run times. The pink line that displays the results of this combination is the same as the results for MinCost alone (see Figure A.27), indicating, that for the median of all results, no improvement could be made on the MinCost result. However, the average graphs show a different behavior (see Figures A.28 and A.29). In these graphs, all SA variants exceed the quality of the MinCost result after a runtime of at most 12 seconds. This different behavior for the averages indicates that there are some tours that can significantly improve the quality of MinCost, but not enough higher quality tours were calculated to achieve the same results in the median. The varying results could be due to the starting point or other starting conditions such as the selected maximum elevation or the selected desirable tags.

While all other algorithms show some quality improvements over time, only the combination of Ant Colony and Simulated Annealing improves enough to exceed the quality of the MinCost-Simulated Annealing combination. However, the results could look very differently for other importance combinations. The MinCost algorithm starts with a tour that has a very high covered area value, greatly impacting the initial quality, which is then

further enhanced by using Simulated Annealing. When the covered area importance is set very low, using this combination can quickly result in a less optimal tour.

The conducted test cases do not cover all possible parameters, configurations and are far from covering all combinations. For all algorithms, especially the combinations of Ant Colony and Simulated Annealing with other algorithms, several additional test cases are possible. However, conducting more than the presented cases significantly exceeds the time frame of this thesis and thus has to be part of possible future work (see Section 5.2).

Chapter 5

Conclusion

In conclusion, the implementations presented in this thesis have successfully achieved the goals set in the introduction (1.2): The previous version of Tour4Me was extended by both parameter options as well as two metaheuristic approaches. Additionally, combinations of these new metaheuristics and the existing algorithms were implemented. The front-end was redesigned and matched to the added customization options, resulting in a better overview and improving the overall layout. For the majority of data-points and tour lengths, creating roundtrips is now possible, using at least one of the available algorithms. The only exceptions are points that fall outside the valid graph or those lacking any connections to valid edges, which are not considered as valid starting points.

5.1 Results

Overall, several significant outcomes can be reported. The primary goal of this thesis was to develop a user-friendly application capable of computing roundtrips for various outdoor activities. This goal included enhancing the interface and incorporating more customization and algorithmic options. To evaluate the results, three areas need to be examined.

Front-end and New Customization Options The front-end was fully remodeled, replacing all pop-up dialogue windows with foldable menus. These menus are divided into a side menu, which contains all customization options, and a bottom menu, which displays information about the generated tours. Unlike the pop-up dialogues, using foldable menus allows the user to keep all options visible as needed. Furthermore, relocating the customization options to a side menu prevents the interface from appearing too cluttered, which makes offering additional customization options easier.

To further prevent the user from feeling overwhelmed with options, preset activities and default values were established. For example, when a user selects “hiking” as an activity,

the elevation is set to 150 meters and the steepness to 100%. The default tour shape is set to the round option. The surroundings are pre-selected with mountain values. Path types such as cycleways and residential areas are deemed undesirable, while footways, paths, and tracks are preferred. Similar presets are available for running or cycling, filling in most options with values suited to the respective activity.

Moreover, several options remain hidden until the user opts to view them. The tour shape can be chosen from the three options of round, U-turn, and complex, but a fourth customization option is also available. Choosing this case reveals three additional sliders with which the user can change the importances for covered area (roundness), elevation and edge profits according to their preferences. The surrounding variables are also only displayed if the user chooses to pick one of the higher level options that are presented in the respective drop-down menu. To enhance comprehension and usability, information buttons were added to each customization option, providing brief explanations of their functions and effects when hovered over with the mouse.

In general, only the length and a tour shape need to be selected to generate a tour. Thus, even users who do not need or want as many customization options can still easily navigate and utilize the new interface.

The new footer menu offers a set of information for the generated tours, allowing users to easily check if all important variables have been taken into account. Moreover, this menu enables users to compare different tours, display the best ones, and delete unwanted results. A search bar has also been added to the map, allowing users to search for a specific starting point without having to manually navigate the map. This feature becomes particularly useful, when more cities are included in the database beyond Dortmund. Manually moving the map, zooming, and scrolling the map can be tedious and less user-friendly.

Several changes have been implemented to create a more user-friendly interface, making the whole app more usable overall. Integrating many additional customization options made this remodeling necessary. The new design features a nature-themed color scheme and aims for ease of use. Every added option includes an explanation, and as many parameters as possible are hidden until needed. The menus are self explaining and can be folded and unfolded according to the user's preferences.

Algorithmic Changes Two new algorithms and five additional combinations have been implemented. All of these algorithms demonstrated an increase in their quality over time and for all analyzed algorithms, the importances impacted the resulting values. The new algorithms allow the user to change the resulting tour according to their preferences. Both MinCost and Greedy show very little change when the importances are adjusted. However, Ant Colony and SA showed an increase in covered area, elevation measure, and edge profit when the respective importance was increased.

Overall, SA, MinCost, and their combinations return very good results with a high covered area. For users specifically desiring a U-turn or even a complex tour with many turns, Ant Colony could be more suitable for achieving the desired shapes.

The goal of calculating tours for (almost) any length while considering user preferences and operating in real-time was largely achieved. For very long tours of more than 10 kilometers, the algorithms are relatively slow in building the graph from the database. This issue is discussed in Section 5.2, which provides ideas for speeding up the graph creation. Other than that, with the calculated parameters for Ant Colony (25 runs and 100 ants per run) and SA (10 inner repetitions and 250 runs), very good results can be generated quickly. The SA calculations take a bit more time (see Figure A.19), but using fewer runs still yields very good results and significantly reduces the run time.

The new algorithms consider user preferences and can generate results for almost any length and starting point. Lengths exceeding the available data and starting points outside the current data table or at dead ends cannot be covered yet. However, with those exceptions, any length and starting point will return a tour.

In conclusion, the improvements across these three areas have resulted in enhanced versions of the previous Tour4Me application. The front-end is more user-friendly and displays new customization options. These new options are considered when calculating tours and the newly implemented algorithms take the user-selected importance values into account.

5.2 Future Work

For this thesis, several changes were implemented to develop an app offering extensive customization options and also utilizing different algorithms to find roundtrips that are suited to the user's preferences. However, there remain several areas for optimization and enhancement.

First, there are only few surrounding information available in the OSM data, despite the fact that the map visualization does have areas colored based on their surroundings. This coloring implies that the information is stored somewhere, and a method to extract them should exist. Using the visualization or the data used to find the correct colors could enable a much wider coverage of points with respective surrounding information. Adding these tags would result in many more edges being labeled according to the area they are in, leading to a more accurate portrayal and filtering of the actual nodes. Currently, the limited surrounding information makes the chance of this data impacting a tour very slim. Only about 8% of the nodes in Germany can be matched to surrounding information[1].

Additionally, the elevation data used could be upgraded to a more fine grained database, resulting in more precise values. Currently, a tour could end up with a steepness of over

[1] https://dashboard.ohsome.org/, last accessed: 03.06.2024

100%, while the real points differ only slightly in their elevation, due to the inaccurate elevation information. Superior versions might be available from NASA, however downloading and using them proved more complicated than the docker setup from Open-Elevation. More detailed research is needed to find better data that can be easily used and queried to provide a reliable solution.

Aside from fixing current data issues, several other enhancements and extensions are possible. The number of parameters to adjust user preferences can be increased, particularly for elevation information. Several additional constraints could be implemented:

- Steepness split into ascending and descending values: This feature would be beneficial for individuals with knee problems, as oftentimes steep ascents might not be problematic, but steep descents can strain their knees. Directed tours that can be adjusted for both steepness values separately would be extremely useful.

- Adding more detailed options: Currently, the overall elevation difference in the tour is calculated, but many other information could be useful: For example a limit on how long any part of the tour is allowed to continuously ascend or descend, which could be split into two variables. Additionally, the maximum ascent per 50, 100, or 1000 m could be more interesting than a maximum steepness of any small part.

- A visualization of the height profile of the tour: This profile view is mainly a front-end extension, but depending on the required data, the way paths are saved might need to be changed to allow for saving and returning elevation information of the tour for this visualization.

Additionally, as highlighted in the evaluation (see Chapter 4), several other test cases could be executed. For Ant Colony, the parameters for the pheromone amount of every ant, evaporation rate, penalty scaling and trail intensity were fixed for all test cases. Adjusting these variables could impact both the quality of the results and the overall run time. Similarly, for SA, variation in the initial temperature and the number of Waypoints could be explored. Furthermore, the way neighboring solutions are determined or the calculated probability distribution could be changed. These parameters are at the core of SA and could change the results significantly.

Aside from testing algorithm-specific configurations, different tour lengths and other tag combinations could be tested as well. The current test configuration uses only desirable tags, but experimenting with marking some values as undesirable and adding other desirable tag values could reveal how these tags influence the resulting tour. The same is true for the allowed maximum steepness, maximum elevation, and the general starting point. These values have been set to fixed values to assure for similar test cases, but building a tour from varying starting points and changing the allowed steepness and elevation could significantly impact the results as well.

Overall, there are several test cases that can be explored without needing to extend the current application. Finding a good configuration of all parameters could improve the results in terms of achieved quality but also regarding the covered area, elevation, and edge profit returned for respective high importance values.

Furthermore, many other algorithms, particularly different metaheuristics, could be implemented. One promising approach is the use of genetic algorithms. This extension would involve not only the genetic algorithm itself but also several combinations. These combinations would require more substantial modifications than the current ones, since generations of the algorithms have to be created, mutated, and optimized. To realize genetic changes, several possible approaches can be thought of. For a combination with the ant algorithm, changing the ant parameters and having the ants with the best results survive and create offspring could greatly increase the quality of the ant solutions. Combining genetic algorithms with SA to experiment with different temperatures, cooling schedules, different methods of creating neighboring solutions, and other base operations could also yield interesting and possibly significantly improved results.

Another possibility is to integrate existing tour data. Apps like Komoot provide user-generated tours that could be beneficial for some users. Furthermore, many hiking trails or predefined cycling tours have been created by cities. Using these routes can increase the options to choose from and also provide a base set of tours to run the implemented algorithms on.

Improving runtime is another crucial area for enhancement. Currently, graph creation takes a long time when calculating tours of 10 kilometers or more. With the implementation that directly accesses the database and reads nodes and edges sequentially, parallelization is not possible. Adopting a different method of database access, using a different database altogether, or implementing better import strategies could significantly improve run time. Additionally, the metaheuristics used could benefit from optimization in terms of data structures, implementation strategies, and parallelization.

In conclusion, while the current version of the app enhances the possibilities and customization options of the base application Tour4Me, there remain several interesting extensions, and numerous areas for improvement. These enhancements will further refine the application's functionality and user experience.

Bibliography

[1] Emile H. L. Aarts, Jan H. M. Korst, and Wil Michiels. "Simulated Annealing". In: *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Ed. by Edmund K. Burke and Graham Kendall. Boston, MA: Springer US, 2005, pp. 187–210. DOI: https://doi.org/10.1007/0-387-28356-0_7.

[2] Saurav Agarwal and Srinivas Akella. "The Correlated Arc Orienteering Problem". In: *Algorithmic Foundations of Robotics XV*. Ed. by Steven M. LaValle, Jason M. O'Kane, Michael Otte, Dorsa Sadigh, and Pratap Tokekar. Cham: Springer International Publishing, 2023, pp. 402–418. DOI: 10.1007/978-3-031-21090-7_24.

[3] Majid Alivand, Hartwig H. Hochmair, and Sivaramakrishnan Srinivasan. "Analyzing how travelers choose scenic routes using route choice models". In: *Computers, Environment and Urban Systems* 50 (2015), pp. 41–52. DOI: 10.1016/j.compenvurbsys.2014.10.004.

[4] Ozalp Babaoglu, Hein Meling, and Alberto Montresor. "Anthill: a framework for the development of agent-based peer-to-peer systems". In: *Proceedings 22nd International Conference on Distributed Computing Systems*. Proceedings 22nd International Conference on Distributed Computing Systems. Vienna, Austria: IEEE, 2002, pp. 15–22. DOI: 10.1109/ICDCS.2002.1022238.

[5] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. "Route Planning in Transportation Networks". In: *Algorithm Engineering: Selected Results and Surveys*. Ed. by Lasse Kliemann and Peter Sanders. Cham: Springer International Publishing, 2016, pp. 19–80. DOI: 10.1007/978-3-319-49487-6_2.

[6] Stuart J. H. Biddle. "Psychological benefits of exercise and physical activity". In: *Revista de psicología del deporte* 2.2 (1993), pp. 0099–107.

[7] Thomas Blondiau, Bruno van Zeebroeck, and Holger Haubold. "Economic Benefits of Increased Cycling". In: *Transportation Research Procedia*. Transport Research Arena TRA2016 14 (2016), pp. 2306–2313. DOI: 10.1016/j.trpro.2016.05.247.

[8] Olli Bräysy and Michel Gendreau. "Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms". In: *Transportation Science* 39.1 (2005), pp. 104–118. DOI: 10.1287/trsc.1030.0056.

[9] Kevin Buchin, Mart Hagedoorn, and Guangping Li. "Tour4Me: a framework for customized tour planning algorithms". In: *Proceedings of the 30th International Conference on Advances in Geographic Information Systems*. SIGSPATIAL '22: The 30th International Conference on Advances in Geographic Information Systems. Seattle Washington: ACM, 2022, pp. 1–4. DOI: 10.1145/3557915.3560992.

[10] Resul Cekin. "Psychological Benefits of Regular Physical Activity: Evidence from Emerging Adults". In: *Universal Journal of Educational Research* 3.10 (2015), pp. 710–717. DOI: 10.13189/ujer.2015.031008.

[11] Boris V. Cherkassky, Andrew V. Goldberg, and Tomasz Radzik. "Shortest paths algorithms: Theory and experimental evaluation". In: *Mathematical Programming* 73.2 (1996), pp. 129–174. DOI: 10.1007/BF02592101.

[12] Andrew R. Curtis, Tommy Carpenter, Mustafa Elsheikh, Alejandro Lopez-Ortiz, and S. Keshav. "REWIRE: An optimization-based framework for unstructured data center network design". In: *2012 Proceedings IEEE INFOCOM*. IEEE INFOCOM 2012 - IEEE Conference on Computer Communications. Orlando, FL, USA: IEEE, 2012, pp. 1116–1124. DOI: 10.1109/INFCOM.2012.6195470.

[13] Daniel Delahaye, Supatcha Chaimatanan, and Marcel Mongeau. "Simulated Annealing: From Basics to Applications". In: *Handbook of Metaheuristics*. Ed. by Michel Gendreau and Jean-Yves Potvin. Cham: Springer International Publishing, 2019, pp. 1–35. DOI: https://doi.org/10.1007/978-3-319-91086-4_1.

[14] Daniel Delling. "Time-Dependent SHARC-Routing". In: *Algorithmica* 60.1 (2011), pp. 60–94. DOI: 10.1007/s00453-009-9341-0.

[15] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. "Customizable Route Planning in Road Networks". In: *Transportation Science* 51.2 (2017), pp. 566–591. DOI: 10.1287/trsc.2014.0579.

[16] Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. "Engineering Route Planning Algorithms". In: *Algorithmics of Large and Complex Networks: Design, Analysis, and Simulation*. Ed. by Jürgen Lerner, Dorothea Wagner, and Katharina A. Zweig. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 117–139. DOI: 10.1007/978-3-642-02094-0_7.

[17] Narsingh Deo and Chi-Yin Pang. "Shortest-path algorithms: Taxonomy and annotation". In: *Networks* 14.2 (1984), pp. 275–323. DOI: 10.1002/net.3230140208.

[18] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. "Ant system: optimization by a colony of cooperating agents". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 26.1 (1996), pp. 29–41. DOI: 10.1109/3477.484436.

[19] Richard W. Eglese. "Simulated annealing: A tool for operational research". In: *European Journal of Operational Research* 46.3 (1990), pp. 271–281. DOI: 10.1016/0377-2217(90)90001-R.

[20] Matthias Ehrgott, Judith Y. T. Wang, Andrea Raith, and Chris van Houtte. "A bi-objective cyclist route choice model". In: *Transportation Research Part A: Policy and Practice* 46.4 (2012), pp. 652–663. DOI: 10.1016/j.tra.2011.11.015.

[21] Giorgio Gallo and Stefano Pallottino. "Shortest path algorithms". In: *Annals of Operations Research* 13.1 (Dec. 1, 1988), pp. 1–79. DOI: 10.1007/BF02288320.

[22] Andreas Gemsa, Thomas Pajor, Dorothea Wagner, and Tobias Zündorf. "Efficient Computation of Jogging Routes". In: *Experimental Algorithms*. Ed. by Vincenzo Bonifaci, Camil Demetrescu, and Alberto Marchetti-Spaccamela. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 272–283. DOI: 10.1007/978-3-642-38527-8_25.

[23] Michel Gendreau and Jean-Yves Potvin. *Handbook of Metaheuristics*. Vol. 146. International Series in Operations Research & Management Science. Boston, MA: Springer US, 2010. DOI: 10.1007/978-1-4419-1665-5.

[24] Alexander Gorobets. "Development of bicycle infrastructure for health and sustainability". In: *The Lancet* 388.10051 (Sept. 24, 2016), p. 1278. DOI: 10.1016/S0140-6736(16)31671-3.

[25] Jack E. Graver. "On the foundations of linear and integer linear programming I". In: *Mathematical Programming* 9.1 (Dec. 1, 1975), pp. 207–226. DOI: 10.1007/BF01681344.

[26] Stefan Irnich, Birger Funke, and Tore Grünert. "Sequential search and its application to vehicle-routing problems". In: *Computers & Operations Research* 33.8 (2006), pp. 2405–2429. DOI: 10.1016/j.cor.2005.02.020.

[27] Mads M. Jensen and Florian F. Mueller. "Running with technology: where are we heading?" In: *Proceedings of the 26th Australian Computer-Human Interaction Conference on Designing Futures: the Future of Design*. OzCHI '14: the Future of Design. Sydney New South Wales Australia: ACM, 2014, pp. 527–530. DOI: 10.1145/2686612.2686696.

[28] Faisal Khamayseh and Nabil Arman. "An Efficient Multiple Sources Single-Destination (MSSD) Heuristic Algorithm Using Nodes Exclusions". In: *International Journal of Soft Computing* 10 (2015). DOI: 10.3923/ijscomp.2015.301.306.

[29] Gilbert Laporte and Frédéric Semet. “5. Classical Heuristics for the Capacitated VRP”. In: *The Vehicle Routing Problem*. Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 2002, pp. 109–128. DOI: https://epubs.siam.org/doi/abs/10.1137/1.9780898718515.ch5.

[30] Eugene L. Lawler and D. E. Wood. “Branch-and-Bound Methods: A Survey”. In: *Operations Research* 14.4 (1966), pp. 699–719. DOI: 10.1287/opre.14.4.699.

[31] Benedikt Loepp and Jürgen Ziegler. “Recommending Running Routes: Framework and Demonstrator”. In: *Workshop on Recommendation in Complex Scenarios*. Workshop on Recommendation in Complex Scenarios. Vancouver, Canada, 2018.

[32] Ying Lu and Cyrus Shahabi. “An arc orienteering algorithm to find the most scenic path on a large-scale road network”. In: *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*. SIGSPATIAL ’15. New York, NY, USA: Association for Computing Machinery, Nov. 3, 2015, pp. 1–10. DOI: 10.1145/2820783.2820835.

[33] Florian F. Mueller, Shannon O’Brien, and Alex Thorogood. “Jogging over a distance: supporting a "jogging together" experience although being apart”. In: *CHI ’07 Extended Abstracts on Human Factors in Computing Systems*. CHI EA ’07. New York, NY, USA: Association for Computing Machinery, 2007, pp. 2579–2584. DOI: 10.1145/1240866.1241045.

[34] Matthew A. Nystoriak and Aruni Bhatnagar. “Cardiovascular Effects and Benefits of Exercise”. In: *Frontiers in Cardiovascular Medicine* 5 (2018), p. 135. DOI: https://doi.org/10.3389/fcvm.2018.00135.

[35] Shannon O’Brien and Florian F. Mueller. “Jogging the distance”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI07: CHI Conference on Human Factors in Computing Systems. San Jose California USA: ACM, 2007, pp. 523–526. DOI: 10.1145/1240624.1240708.

[36] Pekka Oja, Sylvia Titze, Adrian Bauman, Bas de Geus, Patricia Krenn, Ill Reger-Nash, and Theresa Kohlberger. “Health benefits of cycling: a systematic review”. In: *Scandinavian Journal of Medicine & Science in Sports* 21.4 (2011), pp. 496–509. DOI: 10.1111/j.1600-0838.2011.01299.x.

[37] Thomas Pajor. “Algorithm Engineering for Realistic Journey Planning in Transportation Networks”. PhD thesis. Karlsruhe, Germany: Karlsruher Institut f{\"u}r Technologie (KIT), 2013. DOI: 10.5445/IR/1000042955.

[38] Michalis Potamias, Francesco Bonchi, Carlos Castillo, and Aristides Gionis. “Fast shortest path distance estimation in large networks”. In: *Proceedings of the 18th ACM conference on Information and knowledge management*. CIKM ’09. New York,

NY, USA: Association for Computing Machinery, 2009, pp. 867–876. DOI: 10.1145/1645953.1646063.

[39] Gerhard Reinelt. *The Traveling Salesman: Computational Solutions for TSP Applications*. Berlin, Heidelberg: Springer, 2003. 231 pp. DOI: https://doi.org/10.1007/3-540-48661-5.

[40] Stefan Ropke. "Heuristic and exact algorithms for vehicle routing problems". PhD thesis. Technical University of Denmark, 2005.

[41] Gregory N. Ruegsegger and Frank W. Booth. "Health Benefits of Exercise". In: *Cold Spring Harbor Perspectives in Medicine* 8.7 (2018), a029694. DOI: 10.1101/cshperspect.a029694.

[42] Peter Sanders, Kurt Mehlhorn, Martin Dietzfelbinger, and Roman Dementiev. "Shortest Paths". In: *Sequential and Parallel Algorithms and Data Structures: The Basic Toolbox*. Ed. by Peter Sanders, Kurt Mehlhorn, Martin Dietzfelbinger, and Roman Dementiev. Cham: Springer International Publishing, 2019, pp. 301–332. DOI: 10.1007/978-3-030-25209-0_10.

[43] Christian Sommer. "Shortest-path queries in static networks". In: *ACM Computing Surveys* 46.4 (2014), pp. 1–31. DOI: 10.1145/2530531.

[44] Wouter Souffriau, Pieter Vansteenwegen, Greet Vanden Berghe, and Dirk Van Oudheusden. "The planning of cycle trips in the province of East Flanders". In: *Omega* 39.2 (2011), pp. 209–213. DOI: 10.1016/j.omega.2010.05.001.

[45] Attila Szabo and Júlia Ábrahám. "The psychological benefits of recreational running: A field study". In: *Psychology, Health & Medicine* 18.3 (2013), pp. 251–261. DOI: 10.1080/13548506.2012.701755.

[46] Cédric Verbeeck, Pieter Vansteenwegen, and El-Houssaine Aghezzaf. "An extension of the arc orienteering problem and its application to cycle trip planning". In: *Transportation Research Part E: Logistics and Transportation Review* 68 (2014), pp. 64–78. DOI: 10.1016/j.tre.2014.05.006.

[47] Jose Viña, Fabian Sanchis-Gomar, Vladimir Martinez-Bello, and Mari C. Gomez-Cabrera. "Exercise acts as a drug; the pharmacological benefits of exercise". In: *British Journal of Pharmacology* 167.1 (2012), pp. 1–12. DOI: 10.1111/j.1476-5381.2012.01970.x.

[48] Xueyang Wang, Chonghua Liu, Yupeng Wang, and Chengkai Huang. "Application of ant colony optimized routing algorithm based on evolving graph model in VANETs". In: *2014 International Symposium on Wireless Personal Multimedia Communications (WPMC)*. 2014 International Symposium on Wireless Personal Multimedia Communications (WPMC). Beijing, China: Institute of Electrical and Electronics Engineers (IEEE), 2014, pp. 265–270. DOI: 10.1109/WPMC.2014.7014828.

[49] Leonard M. Wankel and Bonnie G. Berger. "The Psychological and Social Benefits of Sport and Physical Activity". In: *Journal of Leisure Research* 22.2 (1990), pp. 167–182. DOI: https://doi.org/10.1080/00222216.1990.11969823.

[50] Muhammad R. Wayahdi, Subhan H. N. Ginting, and Dinur Syahputra. "Greedy, A-Star, and Dijkstra's Algorithms in Finding Shortest Path". In: *International Journal of Advances in Data and Information Systems* 2.1 (2021), pp. 45–52. DOI: 10.25008/ijadis.v2i1.1206.

[51] Shi-hua Zhan, Juan Lin, Ze-jun Zhang, and Yi-wen Zhong. "List-Based Simulated Annealing Algorithm for Traveling Salesman Problem". In: *Computational Intelligence and Neuroscience* 2016 (2016), e1712630. DOI: 10.1155/2016/1712630.

Appendix A

Additional Figures

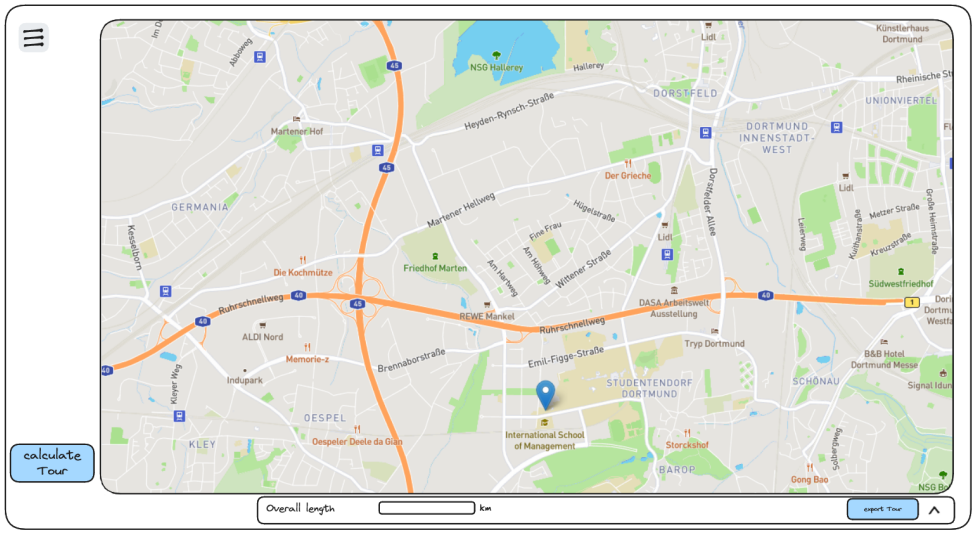
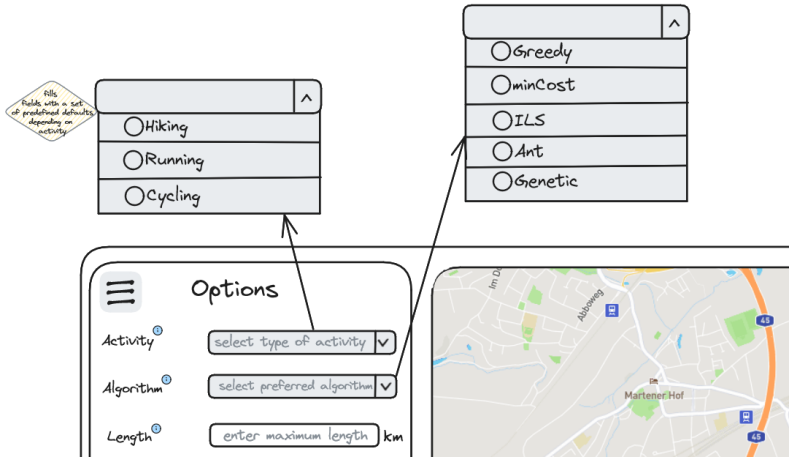
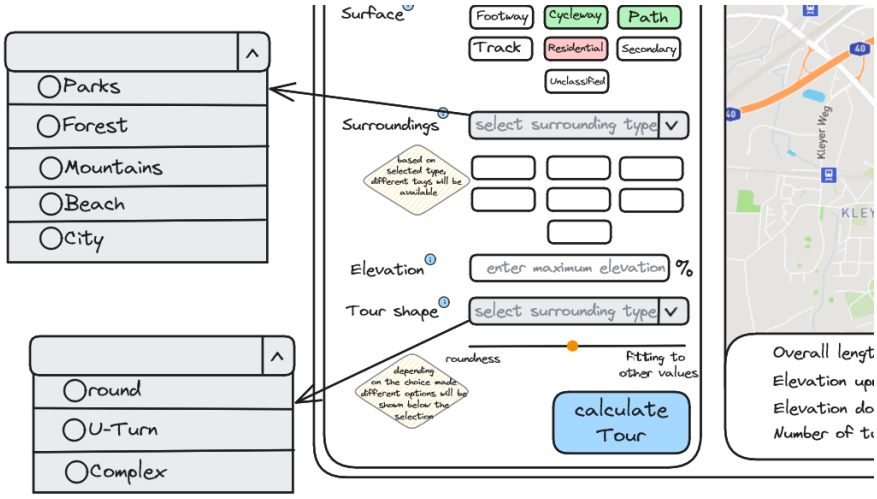


Figure A.1: Design concept for the front-end view with all menus folded



(a) Design concept for the front-end view,closeup of activity and algorithm dropdowns



(b) Design concept for the front-end view,closeup of a surrounding and Tour shape

Figure A.2: Closeups of the side menu design concept

Figure A.3: Design concept for the front-end view, closeup of the results view

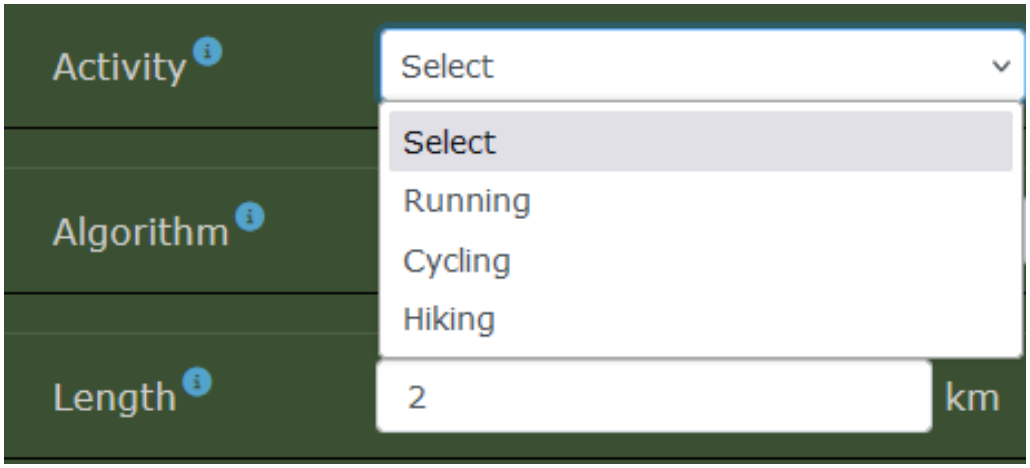


Figure A.4: The options to choose from when the *Activity* drop down is clicked

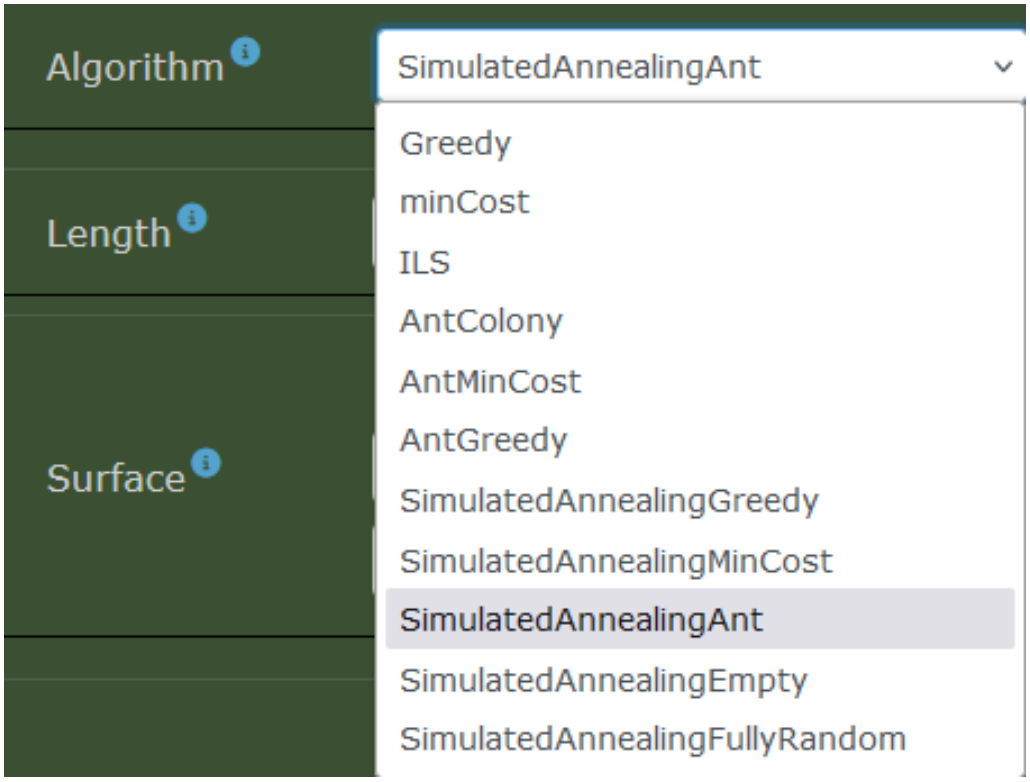


Figure A.5: The options to choose from when the *Algorithm* drop down is clicked

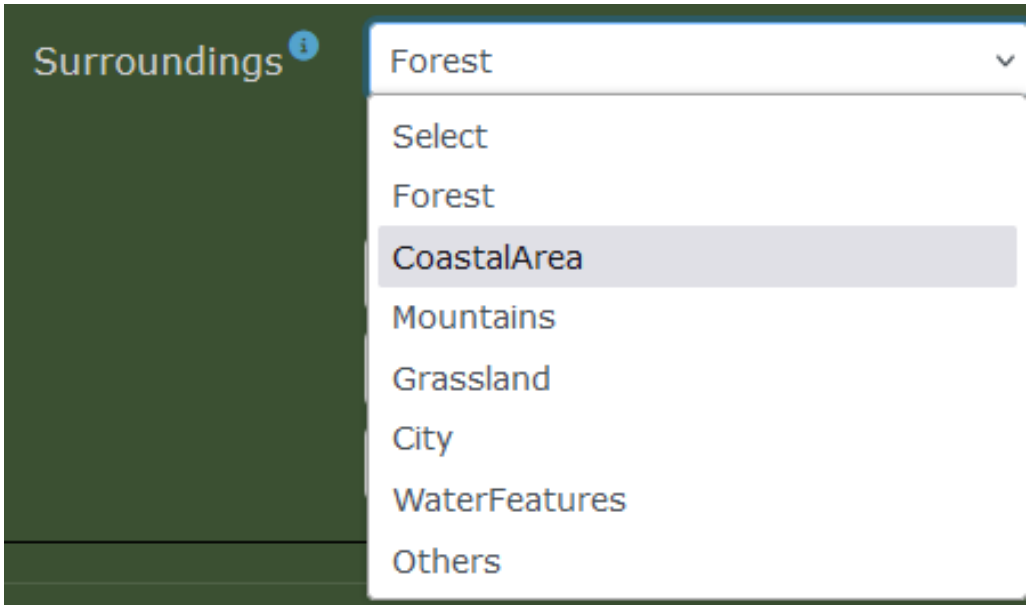


Figure A.6: The options to choose from when the *Surroundings* drop down is clicked

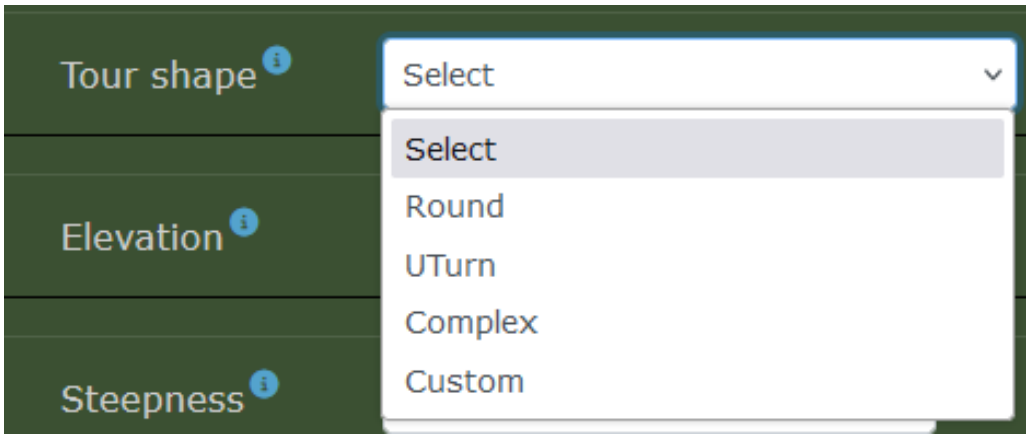


Figure A.7: The options to choose from when the *Tour Shape* drop down is clicked

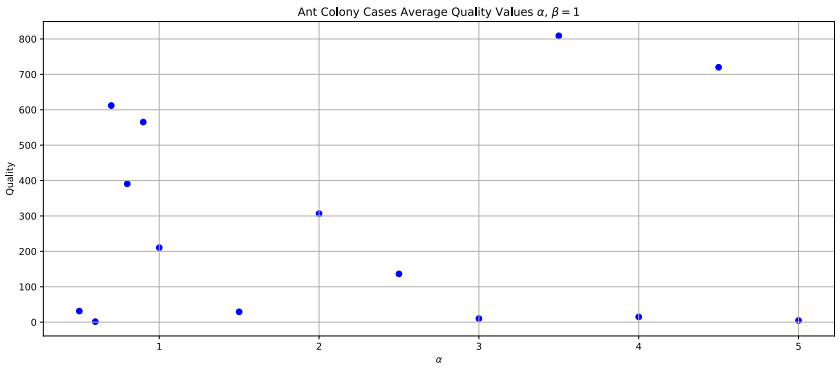


Figure A.8: Ant Colony average quality values over varied α, $\beta = 1$

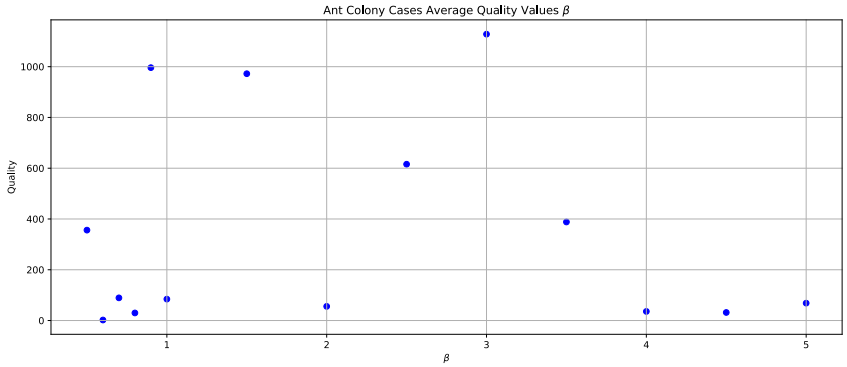
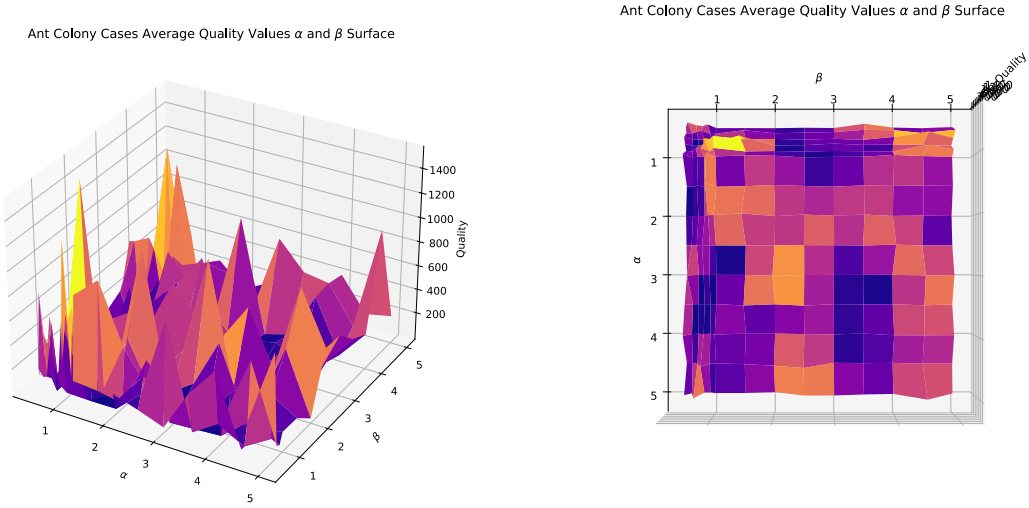


Figure A.9: Ant Colony average quality values over varied β, $\alpha = 1$



(a) Ant Colony average quality values over varied α and β surface plot

(b) Ant Colony average quality values over varied α and β surface plot top down view

Figure A.10: Plot of varied α and β in different views

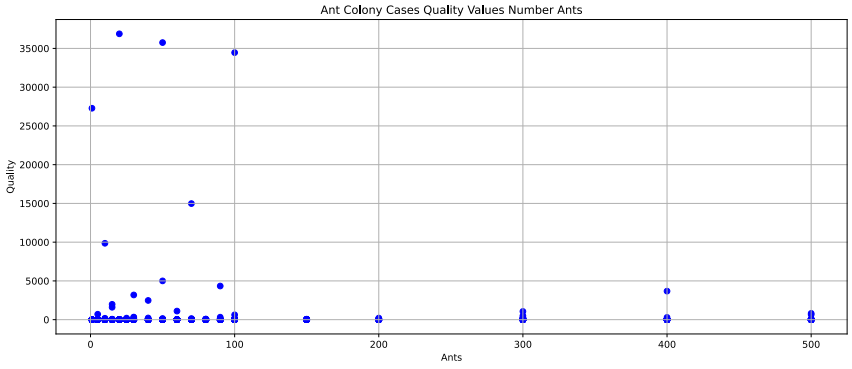


Figure A.11: Ant Colony average quality values over number ants

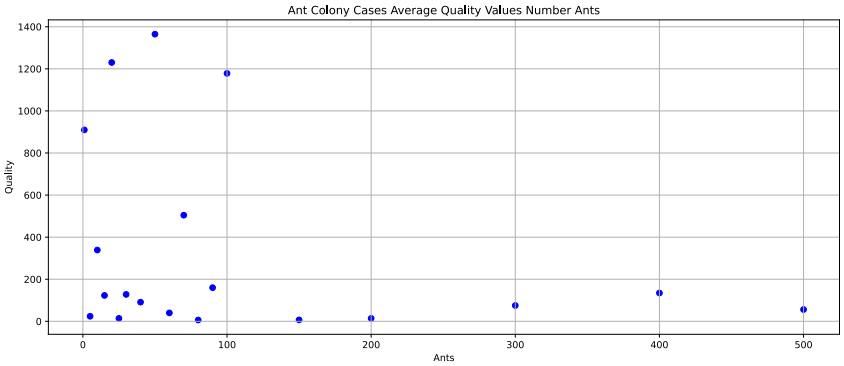


Figure A.12: Ant Colony average quality values over number ants

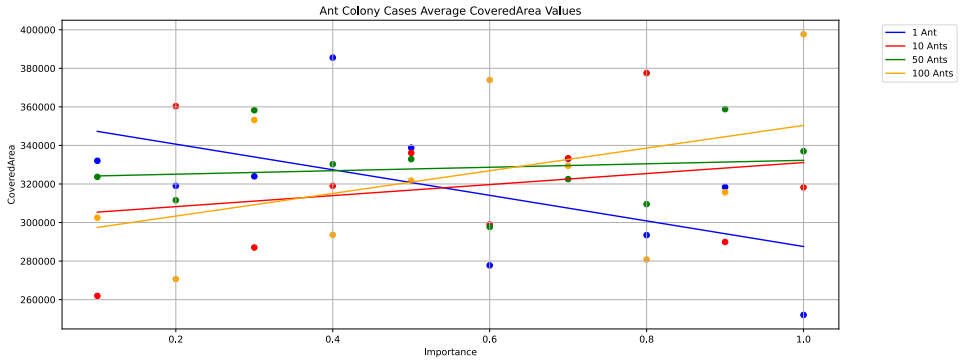


Figure A.13: Ant Colony average covered area values over covered area importance

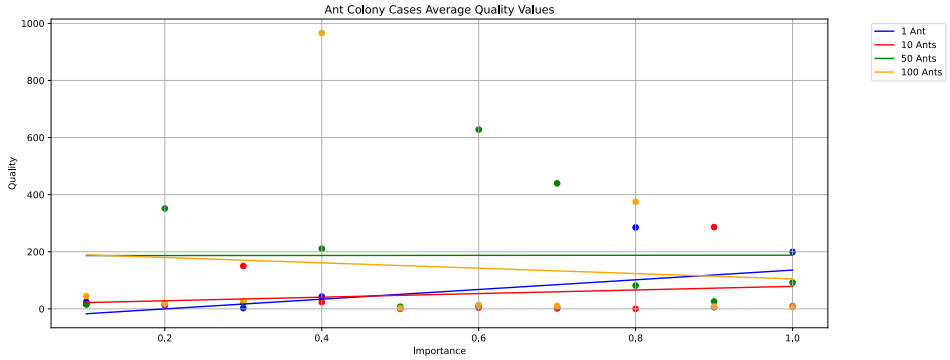


Figure A.14: Ant Colony average quality values over covered area importance

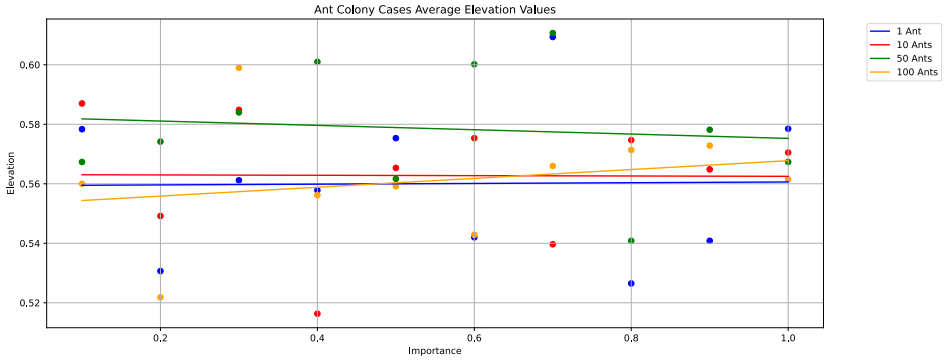


Figure A.15: Ant Colony average elevation values over elevation importance

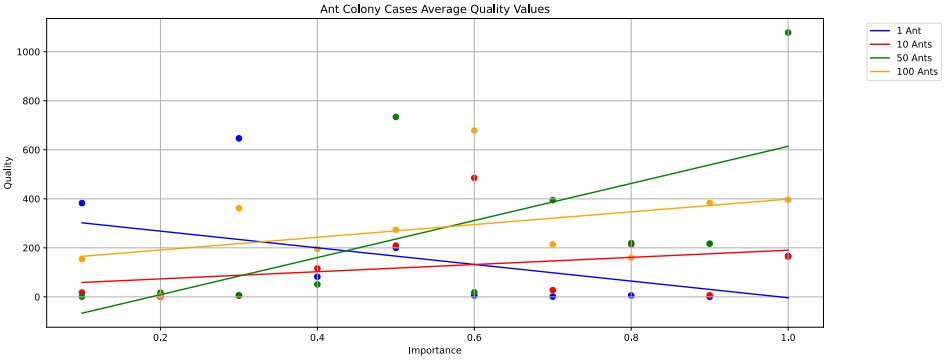


Figure A.16: Ant Colony average quality values over elevation importance

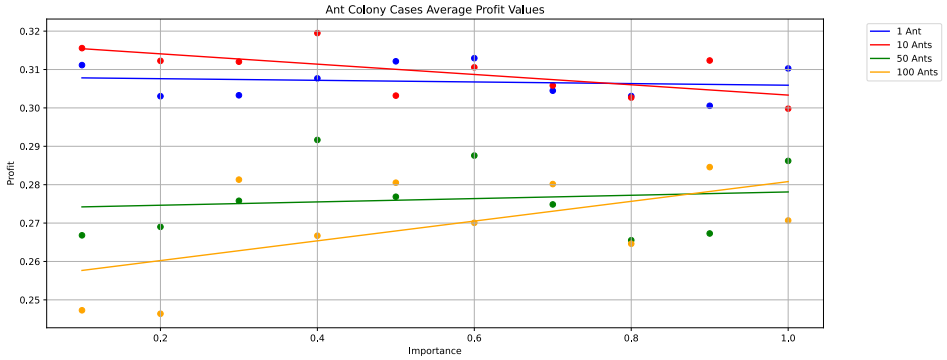


Figure A.17: Ant Colony average profit values over profit importance

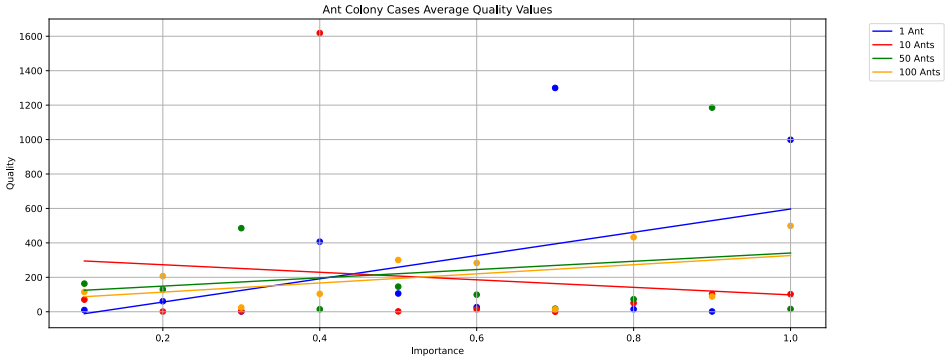


Figure A.18: Ant Colony average quality values over profit importance

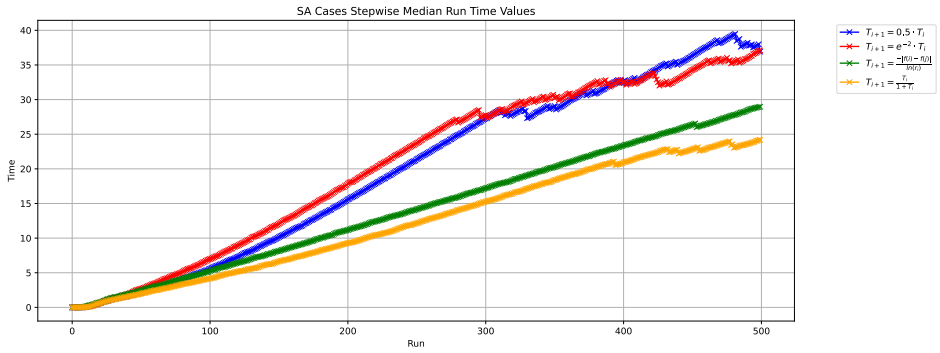


Figure A.19: Simulated Annealing median time values over runs

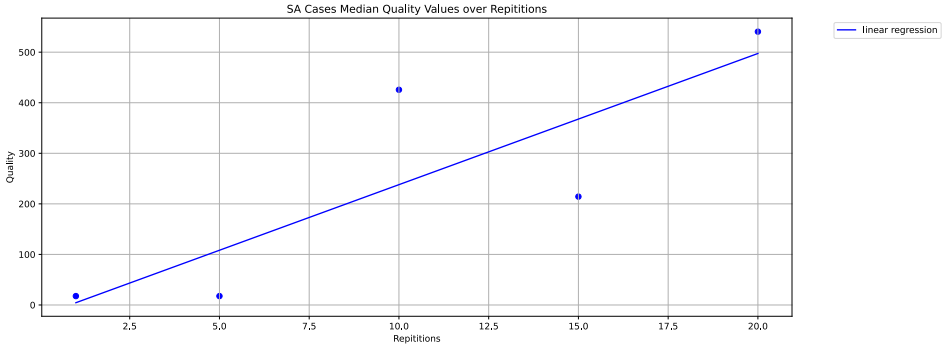


Figure A.20: Simulated Annealing median quality values over repetitions

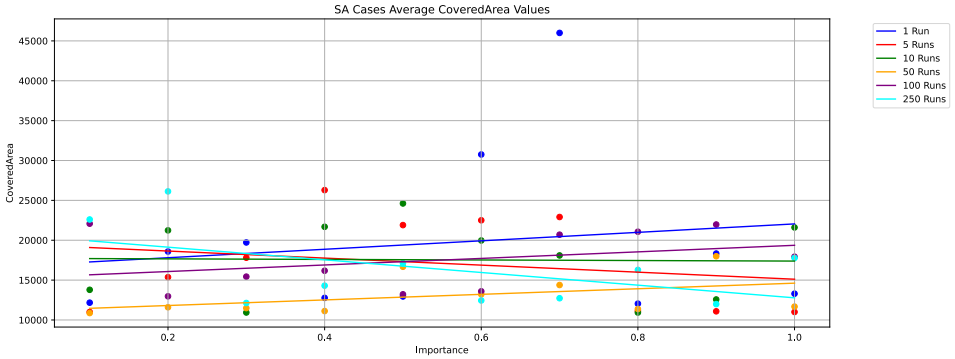


Figure A.21: Simulated Annealing average covered area values over covered area importance

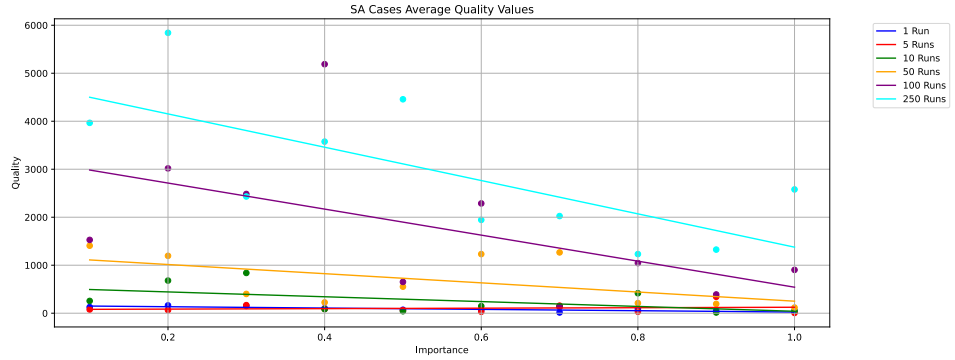


Figure A.22: Simulated Annealing average quality values over covered area importance

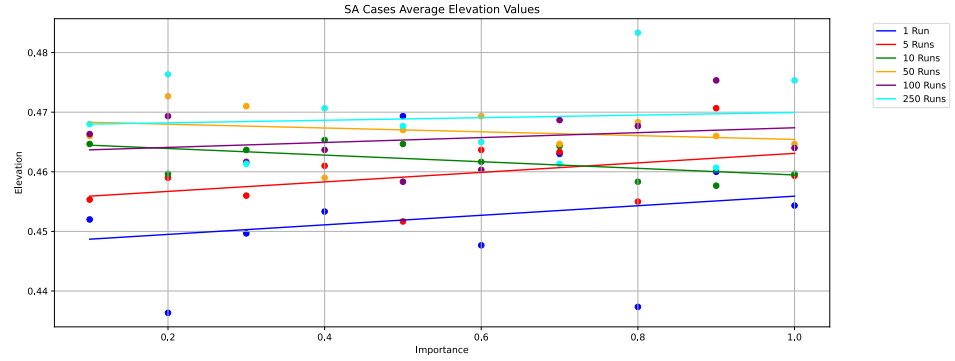


Figure A.23: Simulated Annealing average elevation values over elevation importance

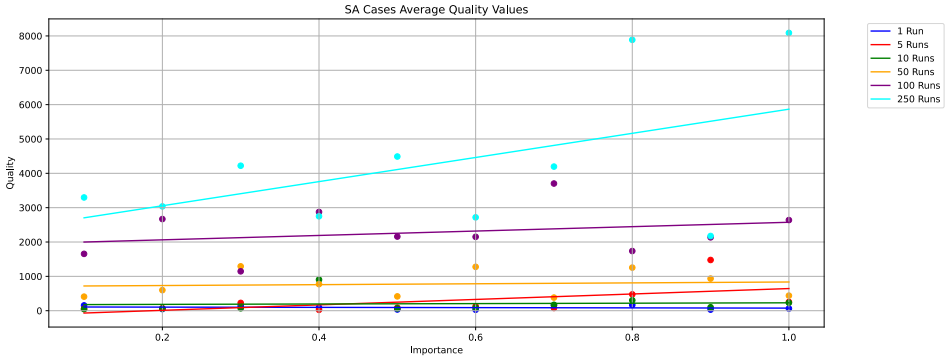


Figure A.24: Simulated Annealing average quality values over elevation importance

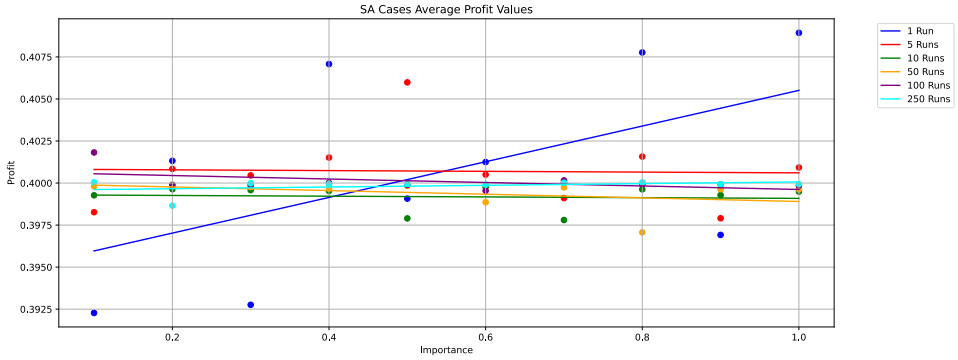


Figure A.25: Simulated Annealing average edge profit values over edge profit importance

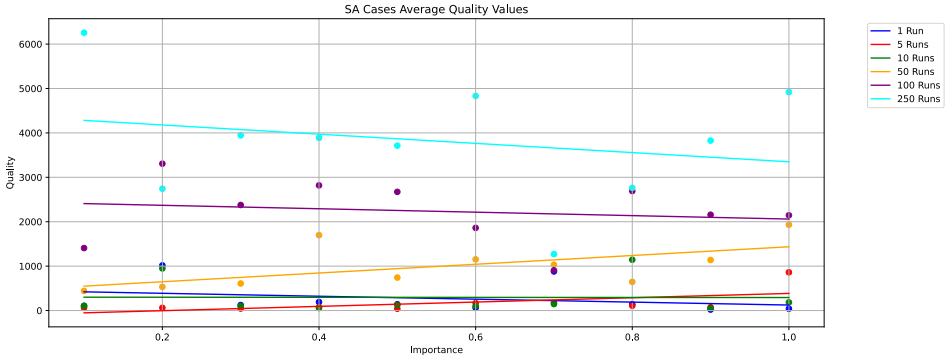


Figure A.26: Simulated Annealing average quality values over edge profit importance

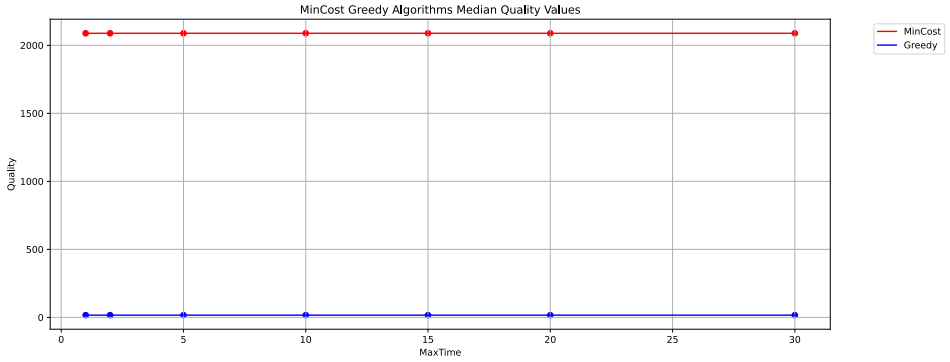


Figure A.27: Greedy and MinCosts median quality values over time

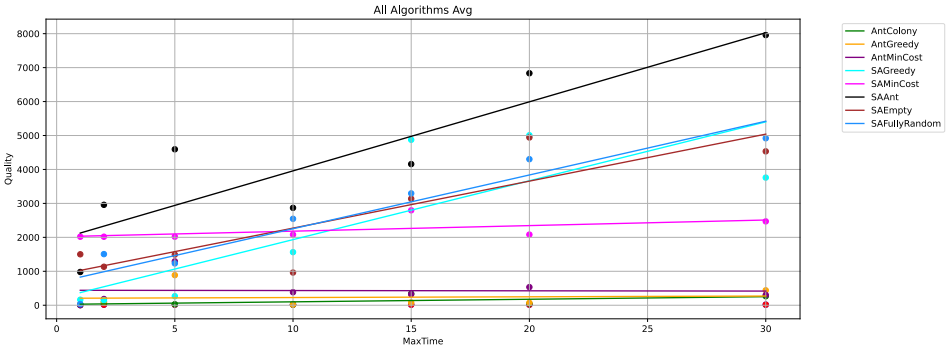


Figure A.28: All implemented algorithms' average quality values over time

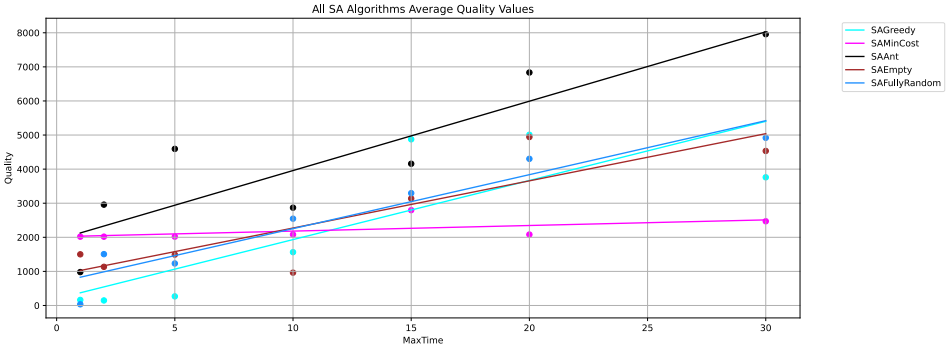


Figure A.29: All SA variants' average quality values over time

Appendix B

Additional Pseudocode

Algorithm B.1 Ant Colony

1: t $\leftarrow$ 0, NC $\leftarrow$ 0
2: set for all edges(i,j) initial $\tau_{ij}(t) \leftarrow c$ (trail intensiy),$\Delta\tau_{ij} \leftarrow 0$
3: Place all m ants on n nodes
4: s (`tabuList` index) $\leftarrow$ 1
5: **for** k = 1 to m **do**
6: add town of m to `tabuList`
7: **while** tabuList not full **do**
8: s $\leftarrow$ s+1
9: **for** k = 1 to m **do**
10: choose nest town to move to with probability $p_{ij}^k(t)$
11: move k-th ant to j
12: add town j to `tabuList`
13: **end for**
14: **end while**
15: **for** k = 1 to m **do**
16: move k-th ant back to `tabuList`(1)
17: calculate length L_k of k's tour
18: Update shortest tour
19: **for** edge(i,j) in allEdges **do**
20: **for** k = 1 to m **do**
21: $\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{L_k} \\ 0 \end{cases}$
22: $\Delta\tau_{ij} = \Delta\tau_{ij} + \Delta\tau_{ij}^k$
23: **end for**
24: **end for**
25: **end for**
26: **for** edge(i,j) in allEdges **do**

```
27:     τ_ij(t+n) = ρ · τ_ij(t) + Δτ_ij
28:   end for
29:   t ← t+1,NC ← NC+1
30:   for edge(i,j) in allEdges do
31:     reset Δτ_ij(t) ← 0
32:   end for
33:   if NC < NC_MAX and not stagnating then
34:     empty tabuList
35:   else
36:     return shortest tour
37:   end if
38: end for
```

Algorithm B.2 High level Simulated Annealing

```
1: i := Select init roundtrip
2: t := Select init temperature
3: for i = 0 to numberRepitions do
4:   for i = 0 to numberRuns per temperature do
5:     j := Generate neighborhood roundtrip
6:     Calculate difference in quality d := f(j)-f(i)
7:     if d ≥ 0 then
8:       use neighboring solution as current best i ← j
9:     else
10:      random ← generate random(0,1)
11:      if random < exp(d/t) then
12:        use neighboring solution as current best i ← j
13:      end if
14:    end if
15:  end for
16:  update temperature t ← T(t)
17: end for
```

Algorithm B.3 Simulated Annealing

```
1: build Waypoint list (starting point only)
2: (optional; for weighted selection of points) calculateDistances
3: calculate probability distribution
4: t := Select init temperature
5: for i = 0 to numberRepitions do
6:   for i = 0 to numberRuns per temperature do
7:     j := Generate neighborhood roundtrip
```

```
8:      Calculate difference in quality d := f(j)-f(i)
9:      if d ≥ 0 then
10:       use neighboring solution as current best i ← j
11:     else
12:       random ← generate random(0,1)
13:       if random < exp(d/t) then
14:         use neighboring solution as current best i ← j
15:       end if
16:     end if
17:   end for
18:   update temperature t ← T(t)
19: end for
```

Algorithm B.4 Add `Waypoint`

1: calculate shortest path between closest `Waypoint` (a) and new point (n)
2: calculate shortest path between new point and closer one of the neighbors of closest `Waypoint` (b)
3: update path from closest `Waypoint` a
4: add new point n into `Waypoint` list
5: update path from new point n to b
6: update full tour to include path part a-n-b

Algorithm B.5 Move closest `Waypoint`

1: calculate shortest path between predecessor of closest `Waypoint` (a) and new point (n)
2: calculate shortest path between new point and successor of closest `Waypoint` (b)
3: update path from successor of closest `Waypoint` a
4: update closest `Waypoint` to be the new point n in `Waypoint` list
5: update path from new point n to b
6: update full tour to include path part a-n-b

Algorithm B.6 Remove closest `Waypoint`

1: calculate shortest path between predecessor of closest `Waypoint` (a) and successor of closest `Waypoint` (b)
2: update path from successor of closest `Waypoint` a
3: remove closest `Waypoint` from `Waypoint` list
4: update full tour to only include path part a-b

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kentlich gemacht habe.

Dortmund, 17.07.2024

Lisa Salewsky

Eidesstattliche Versicherung

(Affidavit)

Salewsky, Lisa

Name, Vorname
(surname, first name)

192742

Matrikelnummer
(student ID number)

☐ Bachelorarbeit
(Bachelor's thesis)

■ Masterarbeit
(Master's thesis)

Titel
(Title)

Customizable Roundtrips with Tour4Me

Metaheuristic Approaches for Personalized Running and Cycling Routes

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit mit dem oben genannten Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present thesis with the above-mentioned title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution before.

Dortmund, 17.07.2024

Ort, Datum
(place, date)

L. Salewsky

Unterschrift
(signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to EUR 50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the Chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, Section 63 (5) North Rhine-Westphalia Higher Education Act (*Hochschulgesetz, HG*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund University will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:*

Dortmund, 17.07.2024

Ort, Datum
(place, date)

L. Salewsky

Unterschrift
(signature)

***Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.**