

Masterthesis

**Customizable Roundtrips with Tour4Me**

Meta-heuristic Approaches for Personalized Running and  
Cycling Routes

Lisa Salewsky

July 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>



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Related Work . . . . .	3
1.1.1	Tour4Me . . . . .	3
1.1.2	Roundtrip paths . . . . .	6
1.1.3	Other running related research . . . . .	9
1.1.4	Apps that assist with sports . . . . .	9
1.1.5	Other tour optimization ideas . . . . .	10
1.2	Goal and Methodology . . . . .	12
1.3	Structure . . . . .	13
<b>2</b>	<b>Fundamentals and Background</b>	<b>15</b>
2.1	Arc Orienteering Problem . . . . .	15
2.2	Shortest Path algorithms . . . . .	16
2.3	Heuristic Approaches . . . . .	18
2.3.1	Ant Colony . . . . .	19
2.3.2	Simulated Annealing . . . . .	24
<b>3</b>	<b>Implemented Changes</b>	<b>31</b>
3.1	Application . . . . .	31
3.1.1	New Architecture . . . . .	31
3.1.2	Database . . . . .	34
3.1.3	Interface and Front end changes . . . . .	36
3.2	Algorithmic changes . . . . .	41
3.2.1	Ant Colony . . . . .	41
3.2.2	Simulated Annealing . . . . .	44
3.2.3	Combinations . . . . .	47
3.3	Parameter changes . . . . .	47
<b>4</b>	<b>Evaluation</b>	<b>49</b>

<b>5 Conclusion</b>	<b>51</b>
5.1 Results . . . . .	51
5.2 Future Work . . . . .	51
<b>A Source Code</b>	<b>i</b>
<b>Bibliography</b>	<b>xiii</b>
<b>List of Figures</b>	<b>xv</b>
<b>List of Algorithms</b>	<b>xvii</b>
<b>Affidavit</b>	<b>xix</b>

# Chapter 1

## Introduction

Algorithms for shortest paths are an important and much studied part of computer science. The topic of finding shortest paths directly influences the lives of many people. However, for outdoor activities, the goal might not always be to find the quickest or shortest route. Whether someone wants to go running, ride their bicycle, go hiking, skateboarding, inline skating or do any other outdoor activity, in most cases the desired route is a roundtrip rather than the shortest path between two (separate) points. Most sports or general outdoor activities are done during free time and are not means to get from one point to another. Rather than wanting a shortest path from A to B, these tours are meant to end at their starting point, e.g. the home. Especially if these hobbies involve driving to a park or into areas where the landscape is more fitted to the person's personal goals. In this case, finding a good roundtrip of the desired length that brings the person back to their starting point can be especially desirable. Furthermore if someone simply wants to run a few times a week, they might want to have a roundtrip that starts and ends at their home.

Better 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, 18, 22, 33, 44, 51]. Furthermore, considerable work on optimizing public transportation [5, 15] and managing traffic jams has been done [14, 16]. Examples are Dijkstra (uni- and bidirectional) [33, 44, 51], A\* search (also uni- and bidirectional) [33, 44, 51], greedy algorithms [33, 51], branch-and-bound algorithms [30], the Bellman-Ford-Moore algorithm [11] and many more [17, 22, 44].

All of these approaches have in common that they always look for the shortest or quickest path between two different points. However, when planning a tour, the goal might not be to simply get to a location as quick as possible. In many cases people plan round trips for outdoor activities to train towards a specific goal. For that purpose, it is strictly necessary to be able to create tours with a set length in order to see and compare their progress. However even if it is a pastime hobby without ambition to reach certain marks, oftentimes, people still want to have roundtrips of a certain length to not overdo

things. Additionally, people typically enjoy running or cycling more appealing paths in the nature and on softer ground rather than between high buildings and on asphalt. Thus, a lot more information has to be taken into account when trying to find good roundtrips for outdoor activities. For these cases, shortest path algorithms become useless as the shortest path from a starting point back to the same point will always be to never leave. Therefore, a different approach is needed for these kinds of routes. A modified version of the arc orienteering problem (AOP) (see section 2.1), which will be called the *touring problem* in accordance with Tour4Me [9] and forms the basis for this thesis (see 1.1.1) is required.

Outdoor activities (like running and cycling as well as other sporty hobbies) can not only be fun but also have many inherent benefits: For overall health [37, 42, 48], the cardiovascular system [35], as a measure against many different diseases [37] as well as for social [34, 36, 50] and psychological benefits [6, 10, 46, 50]. Furthermore, touristic cycling can also be beneficial - in this case for a city gaining more tourism by offering attractive roundtrips for outdoor activities to tour the surroundings [7]. Considering all these advantages and payoffs, finding a solution to the problem at hand becomes all the more important.

Not only are there many joggers and cyclists, who would profit from a tool that returns a roundtrip for their personal well fitting route, but having the option to easily create and plan routes could help convince more people to start any kind of outdoor activity. Having such a tool could increase the amount of people doing some form of exercise and profiting from the previously mentioned benefits of physical activity outdoors. Creating a web app to assist with roundtrip generation lowers the effort to start running or cycling (as route planning is no longer coupled with effort). Furthermore, such an app also helps to show people better or more appealing routes and encourages participation in outdoor activities.

Additionally, as already stated in examples for benefits of outdoor activities, such an app can prove useful for tourism purposes as well. People typically enjoy running or cycling along enticing, exiting routes, which are often hard to find - especially in unfamiliar areas. For any kind of holiday trip, planning new roundtrips for either exercise or touristic purposes or even for several-day roundtrips, as well as for many general outdoor activities this app can be very useful. Especially since users can fully customize the generated tours to their preferences, this app is not limited to only the activities that have been mentioned but can be used for many other outdoor activities as well.

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 arc orienteering problem will be at least NP-hard [2]. Additionally, Gemsa et al. [23] present a proof for 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 (for example [11, 18, 22, 33, 44, 51]), however, for the - more complicated [23] - problem of finding a round trip with several further conditions, not much work has been done yet. 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<sup>1</sup> and RouteYou<sup>2</sup> which both do not allow for much customization of preferences, (surroundings, elevation levels, the steepness of the path, etc.).

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 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, meta-heuristics or even the language they used for programming these solutions wasn't possible. All gathered information are collected from exploring the functionality of the two tools by hand 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

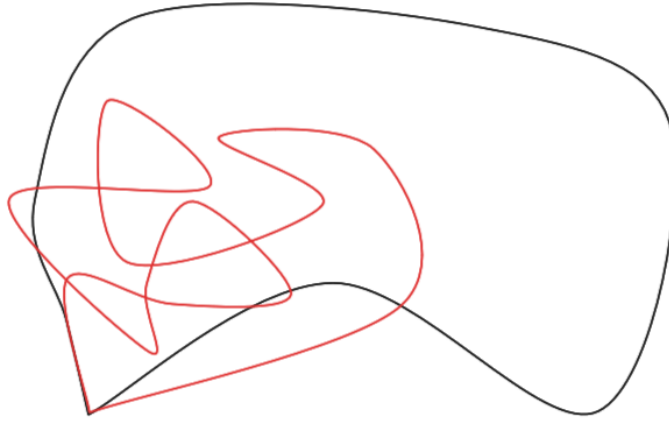
The tool which this thesis will be based off, Tour4Me<sup>3</sup> [9], incorporates some of the mentioned customization options in the created web interface. The app offers the option to choose the favored ground type as well as make selections about preferred route types. Furthermore, the user can also mark certain 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.

---

<sup>1</sup><https://www.routeloops.com/>, last accessed: 22.03.2024

<sup>2</sup><https://www.routeyou.com>, last accessed: 22.03.2024

<sup>3</sup><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 above figure shows the outlines of two possible tours, the black outline being 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 counter clockwise). 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. Second, 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. The offered choices include a Greedy Selection approach, Integer Linear Programming, MinCost with *Waypoints* - a shortest paths variant - and Iterative Local Search (ILS) [9].

The Greedy Selection [9, 51] is the simplest algorithm which only ensures that the chosen route is a roundtrip. The result 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.



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  $\pi$ , the budget (length restrictions)  $B$  and the starting (and end-) point  $v_0$ . Given this encoding, cycles  $P = (v_0, \dots, v_i, \dots, v_0)$ , 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, 23] needs the *waypoints*, which are intermediate points used to calculate shortest paths between them (see 1.1.2 for more details). These waypoints are needed because the algorithm used for the MinCost calculation is typically meant to solve shortest path problems. Thus, the underlying calculations would always result in not leaving the starting position without the added waypoints. 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 simple greedy search. To create an optimized tour, the inefficiency of paths has to be measured. This 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 start-point  $s$  can be calculated. All points that are part of this ring have a shortest path distance of at most  $\pi$ . From these, new rings  $R_r$  with the same requirements can be calculated. The solution path is then obtained through intersecting the sets of all circles and selecting all those that intersect with  $R_s$ . To ensure that the highest profit tour is returned, all possible combinations are calculated and the best of these solutions will be returned [9]. Further details can be found in the original paper [23], 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 from the solution that was built using minCost, Iterative Local Search [9, 32, 47] 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 has to 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 that these paths have must be considered. Since the roundtrip has a given maximum length, the profit has to be maximized while keeping track of and never exceeding this length constraint. The best solution is improved constantly 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 speed up 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 or tours tailored pointedly to cycling. RouteLoops and RouteYou are two commercial tools that offer an interface to calculate tours, but don't have many customization options. However, 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. [23] present two approaches to handle the new routing problem the authors labeled “Jogging Problem”, which is split up into two variants: One being the simple version, that only aims to build a cycle that contains the starting point  $s$  and has the desired length. The other 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” [23]. This relaxation allows to take more factors into account 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 then can 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 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 - which would effectively form a simple U-Turn (turning by 180 degrees) tour - the number of shared edges has to be minimized. The number of turns corresponds to the complexity of the tour and is measured by a percentage of doing a full U-Turn. Gamsa et al. define the point between two edges as a *turn* if the angle is larger than 15 degrees or equal 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.

The ideas presented in this paper are also used by Thomas Pajor in his dissertation [38], where he talks about Computation of Jogging Routes. In the last chapter, he describes the algorithms and gives details about Greedy Faces and Partial Shortest Paths as well.

### Computing Cycling Routes

For running, there are some papers discussing ideas for generating cycling tours. Ehrgott et al. [21] 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 taken into account for this implementation. All objective values that are considered significant for the *suitability* of the tour are accumulated into the one 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 proposes. 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 for their 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. [47] concentrate on cycle trips in their paper, which also builds a foundation for the algorithm used in Tour4Me (see 1.1.1). They 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 meta-heuristic approach of ILS to build roundtrips that optimize the profit of the trip. Since the arc orienteering problem is already NP-hard and the CTPP has an even higher complexity, attempting to solve it 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 meta-heuristic 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 will be too slow for larger problem space instances.

Therefore, Vereeck et al. developed the ILS approach which can be split up into three phases: The initialization, the improvement and the selection. During the initialization 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 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 path was removed - is closed using the same insert move from the initialization, thus improving the previous solution. This 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 it's length lies between a maximum and minimum value and optimize it's profit. They also stated that their ideas can be used as "building blocks" [47] for further development. Something Verbeeck et al. stress, is the fact that vertices can be visited multiple times (except the start vertex), however arcs and (if existent) their complements cannot. Thus they enforce trips to not take the same paths twice.

They did several benchmark tests for their implementations, but the code is not available. Furthermore, there does not seem to be any way to try the existing implementation out and assess how many parameters are used, which of them can be changed and how much customization is 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 can be accessed.

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 can not be used as inputs to get a route with - for example - 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, the difficulty was not shown. 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 created. These points can be removed or more can be added in when editing the tour. Between the waypoints, a 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 can not specify directly what type of ground, 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 especially interested in how to improve these gadgets and apps to make them more usable. In their paper, they discuss the current state of different technologies and propose the following three questions as ideas of what aspects to focus on for further improvement: “How to interact”, which focuses mainly on the question how interaction with any app or gadget can be designed so it won’t hinder the actual activity of running; “What information”, aiming at improving the types of information that are presented to the user while running (for example to change the running style mid run); and “When to assist”, which 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 like one by Loepp and Ziegler [31] used the Partial Shortest Paths algorithm from the idea Gemsal et al. presented [23] to build a recommendation based app. However, they tried to incorporate more criteria, for example 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 of other users, but their following survey revealed that none of their users were interested in that particular feature.

The customized tours that could be generated were received well, perceived as having high quality and the difficulty was seen as low by the users. This app is only a prototype

and was never fully expanded into a full fledged product. Currently, it 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*<sup>4</sup>, *Sportractive*<sup>5</sup> or *Strava*<sup>6</sup> are designed to help runners track the tours they already ran. These apps measure pace, position, height meters and several other stats during a run to then be able to produce feedback for the user. Planning a route is not one of the features these apps offer. And even apps that are meant to assist with the training and which create a plan like *Trainingpeaks*<sup>7</sup> or *SportTracks*<sup>8</sup> do not offer a feature to create routes or roundtrips with a set of preferences [31].

A German app that is meant to provide suitable routes for a variety of different outdoor sports - *Komoot*<sup>9</sup> - does offer a route selection. However, only tours other users have planned and added can be selected. No customization or automated route creation is offered here. As Loepp and Ziegler pointed out in their user study [31], user feedback, their preferences and the option to customize were very important features for a route generation app. The fact that no participant of their study decided to try out a route another member had recorded further stresses the importance of personalized route generation.

### 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 can be related to generating desirable tours. Some papers 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 decisions of users. 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 view along the path. The points they used to decide what can be considered as a scenic view were gathered from a set of geo-tagged photos and from travel blogs.

---

<sup>4</sup><https://www.runtastic.com/>

<sup>5</sup><http://sportractive.com/>

<sup>6</sup><https://www.strava.com>

<sup>7</sup><https://www.trainingpeaks.com/>

<sup>8</sup><https://sporttracks.mobi/>

<sup>9</sup><https://www.komoot.de/>

From their testings, users were happy to take detours that were on average 90% longer than the fastest tour. This observation shows how important the view and surroundings can be when the goal is not necessarily to find a quick or short path, but also to take other subjective parameters into account. 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 single users or even whole user groups. For example, people who prefer running with little to no elevation can end up with a route that takes them uphill for half of the route. While this resulting path still may be a good choice for other users - joggers who prefer more challenging routes or people who want to hike and enjoy ascending - the constant elevation for one half of the tour can be undesirable for beginners. Some people could 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, it can be crucial to the usefulness of an app to give the user as many options to customize as possible without becoming overwhelming.

Thus, the goal of this thesis is to create a usable application for computing running or cycling roundtrips of (almost) arbitrary length. In this case usable means an app that can be used in real time, that produces results of the desired length and prioritizes paths according to the user's input. To achieve this goal, the thesis is built on the already existing prototype Tour4Me [9] and adds meta-heuristic approaches that have been deemed the most fitting for this purpose.

First, an interface for testing the new approaches was built. For adding in 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 could be added and compared with each other to identify the ones that produced promising outputs. However, the definitions of a high quality result can vary drastically based on the criteria that are used. An ideal algorithm would be fast, always generate a route and use all the users' preference inputs. However, achieving all these goals with just one algorithm is not possible. Therefore, different approaches have been implemented and analyzed according to how well they fulfilled the previously mentioned criteria.

The realized approaches are ant colony [4, 24, 49] (as a standalone solution as well as in combination with the previous greedy algorithm and the MinCost implementation) and Simulated annealing. #TODO add more for SA (also add cites)

After the most suitable algorithms for this application had been determined – as well as relatively good parameter configurations for these – they needed to be integrated into the already existing Tour4Me application. The aim was for the app to calculate a high-quality tour for any typical roundtrip requests 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 improves the usefulness of the app. Improving the interface, adding more options like elevation data, including more



information (for example previously used routes) etc. can be 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. The extension of available inputs and sliders to better specify tour parameters is an alternative approach towards the goal of making Tour4Me more usable. Enhancing the interface and overall refining the usability builds a different pillar of improving the app aside from adding more or faster algorithms.

### **1.3 Structure**



## Chapter 2

# Fundamentals and Background

As stated in the [introduction](#), most routing algorithms focus on shortest paths between two or more points. Many of those have been reviewed in several different surveys (see for example [\[33, 51\]](#)). Additionally, there have been many more heuristic approaches, like local search variants [\[8, 26, 41\]](#) or different neighborhood based ideas [\[8, 26, 41\]](#) that offer faster results in exchange for not necessarily finding the one 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) [\[24\]](#), the vehicle routing problem (VRP) [\[8, 26\]](#) or the arc orienteering Problem (AOP) [\[2, 9\]](#)) are NP-hard [\[40\]](#).

Furthermore, finding a shortest path is important in various parts of daily life. Whether the problem is discovering the best (shortest, quickest, most convenient) way to get to work or to a supermarket by car or bike, a good way to minimize travel time by bus or any other trip from one place to another. 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 [\[39\]](#). Here, different algorithms can help calculating friend networks or support the routing of data through virtual networks.

### 2.1 Arc Orienteering Problem

The arc orienteering problem (AOP) is a variant of the orienteering problem (OP) and forms the central concept for roundtrip generation. Souffiau et al. [\[45\]](#) 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 by the user. 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 an edge is part of a path or not is given by  $\chi_{ij}$ . Based on Souffiau et al., the objective is to maximize

$$\sum_{i=1}^n \sum_{j=1}^n c_{ij} \chi_{ij} \quad (2.1)$$

while adhering to the following constraints:

$$\sum_{j=2}^n \chi_{1j} = \sum_{i=1}^{n-1} \chi_{in} = 1 \quad (2.2)$$

$$\sum_{i=1}^n \chi_{ik} = \sum_{j=1}^n \chi_{kj} \leq 1 \quad \forall k = 2, \dots, n-1 \quad (2.3)$$

$$\sum_{i=1}^n \sum_{j=1}^n \chi_{ij} \leq C_{max} \quad (2.4)$$

$$2 \leq v_i \leq n \quad \forall i = 2, \dots, n \quad (2.5)$$

$$v_i - v_j + 1 \leq (n-1)(1 - \chi_{ij}) \quad \forall i, j = 2, \dots, n; \quad i \neq j \quad (2.6)$$

$$\chi_{ij} \in \{0, 1\} \quad \forall i, j = 1, \dots, n \quad (2.7)$$

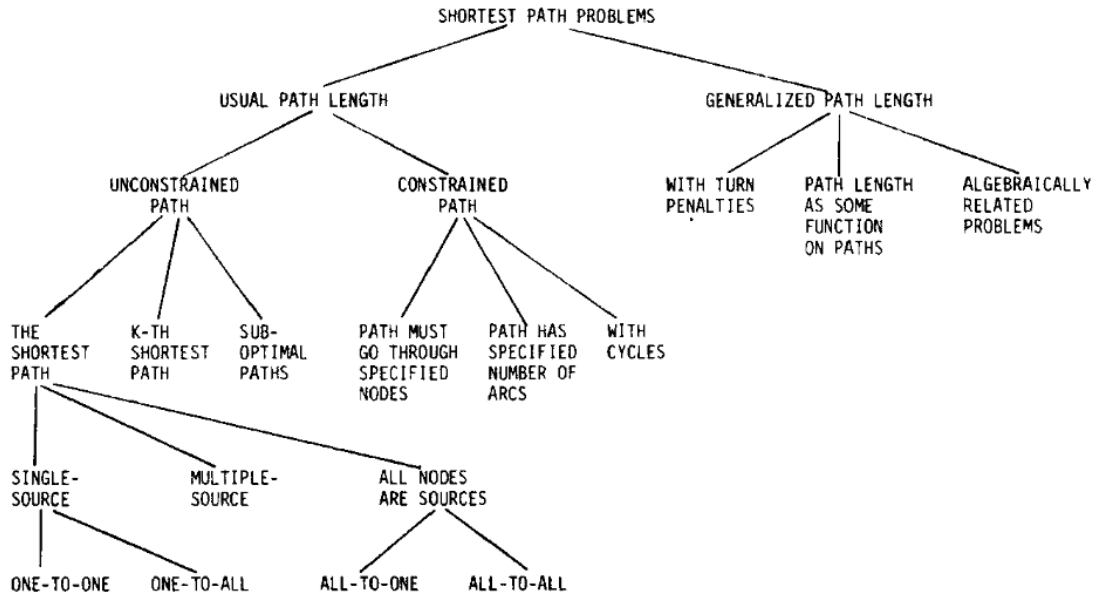
The first constraint (2.2) ensures that both the starting point as well as the end point are part of the resulting path. For roundtrips, this constraint can be simplified to only use the first or the last point. The second constraint (2.3) establishes that the resulting tour is connected. Furthermore, all the contained vertices and edges have to be visited. The third constraint (2.4) guarantees that the resulting path will be at most of length  $C_{max}$ . While constraints four and five (2.5, 2.6) ensure that no sub-tours are created. The last constraint (2.7) defines the permissible values (0 and 1)  $\chi$  can have. These values indicate whether the edge is part of the solution (1) or not (0).

Using these constraints and the base formula (2.1) 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 [18] created an overview tree for different types of shortest path sub-classes to give a better overview 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.



**Figure 2.1:** This image shows a conceptual tree of different variations of shortest path algorithms, taken from [18]

Since the shortest path problem has been well-studied and still continues to advance in terms of the quality of the returned paths as well as in optimizing the running time of algorithms, the number of approaches to solve this problem is enormous. The above tree offers a systematic approach to classify problems and 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).

The first - SSSP - only uses one single starting point and tries to find the one shortest path between this point and one or all other vertices. These kinds of problems have common use cases in many daily routing problems. One to one paths are already described in this chapter's introduction - finding shortest paths to a specified destination [43]. One to all paths can be useful for cases like fire departments or the police that might need a map of quickest routes for every place in their jurisdiction [43].

The second aims to find shortest paths between all vertices of a graph - starting from a specified vertex or from every vertex to every other, which can be necessary for transportation networks and similar use cases. All to one path calculations can be useful in

scenarios where an accident happens and out of all available emergency vehicles, the ones with the shortest paths have to be determined [28]. For all to all paths, many traffic-load calculation problems come to mind. For example cases where trains have to be distributed along the rail network [12].

Aside from these two categories, many more can be found to describe and sort types of approaches.

Which of these shortest path algorithms performs best is typically dependent on the type of graph the implementation is being used on, the graph's structure and the specific problem to be solved. A graph can be categorized as planar or not, directed or undirected, weighted or not, and carry only non-negative weights or allow negative ones as well, they can contain cycles or be acyclic and many more. These different types determine which algorithms can be used as well as which will return better results. Some algorithms like Dijkstra can - without modifications - only be used on a specific type of graph. In this case, the graph needs to have only non negative edges. Others are modified versions, created specifically to fix problems like graphs with negative edges.

## 2.3 Heuristic Approaches

Additionally to exact approaches, heuristics can be used to improve the runtime of an algorithm. A heuristic is a technique that is based on experience or statistical insights [24]. The downside of using such an approach is, that there will no longer be a guarantee that the result is the global optimum, as heuristics specifically only find partial or approximate solutions to a given problem. In many cases where exact algorithms would take too much time or space to find the actual optimal solution, heuristics can be used to find the best possible solution within the given bounds.

For these heuristics, several different ideas have been formed. These can then be categorized into construction heuristics, improvement heuristics and meta-heuristics [29, 41]. However, differentiating between these different types is not always trivial and oftentimes, the line is blurred.

Construction heuristics build their solution from a starting point until a certain boundary is reached. They typically don't have a separate improvement phase. Improvement heuristics try to improve an already existing solution. They perform improvement steps several times until a specified boundary is reached. These boundaries can be e.g. 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.

Meta-heuristics are a form of heuristic approaches. As such, they also try to find an approximate solution to a problem that is as optimal as possible. The distinction between classical heuristics and meta-heuristics is, that the latter are combined with additional

strategies. These are used to enable the meta-heuristics to not produce only solution that are locally optimal, but to broaden the search space they can use for finding optima.

Classical heuristics oftentimes carry the inherent risk of only finding a local optimum that can be far from the actual global one. To reduce this risk, higher level approaches are necessary. These can include using several neighborhood structures to broaden the search space or entirely new concepts like the Ant Colony approach or Genetic Algorithms.

The meta-heuristic ideas that will be used in this thesis will be explained in the following subsections.

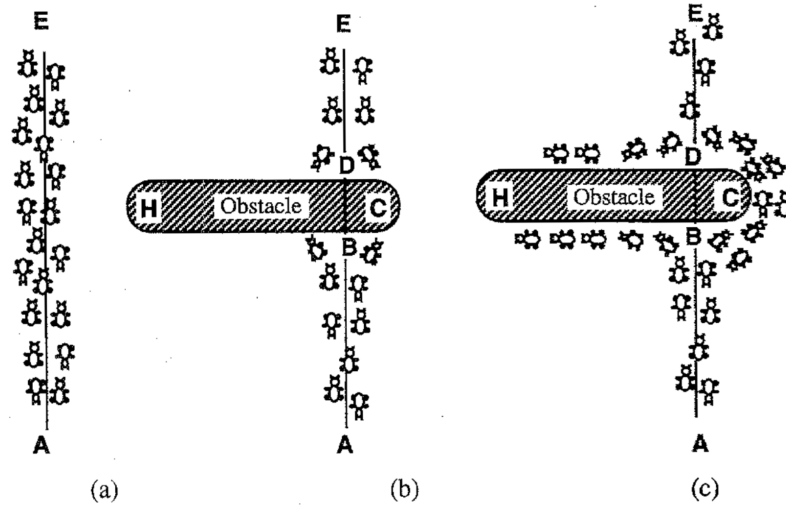
### 2.3.1 Ant Colony

Ant Colony is a meta-heuristic approach that is based on biological ants, ant colonies and how they search food. Real ants start off by walking around on random paths starting from their nest. When they discover a food source, they pick up the food and walk back to their nest. On this way, they distribute a substance called pheromones. These can then be detected by other ants and indicate to them, that a path leads to a potentially good food source. Other ants then are 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 pheromones on good paths. Over time, the pheromones dissipate and when they aren't renewed, will evaporate completely, decreasing the attractiveness of the corresponding path [24, 19].

Furthermore, pheromone distribution also inherently leads to using shorter paths. When several ants have to choose between paths, they will first select at random. However, as soon as one ant discovers the food, turns around and distributes pheromone on the way back, this process automatically increases the likelihood of the respective path being taken by other ants. Here, the shorter paths will be first to receive more pheromones as the ants returning will be quicker. Due to the faster accumulation, more ants will choose this shorter path and thus place even more pheromone on it, leading to a self-reinforcing loop that converges when all ants choose the best path only. Then, all worse paths will loose all their pheromone over time and leave the best result as the only remaining path [24, 19].

To illustrate pheromone distribution, an example illustrates in figure 2.2 how real ants find food and establish the best path towards the source. In part **a** on the left side, there are many ants that run between two points A and E. These could be the nest and an interesting food source. In part **b** in the middle, an obstacle has been added. This construction now leaves the ants with a choice, which path to follow. In the beginning, the likelihood of picking either path will be around 50%. While taking the path, the ants distribute pheromones on it. On the shorter route, the ants will end up reaching the food source earlier, thus returning quicker than the ones 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, the more ants take the short tour and return quicker, the more pheromone will accumulate on that path. This new pheromone placement and distribution leads to a shift in the attractiveness, making the shorter path more likely to be chosen by later ants. These ants will in turn again increase the amount of pheromones placed, making the path even more attractive. Thus, the ants create a self-reinforcing loop of positive feedback through their pheromones which eventually leads to a state where all ants always choose the shorter option.



**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*[19]

This behavior can be replicated in virtual graphs for various routing problems. Ant system has been first introduced in 1990 by Dorigo et al[19] and the following explanations about calculations are based on their findings. In the paper, the authors describe how to use ants for solving the traveling salesman problem (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 amount of  $m$  ants moves through the graph, tries to find a good tour and places pheromones on edges. Every ant has a defined amount of pheromone to place. How much of the pheromone will be laid on



a path can be calculated in several different ways. The authors propose the following three ideas:

$$\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{L_k} & \text{if } (i,j) \in \text{tour described by } \textit{tabu}_k(1) \\ 0 & \text{otherwise} \end{cases} \quad (2.8)$$

$$\Delta\tau_{ij}^k = \begin{cases} Q & \text{if the } k\text{th ant goes from } i \text{ to } j \text{ between time} \\ & t \text{ to } t+1 \\ 0 & \text{otherwise} \end{cases} \quad (2.9)$$

$$\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{d_{ij}} & \text{if the } k\text{th ant goes from } i \text{ to } j \text{ between time} \\ & t \text{ to } t+1 \\ 0 & \text{otherwise} \end{cases} \quad (2.10)$$

Here, equation 2.8 is the default the authors used for solving the TSP. The constant  $Q$  has to be picked according to the problem in question. Variable  $L_k$  describes the length of the whole tour. This property makes sense for the TSP setting, but is relatively useless for the case of tours with a fixed length, as it will be the same value for every ant and every run made. In this case, where the user defines the length of the tour,  $L_k$  will only scale the values picked for  $Q$ .

Equation 2.9 only uses the constant  $Q$  to describe pheromone placement. Here, neither the full tour length nor individual edge costs are taken into account. Pheromone is placed evenly on all edges. This way of placing pheromones equates to a not-scaled version of equation 2.8 with the given use-case of a set length for the tour.

The last equation 2.10 divides the constant by the length - or the cost - of each edge when it is used. This division reduces the amount of pheromone placed on longer edges proportionally to shorter edges. While this equation is not influenced directly by the fixed length, this property can still cause the equation to be less useful for tours with a specified length than for TSP. Since tours that are meant to cover a fixed distance are different from the TSP, where a shortest path that visits all selected cities is to be found, the last equation seems like the least promising candidate for useful pheromone distribution.

The pheromone calculation and placement can be defined with different formulas depending on the problem and what should be optimized. Which option turns out to be the best fitting one will be described in the evaluation chapter 4.

Using 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:

$$\Delta\tau_{ij} = \sum_{k=1}^m \Delta\tau_{ij}^k \quad (2.11)$$

This overall value can then be used to calculate the so called „intensity“ of the placed pheromone trail. Since pheromones evaporate over time, this property has to be modeled as well, using a new parameter  $\rho$ , which describes how much of the pheromone stays on the trail between two time steps. Thus, the overall pheromone intensity can be described by

$$\tau_{ij}(t+n) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}^k \quad (2.12)$$

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. What's left is determining the probability with which ants will choose a certain edge over the other options. For this calculation, two more properties are needed: the visibility of an edge and a tabu-list (or rather a list of allowed nodes). The tabu-list contains all nodes that have been visited before. Since roundtrips should - per default - be round rather than the same path run in two directions, this property is needed to ensure no city is visited more than once. In chapter 4, different configurations are tested to represent different shapes and allow for more options users can define. Thus, for other shapes, this list is not needed. For the TSP-case, the visibility  $\nu_{ij}^k$  is calculated using the length of the edge  $d_{ij}$  as follows:

$$\nu_{ij}^k = \frac{1}{d_{ij}} \quad (2.13)$$

And the transition probability is given by

$$p_{ij}^k = \begin{cases} \frac{[\tau_{ij}(t)]^\alpha \cdot [\nu_{ij}^k]^\beta}{\sum_{k \in allowed_k} [\tau_{ij}(t)]^\alpha \cdot [\nu_{ij}^k]^\beta} & \text{if } j \in allowed_k \\ 0 & \text{otherwise} \end{cases} \quad (2.14)$$

using all previously defined values to calculate visibility  $\nu_{ij}^k$ , trail intensity  $\tau_{ij}$ , pheromone distribution  $\Delta\tau_{ij}$  and  $\Delta\tau_{ij}^k$ . Here,  $\alpha$  and  $\beta$  are parameters that influence the weight of visibility and trail intensity. Higher values of  $\alpha$  increase the significance of the pheromones on the trail (setting  $\alpha$  to 0 would lead to completely ignoring the pheromone placed) and higher values of  $\beta$  increase the importance of the visibility of an edge (making longer edges less attractive as a result). These parameters will be experimented with and their influence will be evaluated in chapter 4.

In their paper, Dorigo et al. suggest picking middling values for  $\alpha$  and  $\beta$  in a range of  $[0.5, 5]$ . They furthermore 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  $\alpha$ , no good results could be generated.

Some of these formulas have to be changed and fitted to the given arc orienteering problem (see section 2.1). The adjusted calculations and explanations of the changes that had to be made can be found in section 3.2.1.

### Pseudocode

The original ant colony algorithm in 2.1 is built for solving TSP instances. A possible structure for an ant colony algorithm is displayed in 2.1, based on the framework provided in [19]. Therefore, all formulas are centered around a 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 2.14. NC and  $NC_{MAX}$  are used for the loop count and the maximum allowed number of loops 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 completes one tour for all ants.

The next steps are needed to reset and re-calculate the needed values for the next run. In the loop, the ants are moved back, 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 trail intensity for all ants ( $\Delta\tau_{ij}$ ) is updated. Furthermore, the new shortest tour is updated, if a new one was found. After the individual pheromone values were updated, the trail intensity  $\tau_{ij}(t+n)$  has to be recalculated. For this step, the evaporation rate  $\rho$  scales the current trail intensity and the calculated resulting pheromones for all ants are added  $\Delta\tau_{ij}$ .

The rest of the loop is updating the loop count NC, the time step  $t$  and resetting the pheromones for each edge to the initial 0. The loop is continued with emptied *tabuList*s if the maximum count wasn't reached and is ended otherwise, returning the shortest tour found.

# TODO in den Anhang?

---

#### Algorithm 2.1 AntColony

---

```

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 kth ant to j
12:     add town j to tabuList
13:   end for
14: end while
15: for k = 1 to m do
16:   move kth 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:    $\tau_{ij}(t+n) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}$ 
28: end for
29: t  $\leftarrow$  t+1, NC  $\leftarrow$  NC+1
30: for edge(i,j) in allEdges do
31:   reset  $\Delta\tau_{ij}(t) \leftarrow 0$ 
32: end for
33: if NC < NCMAX and not stagnating then
34:   empty tabuLists
35: else
36:   return shortest tour
37: end if
38: end for

```

---

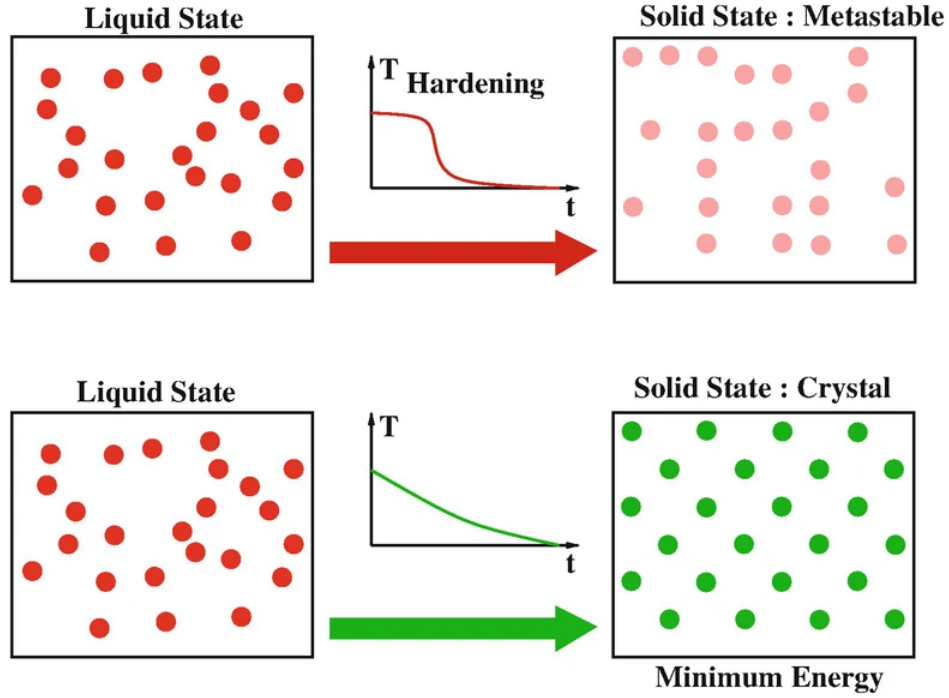
### 2.3.2 Simulated Annealing

Simulated Annealing (SA) builds upon a 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:

- melting the solid through heating
- cooling the material to a specified temperature

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:** 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 [13].

The above figure shows the effects of different cooling schedules on a material and the type of solid state this material will obtain as a result. In the upper part, the top left image shows a liquid state, in which molecules are free and can move around. Cooling down too quickly hinders the atoms to move 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 with a minimum energy state, resulting in a stable crystal structure and a highly durable solid material.

Based on this analogy, simulated annealing was developed as a concept to solve NP-hard computational problems by improving local search variants [1, 20]. Local search approaches a very good at finding locally optimal solutions, but can easily fail to find a global optimum. Simulated annealing addresses this problem with the underlying idea based on the two steps of heating and slow cooling. What makes SA different from a

basic local search is, that it allows – with a certain probability (temperature) – for worse solutions to be accepted.

First, some basic solution for the problem is needed. For the use case of this thesis, a basic solution will be a roundtrip. Furthermore, a function which determines the quality of the solution is needed, so that comparing different solutions to each other is 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 which 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, lowering it over time. In the beginning, the solution is likely to be further away from a global optimum. Thus, a higher temperature allows for an increased likelihood to accept solutions with a lower quality, which in turn allows to escape local optima. The longer SA runs, the lower the temperature gets, lowering the likelihood of accepting worse solutions while nearing the global optimum. To do this, several nested runs are needed. In an outer loop, the temperature changes and gets updated with lower values. In an inner loop, several possible neighborhoods are tested for one temperature state. Solutions that are better than the previous one are always chosen. Solutions that are worse than the previous one are chosen based on the temperature.

For SA to yield a good result, several parameters need to be set. First, a starting roundtrip, which builds the base, has to be calculated. Then, initial temperature as well as the cooling formula have to be determined. Furthermore, a neighboring roundtrip has to be found, as well as a formula for calculating the quality of all found solutions. Additionally, the number of runs has to be decided, while keeping both the runtime as well as the quality of the resulting tour in mind.

For some of the parameters – like the initial temperature and the cooling formula –, thorough testing of different configurations is needed. For others, only a limited set of options is interesting. Determining a neighboring roundtrip as well as an initial solution are the two parameters that have a narrow array of possibilities to calculate them.

For an initial solution, the implemented algorithms – AntColony, Greedy and MinCost (see 1.1.1) – can calculate a base roundtrip. Which of these is to be preferred is dependent on the preferences about the resulting shape. Greedy and AntColony prioritize edge quality while MinCost mostly optimizes for a rounder shape.

To determine a neighboring solution, there are a few options that can yield good results. One approach is to pick two random vertices and switch their position in different ways within the roundtrip path or some variations of this idea [52]. Another is to create waypoints, similarly to how it is done in the MinCost algorithm, but without the length restrictions. The generated initial solution can then be used to generate a set of waypoints.

Based on this node set, neighboring roundtrips are ones where the tour is modified by removing, adding or moving one of the waypoints.

# TODO move to implementation chapter?

To remove a waypoint, the node and the paths that connect this vertex with the neighboring two waypoints need to be deleted. Then, the two ends are connected by a shortest path or a MinCost approach. For adding a waypoint, first, a suitable node needs to be determined. This choice can be done either randomly or based on a probability distribution that gives higher probabilities to nodes closer to the path and lowers in value the further away a node is. After a suitable node was picked, the closest waypoint and the best neighbor needs to be found. This selection is done by calculating the shortest paths from the new waypoint to all existing ones. Then, the connection between the two picked nodes has to be removed and the new waypoint needs to be joined up with the two nodes that now frame the gap. Moving is the combination of removal and adding. For this operation, the new node has to be found first and the closest waypoint will be the one to remove.

### Calculations

For SA, a few parameters have to be calculated. First, the temperature and the respective cooling schedule or a function to decrease the value have to be determined. Here, an several different functions can be applied. The important criteria are that they decrease the temperature and are fit to the problem. For example, a simple function that calculates the Temperature by using the difference of the quality of the neighboring solution  $f(j)$  and the quality of the current solution  $f(i)$  divided by a random number  $r_i$  [52]:

$$T_{i+1} = \frac{-|f(j) - f(i)|}{\ln(r_i)} \quad (2.15)$$

Another option is to multiply the current function value with a factor  $\alpha$  smaller than 1 [20]:

$$T_{i+1} = \alpha \cdot T_i \quad (2.16)$$

Or scaling by using a constant  $b$  to build a fraction [20]:

$$T_{i+1} = \frac{T_i}{1 + b \cdot T_i} \quad (2.17)$$

There are several other options – a few of them are listed in the paper by R. W. Eglese [20] – to calculate the temperature, where the most important part is, that the temperature is decreasing.

The quality function  $f$  is highly dependent on the problem and can thus vary by a lot. For TSP, the quality of a tour is determined by its length. For the act orienteering

problem by the values that are to be collected when generating a roundtrip. How the quality is calculated for the specific case of this thesis is described in detail in section 3.2.2.

Another important calculation is the one of the probability, which is used to determine whether or not a solution should be picked even though the quality is worse than the current quality. To determine this value, the temperature as well as the quality values of the two solutions are necessary. Here, the difference between the quality of the neighboring solution  $f(j)$  and the quality of the current solution  $f(i)$  is always negative, since a positive difference would imply  $f(j) > f(i)$  and thus result in the neighboring solution being picked automatically. Since the probability is only calculated if 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 gets bigger.

$$p = e^{\frac{f(j)-f(i)}{T}} \quad (2.18)$$

### Pseudocode

The simulated annealing algorithm has a basic structure that always stays the same for every problem. This structure is outlined in the pseudocode in 2.2, based on [20] and [52] and visualized in a flowchart diagram in A.1.

First, an initial roundtrip has to be determined. This calculation can be done with any algorithm that results in a valid solution, for example a greedy approach or the MinCost algorithm that is also used for solving the arc orienteering problem in the original Tour4Me application 1.1.1. Furthermore, a starting temperature has to be set, which will be used for the first probability calculations and decreased after the inner loop.

Then, the two loops of the main SA step are run. The outer loop is bound by a selectable number of repetitions and the inner loop is bound by the selectable number of runs that should be performed with a fixed temperature. While the former controls how many new temperatures are tried – in the physics analogy this would be how long the material is left to cool – with decreasing temperatures, the latter controls how many tries are executed before the temperature is adjusted. Alternatively, the outer loop could also have reaching a certain quality value as the stopping condition. Performing several calculations in the inner loop with the same temperature can increase the probability to find a better solution for the set temperature as well as the probability to choose a solution with a worse quality.

---

#### Algorithm 2.2 High level Simulated Annealing

---

- 1: Select init roundtrip ( $i$ )
- 2: Select init temperature ( $t$ )
- 3: **for**  $i = 0$  to numberRepetitions **do**



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

---



## Chapter 3

# Implemented Changes

This work extends Tour4Me<sup>1</sup>, which is an application written in C++, HTML and JavaScript. The implemented interface of this extension uses C# as programming language to enable easy porting of the web application to a desktop or mobile application. To improve the query times and allow for easier coverage of the whole world (see 5.2), a database that offers features of a spacial database was added. Reasons for and positive effects of this decision are described in the following section.

Furthermore, not only the language and data access was changed. New options and parameters to improve the customizability of preferences for a generated tour were added as well. These changes had to be incorporated into an upgraded front end design (see sections 3.1.3 and 3.3) as well as into the back end and all solvers (see section 3.2).

### 3.1 Application

To include the various changes, the whole application – including the front end representation, the back end implementation and the data retrieval – was changed. The Open Street Map (OSM) data are downloaded and stored in a database from which the graph for calculating the roundtrips is build. Furthermore, the design of the front end was changed to improve the overview and general user experience as well as to allow for the addition of new customization options. Lastly, the algorithms to choose from have been extended by two additional meta-heuristic approaches.

#### 3.1.1 New Architecture

For the new application, the architecture had to be re-structured. 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, which contains all the nodes and edges for Dortmund, a database is used to manage the nodes, edges, their additional information and the relationships between

---

<sup>1</sup><http://tour4me.cs.tu-dortmund.de/>

them. It can be filled with the data needed by using an import python script that creates an osmnx-graph<sup>2345</sup> for a user specified location. From this graph, the nodes and edges can be extracted alongside their additional information. For the current use case, nodes are stored with their OSM-ID, which is transformed into a UUID, their latitude and longitude coordinates as well as their elevation profile and tags of the surroundings they are placed in. The elevation data has to be acquired from a different source than OSM, since they do not use a height profile. A few open source providers were available, but ultimately, Open-Elevation<sup>6</sup> was used.

Since 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 very important to assure usability. When using the python script to create and fill the database and its tables, the Open-Elevation data needs to be available. A local docker container 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 version that can be hosted locally is not ideal, it was the easiest method that did not rely on querying a website and being limited by their API-call restrictions. In the section future work (5.2), a few other options of including other data sources are discussed.

The used database is Microsoft SQL Server Management Studio<sup>7</sup>, which can handle spatial data, supports spatial queries and works well in combination with the C# implementation.

The back end is implemented in C#<sup>8</sup>, as this language allows for the opportunity to also create a mobile- or desktop application in addition to the web application that already exists (see 5.2). Furthermore, C# allows for using SQL queries and filtering using LINQ for easy runtime database querying<sup>9</sup>. Here, the Tour4Me application was automatically translated from C++ to C# using ChatGPT. This translation is not part of the thesis and only allows for easier extension, since the basic structures that are used in Tour4Me did not need to be re-implemented from scratch.

---

<sup>2</sup><https://osmnx.readthedocs.io/en/stable/>, last accessed: 15.04.2024

<sup>3</sup><https://networkx.org/>, last accessed: 22.03.2024

<sup>4</sup><https://wiki.openstreetmap.org>, last accessed: 22.03.2024

<sup>5</sup>[https://wiki.openstreetmap.org/wiki/Main\\_Page](https://wiki.openstreetmap.org/wiki/Main_Page), last accessed: 19.04.2024

<sup>6</sup><https://open-elevation.com/>, last accessed: 20.03.2024

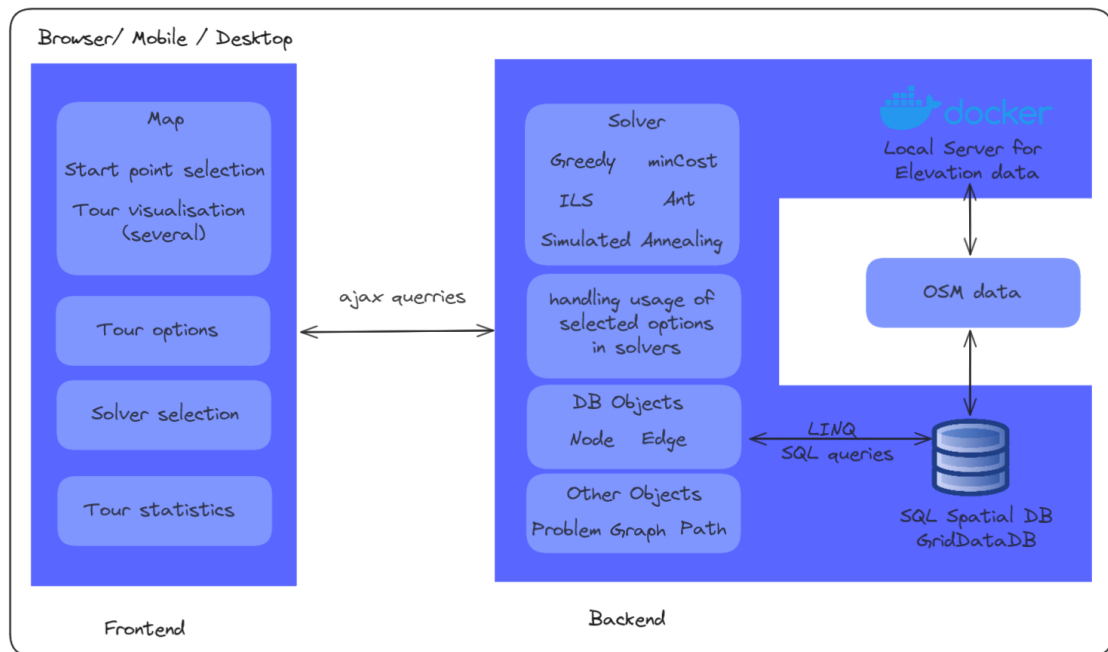
<sup>7</sup><https://learn.microsoft.com/en-us/sql/sql-server/sql-docs-navigation-guide?view=sql-server-ver16>, last accessed: 22.03.2024

<sup>8</sup><https://learn.microsoft.com/en-us/dotnet/csharp/>, last accessed: 22.03.2024

<sup>9</sup><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

The front end is implemented using HTML<sup>10</sup>, CSS<sup>11</sup>, JavaScript<sup>12</sup> and C# code behind. The base-styling is done using bootstrap<sup>13</sup>, but additional custom CSS is added to create a nature-based color palette (#TODO references to color theory stuff?) as well as several custom effects and transitions for the side and bottom menus. To realize the communication between front end and back end, Ajax-queries<sup>14</sup> are used.

The map is a leaflet<sup>15</sup> visualization that shows Open Street Map data. The leaflet map allows to set markers, add a search bar, create polygons - which are used to illustrate the generated routes - and offers an open source map view.



**Figure 3.1:** Visualization of the used architecture

In the above visualization, the whole application, the distinct parts and features as well as communications between them are illustrated. The front end is realized as a web application, running in the browser, but the visualization can also be customized to be executable as a mobile or desktop application (see 5.2). In the visualized map, the marker can be set to the current location - if the permission to access the user's location data is granted. However, simply 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. Furthermore, the visualization of the calculated tours is also realized

<sup>10</sup><https://devdocs.io/html/>, last accessed: 22.03.2024

<sup>11</sup><https://devdocs.io/css/>, last accessed: 22.03.2024

<sup>12</sup><https://devdocs.io/javascript/>, last accessed: 22.03.2024

<sup>13</sup><https://getbootstrap.com/docs/4.3/getting-started/introduction/>, last accessed: 22.03.2024

<sup>14</sup><https://api.jquery.com/category/ajax/>, last accessed: 22.03.2024

<sup>15</sup><https://leafletjs.com/>, last accessed: 20.03.2024

using the map and a polygon built from the respective points. These tours can be toggled to be shown or hidden, so an easier overview is possible after generating several tours. For debug purposes, there is a feature to show the whole graph that is being used for the calculation using the currently selected maximum length.

In addition to the main feature – the map – the front end also 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. Furthermore the back end is responsible for handling the parameters the user chose, 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, the data for Dortmund were retrieved. In addition to the OSM data, using the latitude and longitude, elevation data are obtained from the docker server.

### 3.1.2 Database

The database is a relational database using Microsoft SQL server, administered in Microsoft SQL Server Management Studio. This database also allows to use spatial data, which was an important feature for storing and processing the nodes and edges. Using the spatial features enables the possibilities to filter nodes within a given radius, retrieving only a relevant subset of data points. This filtering option within the database significantly speeds up the data retrieval as well as the graph creation. Compared to the previous method of generating a fixed graph for the city of Dortmund, the database offers further important advantages: Far more nodes than only points within Dortmund can be used. Even though the database creation and the adding of points is relatively slow, this is a process that only needs to be run once – before the application can be used – and does not affect the tour calculation. Once the data has been added, all points can be accessed without needing to retrieve the whole database.

To create the database, a python script is used. # TODO if added, describe parameters and how to use the script with them This script first creates an osmnx graph from OSM data using the `graph_from_place` function

```
ox.graph\textunderscore from\textunderscore place(place\textunderscore
    ↪ name, network\textunderscore type='all', custom\textunderscore
    ↪ filter=custom\textunderscore filter)
```

Here, 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<sup>16</sup>: `all_private`, `all`, `bike`, `drive`, `drive_service`, and `walk`.

The `custom_filter` is defined to select only those edges, where walking, running and cycling is possible by specifically de-selecting respective highway types:

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

The excluded types are used for the following street types according to the OSM Wiki:

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<sup>17</sup>

Next, the elevation data has to be added for the nodes of this graph. These information are not part of osmnx but need to be retrieved from a different source. For this thesis, Open Elevation<sup>18</sup> was used. The data were downloaded and hosted in a local docker that could be queried instead of the API.

<sup>16</sup><https://osmnx.readthedocs.io/en/stable/user-reference.html#module-osmnx.settings>, last accessed: 19.04.2024

<sup>17</sup><https://wiki.openstreetmap.org/wiki/Key:highway>, last accessed: 19.04.2024

<sup>18</sup><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. So, a different query that retrieves areas and then matches the respective tags to the saved nodes by matching the IDs has to be executed.

After gathering all the necessary information, the script first checks if the database already contains matching tables 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 (inserting edges into the IncidentEdges table that manages nodes and all their incident edges as well as adding references to the source- and target node when entering the edge data). 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.

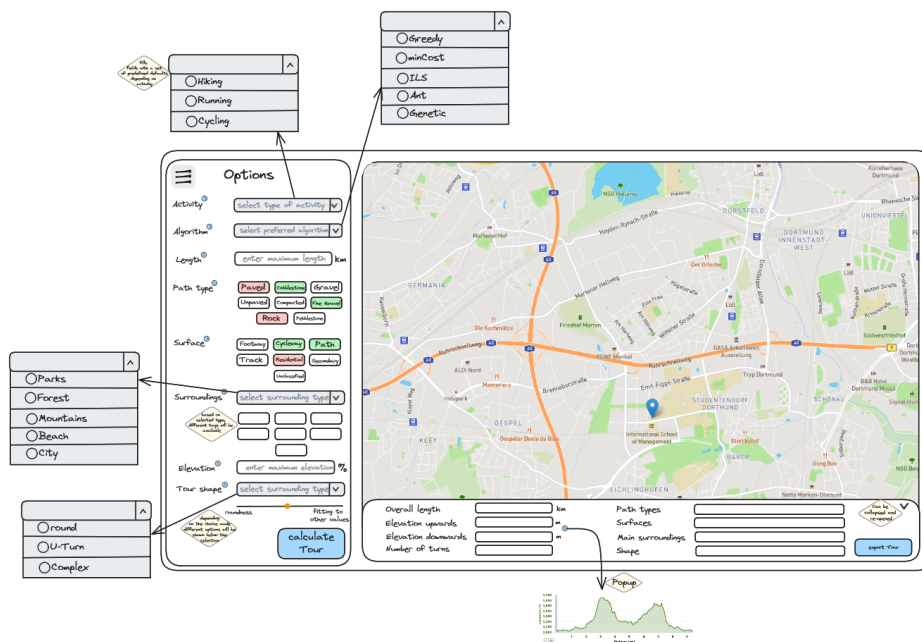
### 3.1.3 Interface and Front end changes

The front end was updated from the existing version of Tour4Me to allow for a better overview after adding in several additional option for the user to select from. First, a conceptual idea see figures (3.2 to A.4) was built, mapping out the general structure of the new interface. Here, the main changes were 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 A.3). The result view was moved from an overlay on the map to a foldable footer menu (A.4). 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 popup.

Displaying all the resulting data while giving the option to fold and unfold them allows the user easier access as well as enables easier comparison between different tours. Previously, only the data of a single tour could be shown in the popup. Now, the information of all tours are visible at the same time.

# TODO move this to the appendix maybe?





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

The new interface now 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 A.5) can be used to set a pre-selection that fills in the following fields with suggested default values. These values can always be customized afterwards, but could offer a higher usability. The next selection shows all currently implemented algorithms (see A.6). Depending on the choice, the results will turn out differently. While greedy always only prioritizes the maximum edge profit and ignores the roundness of the tour, MinCost does the exact opposite. AntColony is more focused on edge profit, but also takes elevation and roundness into account. And finally simulated annealing produces results that are mostly focused on roundness while still taking edge profits and elevation into consideration. The respective combinations start with greedy or MinCost and then strive to improve them.

All following options are direct tour parameters and describe user preferences. The length is an estimate of the final tour length, that is never used as a hard stop 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. # TODO check that this is true lol

Surface and path type show 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 undesirable 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 much interest and would only result in an overwhelmingly large selection. Surroundings have a drop down menu to select a general type – the forest, grasslands or other options – that will then display the respective tags to display (see A.7). This approach was used to minimize the number of tags and allow for an easier overview for the user.

The tour shape offers the options to pick a round tour, a U-turn, a complex tour or do a custom selection (see A.8). Round tours have the highest importance (80 %) on the roundness of a tour and split the remaining 20% between elevation and edge profits. This 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, but currently simply levels the importance of edge profits and elevation while ignoring the covered area. Complex tours are similar, but don't fully remove the importance of the area. For the custom selection, three sliders are displayed. These slider inputs are linked so they influence each other, ensuring that the three probabilities always sum to 100%.

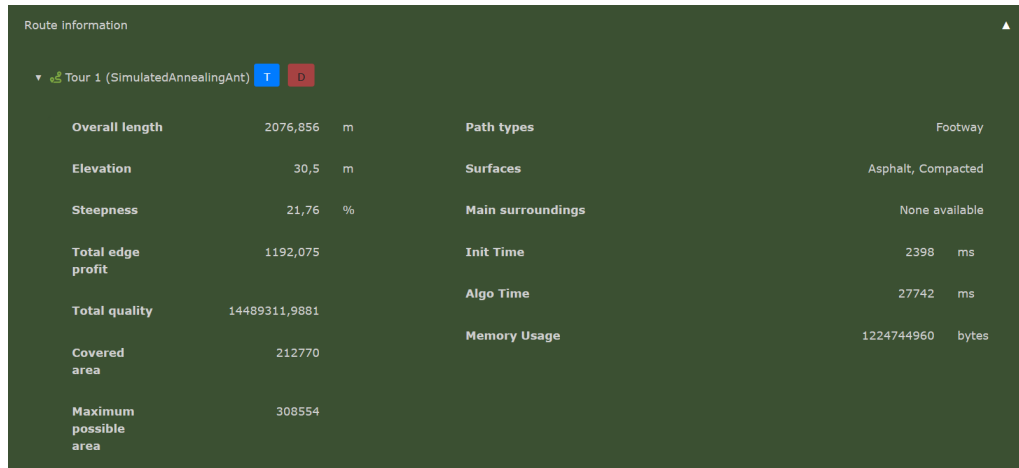
Below the tour shape, elevation and steepness can be selected. The elevation describes the maximum difference in elevation for the whole tour. This property does not differentiate between ascending or descending parts but sums up all differences and divides the overall result by two, to accommodate for the roundtrip. The steepness is used to exclude any part of the tour from being steeper than the chosen limit as much as possible. With the inaccuracy of the elevation data, this 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.

At the bottom, the maximum runtime can be chosen, however no algorithm runs as long as 30 seconds with the current settings. Clicking the button „Compute Path“ will then trigger a query to the back end, where all the selected information are processed, 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 then be displayed and connected to form a polygon that visualizes the resulting tour.



**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*

All additional information are displayed in the *Route information* footer menu. The menu as well as the calculated tours can be folded and unfolded. This option does not decrease the size of the map 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 are two buttons to toggle the visibility of the polyline in the leaflet map or to delete the generated tour entirely.




Route information

▼ Tour 1 (SimulatedAnnealingAnt) T D

Overall length	2076,856	m	Path types	Footway
Elevation	30,5	m	Surfaces	Asphalt, Compacted
Steepness	21,76	%	Main surroundings	None available
Total edge profit	1192,075		Init Time	2398 ms
Total quality	14489311,9881		Algo Time	27742 ms
Covered area	212770		Memory Usage	1224744960 bytes
Maximum possible area	308554			

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

Clicking on the tour name will fold (see 3.5) and unfold (see 3.4) 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 given the same extreme points as the current tour has. 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 not as interesting for the typical user but allow for a deeper insight into the calculation. Removing the outputs for a final version is easily done if needed.



Route information

▶ Tour 1 (SimulatedAnnealingAnt) T D

▶ Tour 2 (SimulatedAnnealingAnt) T D

▶ Tour 3 (SimulatedAnnealingAnt) T D

▶ Tour 4 (SimulatedAnnealingAnt) T D

▶ Tour 5 (SimulatedAnnealingAnt) T D

▶ Tour 6 (SimulatedAnnealingAnt) T D

▼ Tour 7 (SimulatedAnnealingAnt) T D

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

## 3.2 Algorithmic changes

For the tour calculation, two new meta-heuristics were implemented. The first one was ant colony, which was modified to match the arc orienteering problem instead of TSP like the original. The second meta-heuristic was a version of simulated annealing. Here, the calculation of neighboring solutions had to be matched to return roundtrips. In the following sections, the changes that had to be made to result in implementations that can be applied to the AOP will be explained. For both meta-heuristics, the pseudocode and changed formulas will be explained. The base idea and the overall working of the algorithms stays identical to the explanations in 2.3.1 and 2.3.2.

### 3.2.1 Ant Colony

For ant colony, several things had to be changed. First, for solving TSP, finding a shortest path and visiting all specified cities are the main objectives. Whereas for AOP, these 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.

In the original paper, the pheromone amount to be placed on the collected edges was calculated based on the tour length the ant built. This scaling resulted in higher pheromone for shorter tours and lower pheromone 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**  $p$  the edge with the length  $l$  will grant the tour: # TODO check if better use quality here?

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

Here, the length 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 5.2). If the edge has a tag specified as desirable, the **edgeProfit**  $p$  is increased by one. If a tag that is specified as undesirable, the profit will be decreased by one. If the edge has no tag that is part of desirable or undesirable tags, the profit is 0.0001. Like this, edges with desirable tags will gain as much profit as they are long. Edges with undesirable tags will lower the profit by their length and edges without tags that are either desirable or undesirable, will impact the profit only by a fraction of their length.

$$p(i, j) = \begin{cases} 1 & \exists \text{tag} \in \text{tags}(i, j) \text{ tag} \in \text{desirable} \\ -1 & \exists \text{tag} \in \text{tags}(i, j) \text{ tag} \in \text{undesirable} \\ 0.0001 & \text{otherwise} \end{cases} \quad (3.2)$$

The trail intensity itself is calculated the same way as in the original paper (see 2.12), however, for further calculations in the ant, it is scaled by dividing by the desired length  $L$ , based on whether or not an edge is visited twice:

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

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

Furthermore, the visibility cannot be based on the length like the original paper did, since all tours should result in a length close to the desired one. Thus, for the visibility, the edge quality is determined. To calculate the quality, all selectable values have to be taken into account: The **coveredArea**  $A$ , that maps to 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  $i_e$ :

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

The respective importance values are multiplied by 100 to scale them from the percentage values to be bigger than one. For the **coveredArea**, to scale the size and account for the quadratic increase, the square root is taken. Then, the whole Area is scaled by the desired length of the tour  $L$  to account for the larger value the area will have compared to the edge profits. # TODO check if this scaling is actually fine! The full derivation of the calculation of the scaling is shown in equations 3.5 and 3.6, where  $A$  is the area and  $U$  is the circumference of a circle.

$$\begin{aligned} A &= \pi r^2 \\ A &= \pi \cdot \left( \frac{U}{\pi \cdot 2} \right)^2 \\ U &= 2 \cdot \pi \cdot r \\ \frac{U}{\pi \cdot 2} &= r \end{aligned} \quad (3.5)$$

$$\begin{aligned} A &= \pi \cdot \frac{U^2}{\pi^2 \cdot 2^2} \\ A &= \frac{U^2}{\pi \cdot 4} \\ A \cdot \pi \cdot 4 &= U^2 \\ \sqrt{A \cdot \pi \cdot 2} &= U \end{aligned} \quad (3.6)$$

As in the original paper by Dorigo et al. [19], the probabilities for the edges are calculated using the visibility and the trail intensity as well as the two parameters  $\alpha$  and  $\beta$  (see also 2.14), however, here, the scaled value  $\tau'_{ij}$  is used:

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

Additionally, there is no tabuList and thus, this list cannot be used as a stopping criterion. Instead, a set number of tours are 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 rather than in  $n$  different towns. The full updated pseudocode is shown in 3.1

---

**Algorithm 3.1** AntColonyAOP

---

```

1: initialize graph and problem (starting point, graph with nodes max  $\frac{2}{3} \cdot L$  distance)
2: init trailIntensity  $\tau_{ij} \leftarrow 0.0001$  for all edges(i,j)
3: set starting pheromone values  $pheromone(i, j) \leftarrow 0$  for all edges(i,j)
4: generate ants
5: set current node  $\leftarrow$  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 ( $\tau'_{i,j}$ )
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 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 2.12
22:  pick tour with best quality to use for next run
23: end for
24: pick tour with best quality as result

```

---

To calculate the new ant colony implementation, first, the graph and a helper class **problem** have to be initialized. The graph is created from the database, using the selected starting point and  $\frac{2}{3} \cdot L$  to fetch only the needed number of nodes and edges. The problem saves all other user inputs:

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

The edges hold their length, the profit according to the tags, their pheromones 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 pheromones are initialized with 0.

Then, the ants can be created and initialized. Currently, all ants use the same  $\alpha$  and  $\beta$  as well as the same base amount of pheromones they can distribute. However, the values could be changed throughout the runs, which can be of interest for further combinations.

For a set number of runs, all ants calculate a tour in parallel. To do this, 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 maximum selected 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 some edges are still available, they are scaled, and then their visibility and based on that the probability to choose the edge is calculated. Based on these probabilities, one edge is picked, the pheromone values are updated and the ant is moved. The current node is updated accordingly. 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 further calculation. After the last run, again, the tour with the highest probability is chosen as the final result.

### 3.2.2 Simulated Annealing

For simulated annealing, all needed changes are to do 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 with that, the exact calculations can be specified. The updated pseudocode only additionally contains building the waypoint list, calculating distances and probabilities (see A.1)

The quality  $f(j)$  of the solution  $j$  is – like for ant colony – calculated based on the `coveredArea`  $A$  and the respective importance (`coveredAreaImportance`  $i_A$ ), the `edgeProfit`  $p$  and the respective importance (`edgeProfitImportance`  $i_p$ ) and the elevation difference  $e$  from the maximum  $e_{max}$  and the respective importance  $i_e$  and scaled by the difference between the actual length  $L_{calc}$  of the tour from the target length ( $L$ ).

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

The respective importance values are again multiplied by 100 and the `coveredArea` is scaled the same as for the ant colony (see 2.13, 3.5 and 3.6)

The neighboring solution is created based on waypoints, which are used similarly to how they work in the minCost tour creation (see 1.1.1): A the waypoints form base points of the tour that are connected using a weighted version of shortest paths. For this calculation, the shortest path of any point to the starting point is determined by dividing the edge length by the profit that can be gained when picking 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 with 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 the waypoint are the only options available, however when only one waypoint exists, a second one *has* to be added first. After that, adding, moving or removing of a waypoint are possible until 15 points are reached. Then, a waypoint can only be moved or removed. Using these boundaries ensures that the resulting tour will not end up way too short while also assuring that not too many distance calculations are needed. The number of waypoints determines how many times the distance of a new waypoint is of interest, since for moving and adding, the closest existing waypoint has to be known.

# TODO create visualizations for waypoints, adding, moving, removing

Whether a waypoint is moved, removed or a new one is added is picked randomly within the previously given constraints. The starting point can never be pick for neither moving nor removing.

First, a random point is picked 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  $s$ :  $\text{dist}(j, s)^2$ . 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 moving or adding and will determine the waypoint to choose, also for the remove case. 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 have to be calculated. The closest waypoint of the current list will then be picked and:

- **added:** from the two neighbors of the picked waypoint, the closed 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 A.2)
- **moved:** the previous waypoint is deleted, a new shortest path from the predecessor to the new point and a shortest path from the new point to the successor is calculated and the new point is added as the moved version of the selected waypoint (see A.3)
- **removed:** the picked waypoint is deleted and a new shortest path between the predecessor and the successor is calculated (see A.4)

For the new tour, the overall quality is calculated using 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 2.18. This step is done for a set number of inner runs, before the temperature is adjusted based on 2.15 # TODO pick working schedule and fix code accordingly

To better illustrate the added steps to choosing and building a solution, the pseudocode for the inner loop is given in 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  $\leq$  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, only small changes had to be made to the existing code base, which will be shortly illustrated in the following paragraphs.

**Ant + Greedy and Ant + MinCost** To realize the combination of ant with the two already implemented algorithms, the changes were minor. First, the respective algorithm has to be run to have a base tour to build the ant algorithm from. Then, the initialization has to place trail intensity and pheromones on the edges of the existing tour. Lastly, the importance of the trail intensity has to be higher than in the pure ant algorithm to make it more likely that the ants will follow the previous tour.

**SA + Greedy, SA + MinCost, SA + Ant** To implement the combinations with simulated annealing, again, the tours have to be calculated first. Then, the waypoint list can be created from the base tour that has been built, splitting the tour into 10 waypoints and the paths connecting them. Other than that, no changes are needed.

Realizing combinations is relatively easy for the given variations, however more complicated combinations are possible. Especially combining a genetic meta-heuristic could result in more complex ways of combining algorithms and result in interesting new solutions. This could not be implemented due to time constraints, however possible approaches are discussed in the section about future work (5.2).

## 3.3 Parameter changes

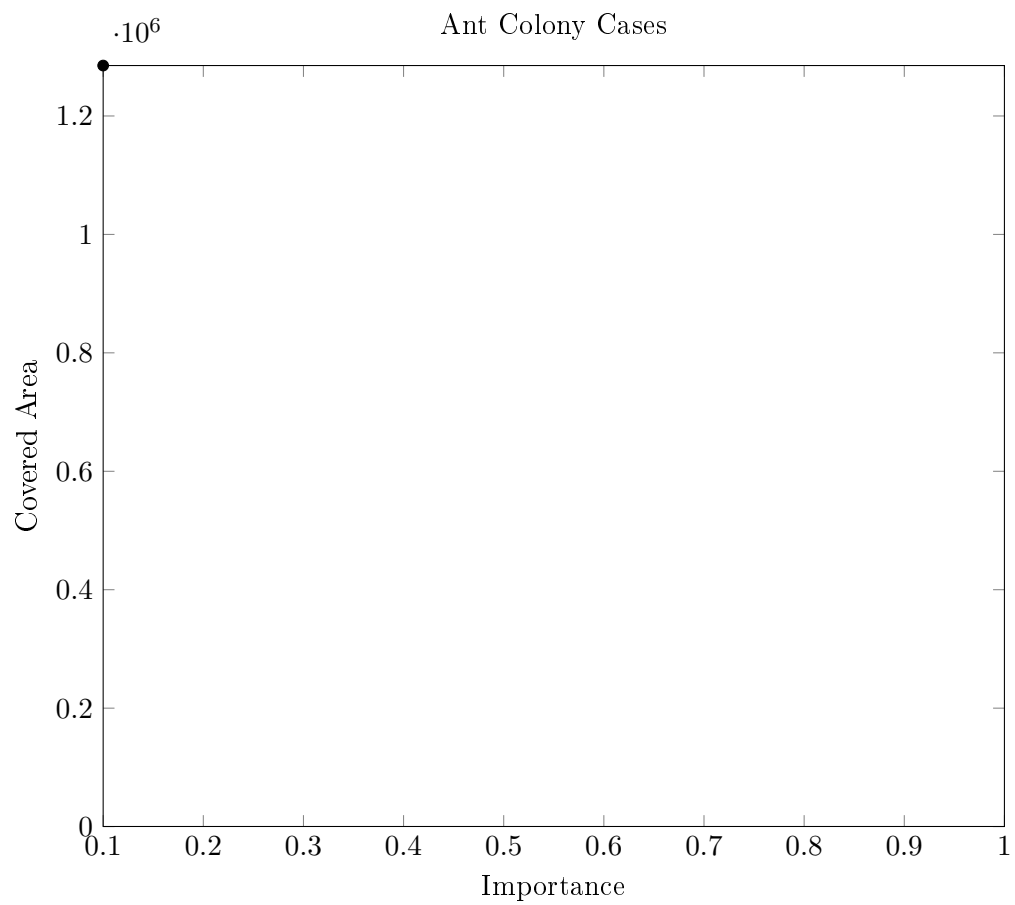
To include the parameters that have been added in the front end, a few minor changes had to be made to the existing algorithms. # TODO check minCost (vsMinCost) spelling

However, since `minCost` already uses the quality calculation to determine the tour quality, nothing else had to be changed. In the quality calculation, the elevation difference and the respective importance was added and the scaling was changed (see 3.8). Other than that, more tags had to be added when creating the problem class, however this was done similarly to the existing tags (path type and surface) and did not need any further changes. The greedy algorithm only takes the profit an edge can add into account and thus also did not need any change.

# TODO add steepness into quality?? check how to do that

## Chapter 4

# Evaluation



**Figure 4.1:** Multiple Line-plots using Tikz



## Chapter 5

# Conclusion

### 5.1 Results

### 5.2 Future Work

# Appendix A

## Source Code

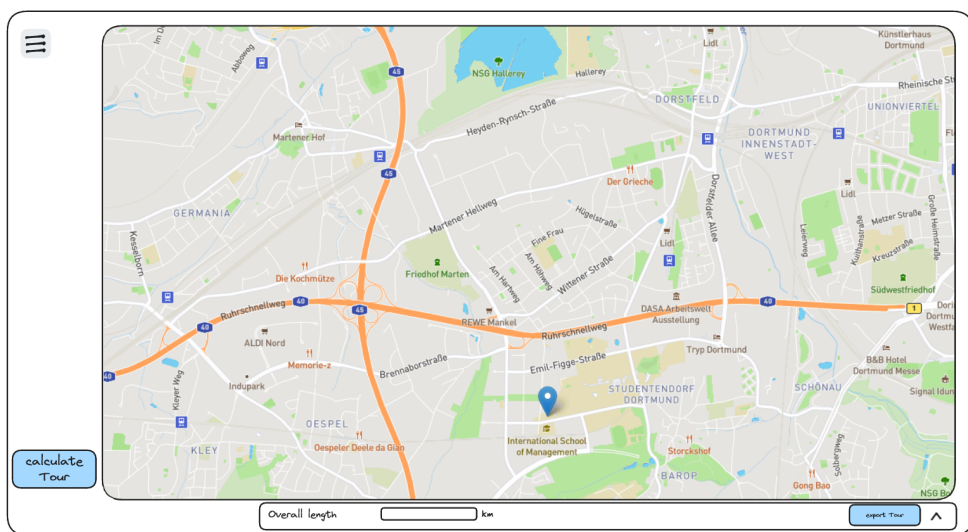
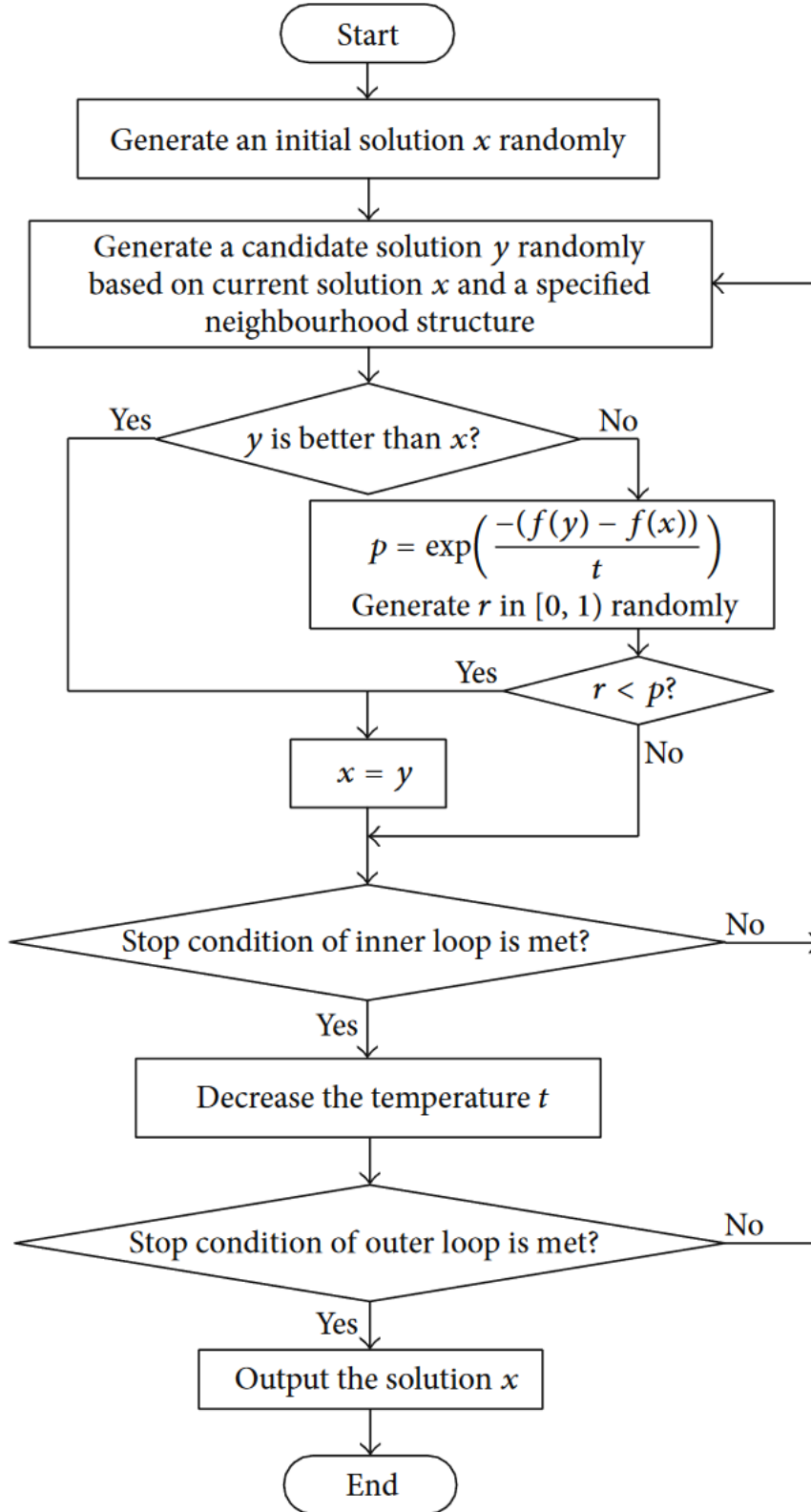
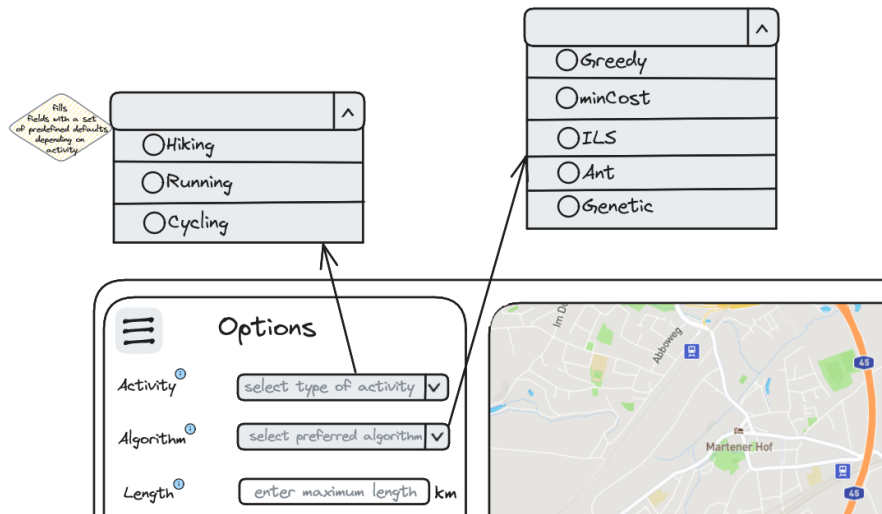


Figure A.2: Design concept for the front end view with all menus folded

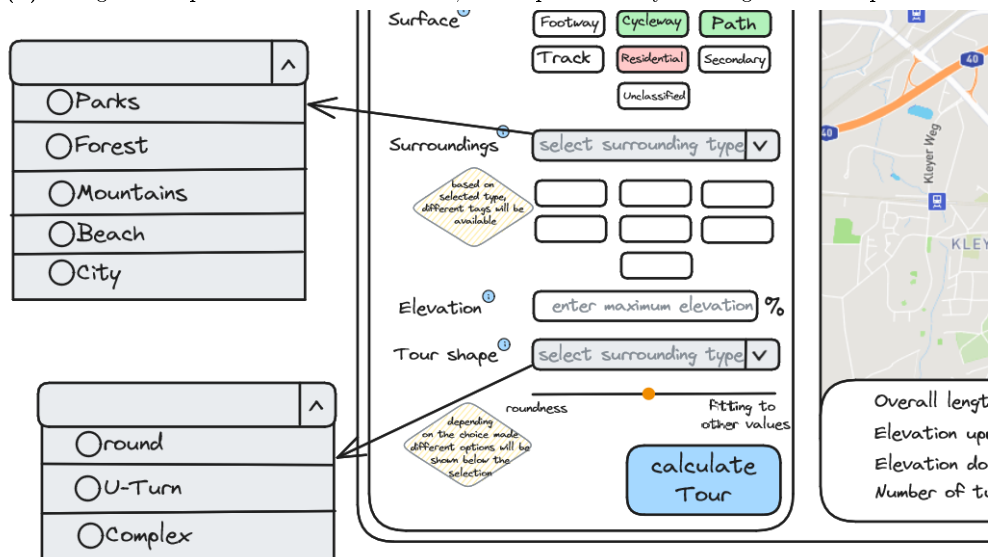




**Figure A.1:** Flowchart of the basic simulated annealing algorithm



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

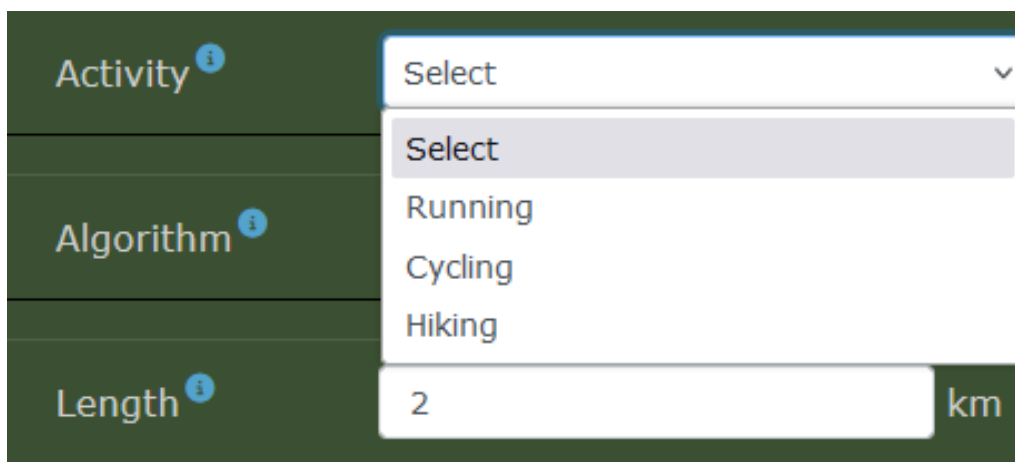


(b) Design concept for the front end view, closeup of a surrounding and Tour shape

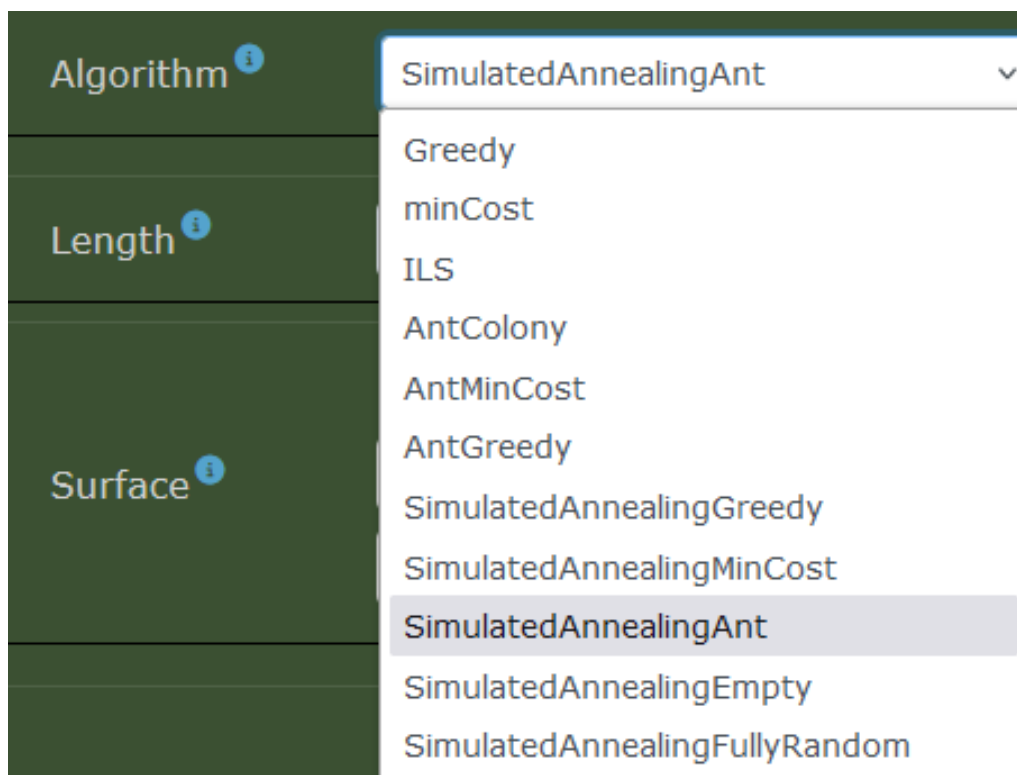
**Figure A.3:** Closeups of the side menu design concept



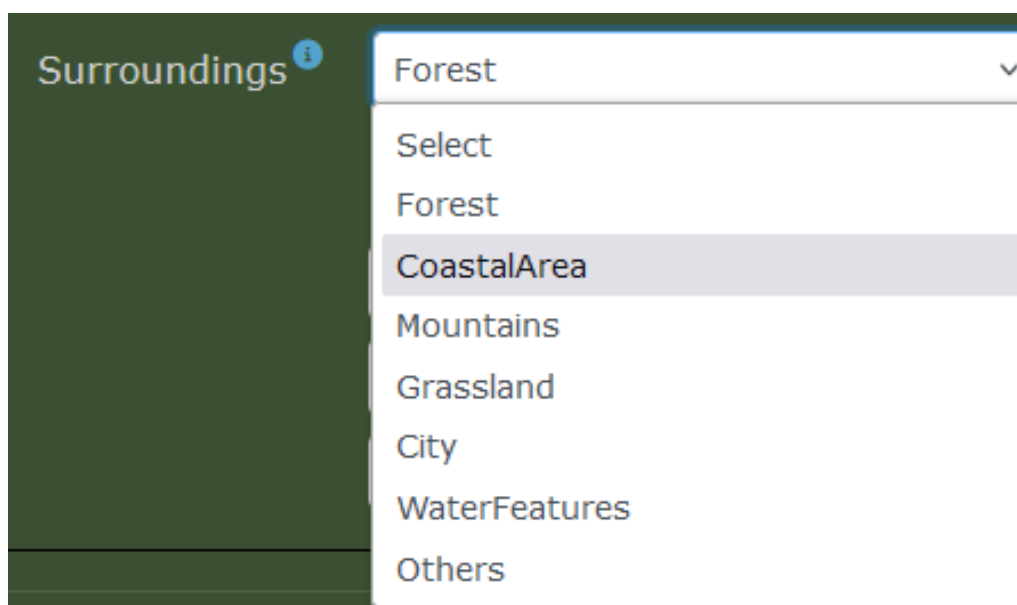
**Figure A.4:** Design concept for the front end view, closeup of the results view



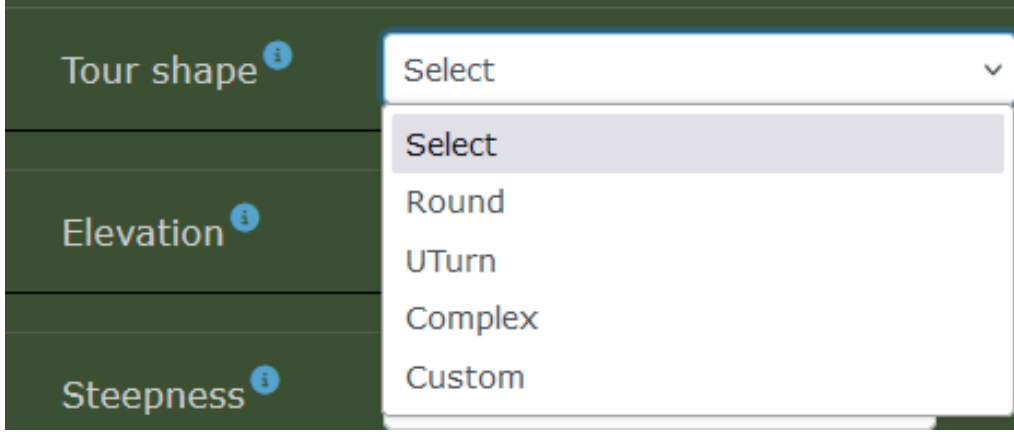
**Figure A.5:** The options to choose from when the Activity drop down is clicked



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



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



**Figure A.8:** The options to choose from when the TourShape drop down is clicked

---

**Algorithm A.1** Simulated Annealing

---

```

1: build waypoint list (starting point only)
2: (optional; for weighted selection of points) calculateDistances
3: calculate probability distribution
4: Select init temperature (t)
5: for  $i = 0$  to numberRepitions do
6:   for  $i = 0$  to numberRuns per temperature do
7:     Generate neighborhood roundtrip (j)
8:     Calculate difference in quality  $d = f(j) - f(i)$ 
9:     if  $d \geq 0$  then
10:      use neighboring solution as current best  $i \leftarrow j$ 
11:   else
12:     random  $\leftarrow$  generate random(0,1)
13:     if random  $< \exp(d/t)$  then
14:      use neighboring solution as current best  $i \leftarrow j$ 
15:   end if
16: end if
17: end for
18: update temperature  $t \leftarrow T(t)$ 
19: end for

```

---

**Algorithm A.2** 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 A.3** 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 A.4** 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
-



# Bibliography

- [1] Emile Aarts, Jan 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.
- [2] Saurav Agarwal and Srinivas Akella. “The Correlated Arc Orienteering Problem”. In: *Algorithmic Foundations of Robotics XV*. Ed. by Steven M. LaValle et al. Cham: Springer International Publishing, 2023, pp. 402–418. DOI: [10.1007/978-3-031-21090-7\\_24](https://doi.org/10.1007/978-3-031-21090-7_24).
- [3] Majid Alivand, Hartwig 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](https://doi.org/10.1016/j.compenvurbsys.2014.10.004).
- [4] O. Babaoglu, H. Meling, and A. 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](https://doi.org/10.1109/ICDCS.2002.1022238).
- [5] Hannah Bast et al. “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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/10.1007/BF02592101).
- [12] Andrew R. Curtis et al. “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/INFOCOM.2012.6195470](https://doi.org/10.1109/INFOCOM.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: [10.1007/978-3-319-91086-4\\_1](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](https://doi.org/10.1007/s00453-009-9341-0).
- [15] Daniel Delling, Thomas Pajor, and Renato F. Werneck. “Round-Based Public Transit Routing”. In: *Transportation Science* 49.3 (2015), pp. 591–604.
- [16] Daniel Delling et al. “Customizable Route Planning in Road Networks”. In: *Transportation Science* 51.2 (2017), pp. 566–591. DOI: [10.1287/trsc.2014.0579](https://doi.org/10.1287/trsc.2014.0579).
- [17] Daniel Delling et al. “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](https://doi.org/10.1007/978-3-642-02094-0_7).
- [18] 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](https://doi.org/10.1002/net.3230140208).
- [19] M. Dorigo, V. Maniezzo, and A. Coloni. “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](https://doi.org/10.1109/3477.484436).

- [20] R. 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](https://doi.org/10.1016/0377-2217(90)90001-R).
- [21] Matthias Ehrgott et al. “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](https://doi.org/10.1016/j.tra.2011.11.015).
- [22] 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](https://doi.org/10.1007/BF02288320).
- [23] Andreas Gamsa et al. “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](https://doi.org/10.1007/978-3-642-38527-8_25).
- [24] 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](https://doi.org/10.1007/978-1-4419-1665-5).
- [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](https://doi.org/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](https://doi.org/10.1016/j.cor.2005.02.020).
- [27] Mads Møller Jensen and Florian ‘Floyd’ 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](https://doi.org/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).
- [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.
- [30] E. 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](https://doi.org/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](https://doi.org/10.1145/2820783.2820835).
- [33] Amgad Madkour et al. *A Survey of Shortest-Path Algorithms*. 2017. arXiv: [1705.02044](https://arxiv.org/abs/1705.02044)[cs].
- [34] Florian ‘Floyd’ 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](https://doi.org/10.1145/1240866.1241045).
- [35] Matthew A. Nystoriak and Aruni Bhatnagar. “Cardiovascular Effects and Benefits of Exercise”. In: *Frontiers in Cardiovascular Medicine* 5 (2018), p. 135.
- [36] Shannon O’Brien and Florian “Floyd” 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](https://doi.org/10.1145/1240624.1240708).
- [37] P. Oja et al. “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](https://doi.org/10.1111/j.1600-0838.2011.01299.x).
- [38] Thomas Pajor. “Algorithm Engineering for Realistic Journey Planning in Transportation Networks”. PhD thesis. Karlsruhe, Germany: Karlsruher Institut für Technologie (KIT), 2013. DOI: [10.5445/IR/1000042955](https://doi.org/10.5445/IR/1000042955).
- [39] Michalis Potamias et al. “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, Nov. 2, 2009, pp. 867–876. DOI: [10.1145/1645953.1646063](https://doi.org/10.1145/1645953.1646063).
- [40] Gerhard Reinelt. *The Traveling Salesman: Computational Solutions for TSP Applications*. Berlin, Heidelberg: Springer, 2003. 231 pp.
- [41] Stefan Ropke. “Heuristic and exact algorithms for vehicle routing problems”. PhD thesis. Technical University of Denmark, 2005.

- [42] 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](https://doi.org/10.1101/cshperspect.a029694).
- [43] Peter Sanders et al. “Shortest Paths”. In: *Sequential and Parallel Algorithms and Data Structures: The Basic Toolbox*. Ed. by Peter Sanders et al. Cham: Springer International Publishing, 2019, pp. 301–332. DOI: [10.1007/978-3-030-25209-0\\_10](https://doi.org/10.1007/978-3-030-25209-0_10).
- [44] Christian Sommer. “Shortest-path queries in static networks”. In: *ACM Computing Surveys* 46.4 (2014), pp. 1–31. DOI: [10.1145/2530531](https://doi.org/10.1145/2530531).
- [45] Wouter Souffriau et al. “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](https://doi.org/10.1016/j.omega.2010.05.001).
- [46] 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](https://doi.org/10.1080/13548506.2012.701755).
- [47] C. Verbeeck, P. Vansteenwegen, and E.-H. 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](https://doi.org/10.1016/j.tre.2014.05.006).
- [48] J Vina et al. “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](https://doi.org/10.1111/j.1476-5381.2012.01970.x).
- [49] Xueyang Wang et al. “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](https://doi.org/10.1109/WPMC.2014.7014828).
- [50] 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.
- [51] Muhammad Rhifky Wayahdi, Subhan Hafiz Nanda 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](https://doi.org/10.25008/ijadis.v2i1.1206).
- [52] Shi-hua Zhan et al. “List-Based Simulated Annealing Algorithm for Traveling Salesman Problem”. en. In: *Computational Intelligence and Neuroscience* 2016 (Mar. 2016), e1712630. DOI: [10.1155/2016/1712630](https://doi.org/10.1155/2016/1712630).



# List of Figures

1.1	An example sketch showing the outlines of two different tours. . . . .	4
2.1	This image shows a conceptual tree of different variations of shortest path algorithms, taken from [18] . . . . .	17
2.2	This figure shows an example of pheromone distribution with real ants. Taken from <i>Ant System: An Optimization by a Colony of Cooperating Ants</i> [19]	20
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 [13]. . . . .	25
3.1	Visualization of the used architecture . . . . .	33
3.2	Design concept for the front end view, including descriptions for drop-downs and pop-ups . . . . .	37
3.3	This figure shows the newly implemented side menu. (a) displays the full side menu (b) is a closeup of the surroundings when <i>forest</i> is selected and (c) shows the three importance sliders when the selected tour shape is <i>custom</i>	39
3.4	This shows the Route information menu with one tour and all the related information unfolded and displayed . . . . .	40
3.5	This shows the Route information menu with several folded tours displayed	40
4.1	Multiple Line-plots using Tikz . . . . .	49
A.2	Design concept for the front end view with all menus folded . . . . .	i
A.1	Flowchart of the basic simulated annealing algorithm . . . . .	ii
A.3	Closeups of the side menu design concept . . . . .	iii
A.4	Design concept for the front end view, closeup of the results view . . . . .	iv
A.5	The options to choose from when the Activity drop down is clicked . . . . .	iv
A.6	The options to choose from when the Algorithm drop down is clicked . . . . .	v
A.7	The options to choose from when the Surroundings drop down is clicked . . . . .	v
A.8	The options to choose from when the TourShape drop down is clicked . . . . .	vi



# List of Algorithms

2.1	AntColony . . . . .	23
2.2	High level Simulated Annealing . . . . .	28
3.1	AntColonyAOP . . . . .	43
3.2	Generate neighborhood roundtrip (j) . . . . .	46
A.1	Simulated Annealing . . . . .	vi
A.2	Add waypoint . . . . .	vi
A.3	Move closest waypoint . . . . .	vii
A.4	Remove closest waypoint . . . . .	vii





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

Dortmund, den June 3, 2024

Lisa Salewsky

