technische universität
dortmund

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

1 Introduction **1**

1.1 Related Work . . . 3

1.1.1 Tour4Me . . . 4

1.1.2 Roundtrip paths . . . 6

1.1.3 Other running related research . . . 9

1.1.4 Apps that assist with sports . . . 10

1.1.5 Other tour optimization ideas . . . 11

1.2 Goal and Methodology . . . 12

1.3 Structure . . . 13

2 Fundamentals and Background **15**

2.1 Arc Orienteering Problem . . . 15

2.2 Shortest Path algorithms . . . 17

2.3 Heuristic Approaches . . . 18

2.3.1 Ant Colony . . . 19

2.3.2 Genetic Algorithms . . . 23

2.3.3 Simulated Annealing . . . 23

3 Implemented Changes **25**

3.1 Application . . . 25

3.1.1 New Architecture . . . 25

3.1.2 Database . . . 28

3.1.3 Interface and Front end changes . . . 29

3.2 Algorithmic changes . . . 32

3.2.1 Ant Colony . . . 32

3.2.2 Genetic Algorithms . . . 32

3.2.3 Simulated Annealing . . . 32

3.3 Parameter changes . . . 32

4 Evaluation **33**

5 Conclusion **35**
5.1 Results 35
5.2 Future Work 35

A Source Code **i**

Bibliography **ii**

List of Figures **iii**

List of Algorithms **v**

Affidavit **vii**

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 [**cherkassky_shortest_1996**, **deo_shortest-path_1984**, **gallo_shortest_1988**, **madkour_survey_2017**, **sommer_shortest-path_2014**, **wayahdi_greedy_2021**]. Furthermore, considerable work on optimizing public transportation [**bast_route_2016**, **delling_round-based_2015**] and managing traffic jams has been done [**delling_time-dependent_20** **delling_customizable_2017**]. Examples are Dijkstra (uni- and bidirectional) [**madkour_survey_20** **sommer_shortest-path_2014**, **wayahdi_greedy_2021**], A* search (also uni- and bidirectional) [**madkour_survey_2017**, **sommer_shortest-path_2014**, **wayahdi_greedy_2021**], greedy algorithms [**madkour_survey_2017**, **wayahdi_greedy_2021**], branch-and-bound algorithms [**lawler_branch-and-bound_1966**], the Bellman-Ford-Moore algorithm [**cherkassky_shortest_1996**] and many more [**delling_engineering_2009**, **gallo_shortest_** **sommer_shortest-path_2014**].

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. Therefor, 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 [**buchin_tour4me_2022**] 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 [**oja_health_2011**, **ruegsegger_health_2018**, **vina_exercise_2012**], the cardiovascular system [**nystoriak_cardiovascular**
as a measure against many different diseases [**oja_health_2011**] as well as for social [**mueller_jogging_2007**, **obrien_jogging_2007**, **wankel_psychological_1990**] and psychological benefits [**biddle_psychological_1993**, **cekin_psychological_2015**, **szabo_psychological**
wankel_psychological_1990]. 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 [**blondiau_economic_2016**]. 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 [**agarwal_correlated_2023**]. Additionally, Gemsa et al.
[**gemsa_efficient_2013**] 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 [**cherkassky_shortest_1996**, **deo_shortest-path_1984**, **gallo_shortest_1988**, **madkour_survey_2017**, **sommer_shortest-path_2014**, **wayahdi_greedy_2021**]), however, for the - more complicated [**gemsa_efficient_2013**] - 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[1] and RouteYou [2] 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] https://www.routeloops.com/, last accessed: 22.03.2024
[2] https://www.routeyou.com, last accessed: 22.03.2024

1.1.1 Tour4Me

The tool which this thesis will be based off, Tour4Me[3] [**buchin_tour4me_2022**], 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. Therefor, maximizing the surrounded area accounts for overlapping parts as well as smaller circles within the tour or generally less round shapes. An example is illustrated in figure 1.1.

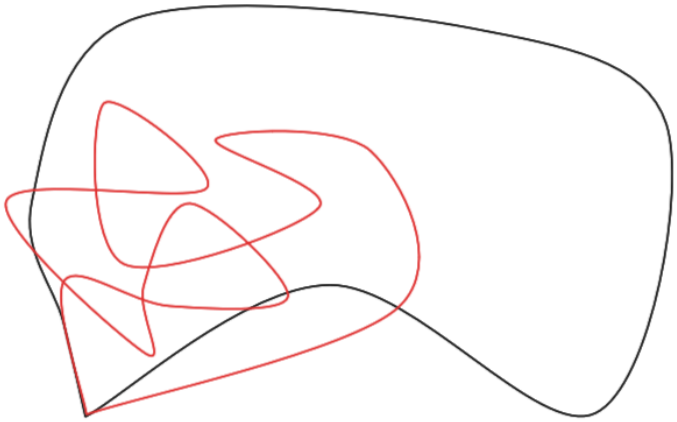


Figure 1.1: An example sketch showing the outlines of two different tours.

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.

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

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) [**buchin_tour4me_2022**].

The Greedy Selection [**buchin_tour4me_2022**, **wayahdi_greedy_2021**] 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.

For Integer Linear Programming [**buchin_tour4me_2022**, **graver_foundations_1975**], the touring problem must be stated in an appropriate form. To do so, a single instance can be encoded as $\mathcal{I}(G, w, \pi, B, v_0)$, containing the Graph G, edge costs w, the profit function π, the budget (length restrictions) B and the starting (and end-) point v_0. Given this encoding, cycles $P = (v_0, ..., v_i, ..., v_0)$, which are always at most of length L, can be built. For the current definition, a few additional variables can be introduced to encode whether or not an edge is part of a solution (and how many times this edge occurs), whether or not an edge is the k-th edge of the solution and whether or not a vertex is the k-th vertex of a solution. Using these variables, constraints can be built to describe the desired behavior of the algorithm.

The MinCost algorithm [**buchin_tour4me_2022**, **gemsa_efficient_2013**] 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 π. 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 [**buchin_tour4me_2022**]. Further details can be found in the original paper [**gemsa_efficient_2013**], 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 this solution, Iterative Local Search [**buchin_tour4me_2022**, **lu_arc_2015**, **verbeeck_extension_2014**] 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 [**buchin_tour4me_2022**].

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. [**gemsa_efficient_2013**] 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" [**gemsa_efficient_2013**]. 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. Gemsa 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 [**pajor_algorithm_2013**], 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. [**ehrgott_bi-objective_2012**] 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. [**verbeeck_extension_2014**] 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" [**verbeeck_extension_2014**] 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 [**jensen_running_2014**] 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 [**loepp_recommending_2018**] used the Partial Shortest Paths algorithm from the idea Gemsa et al. presented [**gemsa_efficient_2013**] 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 [**loepp_recommending_2018**], Loepp and Ziegler also express several issues with and shortcomings of existing apps. Some of these problems have already been identified in the introduction of the two websites RouteLoops and RouteYou (see 1.1.2). They also stress, that most research either concentrates on shortest paths or on the assistance with the training itself rather than finding a good route.

Apps like *Runtastic*[4], *Sportractive*[5] or *Strava*[6] are designed to help runners track the tours they 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*[7] or *SportTracks*[8] do not offer a feature to create routes or roundtrips with a set of preferences [**loepp_recommending_2018**].

A German app that is meant to provide suitable routes for a variety of different outdoor sports - *Komoot*[9] - does offer a route selection. However, only tours other users have planned and added can be selected. No customization or automated route creation is offered here. As Loepp and Ziegler pointed out in their user study [**loepp_recommending_2018**], user feedback, their preferences and the option to customize were very important features

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

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. [**alivand_analyzing_2015**] 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.

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 [**buchin_tour4me_2022**] 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 [**babaoglu_anthill_2002**, **gendreau_handbook_2010**, **wang_application_2014**] (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 [**madkour_survey_2017**, **wayahdi_greedy_2021**]). Additionally, there have been many more heuristic approaches, like local search variants [**braysy_vehicle_2005**, **irnich_sequential_2006**, **ropke_heuristic_2005**] or different neighborhood based ideas [**braysy_vehicle_2005**, **irnich_sequential_2006**, **ropke_heuristic_2005**] 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) [**gendreau_handbook_2010**], the vehicle routing problem (VRP) [**braysy_vehicle_2005**, **irnich_sequential_2006**] or the arc orienteering Problem (AOP) [**agarwal_correlated_2023**, **buchin_tour4me_2022**]) are NP-hard [**reinelt_traveling_2003**].

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 [**potamias_fast_2009**]. 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. [**souffriau_planning_2011**] 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 χ_{ij} Based on Souffiau et al., the objective is to maximize

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}\chi_{ij} \tag{2.1}$$

while adhering to the following constraints:

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

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

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

$$2 \leq v_i \leq n \quad \forall i = 2, ..., n \tag{2.5}$$

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

$$\chi_{ij} \in \{0, 1\} \quad \forall i, j = 1, ..., n \tag{2.7}$$

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) χ 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 [**deo_shortest-path_1984**] 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.

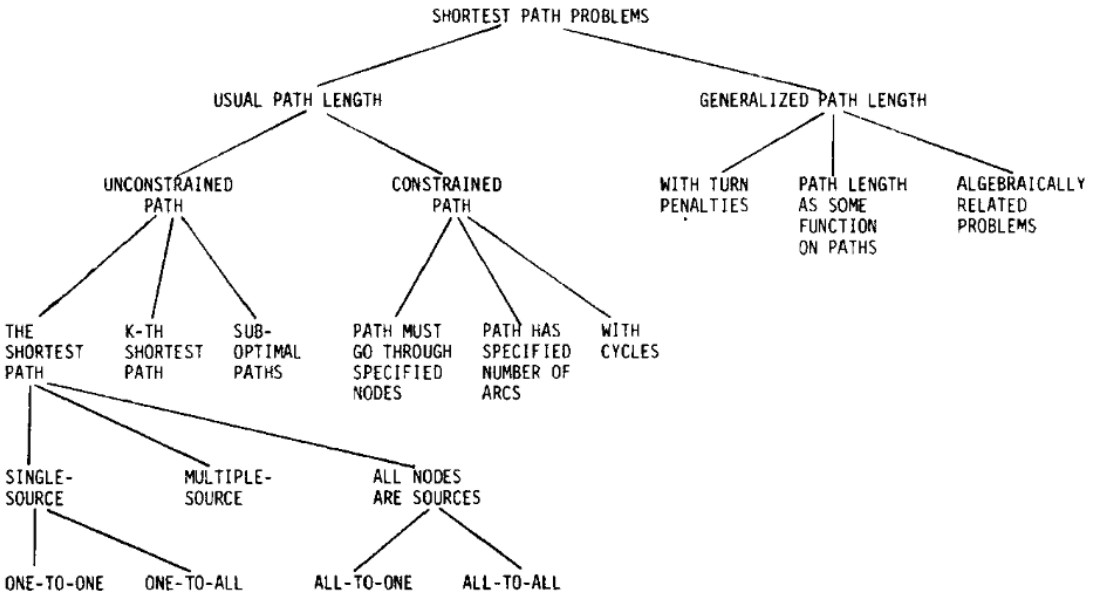


Figure 2.1: This image shows a conceptual tree of different variations of shortest path algorithms, taken from [**deo_shortest-path_1984**]

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

[**sanders_shortest_2019**]. 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 [**sanders_shortest_2019**].

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 [**khamayseh_efficient_2015**]. 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 [**curtis_rewire_2012**].

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 [**gendreau_handbook_2010**]. 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 [**laporte_5_2002**, **ropke_heuristic_2005**]. 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 [**gendreau_handbook_2010**, **dorigo_ant_1996**].

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 [**gendreau_handbook_2010**, **dorigo_ant_1996**].

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.

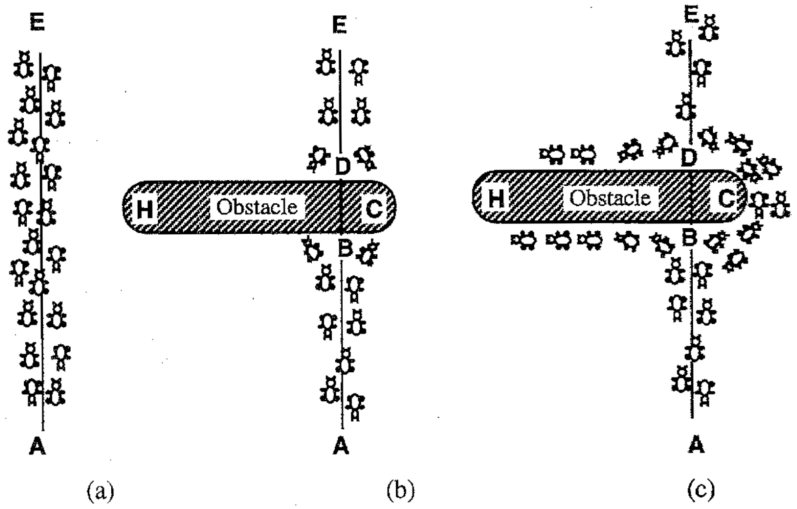


Figure 2.2: This figure shows an example of pheromone distribution with real ants. Taken from *Ant System: An Optimization by a Colony of Cooperating Ants*[**dorigo_ant_1996**]

This behavior can be replicated in virtual graphs for various routing problems. Ant system has been first introduced in 1990 by Dorigo et al[**dorigo_ant_1996**]. 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[**dorigo_ant_1996**].

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. Dorigo et al propose the following three ideas[**dorigo_ant_1996**]:

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

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

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

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[**dorigo_ant_1996**].

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[**dorigo_ant_1996**].

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[**dorigo_ant_1996**].

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[**dorigo_ant_1996**]:

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{2.11}$$

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

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

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[**dorigo_ant_1996**].

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. The visibility ν_{ij}^{k} is calculated using the length of the edge d_{ij} as follows:

$$\nu_{ij}^{k} = \frac{1}{d_{ij}} \tag{2.13}$$

And the transition probability is given by

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

using all previously defined values to calculate visibility ν_{ij}^{k}, trail intensity τ_{ij}, pheromone distribution $\Delta\tau_{ij}$ and $\Delta\tau_{ij}^{k}$. Here, α and β are parameters that influence the weight of visibility and trail intensity. Higher values of α increase the significance of the pheromones

on the trail (setting α to 0 would lead to completely ignoring the pheromone placed) and higher values of β increase the importance of the visibility of an edge (making longer edges less attractive as a result)[**dorigo_ant_1996**]. These parameters will be experimented with and their influence will be evaluated in chapter 4.

In their paper, Dorigo et al suggest middling values for α and β 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 α, no good results could be generated [**dorigo_ant_1996**].

TODO add fomulas and desription how they help constructing paths # TODO add how to use for my work

2.3.2 Genetic Algorithms

2.3.3 Simulated Annealing

Chapter 3

Implemented Changes

This work extends Tour4Me[1], which is an application written in C++ and HTML. The implemented interface uses C# as programming language to enable easy porting of the web application to a desktop or mobile application. To improve the query times, a spatial 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 was changed. The Open Street Map (OSM) data are downloaded and stored in a database. The graph for calculating the roundtrips is thus build from the new database. Furthermore, the whole 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 them. It can be filled with the data needed by using an import python script that creates an osmnx-

[1] http://tour4me.cs.tu-dortmund.de/

graph[2][3][4][5] add all references 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[6] 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.

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

The back end is written in C#[8], 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[9].

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

[2] https://osmnx.readthedocs.io/en/stable/, last accessed: 15.04.2024
[3] https://networkx.org/, last accessed: 22.03.2024
[4] https://wiki.openstreetmap.org, last accessed: 22.03.2024
[5] https://wiki.openstreetmap.org/wiki/Main_Page, last accessed: 19.04.2024
[6] https://open-elevation.com/, last accessed: 20.03.2024
[7] https://learn.microsoft.com/en-us/sql/sql-server/sql-docs-navigation-guide?view=sql-server-ver16, last accessed: 22.03.2024
[8] https://learn.microsoft.com/en-us/dotnet/csharp/, last accessed: 22.03.2024
[9] 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
[10] https://devdocs.io/html/, last accessed: 22.03.2024
[11] https://devdocs.io/css/, last accessed: 22.03.2024
[12] https://devdocs.io/javascript/, last accessed: 22.03.2024
[13] https://getbootstrap.com/docs/4.3/getting-started/introduction/, last accessed: 22.03.2024
[14] https://api.jquery.com/category/ajax/. last accessed: 22.03.2024

The map is a leaflet[15] 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.

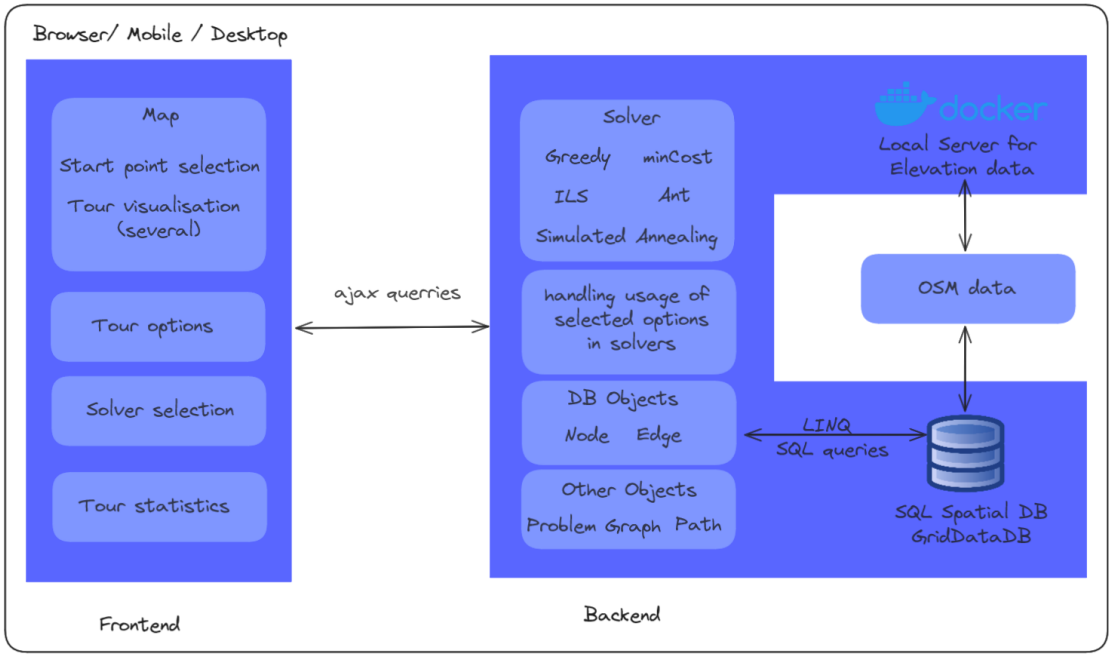


Figure 3.1: Visualization of the used architecture

In the above visualization, the whole application, the distinct parts and features are illustrated. The front end is realized as a web application, running in the browser but can also be customized to be executable as a mobile or desktop application (see 5.2). Here, the map is visualized using leaflet. In this map, the marker can be set to the current location - if the permission to access the 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 using the map and a polygon built from the respective points. For debug purposes, there is a feature to show the whole graph that is being used for the calculation using the current 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.

[15] https://leafletjs.com/, last accessed: 20.03.2024

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. Despite the database creation and the adding of points being relatively slow, this is a process that only needs to be ran once 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_from_place(place_name, network_type='all', custom_filter=
    ↪ custom_filter)
```

Here, the `place_name` is the name of the place for which osmnx data should be gathered. 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[16]: "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:

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[17] was used.

The

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

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

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

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[18]

3.1.3 Interface and Front end changes

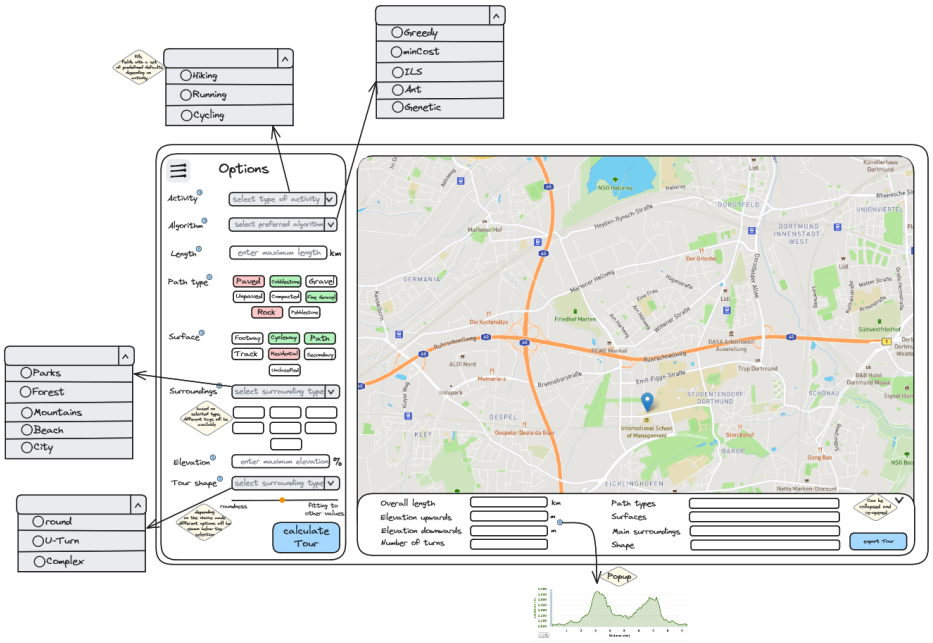


Figure 3.2: Design concept for the front end view, including descriptions for drop-downs and pop-ups

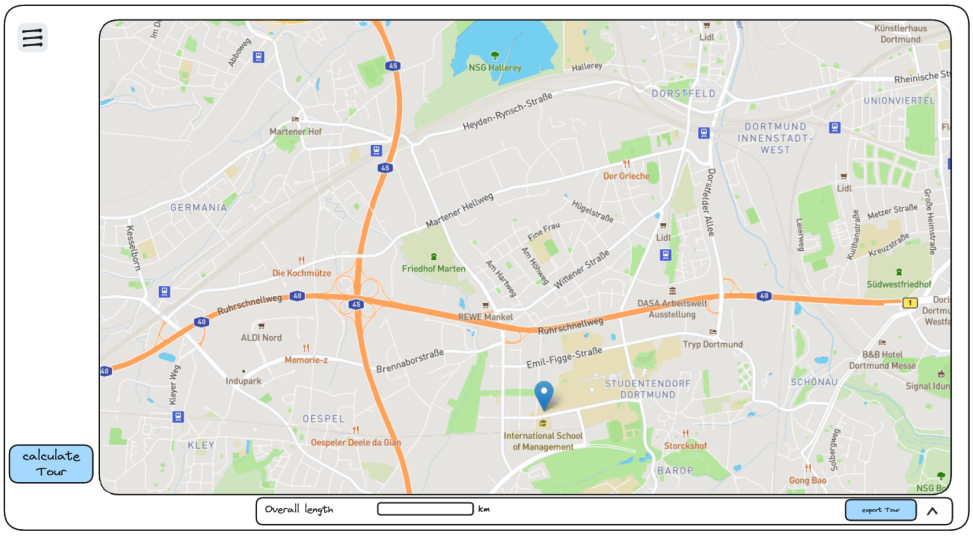
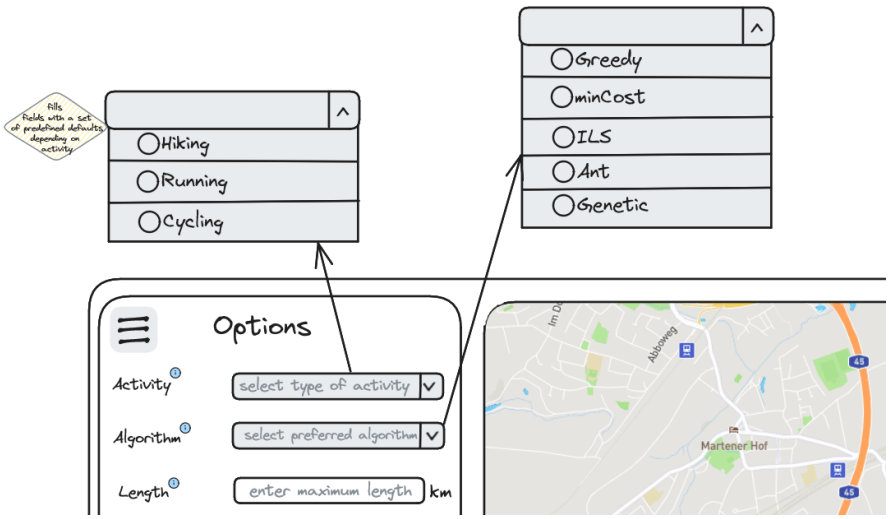
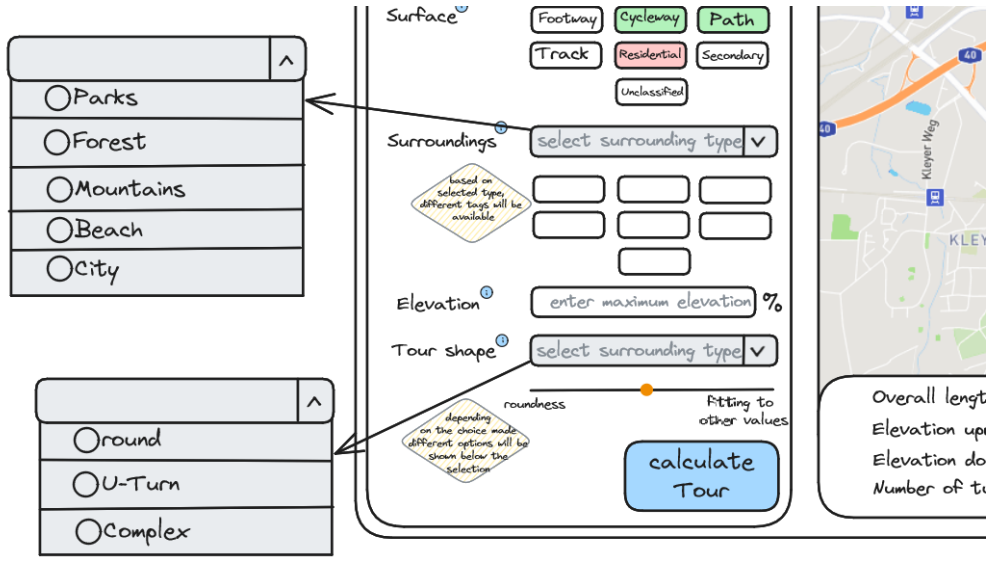


Figure 3.3: Design concept for the front end view with all menus folded



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



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

Figure 3.5: Design concept for the front end view, closeup of the results view

3.2 Algorithmic changes

3.2.1 Ant Colony

3.2.2 Genetic Algorithms

3.2.3 Simulated Annealing

3.3 Parameter changes

Chapter 4

Evaluation

Chapter 5

Conclusion

5.1 Results

5.2 Future Work

Appendix A

Source Code

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 [**deo_shortest-path_1984**] 17
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*[**dorigo_ant_1996**]

3.1 Visualization of the used architecture 27
3.2 Design concept for the front end view, including descriptions for drop-downs and pop-ups 29
3.3 Design concept for the front end view with all menus folded 30
3.5 Design concept for the front end view, closeup of the results view 31

List of Algorithms

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

Dortmund, den April 24, 2024

Lisa Salewsky