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

Chapter 1

Introduction

Algorithms for shortest tours 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 activity outdoors, oftentimes a roundtrip is more in line with these activities than shortest paths between two points. Most sports or general outdoor activities people do in their free time are not means to get from one point to another. Especially if these hobbies involve driving to a park or into areas where the landscape is more fitted to the person's 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. But also 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 [10, 16, 19, 27, 37, 43]. Furthermore, considerable work on optimizing public transportation [4, 13] and managing traffic jams has been done [12, 14]. Examples are Dijkstra (uni- and bidirectional) [27, 37, 43], A* search (also uni- and bidirectional) [27, 37, 43], greedy algorithms [27, 43], branch-and-bound algorithms [26], the Bellman-Ford-Moore algorithm [10] and many more [15, 19, 37].

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 particular, in many cases people plan round-trips, especially for outdoor activities like running and cycling. For training towards a specific goal or even as a pastime hobby, it is often desired to have roundtrips of a certain length. Additionally, people typically enjoy running or cycling on more appealing paths in nature rather than between high buildings and on softer ground rather than on asphalt. Thus, a lot more information have 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, namely a modified version of the arc orienteering problem (AOP) (see section 2.1), which will be called the *touring problem* in accordance with Tour4Me [8] which builds the basis for this thesis (see 1.1.1).

Outdoor activities like running and cycling, but most other sporty hobbies can be fun while also having many inherent benefits: For overall health [31, 35, 40], the cardiovascular system [29], as a measure against many different diseases [31] as well as for social [28, 30, 42] and psychological benefits [5, 9, 38, 42]. Furthermore, touristic cycling for cities can also be considered beneficial - in this case for a city gaining more tourism rather than for an individual [6]). Considering all these advantages and payoffs, 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 of starting any kind of outdoor activity. Having such a tool could result in an overall larger population doing some 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 encourage 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 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 sports 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 arc orienteering problem, the computational complexity will be at least as hard. Thus, the customizable arc orienteering problem will be at least NP-hard [1]. Additionally, Gemsa et al. [20] 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 [10, 16, 19, 27, 37, 43]), however, for the - more complicated [20] - 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.

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, it was not possible to obtain any details about used algorithms, heuristics, meta-heuristics or even the language they used for programming these solutions. 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[3] [8], incorporates some of the mentioned customization options in its 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 or route complexity. However, the tour can be optimized for a circular route by maximizing the covered area of the tour.

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

[1] https://www.routeloops.com/, last accessed: 22.03.2024
[2] https://www.routeyou.com, last accessed: 22.03.2024
[3] http://tour4me.cs.tu-dortmund.de/, last accessed: 18.04.2024

offered choices include a Greedy Selection approach, Integer Linear Programming, MinCost with *Waypoints* - a shortest paths variant - and Iterative Local Search (ILS) [8].

The Greedy Selection [8, 43] is the simplest algorithm which only ensures that the chosen route is a roundtrip. It builds its path 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 [8, 22], 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 it 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, constraints can be built to describe the desired behavior of the algorithm.

The MinCost algorithm [8, 20] needs the *waypoints*, which are intermediate points used to calculate shortest paths between them (see 1.1.2 for more details), because it is typically meant to solve shortest path problems. Thus it would always choose not leaving the starting position without the added points. Even though this algorithm is not originally meant to solve roundtrip problems, it takes into account the cost and profits of edges to create a solution tour, which makes it 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 the highest profit tour is returned, all possible combinations are calculated and the optimum is returned[8]. Further details can be found in the original paper [20], 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, the Iterative Local Search can be applied to improve the found tours. From the returned roundtrip, the algorithm removes partial paths P from the current best solution S and tries to iteratively add new parts that improve the solution profit while always staying within the given budget $(B - w(\frac{S}{P}))$. Since searching for viable edges is performed using a depth first approach, bounding the maximum depth of this step can drastically speed up the algorithm. To keep track of the added length and profit, two variables (l and p respectively) are introduced. These start with an initial value of one and are raised by a single increment for each iteration. p is reset when the starting point is reached by the removal step. l is reset when the maximum length for the solution is reached. The best solution is improved constantly until the user selected time limit is reached [8].

#TODO add more citations? –> see Tour4Me paper

1.1.2 Roundtrip paths

As already stated above, existing tools leave out certain data like elevation or path types. This impacts the quality of the created routes for 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 through a park for half of the route. While this still may be a good choice for other users - joggers who prefer more challenging routes or people who want to hike and enjoy ascending - this can be undesirable for beginners. Some people could prefer running through the city over running through a park when the elevation matches their preferences better in the city. For these users, the created route would be highly unfavorable, even though it 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.

RouteLoops & RouteYou

RouteLoops has two text fields for entering the starting point and the length of the trip. Aside from that, no real customization is possible. It does have a few features to show more information about the route like showing distance markers or elevation, however, these can not be used as inputs to get a route with - for example - as little elevation as possible. Apparently it can also show route difficulty for the United States, however even when creating a route in the United States, no result was shown. RouteLoops also does not actually create loops but rather picks a route that has 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 crate a roundtrip, some „waypoints“ are created. These can be removed or more can be added in when editing the tour. Between the waypoints, it seems like a shortest path is tried

RouteYou offers several different 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. Also, while both do offer the possibility to edit the returned roundtrip, this editing changes the length of the route arbitrarily. Furthermore, it is not possible to specify directly what type of underground or surroundings etc. are preferred.

Computing Running Routes

The problem of calculating good running roundtrips is not new. In addition to the commercial applications, there also are research papers on this subject. One of these papers is „Efficient Computation of Jogging Routes“[20] which presents two ideas to handle the new routing problem the authors labeled „Jogging Problem“. It 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. Hence, it is named „Relaxed Jogging Problem“. 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 [20].

The second problem is chosen as the one to optimize, since it enables the addition of other conditions than just the length of a tour. For this, two different ideas are proposed. The first approach - „Greedy Faces“- is based on the idea of extending previous cycles. It 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 user specified length. The second algorithm was named „Partial Shortest Paths“ and uses via-vertices. These are a number of new points that can be connected with shortest paths. When the via-vertices are connected with each other and the start, they form a roundtrip [20].

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 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 (turning by 180 degrees). They define the angle between two edges as a *turn* if it is larger than 15 degrees (and equal to or less than 180 degrees). These turns can then be used to determine the complexity of a tour: More turns meaning a more complex tour [20].

The ideas presented in this paper are also used by Thomas Pajor in his dissertation „Algorithm Engineeering for Realistic Journey Planning in Transportation Networks“[32], where he talks about Computation of Jogging Routes. In his last chapter, he describes details about the algorithms and mainly focusses on Greedy Faces and Partial Shortest Paths as well.

TODO is this too detailed with the following paragraphs? Should I shorten this by removing greedy faces and partial shortest paths?

Greedy Faces Greedy Faces is built from an already existing path by extending it. For this, blocks outside the given tour that are adjacent to the current path are used. The previous cycle then is changed so that it encloses the chosen block and thus extends the preceding route. New blocks are picked until the desired length is reached. To ensure only blocks that correspond to faces are picked, a preprocessing phase is introduced that identifies faces of the graph. During this step, first, dead-ends are removed, so the resulting graph will be two-connected. Faces then are defined by the edges that surround them. While identifying all faces, a dual graph $G^\star = (V^\star, E^\star)$ for $G = (V, E)$ is built as well.

The Greedy Faces algorithm then works on the dual graph $G^\star$, selects a face f from $V^\star$ which has a surrounding path that contains the starting point s. Then, a Breadth First Search Tree T is built, starting at f, until the desired length (a relaxed version $(1+\varepsilon)L$) is exceeded. The resulting tour will be a simple path iff all vertices in V without the ones in T are connected and contain s. The final jogging path can be extracted by taking the cut edges between the tree T and the remaining vertices. This always forms a cycle and thus builds a roundtrip.

For building a path which optimizes all constraints, the three introduced measures for badness, number of shared edges and the number of turns are used. The badness function is incorporated into a different force function $\varphi(f,p) = \frac{(\mathrm{bad}(f)-0.5)l(f)}{|\vec{d}|^2} \cdot \frac{\vec{d}}{|\vec{d}|}$ which can assign positive and negative badness values to edges. Furthermore, the force function uses the cost of the face and a vector $\vec{d} = \vec{p} - \vec{C}(f)$ which is built from the geometric center $\vec{C}(f)$ of a face to any point $\vec{p}$. This force vector can then be used to calculate the best next edge for extending the current path by maximizing $\varphi(g)cos(\measuredangle(\varphi(g), C(f) - C(P)))$, measuring the angle between the directed force vector and the geometric center point *C(P)* of the path that has been built so far. The force vector is used in the Breadth First Search but it doesn't have to be the only criteria. An extension to include other measures like roundness or complexity can be created as well. [20]

After the tour has been created, it will be smoothed to reduce the complexity. This is done by building a subset (always including s) of the nodes contained in the created path and computing shortest paths between all vertex pairs in this subset. Concatenating them will then return a smoothed path. This approach is extended to again take badness into account as to not create a bad final path because of the smoothing step. [20]

The greedy faces algorithm does extend an initial cycle, but it has no guarantees on the length of the final returned path. It can deviate without constraints from the original user specification, resulting in paths that can be way too short or way too long. [20]

Partial Shortest Paths Since Greedy Faces cannot give any guarantees, the authors pursued a second approach to calculating results that are ensured to deviate at most by a small ε. The partial shortest paths are based on a set of via-vertices and named by the number of intermediate points created. In the paper, 2-via-routes and 3-via-routes are presented.

For two intermediate points, three shortest paths have to be calculated. These furthermore have to have a length of $\frac{1}{3}L \pm \varepsilon$, building a triangle. Again, the shortest path calculation used will also consider the badness of the edges when selecting them. Optimizing this metric will return a set of feasible candidate paths that are „nice“ - as the authors describe this property - and create a ring around the starting point.

From this point, another ring with a diameter of $\frac{1}{3}L \pm \varepsilon$ is calculated from every vertex within the first ring. All elements that are within the intersection of both rings are valid candidates for the third point to be selected. The final path is created by picking the tour with a minimal badness from all feasible combinations of s and the two other selected vertices. [20]

The three point variant 3-via-routes is an extension to improve the smoothness around the two selected vertices for building the initial triangle. The algorithm then builds the ring around the starting point as in the first version but with a narrower radius of $\frac{1}{4}L \pm \varepsilon$. Then, an even narrower ring is created, using a new parameter $\alpha \in [0.5, 1]$ as a condition for the radius of the new ring: $\frac{\alpha}{3}L \pm \varepsilon$. The value of α and a new point m in the middle of the created path control the smoothness around the two other vertices.

From the two triangle points u and v, new points *u'* and *v'* within the narrower ring are obtained by following the shortest path trees. Then, new rings around these vertices are calculated, using a radius of $\frac{2-\alpha}{4}L \pm \varepsilon$ to ensure a distance of $\frac{1}{2}L \pm \varepsilon$ for all vertices within each of these rings. This results in a ring containing possible middle vertices. Then, for all pairs *u'* and *v'*, the intersection of their respective rings can be built and all middle vertices that will yield a smooth path for the two triangle points u and v will be selected as middle point candidates. Finally, the path along the vertices that has minimum badness will be returned as the result. [20, 32]

Both, the Greedy Faces as well as the Partial Shortest Paths offer a solution to the roundtrip problem. They also allow for customization of the tour using different constraints. This is why the Partial Shortest Paths approach is already used as one available option in Tour4Me. The constraints and parameters that can be used to influence properties of the tour offer fewer customization options than it is planned for this thesis. # TODO do I have to specify what is different from my approach for every paper I mention here?

Other running related research Aside from the few concretely related papers and applications, some general research regarding running with technology has been done. Jensen and Mueller focus on the usage 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 on what aspects to focus 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 when trying to produce apps or gear for runners.

Other papers like [**loepp_recommending_nodate**] by Loepp and Ziegler used the Partial Shortest Paths algorithm from [20] to build a recommendation based app. However, they tried to incorporate more criteria, for example elevation or 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 no user was 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 on Android smartphones only.

Computing Cycling Routes

As for running, there are some papers discussing ideas for generating cycling tours. Ehrgott et al. discuss a bi-objective model that takes travel time and „suitability for cycling“ into account. This suitability is defined as a combined measure of objective factors. These contain for example the volume and speed of traffic on the roads, which can impact the safety of these path segments. But elevation and steepness of the terrain and similar values are taken into account. They all are accumulated into the one measure of suitability, so that there are only two values to optimize at the same time. [18]

The authors do offer a solution for the fact that many of the values can have a different importance for every person. While some people might not want hilly routes at all, others could enjoy the challenge they propose. Because of this, they chose to offer a choice set of several alternative routes, from which the user can pick the one that works best for their preferences. [18]

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

Verbeeck et al. do concentrate on cycle trips in their paper. They build a „cycle trip planning problem (CTPP)“ as a different version of the arc orienteering problem. The initial idea is to use a meta-heuristic approach of Iterated Local Search (ILS) to build roundtrips that optimize the profit of the trip. Since the arc orienteering problem is already NP-hard and the CTPP is even more complex, attempting to solve it with an analytic, exact algorithm will not be feasible in terms of time constraints. Because of this, 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, but will be too slow for larger problem space instances. [39]

Therefore, they developed the ILS approach which can be split up into three phases: The initialization, the improvement and the selection. During the initialization gathers a first set of possible solutions by using the insert move that aims to find a path with the highest score. It starts with every arc that leaves the starting point and builds a maximum-profit path until it obtains a feasible solution. This step is done for all possible starting points, so several solutions are created. These are then optimized in the improvement phase. To do so, a part of the solution is removed during every iteration. Then 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. [39]

Using this approach, the authors can 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" for further development. A thing they 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. [39]

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 out the existing implementation 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.

1.1.3 Apps that assist with sports

Aside from Tour4Me, RouteLoops and RouteYou, another prototype for running route recommendations has been developed. In the corresponding paper[**loepp_recommending_nodate**], the authors express the problems with existing apps, some of which have already been identified in the introduction of the two websites (see 1.1.2). They also stress, that most research either concentrates on shortest paths or - if it is research and app development specifically for running - 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

[4] https://www.runtastic.com/
[5] http://sportractive.com/
[6] https://www.strava.com

the tours they already ran. They measure pace, position, height meters and several other stats to then be able to present the user feedback of the run they've done. 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_nodate**].

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, it relies only on tours other users have planned and added. No customization or route creation is offered here. As the authors of the paper "Recommending Running Routes: Framework and Demonstrator"[**loepp_recommending_nodate**] pointed out in their user study, it is very important to take user preferences into account. No participant of the study decided to try out a route another member had recorded, which further stresses the importance of personalized route generation[**loepp_recommending_nodate**].

1.1.4 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, what 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, the authors created a route choice model. This showed 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 scenic view along it. These points that were considered panoramic were gathered from a set of geo-tagged photos and from travel blogs. [2]

From their experiments, users were happy to take detours that were on average 90% longer than the fastest tour. This shows how important the view and surroundings can be when the goal is not only to be quick, but also takes different 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. [2]

[7] https://www.trainingpeaks.com/
[8] https://sporttracks.mobi/
[9] https://www.komoot.de/

1.2 Goal and Methodology

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

First, an interface for testing the new approaches has to be built. This interface also needs an overlay for adding in user options like the length of the desired roundtrip, as well as other preference inputs. Based on this interface, different algorithms can be added and compared with each other to find the ones that will produce relatively good outputs. However, there can be very different definitions of what makes a result good or high quality. An ideal algorithm would be fast, always generate a route and use all the users' preference inputs. However, it is not possible to achieve all these goals with just one algorithm. Therefore, different approaches will be implemented and analyzed according to how well they fulfill the previously mentioned criteria.

Some of the possible approaches include different implementations of genetic algorithms [21], of ant colony or anthill algorithms [21, 3, 41] as well as possible hybrid versions. These hybrids can either be hybrids of one of the meta-heuristics with - for example - local search algorithms [21, 41] or hybrids of these two algorithms joined together. Furthermore, if there is enough time left, it is also possible to include the new algorithms into already implemented ones to improve those.

When the best algorithms for this application have been determined, they will be integrated into the already existing Tour4Me application. The aim is 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. It can be equally important to improve the interface, add more options like elevation data, include more information (for example previously used routes) etc. There are several opportunities and options 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. It is another option to put more work into 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 [27, 43]). Additionally, there have been many more heuristic approaches, like local search variants [7, 23, 34] or different neighborhood based ideas [7, 23, 34] 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) [21] or the vehicle routing problem [7, 23]) are NP-hard [33].

Furthermore, finding a shortest path is important in various parts of daily life. Whether it is about 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. Here, different algorithms can help calculating friend networks or support the routing of data through virtual networks. [27]

2.1 Arc Orienteering Problem

2.2 Shortest Path algorithms

Shortest path algorithms have been studied extensively for many years. In 1994, Deo and Pang 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. [16]

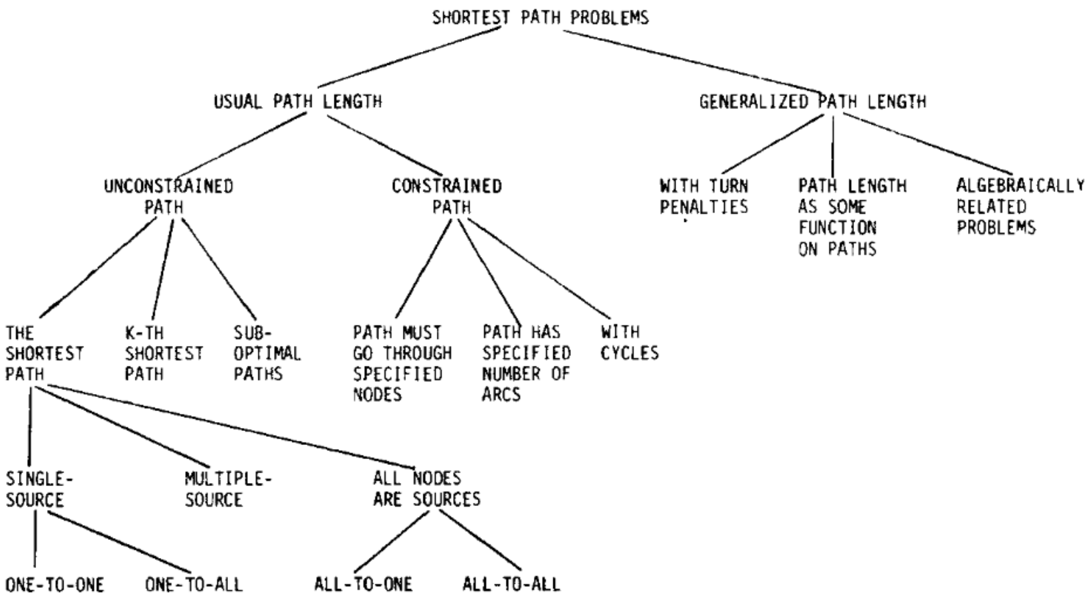


Figure 2.1: This image shows a conceptual tree of different variations of shortest path algorithms, taken from [16]

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 it 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) [27]. The first - SSSP - only uses one single starting point and tries to find the one shortest path between it and one or all other vertices. 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. Aside from these two categories, many more can be found to describe and sort types of approaches. Madkour et al propose a different taxonomy than Deo and Pang did (see [16] and figure 2.1) to help classify the different algorithms into specific categories. [27]

Which of these algorithms performs best is typically dependent on the type of graph it 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.

2.2.1 Single Source Shortest Paths

Following the classification graph (see 2.1), on the bottom left, there is a class „single-source“, which envelops one to one and one to all paths. This class describes all problems that have a single source node and - following the respective parent nodes - have a usual path length, are unconstrained and find a singular short path that is not an approximation. 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 [36]. 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 [36].

For these, many algorithms have been developed over time, trying to solve more types of problems. Some algorithms are for undirected graphs (for example Dijkstra [43]), some for directed graphs with non-negative weights or with arbitrary weights but without negative cycles (like Bellman-Ford-Moore [10]). The respective lists are long and for many more specific cases, there are even more specific algorithms [16, 27].

2.2.2 All Pairs Shortest Paths

The path on the left of the classification graph also shows the class „all nodes are sources“, which encompasses the all to one and all to all cases. In these lie all problems where it is necessary to build paths from every node to either a single destination or to all other nodes. As with single source shortest paths, this branch also describes paths that have a usual length, are unconstrained and give the respective best shortest path for every pair of nodes. 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 [24]. 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 [11].

2.2.3 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. 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 it 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, several different ideas have been formed. These can then be categorized into construction heuristics, improvement heuristics and meta-heuristics [34]. Sometimes, a fourth category for two-phase heuristics is included as well (see [<**empty citation**>]).

#TODO is this correct for heuristics in general? The paper refers to heuristics for VRP

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. [25, 34]

2.3 Meta-heuristics

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. [21]

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 it and will in turn lay down their pheromone as well, leading to an accumulation of these on good paths. Over time, the pheromones dissipate and when they aren't renewed, will evaporate completely, decreasing the attractiveness of the corresponding path [21, 17].

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 its pheromone on the way back, it increases the likelihood of it's path being taken. 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 [21, 17].

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 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 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. So, 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.

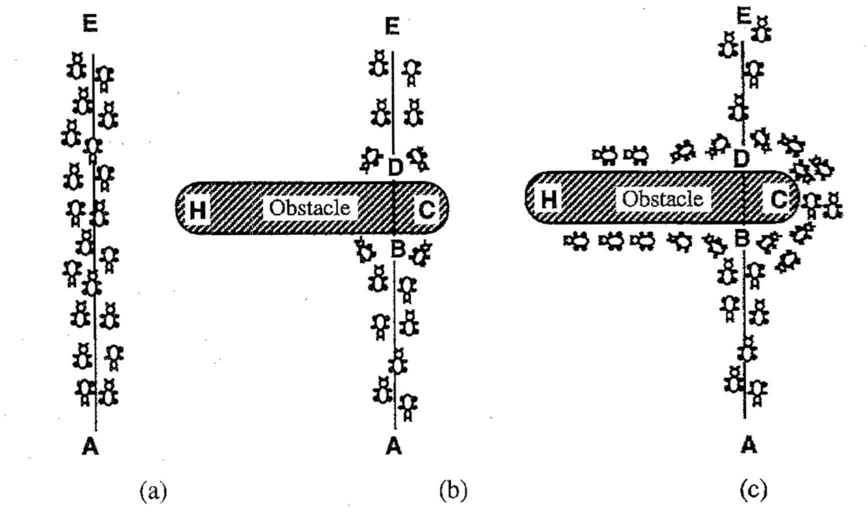


Figure 2.2: This figure shows an example of pheromone distribution with real ants. Taken from *Ant System: An Optimization by a Colony of Cooperating Ants*[17]

This behavior can be replicated in virtual graphs for various routing problems. Ant system has been first introduced in 1990 by Dorigo et al[17]. In the paper, the authors describe how to use ants for solving the traveling salesman problem (TSP). This 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[17].

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 it 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 it will be laid on a path can be calculated in several different ways. Dorigo et al propose the following three ideas[17]:

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

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

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

Here, equation 2.1 is the default the authors used for solving the TSP. Q is a constant that has to be picked according to the problem in question. 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[17].

Equation 2.2 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 equates to a not-scaled version of equation 2.1 with the given use-case of a set length for the tour[17].

The last equation 2.3 divides the constant by the length - or the cost - of each edge when it is used. Doing this 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[17].

It is possible to define other ways to calculate the pheromone placement. 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[17]:

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

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. To do this, a new parameter ρ needs to be introduced. It describes how much of the pheromone stays on the trail between two time steps. So, the overall pheromone intensity can be described by

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

where $\tau_{ij}(t)$ is the previous pheromone intensity and t+n describes the next time step after one full tour was created in n steps[17].

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. To do this, 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:

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

And the transition probability is given by

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

using all previously defined values to calculate visibility ν_{ij}^k, trail intensity τ_{ij}, pheromone distribution $\Delta\tau_{ij}$ 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)[17]. 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 [17].

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, which is an application written in C++ and HTML. The implemented interface uses C# as its 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-graph[1234] 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[5] 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[6], because it can handle spatial data, supports spatial queries and works well in combination with the C# implementation.

The back end is written in C#[7], as it 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, it allows for using SQL queries and filtering using LINQ for easy runtime database querying[8].

The front end is implemented using HTML[9], CSS[10], JavaScript[11] and C# code behind. Here, the base-styling is done using bootstrap[12], 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[13] are used.

[1] https://osmnx.readthedocs.io/en/stable/, last accessed: April 16, 2024
[2] https://networkx.org/, last accessed: April 16, 2024
[3] https://wiki.openstreetmap.org, last accessed: April 16, 2024
[4] https://wiki.openstreetmap.org/wiki/Main_Page, last accessed: April 16, 2024
[5] https://open-elevation.com/, last accessed: April 16, 2024
[6] https://learn.microsoft.com/en-us/sql/sql-server/sql-docs-navigation-guide?view=sql-server-ver16, last accessed: April 16, 2024
[7] https://learn.microsoft.com/en-us/dotnet/csharp/, last accessed: April 16, 2024
[8] https://docs.telerik.com/devtools/aspnet-ajax/controls/grid/asp.net-3.5-features/linq-to-sql---binding-and-automatic-crud-operations, last accessed: April 16, 2024
[9] https://devdocs.io/html/, last accessed: April 16, 2024
[10] https://devdocs.io/css/, last accessed: April 16, 2024
[11] https://devdocs.io/javascript/, last accessed: April 16, 2024
[12] https://getbootstrap.com/docs/4.3/getting-started/introduction/, last accessed: April 16, 2024
[13] https://api.jquery.com/category/ajax/. last accessed: April 16, 2024

The map is a leaflet[14] visualization that shows Open Street Map data. It 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.

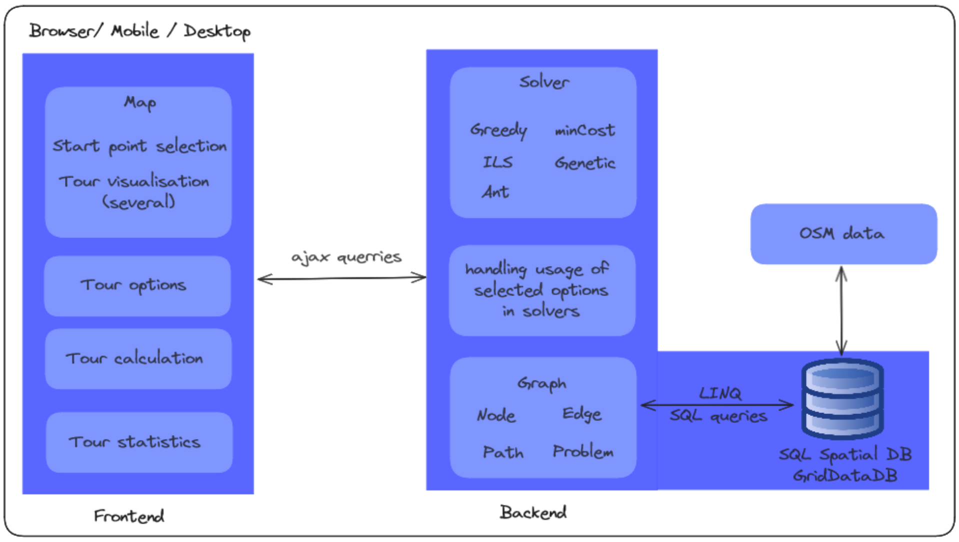


Figure 3.1: Visualization of the used architecture

In the above visualization, the whole application, its 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, it is possible to set the marker to the current location - if the permission to access the data is granted. However it is also possible to simply search for a specific address, to drag and drop the marker on the map or to scroll the map and select a position by clicking on it. Furthermore, the visualization of the calculated tours is also realized using the map and a polygon built from the respective points.

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.

[14]https://leafletjs.com/, last accessed: April 16, 2024

3.1.2 Database

3.1.3 Interface and Front end changes

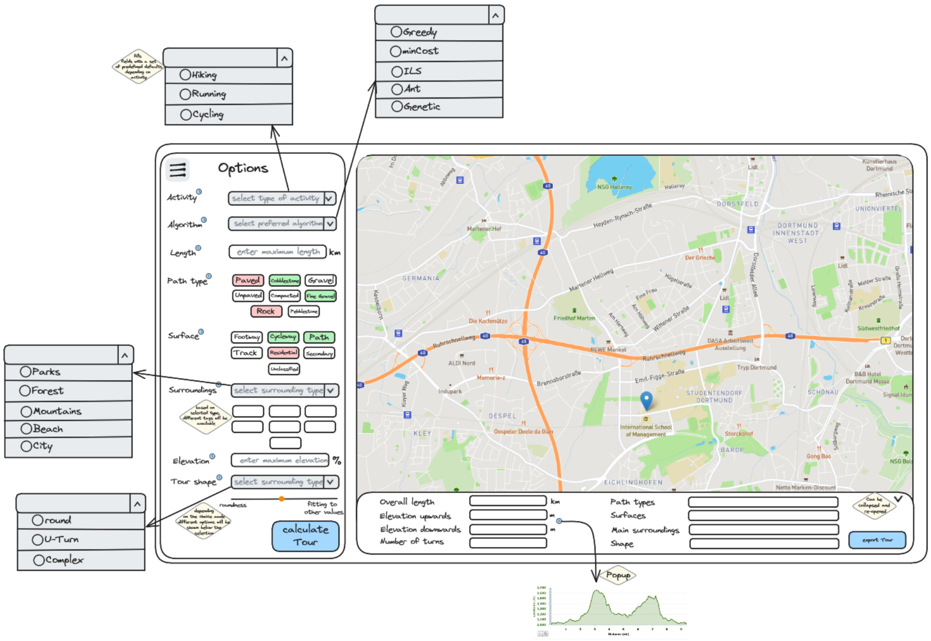


Figure 3.2: Design concept for the front end view, including descriptions for drop-downs and pop-ups

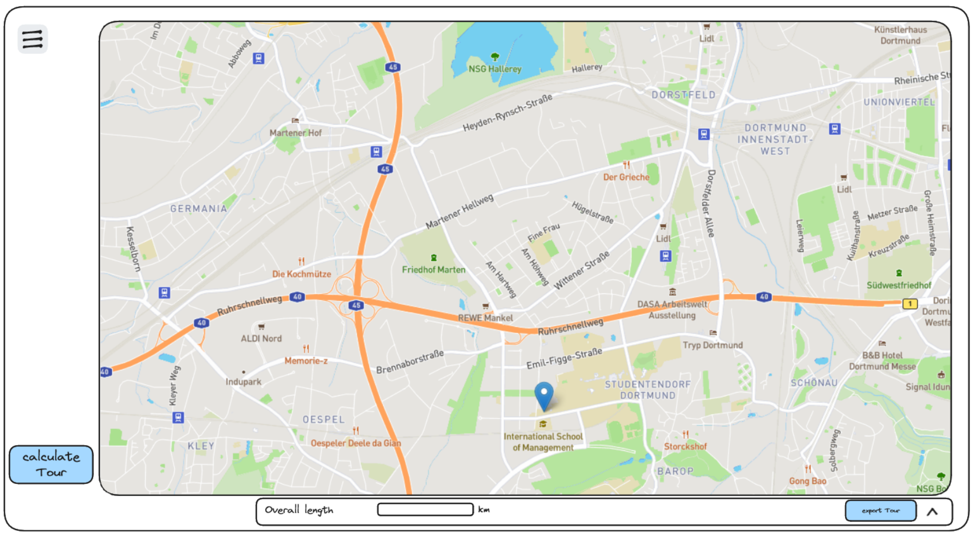
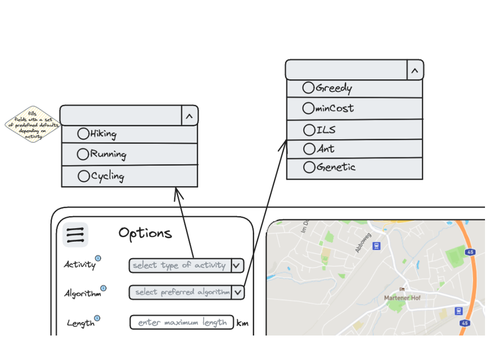
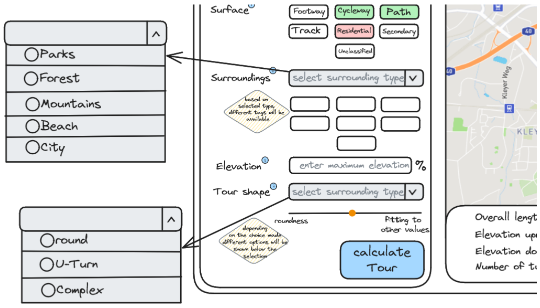


Figure 3.3: Design concept for the front end view with all menus folded



(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

Bibliography

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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/INFCOM.2012.6195470.

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

[13] Daniel Delling, Thomas Pajor, and Renato F. Werneck. “Round-Based Public Transit Routing”. In: *Transportation Science* 49.3 (2015), pp. 591–604.

[14] 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.

[15] 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.

[16] 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.

[17] M. Dorigo, V. Maniezzo, and A. Colorni. “Ant system: optimization by a colony of cooperating agents”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 26.1 (1996), pp. 29–41. DOI: 10.1109/3477.484436.

[18] 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.

[19] 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.

[20] Andreas Gemsa 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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).

[25] 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.

[26] 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.

[27] Amgad Madkour et al. *A Survey of Shortest-Path Algorithms*. 2017. arXiv: 1705.02044[cs].

[28] 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.

[29] Matthew A. Nystoriak and Aruni Bhatnagar. "Cardiovascular Effects and Benefits of Exercise". In: *Frontiers in Cardiovascular Medicine* 5 (2018), p. 135.

[30] 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.

[31] 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.

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

[33] Gerhard Reinelt. *The Traveling Salesman: Computational Solutions for TSP Applications*. Berlin, Heidelberg: Springer, 2003. 231 pp.

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

[35] 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.

[36] 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.

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

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

List of Figures

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 kenntlich gemacht habe.

Dortmund, den April 16, 2024

Lisa Salewsky