Generating Constrained Length Personalized Bicycle Tours

P. Stroobant[a], P. Audenaert[a], D. Colle[a], M. Pickavet[a]

[a] *Ghent University - imec, IDLab, iGent Tower - Department of Information Technology, Technologiepark-Zwijnaarde 15, B-9052 Ghent, Belgium*

Abstract

In the context of recreational routing, the problem of finding a route which starts and ends in the same location (while achieving a length between specified upper and lower boundaries) is a common task, especially for tourists or cyclists who want to exercise. The topic of finding a tour between a specified starting and ending location while minimizing one or multiple criteria is well covered in literature. In contrast to this, the route planning task in which a pleasant tour with length between a maximum *and* a minimum boundary needs to be found is relatively underexplored. In this paper, we provide a formal definition of this problem, taking into account the existing literature on which route attributes influence cyclists in their route choice. We show that the resulting problem is NP-hard and devise a branch-and-bound algorithm that is able to provide a bound on the quality of the best solution in pseudo-polynomial time. Furthermore, we also create an efficient heuristic to tackle the problem and we compare the quality of the solutions that are generated by the heuristic with the bounds provided by the branch-and-bound algorithm. Also, we thoroughly discuss the complexity and running time of the heuristic.

Keywords: Constrained bicycle routing, branch-and-bound algorithm, reach-based routing

1. Introduction

Cycling is a popular means of spending free time. According to Pucher et al. (2011), 49% of all bicycle trips in the US between 2001 and 2009 are for exercise, recreation and sports. In this context, cyclists are often looking for trips that start and end at the same location, satisfy some length constraints and are as pleasant as possible. Furthermore, they typically want to avoid taking the same road or passing through the same area multiple times.

Email addresses: pieter.stroobant@ugent.be (P. Stroobant), pieter.audenaert@ugent.be (P. Audenaert), didier.colle@ugent.be (D. Colle), mario.pickavet@ugent.be (M. Pickavet)

Online Tools. There are two categories of online tools that are currently available to help cyclists in creating such tours. The first type of route planner is limited to finding paths between a specified starting and ending point. Route planners like BikeRouteToaster.com, plotaroute.com or cycle.travel allow cyclists to generate a route by adding and editing route points: users can draw a route by adding route markers on the map, which are connected by a shortest path algorithm in which the lengths of roads are modified to prefer more attractive roads. In the more advanced route planners, users are able to influence how the new edge weights are calculated by specifying the tradeoff between certain road properties. OpenTripPlanner (OTP) for example allows cyclists to specify a trade-off between quick, flat and attractive routes, thus creating a 'bicycle preference triangle'. Creating closed tours with these route planners requires that a user manually searches for some intermediate nodes to form a tour that both satisfies constraints on length and avoids passing through the same areas. This requires a lot of effort by the user and can be cumbersome.

The second type of tools contains route planners such as RouteLoops and RouteYou, that add the needed functionality to generate a closed route by allowing a user to specify a start point and a desired tour length. This type of route planners usually operates by generating a set of via-nodes, after which these points are connected by using a shortest-path algorithm. Another option, taken by RouteLoops, is to generate a random walk through the graph. At each vertex of the walk, the probability of choosing an edge as the next one in the walk depends on the direction in which the edge is heading, the length of the part of the walk that is already generated and the remaining length of the tour. Unfortunately, these route planners limit users to a small set of cycling profiles to express their preferences. Furthermore, tours generated by these planners may contain multiple occurrences of the same road, especially near the via-nodes.

Criteria. Hochmair (2004) suggests that preferences of individual cyclists are highly different and cycling profiles do not suffice to capture the variety of preferences of different cyclists. The algorithms that we develop allow each user to define a function $c_l : E \to \mathbb{R}^+$, which maps each road - represented by an edge $e \in E$ - to a local cost. As in the first category of online tools, this cost should be based on a rescaled road length, where shorter roads are more attractive.

Our implementation calculates c_l using a weighted combination of different criteria, similarly to the formulation of the multi-criteria shortest path problem as given by Hrncir et al. (2014, 2015). Thus, there is a function $\vec{c}_a : E \to (\mathbb{R}^+)^k$, which maps each road to its edge cost with respect to k attributes. Users assign importance to each of the edge attributes by specifying a k dimensional vector of positive numbers $\vec{w} = (w_1, w_2, ..., w_k)$. We assume that this weight vector is normalized, that is: $\sum_{i=0}^{k} w_i = 1$. This allows to define the local cost of an edge e as:

$$c_l(e) = \vec{c}_a(e) \cdot \vec{w}$$

Thus, although our implementation only considers three criteria, the algorithms that are presented work for an arbitrary number of road-dependent

criteria.

The first criterion of our implementation specifies the importance of scenery. Lengths of roads that pass through forests and near waterbodies are scaled down, since they are considered to be more scenic (this is in accordance with Alivand et al. (2015)). In addition, the length of a road is decremented when hotspots for sightseeing or places to drink are situated in its neighbourhood (the impact of these hotspots is discussed by Alivand and Hochmair (2015)).

A second criterion that users can express is road safety (proposed by Hochmair (2004)). The lengths of roads are rescaled to reflect the type of bicycle lane, whether it is separated from car traffic and the maximal allowed speed of nearby vehicles.

Lastly, our implementation allows users to express the importance of road fastness (also suggested by Hochmair (2004)), which penalizes intersections with busy roads and traffic lights.

Similar to OTP, this allows to create an interface in which users can specify $\vec{w}$ by setting a marker inside a triangle with these criteria at the corners, as visualised in Fig. 1.

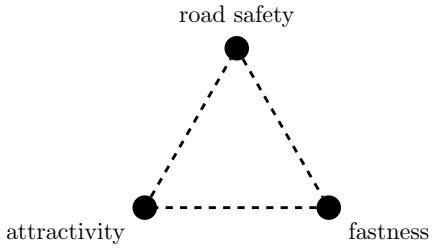


Figure 1: Preference triangle as an interface for a route planning tool

In addition to these road criteria, our algorithms also allow users to specify the importance of the route avoiding to visit the same regions multiple times and it allows to specify the extent of what is considered a 'visited region'.

In this paper, a routing model is formulated which takes these user specified preferences into account and looks for the tour which satisfies them the best, while taking length constraints into account. Before defining this problem, we provide an overview on some related work in Section 2. After this, we formalize the problem in Section 3. In Section 4, a description of a branch-and-bound algorithm to find the best tour is provided. Section 5 discusses a heuristic to solve this problem and several optimizations to speed up the heuristic. Subsequently, the performance of the heuristic is discussed in Section 6. Finally, there is a conclusion (Section 7).

2. Related work

While there is plenty of literature on calculating shortest paths in both a static (Bast et al. (2015)) and dynamic (Demeyer et al. (2014)) context, previ-

ous work on finding tours with a given length and starting node is very limited. Gemsa et al. (2013) studies the problem of calculating closed jogging routes from a given starting position. They also scale the weight of roads to avoid unattractive routes. Since their problem turns out to be NP-hard, they develop several heuristics. The greedy faces algorithm creates tours by joining adjacent faces of a planarized version of the street network. They also introduce several other heuristics, which look for appropriate via-nodes to create a tour. These algorithms differ from the work presented in this paper in that they foremost focus on finding routes with a length as close as possible to the specified input length, while the heuristics presented in this work assume that the length boundaries allow sufficient variation to foremost focus on finding pleasant tours, while achieving the length constraints as a side-effect of the way in which the tours are constructed. Also, the lengths of the cycling tours considered in this paper (up to 150 km) are far larger than the lengths of the jogging routes (up to 10 km). Furthermore, their work measures the amount of shared edges, whilst the metric presented in this work is more general in the sense that is allows to avoid coming close to earlier visited edges.

Literature that is concerned with bicycle routing typically focuses on finding a path between a specified starting and ending node, while minimizing one or more criteria. There are several works that describe an implementation of a web based bicycle planner that finds the shortest path between two nodes based on a user-specified tradeoff between road properties, for example Fu and Hochmair (2009); Hrncir et al. (2014); Su et al. (2010); Turverey et al. (2010). Other work approaches the multi-criteria aspect of bicycle routing by searching for a set of Pareto optima routes Hrncir et al. (2015); Song et al. (2014). Storandt (2012) focuses on the problem of finding Pareto optimal routes with an upper limit on one of the routing criteria. Despite advances in finding Pareto-optimal paths by employing advanced techniques like contraction hierarchies (Geisberger et al. (2008)), the runtime required to find these paths remains too high for routes of the size considered in this paper. Therefore, the heuristic route planner discussed in this work uses a weighted combination of criteria when searching for paths between two nodes.

The heuristic routeplanner presented in this paper roughly operates by selecting a 'forward path' that starts from the starting node of the tour and subsequently calculates (in an efficient way) an alternative route from each of the nodes on this forward path back to the starting node. This means that literature concerning alternative paths is also relevant to our work. An overview of such methods is presented in Dees et al. (2010). However, many of the presented alternate path methods require that both the starting and the ending node between which an alternate path has to be calculated are known. Applying these methods would require calculating an alternate path between the starting node and each of the nodes on the forward path and thus be very time-consuming. An idea which can be extended for this use case is the idea of penalization, which is also presented in Paraskevopoulos and Zaroliagis (2013); Schieferdecker et al. (2013). When applying penalization to find alternate paths, the weight of all edges of the path for which an alternative has to be found is scaled by a factor

$1+\alpha$ where α is a small positive number. After this, the shortest path is calculated using these new weights and the segment of this new shortest path which overlaps with the old path is scaled once again. By reiterating this procedure and combining all the resulting shortest paths into a subgraph, an 'alternative path graph' is found. In this work, the method to avoid the 'forward path' when searching for a path from each node in the forward path back to the starting node also uses penalization, although in a slightly different way.

The quadratic shortest path problem (QSP), as presented in Rostami et al. (2016), searches for a minimum cost path between two given vertices node whilst not only taking the edge cost into account, but also a quadratic cost, which depends on the edge pairs that are present in the solution. This is related to our work in the sense that assigning a non-zero quadratic costs to physically nearby edges can penalize solutions for staying in the same neighbourhood. A key difference with this work will turn out to be that in our approach, the cost assigned to visiting two nearby edges depends not only on the edges themselves, but also of the distance traveled between those edges, i.e. visiting two nearby edges is more problematic as the travelled distance between those edges increases. Nonetheless, there are some similarities is the exact solution approaches, in particular between the lower bound for the branch-and-bound algorithm presented in our work and the Gilmore-Lawler bound, that can be adapted to the QSP and is prevalent in the quadratic assignment problem in general, see Loiola et al. (2007).

3. Problem

Before defining the actual problem of finding cycling tours, some useful terminology needs to be introduced. Subsection 3.1 starts by defining some general graph terminology and definitions that will prove useful throughout this work. Subsection 3.2 focuses on the issue of quantifying the shape of a tour and introduces a measure that allows to evaluate the general 'roundness' of a tour. This roundness is a key ingredient of the Cycling Problem, which is formally defined in subsection 3.3.

3.1. General Terminology

A cycleway graph is a directed embedded multigraph defined as $G = (V, E, \vec{c}, l)$. Here, V is the set of vertices and $E \subseteq V \times V$ is the set of arcs representing road segments.

Furthermore, $\vec{c} : V \to \mathbb{R}^2$ maps each vertex to its coordinates. Related to these coordinates, we define a metric $d : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}^+$ that allows to calculate the displacement (that is, the great-circle distance or the distance as the crow flies) between a pairs of coordinates. This allows to define $d_v : V^2 \to \mathbb{R}^+$, the displacement between any couple of vertices as follows: $d_v(v_1, v_2) = d(\vec{c}(v_1), \vec{c}(v_2))$. Similarly, $d_e : E^2 \to \mathbb{R}^+$, the displacement between any couple of edge(center)s is defined as:

$$d_e(\langle u_1, v_1\rangle, \langle u_2, v_2\rangle) = d\left(\frac{\vec{c}(u_1) + \vec{c}(v_1)}{2}, \frac{\vec{c}(u_2) + \vec{c}(v_2)}{2}\right)$$

Finally, $l : E \to \mathbb{R}^+$ is a function that maps each edge to its corresponding length. For all $e = \langle u, v\rangle \in E$, we require that $l(e) \geq d_v(u, v)$.

A walk is a sequence of edges through the graph in which for each pair (e, e') of subsequent edges the ending vertex of e equals the starting vertex of e'. Whenever this work applies a function $f : E \to Y$, that takes an edge as an input, on a walk π, this refers to the sum of f applied on each of the edges of the walk, e.g. the length of a path is: $l(\pi) = \sum_{\forall e \in \pi} l(e)$.

A closed walk is a walk that starts and ends with the same vertex. The terms closed walk and cycle are used interchangeably throughout this work and are not to be confused with a *simple* cycle, in which vertices cannot be revisited.

3.2. Tour roundness

As previously stated in section 1, users can specify the importance of the tour avoiding to revisit the same regions multiple times and they can specify what is considered a previously 'visited region'.

Evaluating these requirements using 'hard constraints' such as forbidding a tour to revisit the (neighbourhood of the) same street multiple times runs into problems due to cul-de-sac neighbourhoods (with only one inlet/outlet) and, more generally, regions that are only accessible through a single access point within the length constraints of the tour (e.g. a bridge across a river). Another issue with this approach is that it classifies roads that are very close each other as either 'completely unacceptable' or 'perfectly fine', which is counterintuitive.

Therefore, we define a measure that quantifies how well any solution cycle satisfies the requirements considering revisiting regions. We assign to each tour a penalty from 0 to 1, where zero corresponds to a tour that perfectly satisfies its shape requirements and one corresponds to a tour that keeps revisiting the same spot over and over again.

This metric should satisfy several properties:

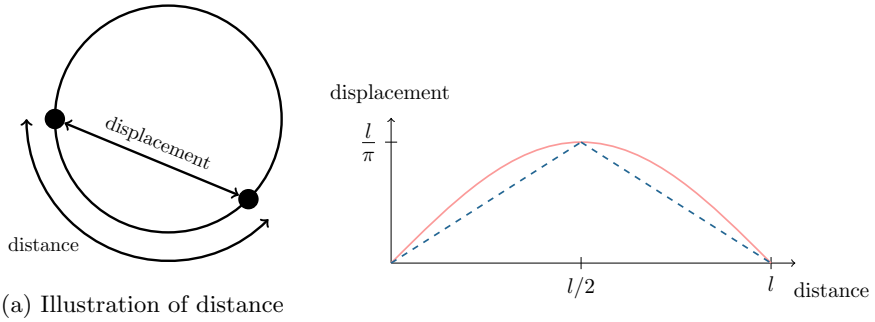
1. The metric should be able to detect whether a tour visits a similar region multiple times

2. The 'size' of what is considered to be a similar region should be adaptable to express the preferences of both cyclists that prefer round tours and bicyclists that just want to avoid reusing the same edges.

3. The boundaries of what is considered to be the neighbourhood of the tour should be 'soft': edges that are further from a previously visited road should be penalized less.

For an ideal tour (a tour that is a perfect circle), the displacement between two points is sinusoidal in the distance between those points (the distance is the length of the path traveled along the circle). This is shown in Figure 2, by the full pink curve. The roundness metric uses the distance traveled along the tour

between the centers of the two edges to calculate the expected displacement $d_{exp} : E^2 \to \mathbb{R}^+$ between these centers. This expected displacement is compared to the (actual) displacement d_e between the edge centers. In order to use the sinusoidal relation to exactly calculate the expected displacement between two points, the total tour length has to be known. Therefore, the expected displacement is calculated using an approximation of this sine by a piecewise linear function, as shown in Fig. 2 by the dashed blue curve. Notice that the slope of the piecewise linear approximation is independent of the tour length. Furthermore, for a tour of length l, we have $d_{exp}(x) = d_{exp}(l - x)$. We conclude that the expected displacement between two edges depends exclusively on the minimal distance between those edges.



(a) Illustration of distance and displacement

(b) Displacement as a function of traveled distance

Figure 2: Distance & displacement in a circular tour of length l

Using the comparison of the expected displacement and the actual displacement between two edges, a penalty $p : E^2 \to \mathbb{R}^+$ is assigned if a pair of edges is too close to each other. Since preferences about the required roundness of a tour may vary between cyclists, cyclists are allowed to specify a strictness factor $\sigma \in [0, 1]$, that is multiplied with the expected distance. Users that prefer very round routes should choose a high strictness factor, users who just want to avoid the close neighbourhood of the roads they already visited should prefer a strictness factor close to zero. Taking this into account, the penalty is defined as follows:

$$p(e_i, e_j) = \begin{cases} \dfrac{\sigma d_{exp}(e_i, e_j) - d_e(e_i, e_j)}{\sigma d_{exp}(e_i, e_j)} & \text{if } d_e(e_i, e_j) < \sigma d_{exp}(e_i, e_j) \\ 0 & \text{otherwise} \end{cases}$$

It is worth noting that the use of d_e, the distance as the crow flies, rather than the shortest path distance between two edge centers, results in increased penalties between edges that are close to one another in terms of displacement, but at a large distance when travelling through the graph. The reasoning behind this choice is that locations with a small displacement tend to be similar, and thus revisiting them is more repetitive (e.g. there will be a high penalty between edges that are on opposite sides of a river). While this assumption is typically

reasonable, there are some scenarios in which nearby roads offer distinct views, but still require to travel a large distance through the graph. Roads on a mountainside at different elevation levels are one example, but this also can occur between streets in cul-de-sac neighbourhoods, where there can be a large travelling distance between roads that are physically close to each other.

The penalty of a tour $\pi = (e_1, e_2, ..., e_N)$ is now defined as the average penalty between two points on π that are randomly chosen, which is approximated by:

$$p_{avg}(\pi) = \frac{\sum_{i=1}^{N} \sum_{j=1}^{N} p(e_i, e_j)\ l(e_i)\ l(e_j)}{l(\pi)^2}$$

Due to the fact that the formula for calculating penalties represents each edge by a single point, the value of $p_{avg}(\pi)$ gets increasingly more accurate as the length of the tour and the number of edges increases (assuming edges that are quite regular). We discuss the accuracy of this approximation in subsection 6.3.

3.3. Cycling Problem

The problem requires a starting (and ending) point s, a lower and an upper bound on the allowed cycle length l_{min} and l_{max} and a local cost function $c_l : E \to \mathbb{R}^+$, assigning an unpleasantness to each edge of the graph. Defining this as an input parameter rather than as a part of the cycling graph allows each users to customise this function, allowing it to better reflect their personal perception of unpleasantness. The average local cost of a path π is defined as: $c_{l,avg}(\pi) = c_l(\pi)/l(\pi)$.

Simply looking for the cycle that satisfies the length boundaries while minimizing the average local cost, will typically result in a route that travels to a cycle with a (locally optimal) minimal cost-to-length ratio, takes a couple of tours along this cycle and then returns to the starting point. In fact, this property is well known and can be used to identify the minimum weight cycle in a graph, as discussed by Karp (1978).

Thus, to keep the algorithm from creating tours that visit similar regions multiple times, the average tour penalty must be incorporated into to the minimization problem. For smaller σ values (that is: $\sigma < 0.5$) and cycling tours of 10 or more kilometers, there are typically many tours with a zero (or almost zero) average penalty. In these cases, it makes sense to trade off the average penalty and the average local path cost (e.g. a higher penalty tour may be preferable over a low penalty tour with very unattractive roads). Thus, rather than adding a constraint that specifies an upper bound on $p_{avg}(\pi)$, the average tour penalty is multiplied with a positive parameter λ and added to the minimization term. The parameters λ allows users to trade off the average penalty and the average local path cost: for $\lambda = 0$, the cyclist does not cares about visiting the same region or even the same road multiple times, higher values of λ will result in tours that avoid revisiting regions that have been visited before.

Finally, the Cycling Problem can be formulated: Given a cycleway graph G, a starting node $n \in V(G)$, a local cost function $c_l : E \to \mathbb{R}^+$, a factor $0 < \lambda$

that specifies the weight of the average penalty and a strictness $0 < \sigma < 1$, we search for the cycle $\pi = (e_1 = (s, v_1), e_2, ..., e_N = (v_{N-1}, s))$ with a minimal average total cost $c_{t,avg}(\pi)$:

$$c_{t,avg}(\pi) = c_{l,avg}(\pi) + \lambda \cdot p_{avg}(\pi) \tag{1}$$

under the constraint that $l_{min} \leq l(\pi) \leq l_{max}$.

In other words, the Cycling Problem asks for the cycle for which the sum of the local average cost and λ times the average penalty is minimal.

4. Exact Solution

In this section, the Cycling Problem is shown to be strongly NP-hard by reduction from the Planar Hamiltonian Cycle Problem (Planar HCP). In addition, we prove that Cycling Problem cannot be approximated in polynomial time unless P = NP. While this implies there is no pseudo-polynomial time algorithm to solve Cycling Problem, it is possible to devise a pseudo-polynomial time algorithm to check whether the length constraints allow a solution. We extend this algorithm such that it provides a lower bound on the minimum score that can be achieved and use it as a building block to create a branch-and-bound algorithm to find an exact solution to this problem.

4.1. NP-hardness

Assume that an instance of Planar HCP is provided: a directed graph $G = (V, E)$ with $V = \{v_1, v_2, ..., v_n\}$ and a planar embedding $\vec{c}_p : V \to \mathbb{R}^2$. The Planar HCP requires to determine whether G contains a cycle that visits each of the vertices exactly once.

A corresponding cycleway graph $G' = (V', E', \vec{c}, l)$ is constructed as follows:

$$V' = \{v_{1,in}, v_{2,in}, ..., v_{n,in}\} \cup \{v_{1,out}, v_{2,out}, ..., v_{n,out}\}$$
$$E' = \{(v_{i,out}, v_{j,in}) \mid (v_i, v_j) \in E\} \cup \{(v_{i,in}, v_{i,out}) \mid i \in \{1, ..., n\}\}$$
$$\vec{c}(v) = \vec{c}_p(v_i) \text{ if } v = v_{i,in} \text{ or } v = v_{i,out}$$

In other words, every vertex from V is replaced by an in- and an out-vertex and each arc from E is adapted to start and end in the appropriate out-vertex and in-vertex of E' respectively. The coordinates of the in- and out-vertices are identical to the coordinates of the original vertices. As the required metric d, we choose the Euclidean distance. Lastly, we choose $l(e) = L, \forall e \in E'$, with $L = \max(\{d_v(u, v) \mid (u, v) \in E\})$. This ensures that for every arc $e = (u, v)$ the following property holds: $d_v(u, v) \leq l(e)$.

Solving Planar HCP in G corresponds to solving Cycling Problem in G' with some appropriate choices of parameters. Setting $l_{min} = l_{max} = 2|V|\, L$ ensures that the resulting solution will correspond to a closed walk with $|V|$ nodes in the original graph.

The coordinates of the edge centers of each couple of non anti-parallel edges in E' correspond to the coordinates of different vertices and/or edgecenters of

G. Thus, since G is planar, the minimum Euclidean distance D between each two edges in E' that are not anti-parallel must be strictly positive. By choosing $(2D)/(\pi|V|L) > \sigma > 0$, one can easily show that for $|V|L$, the largest possible distance between two edges, $0 < d_{exp} < D$. Thus, for this choice of σ, there is only a penalty if a cycle revisits an edge, which results in a penalty of one (since in this case the displacement equals zero). This implies that all Hamiltonian cycles will have an average cycle penalty of zero.

By choosing $\lambda = (2|V|\ L)^2$, revisiting even one edge results in an average cycle penalty that is at least equal to one. Since each vertex is either an in-vertex (with only one outbound edge) or an out-vertex (with only one inbound edge), revisiting a node implies that an edge is revisited as well. Therefore, all non-Hamiltonian cycles will result in an average cycle penalty of at least one.

By defining $c_l(e) = 0, \forall e \in E$, the minimum average total cost of the Cycling Problem will be zero in a Hamiltonian graph and at least one in a non-Hamiltonian graph. Thus, it can be concluded that Cycling Problem is NP-hard. Additionally, since any solution has an objective value of zero or at least one, Cycling Problem cannot be approximated unless $P = NP$.

4.2. Solvability and lower bound

Since Cycling Problem requires that the length of a solution cycle is between l_{min} and l_{max}, another question arises: when is it actually possible to find such a cycle? It turns out that this problem is NP-hard too. This can easily be seen by reduction from the Unbounded Subset Sum Problem, which is discussed by Kellerer et al. (2004). This problem provides a set of weights $\{w_1, w_2, ..., w_N\}$, a target weight W and asks whether it is possible to find $x_i \in \mathbb{N} \cup \{0\}$ such that $\sum_{i=0}^{N} x_i w_i = W$. This problem can be transformed in a graph representation by drawing a single vertex v with each of the weights w_i being represented by a loop edge with length w_i. If this graph contains a cycle starting from v with length $l_{min} = l_{max} = W$, then this cycle corresponds to a solution of the Unbounded Subset Sum Problem.

When restricting the edge lengths to integer values, the solvability problem turns to be weakly NP-hard, since it is solvable in pseudo-polynomial time using the algorithm used to solve the Simple Jogging Problem that is introduced by Gemsa et al. (2013). Here, we present an extension of this algorithm that has the additional benefit of also providing a lower bound on the score that can be achieved in the Cycling Problem.

The problem that we focus on is the following: given a graph $G = (V, E)$, a length function $l : E \to \mathbb{N}$, a weight function $w : E \to \mathbb{R}^+$, a source vertex s, a target vertex t, a starting weight w_s, a starting length $l_s \in \mathbb{N} \cup \{0\}$ and length bounds l_{min} and l_{max}, what is the minimum average weigth associated with a walk $\pi = (e_1 = (s, v_1), e_2, ..., e_N = (v_{N-1}, t))$ for which $l_{min} \leq l_s + l(\pi) \leq l_{max}$. In this problem, we define this minimum average weight as:

$$\min_{\pi} \left(\frac{w_s + w(\pi)}{l_s + l(\pi)} \right)$$

This problem can be solved by maintaining, for each vertex v and for each length l', the weight corresponding to a minimum weight walk of length l' from s to v. Thus, the algorithm maintains a matrix $Q : V \times (\mathbb{N} \cup \{0\}) \to (\mathbb{R} \cup \{+\infty\})$. Initially all values of Q are set to $+\infty$, except for $Q(s, l_s)$ which is set to w_s. The algorithm operates by iterating over all edges, for lenghts l' incrementing from l_s to l_{max}. For each edge $e = (u, v)$, the matrix is updated according to the rule:

$$Q(v, l' + l(e)) = \min(Q(v, l' + l(e)), Q(u, l') + w(e))$$

When these minimum weights are calculated, the minimum average weight can be found by iterating over $Q(t)$ for length values increasing from l_{min} to l_{max} and calculating the average weight, since:

$$\begin{aligned}\min_{\substack{\pi=(s,\ldots,t) \\ l_{min} \leq l_s + l(\pi) \leq l_{max}}} \left(\frac{w_s + w(\pi)}{l_s + l(\pi)} \right) &= \min_{l_{min} \leq l_s + l' \leq l_{max}} \left(\min_{\substack{\pi=(s,\ldots,t) \\ l(\pi)=l'}} \left(\frac{w_s + w(\pi)}{l_s + l'} \right) \right) \\ &= \min_{l_{min} \leq l_s + l' \leq l_{max}} \left(\frac{Q(t, l_s + l')}{l_s + l'} \right)\end{aligned}$$

The worst case running time of this algorithm is $O(l_{max}|E|)$ and the required amount of space is $O(l_{max}|V|)$. Since the underlying graph corresponds to a road network, which typically has a spatial distribution of nodes that is approximately uniform and an average amount of edges per node that is constant, a more realistic estimate can be found by only considering edges adjacent to nodes within a radius of l_{max} from s. This leads to a running time of $O((l_{max})^3)$ and space requirements of the same order.

4.3. Branch-and-bound algorithm

Using the minimum average weight algorithm from the previous section, a branch-and-bound algorithm can be built. This algorithm maintains a priority queue that orders partially completed paths by a lower bound on the best score that can be achieved by extending the respective paths. As long as no solution is found or as long as there are paths in the queue which may result in a solution with a better score, the algorithm keeps running. In these cases, the 'most promising path' (the one with the lowest lower bound on the score) is extracted from the priority queue and for all out edges departing from the end of this path, an extended path is created and bounded. All these extensions are added to the priority queue. Stated otherwise, the algorithm is of the 'branch from lowest bound' type as described by Lawler and Wood (1966).

There are aspects of this technique that require some additional explanation. It is essential to take the average penalty of a cycle into account to get a tight lower bound on the scores that can be achieved by extending a path. To take this into account, the fact that any walk π that solves Cycling Problem has the property $l(\pi) \leq l_{max}$ can be used, which gives rise to the following inequality:

$$s(\pi) \geq \frac{w(\pi) + \frac{\lambda}{l_{max}} \sum_{i=1}^{N} \sum_{j=1}^{N} p(e_i, e_j)\, l(e_i)\, l(e_j)}{l(\pi)}$$
$$= \frac{w(\pi) + \frac{2\lambda}{l_{max}} \sum_{i=1}^{N} \sum_{j=1}^{i-1} p(e_i, e_j)\, l(e_i)\, l(e_j)}{l(\pi)}$$

The second line follows from the properties $p(e_i, e_i) = 0$ and $p(e_i, e_j) = p(e_j, e_i)$. After choosing the first M edges of a path for which a lower bound on the score has to be found, every solution π consists of a subpath $\pi_{s\to t}$ (starting from s and ending in t), containing those first M edges, and a subpath $\pi_{t\to s}$ (starting from t and ending in s), which contains all subsequent edges. This allows to rewrite the bound on the score as:

$$s(\pi) \geq \frac{w(\pi_{s\to t}) + w(\pi_{t\to s}) + \frac{2\lambda}{l_{max}} \sum_{i=1}^{N} \sum_{j=1}^{i-1} p(e_i, e_j)\, l(e_i)\, l(e_j)}{l(\pi_{s\to t}) + l(\pi_{t\to s})}$$
$$\geq \frac{w(\pi_{s\to t}) + \lambda\, p_{avg}(\pi_{s\to t}) + \sum_{i=M+1}^{N} \left(w(e_i) + \frac{2\lambda}{l_{max}} \sum_{j=1}^{M} p(e_i, e_j)\, l(e_i)\, l(e_j) \right)}{l(\pi_{s\to t}) + l(\pi_{t\to s})}$$

where $p(\pi_{s\to t}) = \frac{2}{l_{max}} \sum_{i=1}^{M} \sum_{j=1}^{i-1} p(e_i, e_j) l(e_i) l(e_j)$ is the average penalty associated with the subpath $\pi_{s\to t}$. The algorithm from the previous section can be used to find this last bound by choosing $w_s = w(\pi_{s\to t}) + \lambda p(\pi_{s\to t})$, $l_s = l(\pi_{s\to t})$ and by modifying the weight of the graph edges to account for the average penalty between those edges and $\pi_{s\to t}$.

To find a lower bound on the average penalty associated with $\pi_{s\to t}$, a lower bound on each of the $p(e_i, e_j)$ (with $i > j$) terms in the summation will be deduced. These penalties increase as the expected distance between e_i and e_j increases. Since this expected distance is determined by the minimal distance $d_{min}(e_i, e_j)$ traveled along the cycle between e_i and e_j, it is sufficient to find a lower bound on this distance. The distance $d_{e_j\to e_i}$ to travel from e_j to e_i (corresponding to increasing edge indices) can be calculated directly, since these edges are part of $\pi_{s\to t}$, as illustrated in Figure 3. The distance covered by travelling along the tour from e_j to e_i by passing through the starting vertex s can be split up in three parts: moving from e_i to t, travelling over $\pi_{t\to s}$ and moving from s to e_j. The only unknown distance is the second part, the length of $\pi_{t\to s}$, which can be bounded by: $l(\pi_{t\to s}) \geq \max(l_{min} - l(\pi_{s\to t}), d_v(t, s))$. This results in the following bound on the minimal distance between e_i and e_j:

$$d_{min}(e_i, e_j) \geq \min(d_{e_j\to e_i},\ l(\pi_{s\to t}) - d_{e_j\to e_i} + \max(l_{min} - l(\pi_{s\to t}), d_v(t, s)))$$

Similarly, bounding the average penalty between an edge $e = (u, v) \in E(G)$ and $\pi_{s\to t}$ is achieved by finding a lower bound on the minimum distance $d_{min}(e, e_j)$ required to travel along the cycle between e and each edge e_j of $\pi_{s\to t}$. Like before, two alternative paths are possible: travelling from e_j to t (this distance will be noted as $d_{e_j\to t}$) and from t to the center of e or travelling

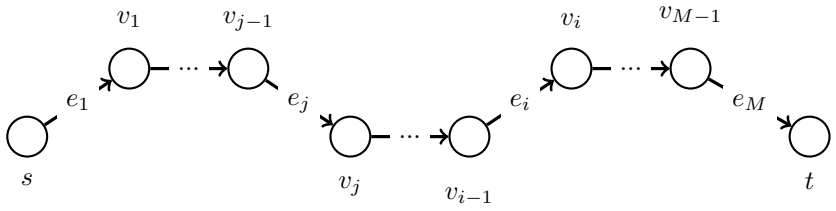


Figure 3: Partially completed path $\pi_{s\to t}$

from the center of e to s and from s to e_j (noted as $d_{s\to e_j}$). This results in the following bound on the minimum distance between e and e_j:

$$d_{min}(e, e_j) \geq \min(d_{e_j\to t} + d_v(t, u) + l(e)/2,\ l(e)/2 + d_v(v, s) + d_{s\to e_j})$$

Finally, we analyse the running time to calculate these bounds. The branch-and-bound algorithm discards paths $\pi_{s\to t}$ that are longer than l_{max}, so the amount of edges in $\pi_{s\to t}$ is $O(l_{max})$. Thus, a bound on the average penalty of $\pi_{s\to t}$ is calculated in worst-case time $O((l_{max})^2)$ and adapting the weight of a single edge is possible in $O(l_{max})$. As discussed in the previous paragraph, the amount of edges within length l_{max} of s in a typical road network is $O((l_{max})^2)$, meaning that adapting the weight of all relevant edges can be done in $O((l_{max})^3)$. Combining this result with the running time of the algorithm of the previous paragraph, we conclude that the running time of a single iteration of the branch-and-bound algorithm is $O((l_{max})^3)$.

5. Heuristic

Since Cycling Problem is NP-hard, a heuristic is developed. Whilst the heuristic described below provides no guarantee on finding a (good) solution, the results that are generated turn out to be close to optimal in practice. Furthermore, when the length bounds allow for some variation in the tour length, a solution that satisfies these bounds is almost always found. This section starts with a general overview of the heuristic. After this, some optimizations are discussed and the running time is analyzed.

5.1. General overview

The algorithm searches for pleasant routes in a way that resembles the route marker based bicycle planning tools that were discussed in Section 1. Basically, it looks for a 'well-placed' intermediary/turning point and generates a tour by searching a path to the intermediary node and an alternative path back to the starting point s. This method results in a heuristic that naturally consists of two major phases: in a first phase, that can be described as 'forward routing', a path from the starting point to an intermediary point is constructed. In a

second step, called 'backward routing', the algorithm looks for an alternative path starting from the intermediary node and ending at the starting point.

In a first, unoptimized version of the algorithm, the forward searching is performed by executing Dijkstra's algorithm to construct, starting from s, a *forward* shortest path tree f that contains a least-cost path $\pi_f(v)$ to each nearby node v. The local edge cost, c_l is used is for building this shortest path tree. These 'nearby nodes' are the nodes v for which $l(\pi_f(v)) + d_v(s, v) \leq l_{max}$. This avoids that the algorithm considers nodes that are too far away to get back to the starting node without exceeding the maximal allowed tour length. The main motivation for using a shortest path algorithm to find the forward path is that these paths, when they are sufficiently long, tend to be (more or less) straight, resulting in a low penalty between edges of the forward path. Furthermore, minimal-cost paths tend to have a low cost-to-length ratio.

In a next step, a node t is chosen that will be the *turning* point that is 'aimed for' during the backwards routing process. To find suitable turning points that are not too close to s, the algorithm collects from the shortest path tree a set C of *candidate* turning points, containing all vertices v for which $l(\pi_f(v)) + d_v(s, v) \geq l_{min}$. In addition to this, vertices that have a neighbour that is not in the shortest path tree f (since this neighbour is too far from s) are added to C. This ensures that C is non-empty and contains vertices that are not too close to s. From this set of candidate nodes, a turning vertex t can be chosen randomly, but in order to favor turning points with a low cost-to-length ratio, it is recommended to tune the selection probabilities of the candidate nodes. When executing the heuristic multiple times, each time a candidate node is chosen, the node probabilities are adapted to decrease the probability of candidate nodes that are in the neighbourhood of previously selected nodes. Thus, when a user generates multiple tours, the resulting tours are unlikely to head for similar directions.

To avoid penalties between the edges that are part of the return path, a shortest path algorithm is also employed to find the return path. Here, Dijkstra's algorithm is used to find a least-cost *return* path $\pi_r(v)$ that ends in s. Such a return path $\pi_r(v)$ is calculated for every vertex v on the forward path (thus, by constructing a sink tree). For each of these vertices, the length and the score of the cycle consisting of $\pi_f(v)$ followed by $\pi_r(v)$ is calculated and the best scoring cycle that also satisfies the length constraints is returned. Thus, rather than searching directly for a cycle that satisfies the length constraints, a cycle is found indirectly by considering every vertex of the forward path as a possible point to start returning to the starting vertex s. This principle is illustrated in Fig. 4.

To take the penalties between the forward path $\pi_f(t)$ and the return paths into account, the costs of the graph edges are adapted when searching for a return path. Dijkstra's algorithm requires the adapted weight of an edge $e = (x, y)$ only when the vertex y is added to the shortest path tree and thus $\pi_{y \to s}$, the least-cost path from y to s is known. Calculating the penalty between e and an edge $e' \in \pi_f$ requires estimating the minimum distance required to travel between e and e' over the final cycle. Since $\pi_{y \to s}$ is known, calculating the

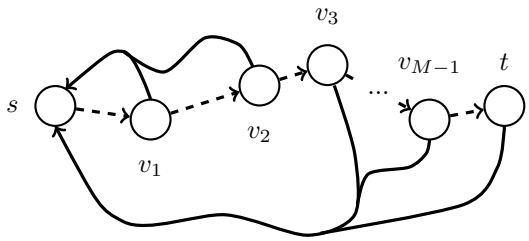


Figure 4: Operation of the heuristic
($\pi_f(t)$ is dashed, $\pi_r(v)$ to each vertex v of $\pi_f(t)$ in full lines)

distance $l_{e\to e'}$ to travel along the tour from e to e' over s is straightforward. Edges e' for which $l_{e\to e'} > l_{max}$ are not taken into account when calculating the penalty, since they cannot be part of a valid cycle that also contains e. To avoid underestimating penalties for cycles with a length that is close to l_{max}, the estimated penalties assume that the tour length equals l_{max}. This means that the estimated length to travel from e' to e (without passing through s) equals $l_{max} - l_{e\to e'}$. This results in the following estimated minimum distance $l_{min,est}(e, e')$ between e and e':

$$l_{min,est}(e, e') = \min(l_{e\to e'}, l_{max} - l_{e\to e'})$$

This corresponds to a new cost $c'(e)$ for e according to:

$$c'(e) = c(e) + \frac{2\lambda}{l_{max}} \sum_{\substack{e'\in\pi_f(t) \\ l_{e\to e'}\leq l_{max}}} p(e, e')\, l(e)\, l(e') \tag{2}$$

5.2. Optimizations

The most time-consuming part of the proposed algorithm is adapting the costs of the edges that are encountered during the backward routing. Indeed, assuming that the underlying graph corresponds to a typical road network, the amount of edges encountered during backward routing is $O((l_{max})^2)$. Since the length of $\pi_f(t)$ is $O(l_{max})$, the time required to apply the formula from the previous section to calculate the adapted costs of all edges encountered during backward routing is $O((l_{max})^3)$. Using a binary heap implementation of Dijkstra's algorithm, the time required for forward and for backward routing is $O((|V| + |E|)\log(|V|))$ according to Bast et al. (2015). In a road graph this equals $O((l_{max})^2 \log(l_{max}))$. This section proposes a method to approximate the adapted cost of the edges encountered during backward routing in the same running time as required by the routing steps. Furthermore, some optimizations for the forward and backward routing are proposed.

5.2.1. Forward path approximation

In order to reduce the running time required for calculating the costs of the edges that are encountered to $O((l_{max})^2 \log(l_{max}))$, an approximation of the

forward path is used to calculate the adapted cost. The approximation contains $O(\log(l_{max}))$ edges rather than $O(l_{max})$ edges.

The path approximation uses the observation that for an edge $e \in \pi_f(t)$ for which the minimal distance to any edge e' of the backward tree is large, a small error in $d_e(e, e')$ has a limited impact on the value of $p(e, e')$. Assuming that the turning point that results in the path with minimal cost is situated close to the end of $\pi_f(t)$, the minimal distance between e and any e' is approximated as the minimum of $l_{s\to e}$, the distance to travel along $\pi_f(t)$ from s to the center of e and $l_{e\to t}$, the distance to travel from the center of e to t. Under this assumption, the edges e_i up to e_j (with $1 \leq i \leq j \leq M$) of $\pi_f(t) = (e_1, e_2, ..., e_M)$ can be replaced by a single edge $e_{i\to j}$, whilst still allowing to calculate the average penalty with an error smaller than the penalty error threshold $0 < \epsilon < 1$.

Before describing how to create such an approximation, some additional notation is introduced. We define $d_{max}(e_{i\to j})$ as the maximum displacement between the center of $e_{i\to j}$ and the center of e_k, for $i \leq k \leq j$, as shown in Figure 5. Furthermore, $l_{e_i\to e_j}$ is the distance required to move from the center of e_i to the center of e_j along $\pi_f(t)$. The approximated path is created by moving along the original path $\pi_f(t)$ and joining edges $e_i = (v_{i-1}, v_i)$ to $e_j = (v_{j-1}, v_j)$ in $e_{i\to j} = (v_{i-1}, v_j)$ if:

$$\frac{2d_{max}(e_{i\to j}) + \sigma d_{exp}(l_{e_i\to e_j})}{\sigma\ d_{exp}(\min(l_{s\to e_j}, l_{e_i\to t}))} < \epsilon \tag{3}$$

The penalty term between $e_{i\to j}$ and e' is calculated as:

$$p(e, e_{i\to j}) = \max\left(0, 1 - \frac{d_e(e, e_{i\to j}) + d_{max}(e_{i\to j}) + \sigma d_{exp}(l_{e_i\to e_j})}{\sigma d_{exp}(\min(l_{s\to e_j}, l_{e_i\to t}))}\right)$$

In equation 2, the terms corresponding to e_i to e_j are replaced by:

$$\frac{2\lambda}{l_{max}}\, p(e, e_{i\to j})\, l(e)\, l(e_{i\to j})$$

In this equation, $l(e_{i\to j})$ is the sum of the lengths of all edges e_k $(i \leq k \leq j)$ that are replaced by $e_{i\to j}$. One can easily show that $p(e, e_{i\to j}) \leq p(e, e_k)$ using the properties that $d_e(e, e_{i\to j}) + d_{max} \geq d_e(e, e_k)$ and for some $0 \leq f \leq 1$:

$$\sigma d_{exp}(\min(l_{s\to e_j}, l_{e_i\to t})) = \sigma d_{exp}(\min(l_{s\to e_k}, l_{e_k\to t})) + f\sigma d_{exp}(l_{e_i\to e_j})$$

Furthermore, it can be shown that $p(e, e_k) - p(e, e_{i\to j})$ is smaller than the left hand side of inequality 3, which leads to the conclusion:

$$p(e, e_k) - \epsilon \leq p(e, e_{i\to j}) \leq p(e, e_k)$$

It remains to be shown that applying inequality 3 results to a number of edges that is logarithmic in l_{max}. This can be seen by noting that in the first half of $\pi_f(t)$ the expected length in the left hand side of the inequality is determined by $l_{s\to e_j}$, which is larger than the sum of the lengths of all approximate edges

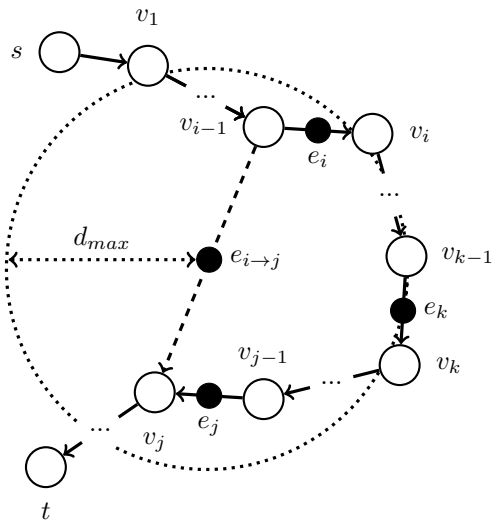


Figure 5: Replacing edges of the forward path with an approximation edge

that precede $e_{i\to j}$. Since $d_{max}(e_{i\to j}) \leq l(e_{i\to j})$ and $s\, d_{exp}(l_{e_i\to e_j}) \leq l(e_{i\to j})$, the length of $e_{i\to j}$ is bounded by a fraction of the sum of the lengths of the approximate edges that precede $e_{i\to j}$. This implies that the series of approximate edge lengths in the first half of the path grows exponentially, resulting in a logarithmic number of edges required to represent this half of the path. A similar reasoning holds true for the last part of the path.

5.2.2. Reach Based Forward Routing

A possibility to gain time in the forward routing step is by applying the concept of reaches, which was introduced by Gutman (2004). We define the reach $r(v, \pi_{s\to t})$ of a node v on a path $\pi_{s\to t}$, starting in a vertex s and ending in a vertex t, with $\pi_{s\to v}$ the subpath of $\pi_{s\to t}$ from s to v and $\pi_{v\to t}$ the subpath from v to t as $r(v, P) = \min(m(P_{s\to v}), m(P_{v\to t}))$. In this expression, $m : E \to \mathbb{R}$ is the *reach metric*. The reach of a node v in G is the maximum of the reach of $r(v, Q)$ over all least-cost paths Q in G, which are calculated over a given *cost metric*. *Reach* and *cost* metric can be the same, but this is not required.

The reach metric over which the reaches are calculated in this algorithm is l and the cost metric is c. The reaches can be efficiently computed using the algorithm that is described in Gutman (2004). In the forward routing step, a search is conducted for the most pleasant path to each of the candidate nodes. When adding a non-candidate node v with a least cost path $\pi_{s\to v}$, with $l(\pi_{s\to v}) + d_v(s, v) < l_{min}$, we have for each candidate node t with a least cost path passing through v and $\pi_{v\to t}$ being the subpath from v to t:

$$l(\pi_{s\to v}) + l(\pi_{v\to t}) + d_v(s, t) \geq l_{min}$$

This situation is visualized in Fig. 6.

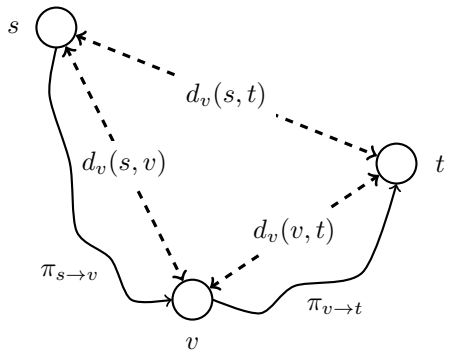


Figure 6: Forward search tree diagram

The triangle inequality implies that $d_v(s,v)+d_v(v,t) \geq d_v(s,t)$ and because of $l(\pi_{v\to t}) \geq d_v(v,t)$, it follows that:

$$l(\pi_{s\to v}) + 2l(\pi_{v\to t}) + d_v(s,v) \geq l_{min}$$

so

$$l(\pi_{v\to t}) \geq \frac{l_{min} - l(P_{s\to v}) - d_v(s,v)}{2} = l_{rem}(\pi_{s\to v})$$

Otherwise: when a tentative node v is added during forward search, the path will at least continue for an additional walking distance of at least $l_{rem}(\pi_{s\to v})$ before reaching a candidate node t. So if $r(v) < \min(l_{rem}(\pi_{s\to v}), l(\pi_{s\to v}))$, v is too far from the starting node and too far from an ending node to be part of a least cost path.

The effect of applying this condition is that half-way between starting node and candidate node, many node insertions are avoided, resulting in a shortest path tree that is dense at the edges and in the center, but sparse in-between, as visualized in Fig. 7.

A problem with this approach is that it requires that the cost function $c(e)$ of an edge e is known during preprocessing, which is not the case if the user has full freedom in choosing the parameters $\vec{w}$ (since $c(e) = \vec{w} \cdot \vec{c}(e)$). We solve this problem by calculating reaches for varying choices of $\vec{w}$ and by storing for each node the maximal reach. An alternative approach is as follows: since the calculated reaches do not have to be stored very accurately to gain speed, and since most nodes have a small reach, the storage requirements for reaches are very small. This makes it possible to easily calculate and store the reach for several combinations of parameter values. During routing, the parameter combination that most closely fits the parameters specified by the user can be chosen.

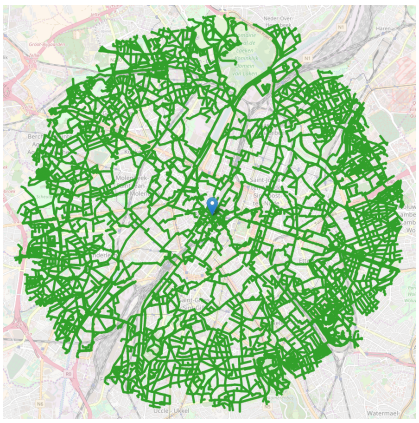


Figure 7: Example forward search tree using reach based routing

5.2.3. Backwards Routing using Inequalities

Since the weights of edges in the neighbourhood of the forward route are increased, it is hard to preprocess the graph with the goal of gaining time in the backward routing step. However, the number of vertices involved in the backward routing can be limited by making use of inequalities. During backwards routing, the system keeps track of c, the first vertex in the forward path that has not yet been reached by the backward search tree. $\pi_f(c)$ is the part of the forward path beginning in vertex s and ending in c. Since all closed tours that have not yet been found will pass through c, any node w on a path $\pi_r(w)$ from w to s, should satisfy the following inequality, as illustrated in Fig. 8:

$$l(\pi_f(c)) + d_v(c, w) + l(\pi_r(w)) < l_{max}$$

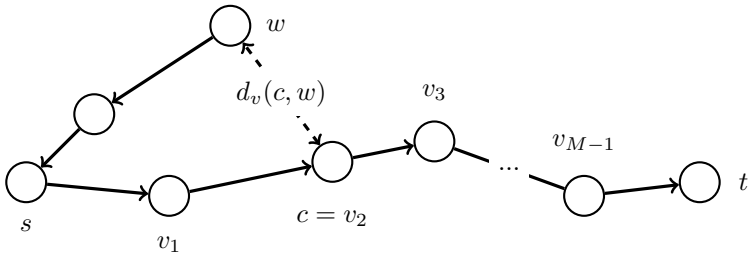


Figure 8: Backward search tree diagram

The nodes of the forward path that are close to the starting point will usually not be used as turning point, since that would typically result in a route that is too short. The search space can be limited even more by assuming that the walking length of the starting point to the turning point will be at least $\beta\, l_{min}/2$ $(0 < \beta < 1)$. Under this assumption, c is initialized as the first node

of the forward path for which this property is fulfilled. An example of such a backward search tree (using $\beta = 0.8$) is shown in Fig. 9.

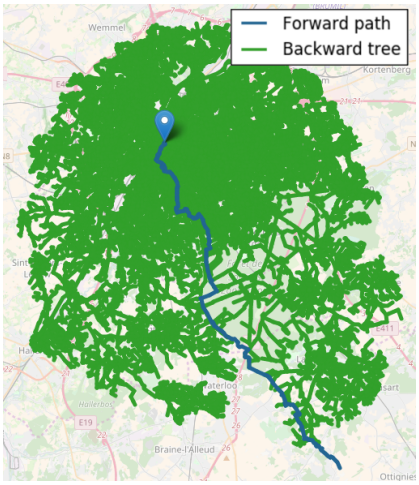


Figure 9: Example backward search tree

6. Results

6.1. Data

The algorithm is tested on an OpenStreetMap (OSM) dataset that stores all roads in Belgium. The dataset is preprocessed to convert it into a graph representation that contains the parameters required by the algorithm. Each of the edges has weights assigned to it, representing the road safety and scenic value of the represented road, as discussed in the introduction. All results that are presented in this section are calculated for a weight vector that assigns equal importance to these attributes.

country	# nodes	area (km)	# edges
Belgium	558 585	30 528	1 646 731

Table 1: Graph characteristics of the test dataset

This section presents the results of two series of experiments. The first experiments aim to assess the quality of the heuristic, by comparing the routes that are found by the heuristic with (bounds on) the optimal tours that are calculated by the branch-and-bound algorithm. To limit the number of walks that are evaluated by the branch-and-bound algorithm, these first experiments are executed on a modified graph that represents a more constrained version of the Cycling Problem. The second series of experiment studies the running

time of the heuristic and the effect of some input parameters on whether the heuristic successfully finds a tour that satisfies the length constraints.

All results that are presented, unless stated otherwise, are calculated for a (roundness) strictness σ of 0.4 and a weight $\lambda = 12$. These parameters were experimentally determined and represent, to our opinion, a reasonable trade-off between edges costs and roundness penalties. For the heuristic, the optimization parameters $\beta = 0.6$ and $\epsilon = 0.01$ were used.

6.2. Route quality

In this experiment, 500 random starting points were chosen and for each of these points, the optimal tour was searched with a length in $[5i, 5i + 5]$ km, for $i \in \{0, 1, ..., 13\}$. To limit the search space of the branch-and-bound algorithm, an additional constraint is imposed on the solution: at any moment along the tour, an edge can only be traversed if, for each node encountered during the last $l_{min}/20$ km of the tour, the edge is part of a least cost path departing from that node. Intuitively, the constraint avoids that the resulting tour features 'sharp turns'. The limitation can be encoded by replacing the original cycling graph G by a graph G' in which each vertex $u_{v\to w} \in V(G')$ represents a least-cost path between nodes $v, w \in V(G)$ with a length that is less than or equal to $l_{min}/20$. Furthermore, the graph G' was adapted such that no anti-parallel edges were present (this can be done similarly to the other constraint, by making sure that to $\forall e = \langle v, w\rangle \in E : u_{v\to w} \in V(G')$, where vertex $u_{v\to w}$, represents a shortest path between v and w). Both of these requirements aim to remove the presence of short, low cost-to-length cycles, since the bounding algorithm cannot detect the penalty associated with traversing these cycles multiple times.

Even with these additional constraints, the branch-and-bound algorithm requires an average routing time varying from 2 to 20 minutes to find tours of varying lengths. This is acceptable when calculating theoretical lower bounds to evaluate the quality of tours, but too slow for applications with user interaction.

In Fig. 10, the score (see equation 1) of the optimal algorithm and the heuristic is visualised for varying tour lengths. In the case of the heuristic, the results are shown for attempting four times to find a tour (which is what we recommend) and selecting the best one and attempting a single time to find a tour. The average cost-to-length ratio in this graph is 1.52. Both the optimal algorithm and the heuristic perform increasingly better than this average, the quality of the generated tours improves as the length of the tour increases. For the optimal algorithm, this is a result of the increasing number of possible walks that are available as the length increases, allowing to pick a tour with both a near-zero penalty and a low cost. In the case of the heuristic, phenomena with similar effects occur: for short paths, it can be hard to avoid the forward path during backward routing due to all sorts of physical boundaries. As the length of the tour increases, the size of these boundaries become smaller in comparison to the tour, thus making it more easy to achieve a low penalty. Furthermore, longer forward and backward paths feature more low cost edges (i.e. 'bicycle highways') , which typically also have a low cost-to-length ratio.

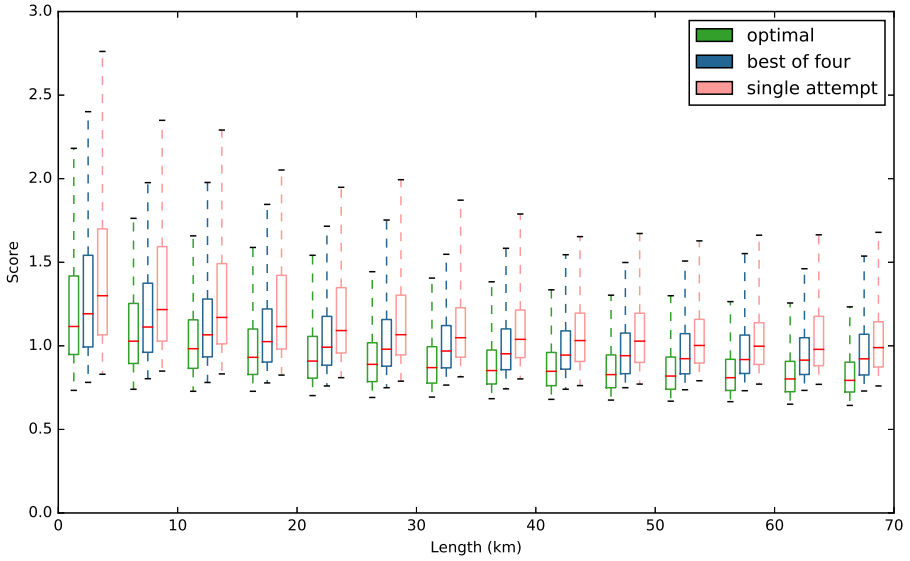


Figure 10: Scores for tours of varying length as calculated by the exact algorithm and 4 or 1 iterations of the heuristic
(boxplots show 5, 25, 50, 75 and 95 percentiles)

6.3. Error of average tour penalty

For the second series of experiments, 500 random starting points were chosen and tours of different lengths were calculated starting from each of these points. Tours were calculated for lengths $[5i, 5i + 5]$ km, for $i \in \{0, 1, ..., 29\}$. For each of these 30 required lengths per node, the heuristic attempts four times to find a tour.

In order to assess the accuracy of the average tour penalty formula $p_{avg}(\pi)$ for tours of different sizes, we calculated for each tour the average tour penalty using the graph representation used by the heuristic and using a high quality graph representation, in which none of the edges have a length that exceeds 10m. The absolute error between the standard and the high quality representation $\Delta p_{avg}(\pi)$, as well as the high quality penalty is shown in Fig. 11 for tours of varying sizes.

We notice that, although $\Delta p_{avg}(\pi)$ is significant for short tours (especially when compared to $p_{avg}(\pi)$), the error quickly decreases as the length of the tour increases. For tours longer than 20 km, $\Delta p_{avg}(\pi)$ is negligible. This is somewhat unsurprising, since larger tours feature more edges and thus a smaller edge size when compared to the tour length. The large absolute error for short tours can easily be remidied by using a higher quality graph representation when searching for short tours. The resulting increase in running time when calculating these short tours would not pose a problem, juding by the timing results presented in subsection 6.5.2.

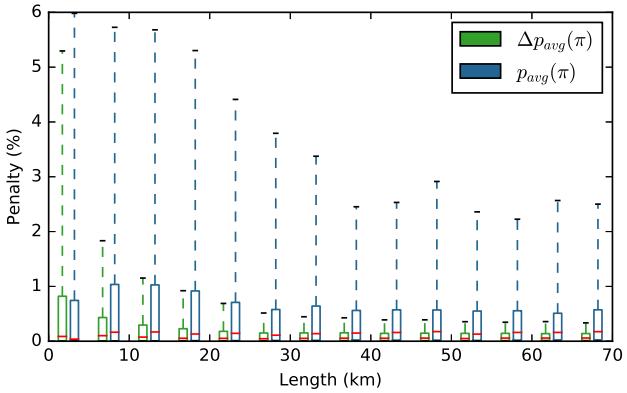


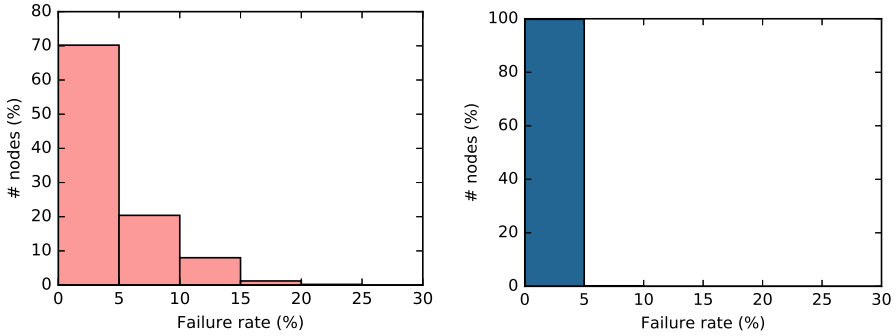
Figure 11: Average tour penalty and error of the average tour penalty for varying tour lengths
(boxplots show 5, 25, 50, 75 and 95 percentiles)

6.4. Solvability

The measurements for evaluating the error of the average penalty are reused to study the effect of the starting location on the chance of such an attempt failing to find a solution. It should be noted that the used length constraints allow for less variation in tour lengths than what would typically be required in practice, especially for longer tours. Figure 12a shows that although most nodes have a chance of failure around 0-5%, there are nodes which perform significantly worse. Figure 13a reveals that these difficult starting points are situated in the southern part of Belgium (especially near the border). This makes sense, considering that the heuristic finds a tour for each of the vertices along the forward path. Since the road network in the south of Belgium is more sparse, the forward path contains less intersections and thus the chance of one of these intersections resulting in a tour with a length within the specified constraints decreases.

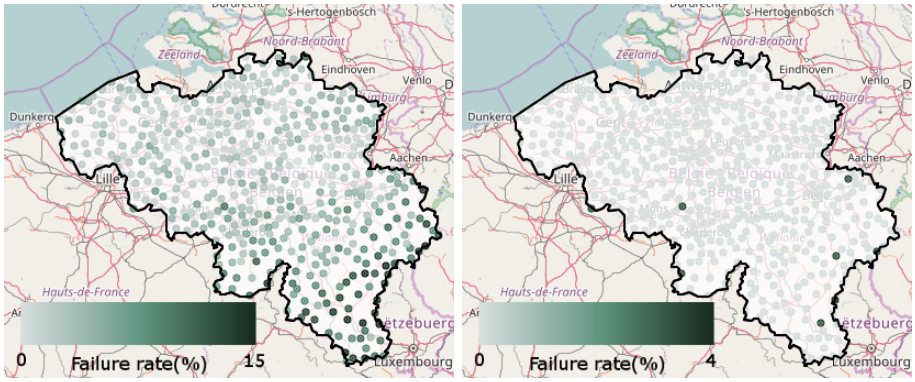
As one would expect, searching multiple times for a tour significantly improves the chance of finding one. This is illustrated in Fig. 12b and 13b, that show the probability of at least one of four attempts being successful. Almost all of the remaining failed configurations correspond to short length requirements, as shown in Fig. 14. These failures are caused by nodes that are situated in remote area's or at a dead end of a long road, making it impossible to find short tours without turning around and returning to the starting point without travelling back along the forward path. Also, for nodes that are close to a border the directions in which the forward path can head are limited, resulting in more overlap and a lower chance of success. It should be noted that because both border nodes and short tours result in a smaller search space, the running time of the algorithm is lower in these cases and one could execute more routing attempts to increase the success rate.

We also notice from Fig. 14 that the chance of a single attempt failing to



(a) One failure (b) Four subsequent failures

Figure 12: Chance of the heuristic failing to achieve a configuration



(a) One failure (b) Four subsequent failures

Figure 13: Chance per node of the heuristic failing to achieve a configuration

find a tour increases slightly for increasing tour lengths. This is because longer forward paths tend to feature more low-cost edges, typically corresponding to long, uninterrupted sections of cycling way. The lower number of intersections per distance unit of these roads results in fewer tours that can satisfy the length constraints. This effect is however limited, barely affecting the success rate for four executions.

In conclusion, for length constraints that allow for tours with a length within a range of 5 km (or more), the heuristic is almost always able to find tours with satisfying length.

6.5. Running times

To study the running time, the same measurements are used as for looking at solvability. These measurements were performed using an Intel Core i7-6700HQ CPU running at a clock frequency of 2.60 GHz. The graph in which

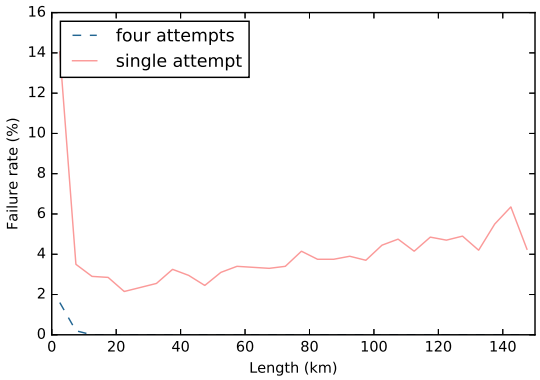


Figure 14: Chance of the heuristic failing to achieve a configuration for varying tour lengths

the routing is performed was loaded into RAM-memory before the start of the time measurements.

6.5.1. Optimizations

Since the use of reaches does not result in a decrease of the worst case running time, the running time of the forward routing step is measured with and without reaches. For the measurement with reaches, reaches of a length up to 5 km were exactly calculated (the reach of all other nodes was set to infinity). As shown in Fig. 15, 70% of the vertices have a reach that is below this length. This is important, since the exact calculation of reaches requires to calculate all shortest paths, whilst calculation up to a specified length can be achieved using local shortest path trees, see Gutman (2004).

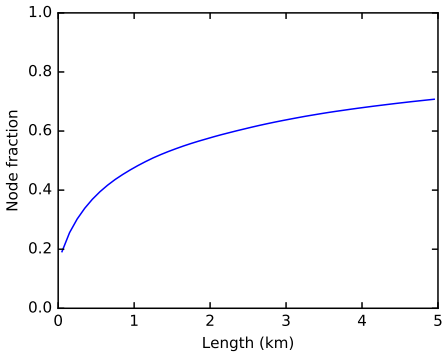


Figure 15: Cumulative distribution of reaches

The effect of using reaches for calculating tours of various lengths is illustrated in Fig. 16. For long tours, a speedup of a factor four is achieved.

To evaluate the impact of the optimisations on the backward routing time,

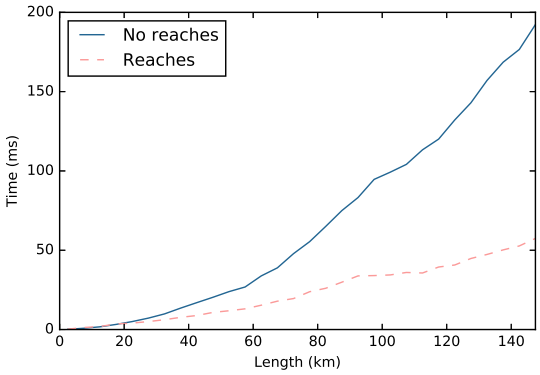


Figure 16: Time required for forward routing

this time is compared for $\beta = 0$ (no assumptions on the position of the turn node along the forward path) and $\beta = 0.6$. The higher β value results in a significant reduction in the search space of the heuristic, resulting in a speedup of a factor two.

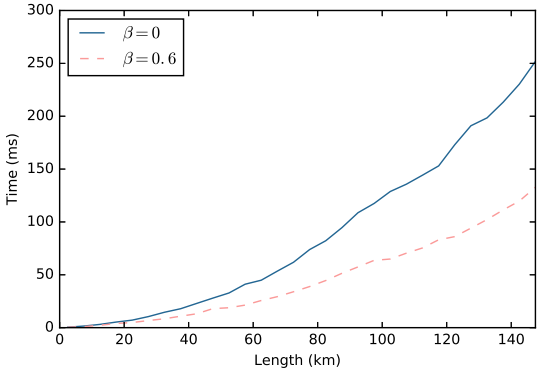
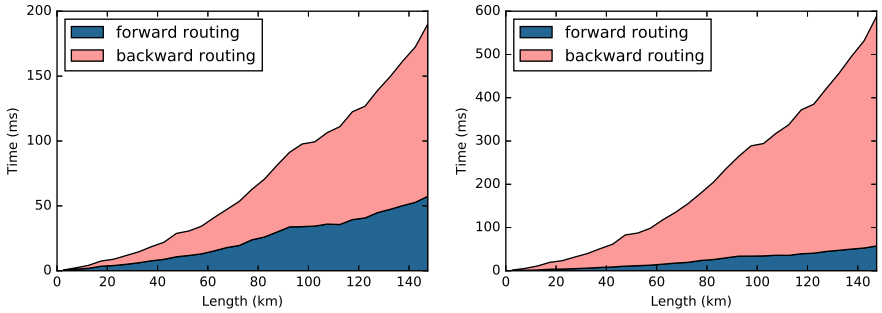


Figure 17: Time required for backward routing (1 attempt)

6.5.2. Total running time

In Fig. 18, the total running time of the heuristic and the different components of which this time consists are plotted. The time required by the heuristic for executing a single attempt and for executing four attempts is shown. Note that executing multiple attempts does not require to re-execute the forward routing step. The plots illustrate the ability of the heuristic to generate constrained cycling routes up to 150 km in the sub-second timescale. This is an increase of an order of magnitude in tour length compared to the previous work on closed tours (see section 2).



(a) Single attempt (b) Four attempts

Figure 18: Total routing time for tours of varying length

7. Conclusion

A routing algorithm is designed that allows cyclists to generate personalized tours within given length constraints. This problem is shown to be NP-hard. An exact branch-and-bound algorithm is introduced and a heuristic is designed.

The heuristic generates routes by selecting an intermediary node and searching a path to this node and a backward path to the starting point, while taking into account personalized user preferences. The running time of this algorithm is shown to be $O((l_{max})^2 \log(l_{max}))$, where l_{max} is the maximal allowed tour length. This running time was improved using reaches and inequalities to limit the search space.

The resulting algorithm is tested and validated on an OpenStreetMap-based graph of the Belgian road network. The results are compared to the exact solutions. Extensive studies are performed regarding the time behaviour and the success rate of the algorithm. The heuristic is able to generate high-quality routes of up to 150 km in the subsecond time scale.

8. Future work

To allow designing even longer routes, for instance tours for motorized vehicles or multi-day cycling trips, research would be needed to speed up the heuristic even more.

Acknowledgement

We wish to thank the reviewers for their valuable comments and their constructive input in improving our work.

Pieter Stroobant is funded by a Ph.D. grant of Ghent University, Special Research Fund (BOF). The authors wish to thank Ghent University - IMEC for their support and their funding through the IOP research project “Modelling uncertainty in hub location planning through interdisciplinary research”.

Compliance with Ethical Standards

Conflict of Interest: The authors declare that they have no conflict of interest.

References

Majid Alivand and Hartwig H. Hochmair. Choice Set Generation for Modeling Scenic Route Choice Behavior with Geographic Information Systems. *Transportation Research Record: Journal of the Transportation Research Board*, 2495:101–111, jan 2015. ISSN 0361-1981. doi: 10.3141/2495-11. URL http://trrjournalonline.trb.org/doi/10.3141/2495-11.

Majid Alivand, Hartwig Hochmair, and Sivaramakrishnan Srinivasan. Analyzing how travelers choose scenic routes using route choice models. *Computers, Environment and Urban Systems*, 50:41–52, mar 2015. doi: 10.1016/j.compenvurbsys.2014.10.004. URL http://dx.doi.org/10.1016/j.compenvurbsys.2014.10.004.

Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. Route Planning in Transportation Networks. Technical report, apr 2015. URL http://arxiv.org/abs/1504.05140.

Jonathan Dees, Robert Geisberger, Peter Sanders, and Roland Bader. Defining and Computing Alternative Routes in Road Networks. *CoRR*, abs/1002.4, feb 2010. URL http://arxiv.org/abs/1002.4330.

Sofie Demeyer, Pieter Audenaert, Mario Pickavet, and Piet Demeester. Dynamic and stochastic routing for multimodal transportation systems. *IET Intelligent Transport Systems*, 8(2):112–123, 2014. ISSN 1751-956X. doi: 10.1049/iet-its.2012.0065. URL http://digital-library.theiet.org/content/journals/10.1049/iet-its.2012.0065.

Zhaohui J. Fu and Hartwig Hochmair. Web Based Bicycle Trip Planning for Broward County, Florida. *GIS Center*, 31, 2009.

Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In *Experimental Algorithms*, volume 5038 LNCS, pages 319–333. Springer Berlin Heidelberg, Berlin, Heidelberg, jul 2008. ISBN 3540685480. doi: 10.1007/978-3-540-68552-4_24. URL http://link.springer.com/10.1007/978-3-540-68552-4_24.

Andreas Gemsa, Thomas Pajor, Dorothea Wagner, and Tobias Zündorf. Efficient Computation of Jogging Routes. In *Lecture Notes in Computer Science*, volume 7933 LNCS, pages 272–283. 2013. ISBN 9783642385261. doi: 10.1007/978-3-642-38527-8_25. URL http://link.springer.com/10.1007/978-3-642-38527-8_25.

Ron Gutman. Reach-based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks. In *Proceedings 6th Workshop on Algorithm Engineering and Experiments*, pages 100–111, Philadelphia, 2004. Society for Industrial and Applied Mathematics. ISBN 0-89871-564-4. URL `http://www.siam.org/meetings/alenex04/abstacts/rgutman1.pdf`.

Hartwig Hochmair. Decision Support for Bicycle Route Planning In Urban Environments. In *Proceedings of the 7th AGILE Conference on Geographic Information Science*, pages 697–706. Association of Geographic Information Laboratories in Europe, 2004.

Jan Hrncir, Marcel Nemet, Jan Hrncir, Qing Song, Pavol Zilecky, Marcel Nemet, and Michal Jakob. Bicycle Route Planning with Route Choice Preferences. In *Proceedings of the Twenty-first European Conference on Artificial Intelligence*, pages 1149–1154. Elsevier and Springer, 2014. ISBN 9781614994190. doi: 10.3233/978-1-61499-419-0-1149.

Jan Hrncir, Pavol Zilecky, Qing Song, and Michal Jakob. Speedups for Multi-Criteria Urban Bicycle Routing. *OASIcs - OpenAccess Series in Informatics*, 48:28, 2015. ISSN 2190-6807. doi: 10.4230/OASIcs.ATMOS.2015.16. URL `http://drops.dagstuhl.de/opus/volltexte/2015/5458/`.

Richard M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23(3):309–311, 1978. ISSN 0012365X. doi: 10.1016/0012-365X(78)90011-0. URL `http://linkinghub.elsevier.com/retrieve/pii/0012365X78900110`.

Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Springer-Verlag, Berlin, Heidelberg, 1 edition, 2004. ISBN 3642073115. doi: 10.1007/978-3-540-24777-7.

Eugene L. Lawler and David E. Wood. Branch-and-Bound Methods: A Survey. *Operations Research*, 14(4):699–719, aug 1966. ISSN 0030-364X. doi: 10.1287/opre.14.4.699. URL `http://pubsonline.informs.org/doi/abs/10.1287/opre.14.4.699`.

Eliane M. Loiola, Nair M. M. de Abreu, Paulo O. Boaventura-Netto, Peter Hahn, and Tania Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176(2):657–690, 2007. ISSN 03772217. doi: 10.1016/j.ejor.2005.09.032.

Andreas Paraskevopoulos and Christos Zaroliagis. Improved alternative route planning. In *13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, volume 33, pages 108–122. Optimization and Systems, 2013. URL `http://hal.archives-ouvertes.fr/hal-00871739/`.

John Pucher, Ralph Buehler, Dafna Merom, and Adrian Bauman. Walking and cycling in the United States, 2001-2009: Evidence from the National

Household Travel Surveys. *American Journal of Public Health*, 101(SUPPL. 1):310–317, 2011. ISSN 00900036. doi: 10.2105/AJPH.2010.300067.

Borzou Rostami, André Chassein, Michael Hopf, Davide Frey, Christoph Buchheim, Federico Malucelli, and Marc Goerigk. The Quadratic Shortest Path Problem: Complexity, Approximability, and Solution Methods. Technical report, 2016. URL `http://www.optimization-online.org/DB_HTML/2016/02/5341.html`.

Dennis Schieferdecker, Moritz Kobitzsch, and Marcel Radermacher. Evolution and Evaluation of the Penalty Method for Alternative Graphs. In *Proceedings of the 13th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'13)*, volume 33, pages 94–107, 2013. ISBN 9783939897583. doi: 10.4230/OASIcs.ATMOS.2013.94. URL `http://drops.dagstuhl.de/opus/volltexte/2013/4247`.

Qing Song, Pavol Zilecky, Michal Jakob, and Jan Hrncir. Exploring pareto routes in multi-criteria urban bicycle routing. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, page 1781. IEEE Intelligent Transportation Systems Society, 2014. ISBN 2153-0009;21530009. doi: 10.1109/ITSC.2014.6957951. URL `http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6957951`.

Sabine Storandt. Route Planning for Bicycles — Exact Constrained Shortest Paths Made Practical Via Contraction Hierarchy. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling*, pages 234–242, 2012. ISBN 9781577355625.

Jason G. Su, Meghan Winters, Melissa Nunes, and Michael Brauer. Designing a route planner to facilitate and promote cycling in Metro Vancouver, Canada. *Transportation Research Part A: Policy and Practice*, 44(7):495–505, 2010. ISSN 09658564. doi: 10.1016/j.tra.2010.03.015. URL `http://dx.doi.org/10.1016/j.tra.2010.03.015`.

Robert J. Turverey, David D. Cheng, Owen N. Blair, Joseph T. Roth, and Gregory M. Lamp. Charlottesville bike route planner. *2010 IEEE Systems and Information Engineering Design Symposium, SIEDS10*, pages 68–72, 2010. doi: 10.1109/SIEDS.2010.5469679.