Utrecht University

The Quality Arc Orienteering Problem for Multi-Objective Scenic Route Planning

Master's Thesis

ICA-3725154

Jarno Le Conté

1^{st} supervisor
prof. dr. M.J. van Kreveld
2^{nd} supervisor
prof. dr. H.L. Bodlaender

Department of Computing Science
Utrecht University, The Netherlands
May 15, 2017

Abstract

The Quality Arc Orienteering Problem (Quality-AOP) is to find a path in a directed graph subject to a travel cost budget and in which the quality of the path is optimized. Due to our restrictive problem definition and the use of a strong quality measure in the objective function, artifacts such as subtours and detours do not occur in Quality-AOP where they did occur in the Arc Orienteering Problem. The budget interval allows us to find higher quality routes close to the desired costs. We give a polynomial time heuristic for this problem, which is NP-hard. Our algorithm is based on a Greedy Randomized Adaptive Search Procedure. Our experiments show the reduction of artifacts and also improvements on the route's quality up to 10%.

Contents

1 Introduction **4**

2 Related work **5**

3 Preliminaries **6**
3.1 Arc Orienteering Problem (AOP) . . . 6
3.2 Examples of the AOP . . . 7

4 Problem description **8**
4.1 Quality Arc Orienteering Problem (Quality-AOP) . . . 9
4.1.1 Examples of the Quality-AOP . . . 10
4.1.2 Complexity classification . . . 12
4.1.3 Example showing that Quality-AOP $\neq$ AOP . . . 13
4.2 Quality Orienteering Problem (QOP) . . . 14
4.2.1 Examples of the QOP . . . 14

5 Solving the Quality-AOP **15**
5.1 Mathematical formulation . . . 15
5.2 Metaheuristics . . . 17

6 An algorithm for the Quality-AOP **18**
6.1 The algorithm . . . 19
6.2 Example . . . 27
6.3 Design choices . . . 30
6.4 Running time reduction . . . 32
6.5 Running time analysis . . . 33

7 Experiment **34**
7.1 Research questions . . . 34
7.2 Setup . . . 35
7.2.1 Road network . . . 35
7.2.2 Measured properties . . . 36
7.2.3 Configuration . . . 38
7.3 Results . . . 39
7.3.1 Optimizing objectives . . . 39
7.3.2 Quality measure . . . 41
7.3.3 Budget and quality . . . 43
7.3.4 Detours and artifacts . . . 46
7.3.5 Route properties and artifacts . . . 48

8 Conclusion **51**

1 Introduction

Finding a scenic route (i.e. an attractive route for a traveler) can not be done easily using the traditional route planning techniques such as Dijkstra's [12], because they will only search for the least cost path between two locations. Costs can be defined as the amount of travel time or travel distance for example, which should be minimal in a route from source to destination. In the case of scenic route planning we would like to allow detours to increase the scenicness of the route but still we want to reach a destination in reasonable time. Therefore we have to deal with two conflicting criteria, maximize scenicness while minimizing costs, and therefore there is no straightforward solution to this problem. In the first place because the amount of detour that would be allowed depends on the user's preference and furthermore because the number of possible detours that exist in a road network is large which makes it hard to find the best route that matches the user's preference.

More generally we have to deal with a multi-objective optimization problem where at least two conflicting objectives should be optimized. In reality there are even more objectives because the scenicness criterion consists of many sub criteria that individually add scenic value to a route, such as road quality, view from the road, points of interest, ease of driving, etc. Also the costs can be broken down into multiple objectives such as minimizing travel distance, travel time, fuel consumption, etc. Although there are many models for multi-objective optimisation, the disadvantage is that they generate many solutions in the Pareto Front [21], i.e. solutions that are incomparable in terms of their optimisation goals, but they do not take into account domain specific knowledge such as the user's preference.

In scenic route planning, we can use domain specific knowledge, because in most cases the user knows how much time or distance he wants to travel. Therefore we can model this as a new problem, namely finding scenic routes within a limited cost budget. Such budget can be the amount of travel time or travel distance the route may cost. This problem is known as the Orienteering Problem (OP) [17] where a route between source and destination should be found within a limited cost budget while the scenic score collected from crossroads should be maximized. A variant to this problem is the Arc Orienteering Problem (AOP) [33] where scenic scores are attached to road segments instead of road crossings, such that we are optimizing the traveling experience on the road rather than visiting attractive locations and therefore it models the problem of scenic routing better.

We can improve upon the AOP even more, because maximizing the scenicness will not guarantee the best route in the sense of the amount of collected scenic score in proportion to the travel costs. We argue that we need a new quality measure to measure the average scenic score along the route as a whole. This requires a modification to the AOP and a new algorithm to solve it. We will present a mathematical solution (in the form of an integer program) as well as an algorithm (using a greedy randomized metaheuristic) to find a route in reasonable time. We set up an experiment using this algorithm to find routes on a real road network. The data originates from OpenStreetMap [30]. We compare the results between the usage of the new quality measure and the measure specified by the AOP.

In extension, our algorithm will work with different cost measures, such as travel distance or travel time. Also it can handle different quality measures, which make it suitable to solve the AOP as well as our new problem definition using the average score as quality measure. Furthermore the quality measures can not only measure scenicness of roads but also scenicness based on properties of the route as a whole, which makes it more general than most of the related work in this research area. It can plan routes from A to B, including a tour from A to A. We also add constraints to avoid traveling roads twice and avoid traveling

the same road in opposite direction. These extensions make that the problem definition has to be changed.

2 Related work

There are several variations on the scenic routing problem, each with its own definition. There exist modified versions of the Traveling Salesman Problem (TSP) that work with profits [14], where an objective function will maximize the profit while the restriction on visiting all vertices is dropped. For example:

- The Profitable Tour Problem (PTP) [11] is to find a tour with the least costs by decreasing the costs with the collected score from vertices.
- In the Profitable Arc Tour Problem (PATP) [13] scores are attached to arcs.
- In the Tour Orienteering Problem (TOP) [25] scores collected from vertices will be maximized and the cost of the route is limited by an upper bound.
- In the Prize-collecting TSP (PCTSP) [7] a lower bound restriction on the collected score is added.
- In the Prize-collecting Rural Postman Problem (PRPP) [4][3] scores are attached to arcs instead of vertices.

These TSP variants have in common that the route returns to the starting location. The Orienteering Problem (OP) [36] and Selective Travelling Salesman Problem (STSP) [20] are variants that do not require to return to the starting location. Archetti and Speranza [5] give an overview of arc routing problems with profits. One of them is the Arc Orienteering Problem (AOP) [33] where the scores from arcs should be maximized along the route. The following literature is based upon the AOP.

- The Most Scenic Path Problem (MSPP) [23] introduces speedup methods to solve the AOP quickly on real-scale road networks.
- The Cycle Trip Planning Problem (CTPP) [37] specifies a lower and upper bound on travel costs and allow multiple starting locations to choose from.
- The Outdoor Activity Tour Suggestion Problem (OATSP) [24] introduces the notion of average scores for arcs, but include non-average scores of vertices as well, and furthermore it will only work with the cost measure based on travel distance.

In our thesis we mainly focus on extending the AOP, and give an algorithm to solve it. Souffriau *et al.* [33] have given an algorithm to solve the AOP based on Greedy Randomized Adaptive Search Procedure (GRASP) [15], while Lu and Shahabi [23] and Verbeeck *et al.* [37] use Iterated Local Search (ILS) [22] because it has faster iterations compared to GRASP.

Mercier *et al.* [26] give an overview of research about determining the scenicness of an arc. This includes research to use geo-tagged photos, GPS tracks, and visibility of scenery to determine attractiveness of a road segment. Several studies on this subject are done by Alivand and Hochmair [1], Alivand *et al.* [2], Aultman-Hall *et al.* [6], Golledge [18], Hochmair [19], Navratil [28], Niaraki and Kim [29], Salomon and Mokhtarian [32], Zheng *et al.* [38] and Qin *et al.* [31].

3 Preliminaries

Notation We will represent the road network as a directed graph $G = (V, E)$ with V the set of n vertices (nodes) and E the set of m arcs (edges) connecting pairs of vertices. Road segments will be represented by arcs and road intersections will be represented by vertices. An arc $e \in E$ can be represented as a pair of vertices $(u, v) \in V \times V$ which denotes a directed connection from vertex u to vertex v. If there is also a connection from v to u than we call that the twin arc $\bar{e}$ of e. A path P is a sequence of arcs $\langle e_1, e_2, ..., e_k \rangle$ of length k with the restriction that these arcs form a contiguous chain of connected vertices, i.e. $(v_i, v_{i+1}) = e_i \in P$ for $i = 1, 2, ..., k$, and each arc may occur only once in the sequence. To represent a path P from a source vertex s to a destination vertex d that includes certain arcs, we use the notation $P(s \rightsquigarrow e_1 \rightsquigarrow e_2 \cdots e_k \rightsquigarrow d)$ in which arrows always represent a least cost path between the two noted entities.

Problem description We want to find a *scenic* route between two locations on a real road network, but more precisely we want to find the *most scenic route*. Due to the fact that such route could be short or even can have infinite length, we need to add additional constraints to manage these artifacts, e.g. such constraint can be the amount of time or distance the user wants to travel, or a constraint to avoid traveling arcs twice. See Definitions 3.1, 3.2, and 3.3 for the definitions of *route*, *scenic route*, and *most scenic route*.

Definition 3.1 (Route)**.** *A path in a graph between two vertices.*

Definition 3.2 (Scenic Route)**.** *A route with a quality measure to determine the scenicness (attractiveness) of the route based on the route's properties, and whose quality value is comparable with others.*

Definition 3.3 (Most Scenic Route)**.** *A scenic route that satisfies a specified set of constraints and whose quality value is the highest among all possible routes in the graph that satisfy the constraints.*

3.1 Arc Orienteering Problem (AOP)

The Arc Orienteering Problem (AOP) [33] as defined in Lu and Shahabi [23] is a formulation that conforms with our definition of finding the most scenic route. The AOP is to find a route that has the most scenic score collected along the route and also satisfies the constraint that the amount of travel costs stay below an upper limit. The starting and ending locations may or may not be the same.

Let $G = (V, E)$ be a graph where each arc $e \in E$ is associated with a cost $e.cost$ (e.g. travel distance) and a score $e.score$ (e.g. the attractiveness value for the view from the road that can be earned when traveling the arc). Furthermore the travel costs should be a positive real value ($e.cost > 0$) and $e.score$ gets assigned a non-negative real value ($e.score \geq 0$). In

this graph we can find a path P that is a sequence of arcs $\langle e_1, e_2, ..., e_k \rangle$ between a source vertex s and destination vertex d, in which each arc may occur only once. The sum of scores of individual arcs in P determine the score of the path P, i.e. $P.score = \sum_{e \in P} e.score$. The costs of P is the sum of the costs of the individual arcs, i.e. $P.cost = \sum_{e \in P} e.cost$. Further there is a restriction on the upperbound of costs a path may have, which is limited by the cost budget B.

The formal definition of the Arc Orienteering Problem is given in Definition 3.4.

Definition 3.4 (AOP)**.** *Given a graph $G = (V, E)$ and a source vertex s and a destination vertex d, further given a total travel cost budget B, the* ***Arc Orienteering Problem (AOP)*** *is finding a path $P = \langle e_1, e_2, ..., e_k \rangle$ from s to d such that the total travel cost is no more than the cost budget B and the total collected score is maximized, i.e.*

Maximize:

$$\sum_{e \in P} e.score \tag{1}$$

Subject to:

$$\sum_{e \in P} e.cost \leq B \tag{2}$$

3.2 Examples of the AOP

In Figure 1 an example is given of a directed graph $G = (V, E)$ with 12 vertices and 24 arcs. Arcs can only be traveled in one direction. Each arc has cost 1, and a scenic score that is indicated by a boxed number on the arc, or will be 0 if no box is present.

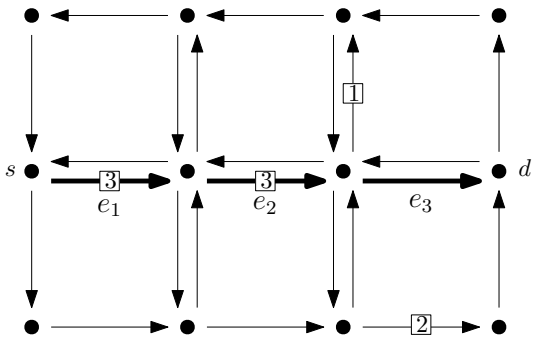


Figure 1: The shortest path $\langle e_1, e_2, e_3 \rangle$ between s and d has score 6 and cost 3.

Shortest path The shortest path in Figure 1 between vertices s and d is path $P = \langle e_1, e_2, e_3 \rangle$, indicated by thicker stroked arcs. The arcs of P are labeled in order, e.g. in this

case e_1, e_2, and e_3. The score of the path is the sum of the scores of e_1, e_2, and e_3, which equals 6 because the path contains two arcs with score 3. The cost of P is 3 because the path contains 3 arcs, each costs 1 to traverse.

Most scenic route If we compute a path based on the AOP, we could possibly find a path with higher scenic score depending on the travel budget B. If the travel costs become limited by a budget $B = 5$ we find a path as in Figure 2. The most scenic path $P = \langle e_1, e_2, e_3, e_4, e_5 \rangle$ has three scenic arcs summing up to a scenic score of $3 + 3 + 2 = 8$ (which is more than the shortest path) and has a total costs of 5.

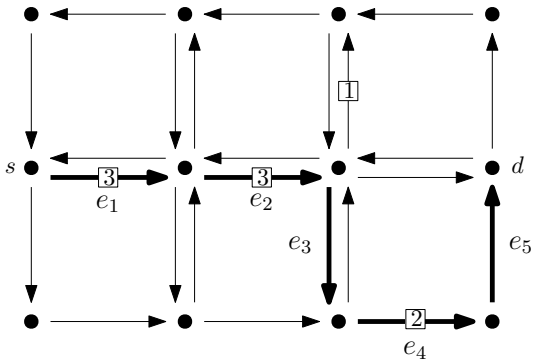


Figure 2: Path $P = \langle e_1, e_2, e_3, e_4, e_5 \rangle$ is the most scenic path between s and d limited by cost budget $B = 5$ and has score $3 + 3 + 2 = 8$ and cost 5.

4 Problem description

This thesis builds upon the Arc Orienteering Problem (AOP) [33] because it closely matches our Definition 3.3 of the most scenic route, given in Section 3. Because we think the AOP can be further improved, we will give a new definition in Section 4.1 that replaces the quality measure in the AOP by a new quality measure that takes into account the average score. Furthermore we add additional constraints to obtain routes that better match the expectation of a user in terms of scenicness.

Then in Section 4.2 we will give a more general definition of the problem that will not only take into account scenicness of arcs but also can determine scenicness based on other route properties, including scenic score attached to vertices, and even properties that should be measured globally like the shape of the route to avoid travelling in a small area too much.

We have chosen to present both definitions in this thesis, because we think they both introduce new important concepts. The first one closely relates to existing research and can be easily represented in mathematical formulation, while the second definition is more general and adds more aspects of the real problem to find routes that match travelers' wishes, but at the same time it is harder to build an efficient algorithm for it. Therefore both are interesting in their way, although the first definition is a specific instance of the second definition.

4.1 Quality Arc Orienteering Problem (Quality-AOP)

As stated by Souffriau *et al.* [33], recreational cyclists aim at riding a certain number of kilometres (specified by the budget B), but they do not mind riding a bit more if this improves the quality of the route. Therefore a budget B is too restrictive because it sets a hard upper limit on the travel costs that may be used and prevent from finding high quality routes just above the budget. Therefore Souffriau *et al.* [33] always enlarge B by a small amount such as $1km$. Clearly, this is not a real solution because, in the first place there is still a hard upper limit (which is only shifted a bit), but more importantly it now always tries to find routes of length $B + 1km$, even if the collected score barely increases, and therefore sometimes unnecessarily leads to routes with more travel time than requested.

In the AOP we can not find high quality routes above a budget, but also the quality of the route below the budget is not considered correctly, namely it will always tries to collect as much score as possible even if it generates a detour to collect them all, which could lessen the average quality of the route. Therefore we argue that we should not sum scenic scores but we need to look at the average scenic score along the route, which is the amount of collected score divided by the total travel costs. So if we have a path P, we will not try to maximize $P.score$, but instead maximize $P.score/P.cost$.

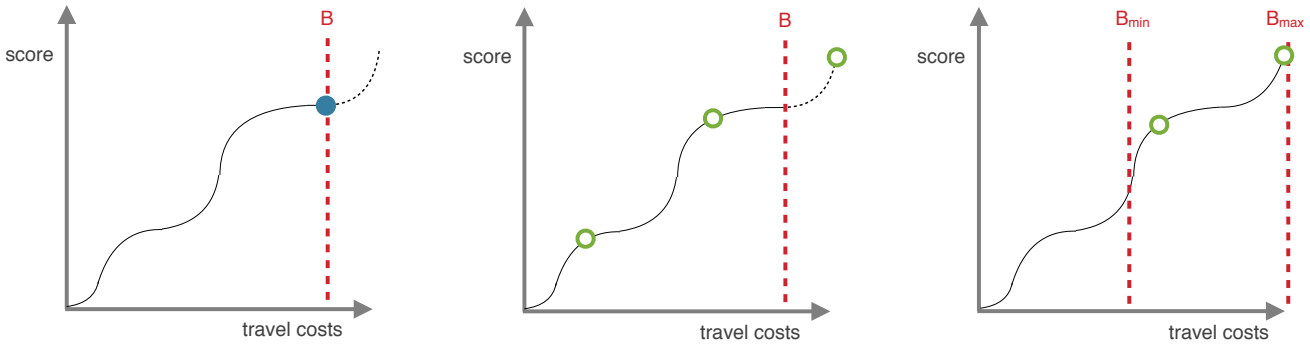


Figure 3: In the AOP (left) a route with highest score is found within a given budget. In the Quality-AOP we look at routes with the highest quality ($score/costs$) within a given budget interval specified by by B_{min} and B_{max} (right).

If we look at Figure 3 we can see that more score can be collected if the travel budget increases. For a fixed budget in the AOP we find a route that collects as much score as possible within the given budget (left graph). Let f be the function that represents the course of the collected score and let $\hat{f}$ be the normalized variant. We could clearly identify the local optima of $\hat{f}$ (middle graph) in which the quality is the highest (i.e. which amount of collected score relative to the amount of travel costs is high). These local optima are exactly at the positions in which the derivative of $\hat{f}$ changes from greater than 1 to less than 1, i.e. at the positions in which $\hat{f}'(x) = 1 \wedge \hat{f}''(x) < 0$. We argue that one of these routes are preferred over the route found for the AOP. To find one of these routes we must redefine the notion of budget. To find the third route, we must enlarge the budget, but within a range that is acceptable for the traveller. In the same way the traveller would travel at least a certain amount of time, and therefore a minimum budget is required as well (right graph)

to exclude the short travels of high quality. Our reformulation of the AOP let us find a high quality route within a travel costs interval for which the score in proportion to the travel costs is the highest.

Definition 4.1 (Quality-AOP)**.** *Given a directed graph $G = (V, E)$ and a source vertex s and a destination vertex d, further given a total travel cost budget B_{max} and minimum budget B_{min}, the* ***Quality-AOP*** *is finding a path $P = \langle e_1, e_2, ..., e_k \rangle$ from s to d, in which an arc is not traversed multiple times, nor traverses an arc another time in opposite direction ($\forall e \in P : \exists \bar{e} : \bar{e} \notin P$), and such that the total travel cost of P is at least B_{min} but not more than the cost budget B_{max}, and the average collected score is maximized, i.e.*

Maximize:

$$\frac{\sum_{e \in P} e.score}{\sum_{e \in P} e.cost} \tag{1}$$

Subject to:

$$B_{min} \leq \sum_{e \in P} e.cost \leq B_{max} \tag{2}$$

To avoid subtours we do not allow traversing arcs multiple times. We also do not want to traverse roads another time in opposite direction. Therefore we also added an additional restriction in the Quality-AOP to avoid traversing a twin arc. This is something that is not part of the AOP (Definition 3.4), because in the work of Souffriau *et al.* [33] where the AOP has been introduced it is only stated that arcs may be visited at most once. In directed graphs roads with two sided access can be represented as two directed arcs pointing in opposite direction, we call them a twin. We have explicitly specified that a restriction on traversing arcs also implies the same restriction on their twin arcs.

4.1.1 Examples of the Quality-AOP

To point out what is wrong with the AOP and how the Quality-AOP improves upon this, consider the example given in Figure 4. In the AOP it will try to collect the highest score as possible. If budget $B = 9$ it can generate a path that collects all scores. This path P has score $3 + 3 + 1 + 2 = 9$ and cost 9. Compared to the path we found in Figure 2 with scenic score 8, this path only adds 1 to the total score, while the costs increase from 5 to 9. We could say that P contains a detour in order to collect all scores, but therefore also include many more arcs that do not improve the path. If we express the quality of the path by the average score collected along the path, then we get the score in proportion to the costs, which is $\frac{score}{cost} = \frac{9}{9} = 1$ while the score of the path from Figure 2 is $\frac{8}{5} = 1.6$. We could argue that it is not worthwhile to include arc e_3 in order to collect the extra scenic score and we should look at quality of the route as a whole rather than summing up to a maximal score. Therefore we should add arcs to the path only if it improves the quality.

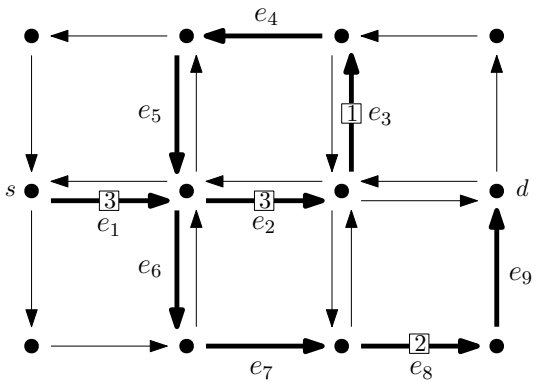


Figure 4: The AOP could easily generate a detour. For budget $B = 9$ it will find a path $P(s \rightsquigarrow e_3 \rightsquigarrow e_8 \rightsquigarrow d)$ between s and d collecting all scores. The path P has score $3 + 3 + 1 + 2 = 9$ and cost 9, and contains many arcs without score. The quality can be defined as $\frac{9}{9} = 1$

Minimum budget Together with the new quality measure we need a minimum budget. Consider the example given in Figure 5. If we again try to find the most scenic route with a cost budget $B = 9$ it will find the path $P = \langle e_1, e_2, e_3 \rangle$ with quality $\frac{6}{3} = 2$ (two times better than the AOP). So in this case the shortest path is the most scenic. Although, thinking of travelers who aim to travel at least a certain amount of time, this new path is probably not desirable. Therefore we need a constraint on the minimum amount of costs as well, so that all routes whose travel costs are within the budget are desirable. Now we should only find the route with the highest quality. Suppose $B_{min} = 4$ and $B_{max} = 9$, it will find the path as presented in Figure 6 and has quality $\frac{8}{5} = 1.6$ which is the best route that can be found and accepted by the budget restriction.

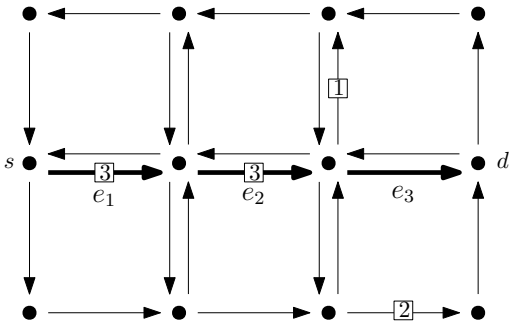


Figure 5: For the Quality-AOP we find a path P limited by only an upper bound cost limit $B = 9$ that has score 6, cost 3, and quality $\frac{6}{3} = 2$.

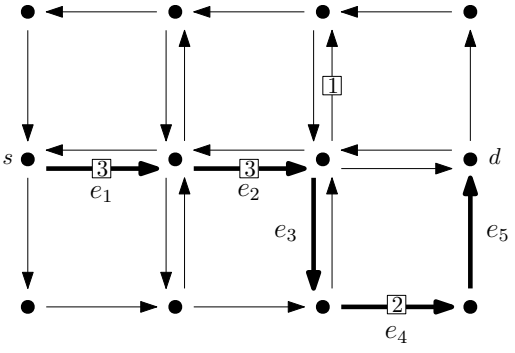


Figure 6: For the Quality-AOP we find a path P limited by $B_{min} = 4$ and $B_{max} = 9$ with score $3 + 3 + 2 = 8$, cost 5, and quality $\frac{8}{5} = 1.6$.

Restricted traversing To avoid subtours it is not allowed to traverse arcs multiple times. We also do not want to traverse roads another time from opposite direction. Therefore we added an additional restriction in Definition 4.1, so that the path from Figure 7 is not valid in the Quality-AOP.

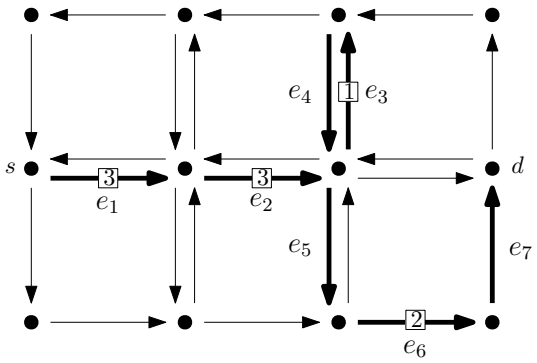


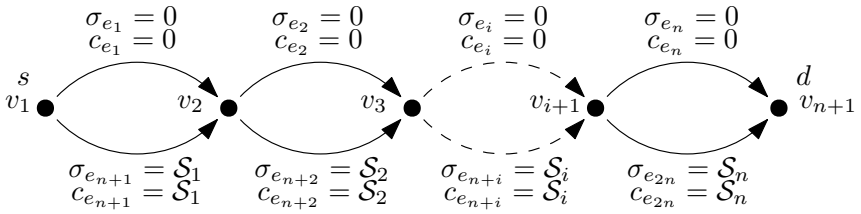
Figure 7: The u-turn between e_3 and e_4 is not allowed because an edge (that contains of two directed arcs) may not be visited multiple times. So this path is not valid in the Quality-AOP.

4.1.2 Complexity classification

In computing science the Arc Orienteering Problem (AOP) [33] is one of the harder problems to solve, due to the many possible routes to consider. First, Golden *et al.* [17] have proven that a simplification of the Orienteering Problem (OP) [36] can be used to construct an instance that expresses the Generalized Traveling Salesman Problem (GTSP) [35]. Because GTSP is classified NP-hard, OP is NP-hard as well. Then Gavalas *et al.* [16] have proven the AOP to be NP-hard by transforming OP to an instance of the AOP. Now we will prove that our definition of the Quality-AOP is NP-hard as well.

Theorem 1. *The Quality-AOP is NP-hard.*

Proof. Given an arbitrary instance of the partition problem, consisting of a multiset $\mathcal{S}$ with n positive integers, and given an instance of the Quality-AOP ($G = (V, E), s, d, B_{min}, B_{max}$) containing a directed graph G with $n+1$ vertices V, $2n$ arcs E, a source vertex $s \in V$ and a destination vertex $d \in V$, and two positive integers for a bounded budget $[B_{min}, B_{max}]$, and in which graph's arcs and vertices form a double linked sequence such that ($\forall i \in 1, 2, .., n : e_i \in E$ connects $v_i \in V$ with $v_{i+1} \in V$ and has score $\sigma_{e_i} = 0$ and cost $c_{e_i} = 0$) and ($\forall i \in 1, 2, .., n : e_{n+i} \in E$ connects $v_i \in V$ with $v_{i+1} \in V$ and has score $\sigma_{e_i} = \mathcal{S}_i$ and cost $c_{e_i} = \mathcal{S}_i$), and in which $s = v_1$ and $d = v_{n+1}$, and $B_{min} = B_{max} = \frac{\sum \mathcal{S}}{2}$, then for a path P of a solution to the Quality-AOP the following two sets $\mathcal{S}_a = \{\sigma_e \mid e \in P\}$ and $\mathcal{S}_b = \{\sigma_e \mid e \in E \setminus P\}$ are the subsets of partition in which $\sum \mathcal{S}_a = \sum \mathcal{S}_b$. This has proven that the partition problem, which is NP-complete, can be reduced to Quality-AOP and therefore Quality-AOP is NP-hard.



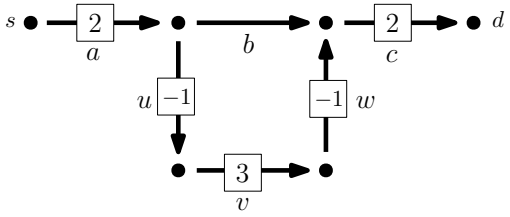
□

4.1.3 Example showing that Quality-AOP ≠ AOP

One could think that our definition of the Quality-AOP would be the same as the AOP if negative scores are used for non-attractive arcs, because then it would not generate a detour that passes through non-attractive arcs in order to collect a small amount of score. Still, the following could be happen, see Observation 1.

Observation 1. *The AOP that will also allow negative scores on arcs does not always give the same results as the Quality-AOP.*

Consider the following graph with either a positive or negative score assigned to an arc. Each arc costs 1 to traverse. A path between s and d that traverses arcs $\langle a, b, c\rangle$, has cost 3 and score 4. In the AOP with a budget of 5 it finds a path $\langle a, u, v, w, c\rangle$ with cost 5 and score 5. The quality of that path (score divided by costs) is $\frac{5}{5}$, which is worse than the quality of the initial path which was $\frac{4}{3}$. Despite of the presence of negative scores in the detour $\langle a, u, v, w, c\rangle$, in the AOP still a path can be found with a small profit of 1 that increases the total score but lowers the quality.

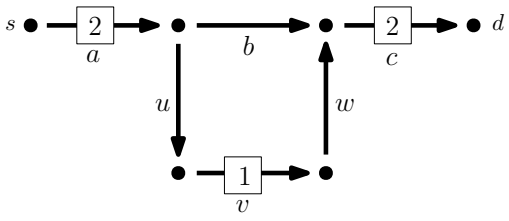


Analogous to Observation 1 we could see that local improvements do not imply the improvement of the route as a whole, see Observation 2. This means that we must be careful when solving the Quality-AOP so that we not search for local improvements.

Observation 2. *An quality improvement of a local part of a path does not imply the improvement of the quality of the path as a whole.*

Consider the following graph with scores assigned to arcs. Each arc costs 1 to traverse. A path between s and d that traverses arcs $\langle a, b, c\rangle$, has cost 3 and score 4. The part of the path that consists of arc b can be replaced by a segment $\langle u, v, w\rangle$. The quality of that new part (score divided by costs) is $\frac{1}{3}$, which is an improvement above the previous part $\frac{0}{1}$. Still, the path $\langle a, u, v, w, c\rangle$ as whole has quality $\frac{5}{5}$ which is not better than the

initial path $\langle a, b, c \rangle$ with quality $\frac{4}{3}$.



4.2 Quality Orienteering Problem (QOP)

Our Quality-AOP definition already allows us to compute interesting scenic routes, but requires the scenic scores to be attached to arcs. Although, there exist properties that contribute to the scenicness of a route but whose scenicness does not depend on individual arcs. Some of these need to be measured globally. We generalise the problem definition by using a global score function instead of summing scores of arcs. This function will return, in the context of the graph, a score for a path that is given as input to this function.

Definition 4.2 (QOP)**.** *Given a directed graph $G = (V, E)$ and a source vertex s and a destination vertex d, further given a total travel cost budget B_{max} and a minimum budget B_{min}, and given a f-algebra F^G_{score} for score in the context of G, the catamorphism $f^G_{score} : P \mapsto \mathbb{R}_{\geq 0}$ maps a path P to a score, and the **quality orienteering problem (QOP)** is finding a path $P = \langle e_1, e_2, ..., e_k \rangle$ from s to d, in which an arc is not traversed multiple times, nor traverses an arc another time in opposite direction ($\forall e \in P : \exists \bar{e} : \bar{e} \notin P$), and such that the total travel cost is at least B_{min} but not more than the cost budget B_{max}, and the average collected score is maximized, i.e.*

Maximize:

$$\frac{f^G_{score}(P)}{\sum_{e \in P} e.cost} \tag{1}$$

Subject to:

$$B_{min} \leq \sum_{e \in P} e.cost \leq B_{max} \tag{2}$$

Provided:

$$f^G_{score} : P \mapsto \mathbb{R}_{\geq 0} \tag{3}$$

4.2.1 Examples of the QOP

The Quality Orienteering Problem (QOP) allows us to determine the quality of a route in many different ways. Now for example we can attach scenic scores to vertices too. Also

we can measure the density of a route to avoid parts being close together, which can be desirable to prevent traveling back and forth on parallel roads that both are scenic. Another measurement can be counting the number of left and right turns or the distribution of turns along the route. This information can not been attached statically to arcs or vertices because it depends on the actual choice of orienteering. It is probably more interesting to sum the degrees of change in direction. Based on the user's preference, the QOP could find a route containing less or more degrees of turn. An example of such score function is:

$$f^G_{score}(P) = \sum_{i=1}^{|P|-1} \theta(e_i, e_{i+1}) \quad \text{with } e \in P$$
$$\text{where } \theta(e_1, e_2) = \textbf{let } \hat{v}_1 = \frac{e_1}{|e_1|}, \hat{v}_2 = \frac{e_2}{|e_2|} \textbf{ in } cos^{-1}(\hat{v}_1 \bullet \hat{v}_2)$$

which sums the degree of change in direction between successive arcs by calculating the angle based on the dot product between the unit vectors of these arcs. The more change in direction, the higher the score of the route.

5 Solving the Quality-AOP

Due to the difficulty of the problem it can be time-consuming to find the optimal answer. Therefore in practice heuristic approaches are preferred to find a solution in reasonable time that is close to optimal.

In Section 5.1 we will first give an exact solution and in Section 5.2 we will discuss heuristics that can be used to quickly find solutions to the Quality-AOP that are close to optimal. In Section 6 we actually give an algorithm that uses heuristics to find solutions to the Quality-AOP.

5.1 Mathematical formulation

The AOP can be formulated as a 0-1 integer linear program, which is done by Souffriau *et al.* [33]. They defined a more restrictive integer program compared to Definition 3.4, because they do not allow vertices to be visited multiple times. We give a mathematical formulation for the Quality-AOP in the form of a mixed-integer linear fractional program in Definition 5.1, based on Vansteenwegen *et al.* [36], Souffriau *et al.* [33], and Verbeeck *et al.* [37].

Definition 5.1 (Quality-AOP Integer Program)**.** *Let $G = (V, E)$ be a directed graph. Each arc has as cost c_e, a score σ_e, and a twin arc $\bar{e}$ directing in opposite direction. $\delta^-(S)$ represents the set of outgoing arcs from vertices in S to vertices outside S, i.e. $\{(v_1, v_2) \in E \mid v_1 \in S, v_2 \in V \backslash S\}$, and $\delta^+(S)$ represents the set of incoming arcs from vertices outside S to a vertex inside S, i.e. $\{(v_1, v_2) \in E \mid v_1 \in V \backslash S, v_2 \in S\}$, and let $\delta^-(v)$ and $\delta^+(v)$ be a short notation for $\delta^-(\{v\})$ and $\delta^+(\{v\})$ respectively that wraps v into a singleton set. The start vertex s and destination vertex d should be included in the solution. Further the travel cost is bounded by a minimum and maximum cost*

budget, denoted as B_{min} and B_{max}.

The five decision variables are x_e (which is 1 if the arc e is part of the solution and 0 otherwise), z_v (the number of times vertex v is visited), s_v (which is 1 if the path starts at vertex v and 0 otherwise), d_v (which is 1 if vertex v is the destination and 0 otherwise), and λ_v (is 1 if v is both the start and destination vertex, otherwise 0). Then:

Maximize:

$$\frac{\sum_{e \in E} \sigma_e x_e}{\sum_{e \in E} c_e x_e} \tag{1}$$

Subject to:

$$B_{min} \leq \sum_{e \in E} c_e x_e \leq B_{max} \tag{2}$$

$$\sum_{e \in \delta^-(s)} x_e \geq 1 \tag{3}$$

$$\sum_{e \in \delta^+(d)} x_e \geq 1 \tag{4}$$

$$s_v - d_v + \sum_{e \in \delta^+(v)} x_e - \sum_{e \in \delta^-(v)} x_e = 0 \qquad \forall v \in V \tag{5}$$

$$(1 - \lambda_v) s_v + \sum_{e \in \delta^+(v)} x_e = z_v \qquad \forall v \in V \tag{6}$$

$$\sum_{e \in \delta^-(S)} x_e \geq \frac{\sum_{v \in S} z_v}{\sum_{v \in S} |\delta^-(v)|} \qquad \forall S \subseteq V \backslash \{s\} \tag{7}$$

$$x_e + x_{\bar{e}} \leq 1 \qquad \forall e \in E : \exists \bar{e} \in E \tag{8}$$

$$s_v d_v = \lambda_v \qquad \forall v \in V \tag{9}$$

$$x_e, s_v, d_v, \lambda_v \in \{0, 1\} \tag{10}$$

The objective function (1) will maximize the quality of the route, i.e. the route's average score based on the total amount of collected scores in proportion to the travel cost. The constraint (2) bounds the travel costs to the specified budget interval $[B_{min}, B_{max}]$. In (3) and (4) is stated that the source and destination vertex should be part of the solution. The constraint (5) let be the number of incoming arcs equal to the number of outgoing arcs of a vertex, taking into account that the start and end vertices may or may not be the same. Constraint (6) sets the number of times a vertex is visited. Constraint (7) by Dantzig *et al.* [10] ensures subtour elimination which is preferable over Miller *et al.* [27] because we can not order vertices due to the multiple visits relaxation, stated by Verbeeck *et al.* [37]. In constraint (8) we ensure that the same arc can not be traveled another time in opposite direction. In (9) the λ_v variable is set and in (10) we bound integers to be 0-1.

The presented mixed-integer linear fractional program can be rewritten to a mixed-integer linear program using the Charnes-Cooper transformation specified in Charnes and Cooper [8]. This transformation gets rid of the ratios in the objective function and constraints. This can be done only under the restriction that the denominator is strictly positive, which is the case in the objective function because we define the costs of an arc to be strictly

positive (see Definition 4.1). Also in the denominator of constraint (7) there will be always at least one incoming arc. So the Charnes-Cooper transformation can be applied to get a valid integer linear program for this problem.

5.2 Metaheuristics

Beside the mathematical specification that solves the theoretical version of the problem, in practice heuristic approaches are required to compute solutions fast. There is a wide range of metaheuristics available (see Figure 8) to find solutions that are close to optimal. Each metaheuristic has its own characteristics and one will fit certain kind of problems better than others.

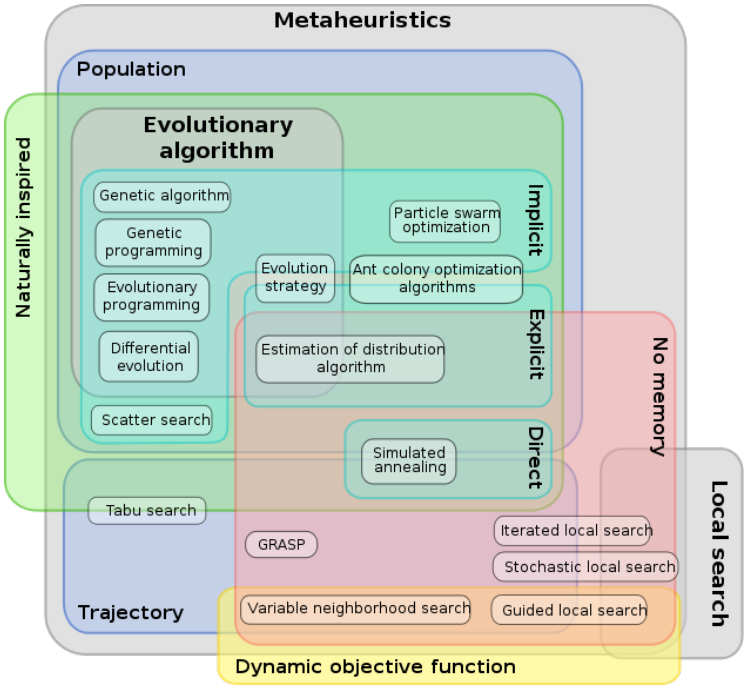


Figure 8: Overview of metaheuristics
By Johann "nojhan" Dréo, Caner Candan, via Wikimedia Commons[1]

Multiple objectives Although we have to deal with multiple objectives, and in the research area of multi-objective optimization evolutionary algorithms are used often, we have already explained that we can reduce the essence of the problem to two objectives. The two objectives travel cost and route quality based on the collected score is what we want to minimize and maximize respectively. Furthermore we also mentioned that the travel costs have additional constraints to stay within a specified budget, therefore evolutionary algorithms are not necessarily the first choice to use for this problem.

[1] https://commons.wikimedia.org/wiki/File%3AMetaheuristics_classification.svg

ILS and GRASP Methods that are used in literature to find solutions to the Arc Orienteering Problem [33] are Greedy Randomized Adaptive Search Procedure (GRASP) [33] and Iterated Local Search (ILS) [23]. Souffriau *et al.* [33] has introduced the AOP together with an algorithm that uses GRASP. Later Lu and Shahabi [23] introduced an algorithm that quickly finds solutions to the AOP that are close to optimal even on a large scale road network, and their algorithm is based on the ILS framework. The choice to use ILS is motivated by the fact that ILS has a faster iteration cycle and therefore performs more iterations within the restricted time limit. Furthermore Verbeeck *et al.* [37] motivated the choice for ILS as follows: "The choice for this framework was motivated by the fact that generally very complex problems require simple solution frameworks and ILS has proven to be successful for other variants of the node orienteering problem but was not used on the arc-variant yet".

Motivation to use GRASP We shall use an expensive bidirectional search [12] to avoid traversing arcs twice. To compensate the expensive call we want to utilize the knowledge from the search to make a greedy choice, but also we have to prevent getting stuck in a local optimum and therefore we use a semi-greedy heuristic [34] to make a random greedy choice among a set of high quality routes which are available after performing the bidirectional search, and make as much progress as possible. This is the reason why we use GRASP. If we would use ILS, then more iterations are required to escape from local optima and more expensive bidirectional searches should be performed. Furthermore not all local search methods could be applied, because quality should be measured globally (see Observation 2), so it must take into account the route as a whole.

6 An algorithm for the Quality-AOP

We will present a new algorithm in Section 6.1 called scenicRoute that can solve both the Arc Orienteering Problem (AOP) [33] as well as our reformulated Quality Arc Orienteering Problem (Quality-AOP) which we have introduced in Section 4.1. Which of the two problems the algorithm will solve (the AOP or the Quality-AOP) depend on the algorithm type argument $\mathcal{Q}$ that is given to the input of the algorithm, which can be either SUM or AVG. The quality measure of the AOP will be used if $\mathcal{Q} = \text{SUM}$ and the quality measure of the Quality-AOP will be used if $\mathcal{Q} = \text{AVG}$. To determine the quality of the route we will sum the scores of individual arcs in the path (i.e. $\sum_{e \in P} e.score$) if $\mathcal{Q} = \text{SUM}$, and we look at the average score along the path (i.e. $\frac{\sum_{e \in P} e.score}{\sum_{e \in P} e.cost}$) if $\mathcal{Q} = \text{AVG}$. The reason to present a single algorithm that can solve both problems is that we can easily point out steps in the algorithm that depend on the quality measure, while the other steps are the same for both problem definitions. Using the same algorithm for both lets us fairly compare the two problems in our experiment (Section 7). Besides the new quality measure the algorithm should be designed to handle additional constraints such as avoiding to traverse arcs twice or traverse an arc another time in opposite direction. Furthermore the starting and ending vertices may or may not be the same vertex. Design choices are explained in more detail in Section 6.3.

The algorithm scenicRoute is loosely based on the algorithm presented by Lu and Shahabi [23]. Instead of making it fully based on the Iterated Local Search (ILS) framework as in Lu and Shahabi [23], we will also involve the Greedy Randomized Adaptive Search Procedure (GRASP) that is presented in an earlier paper by Souffriau *et al.* [33] to make

semi greedy choices among a set of candidate routes to make as much progress as possible in a single iteration. Where Lu and Shahabi [23] only take into account a cost measure based on travel distance because their speed up methods use Euclidean distance heuristics, our goal is to make cost measures interchangeable and do not let it depend on the algorithm. For example we can easily swap travel distance for travel time.

An example of runing the algorithm is given in Section 6.2.

6.1 The algorithm

First, we show a textual and visual outline of the algorithm scenicRoute. Then the pseudocode of the algorithm is shown in Algorithm 6.1. The algorithm makes use of two subprocedures generatePath and shortestPaths, whose pseudocode is given in Algorithm 6.2 and 6.3. With the code, a detailed description is given line by line. In Section 6.2 an example is given of running the algorithm.

Outline The outline of scenicRoute along with a visualisation in Figure 9.

1. Calculate an initial path P between source s and destination d that stays within cost budget B.
2. Randomly partition the path into three path segments s_1, s_2, s_3 and remove the arcs s_2 from P.
3. Calculate a new path (generatePath) between the end of s_1 and the beginning of s_3 that should replace the path s_2 in P. It uses bidirectional search (shortestPaths) to find shortest paths between s_1 and s_3 that travels through an arc from the candidate arc set (CAS). The CAS is determined by budget B.
4. We greedily select an arc e from CAS whose path is in the bidirectional search and has a quality that is at least the average quality of all paths found.
5. To improve the amount of progress made in a single iteration, one level of recursion is performed. Call generatePath to find a path that replaces the first part (s_1 to e) and another call to replace the second part (e to s_2) in which steps 3 and 4 are repeated.
6. After replacing s_2 with the new path from steps 3-5, a new path P is obtained. Then repeat from perturbation (step 2) until it exceeds the maximum number of iterations. Then return P.

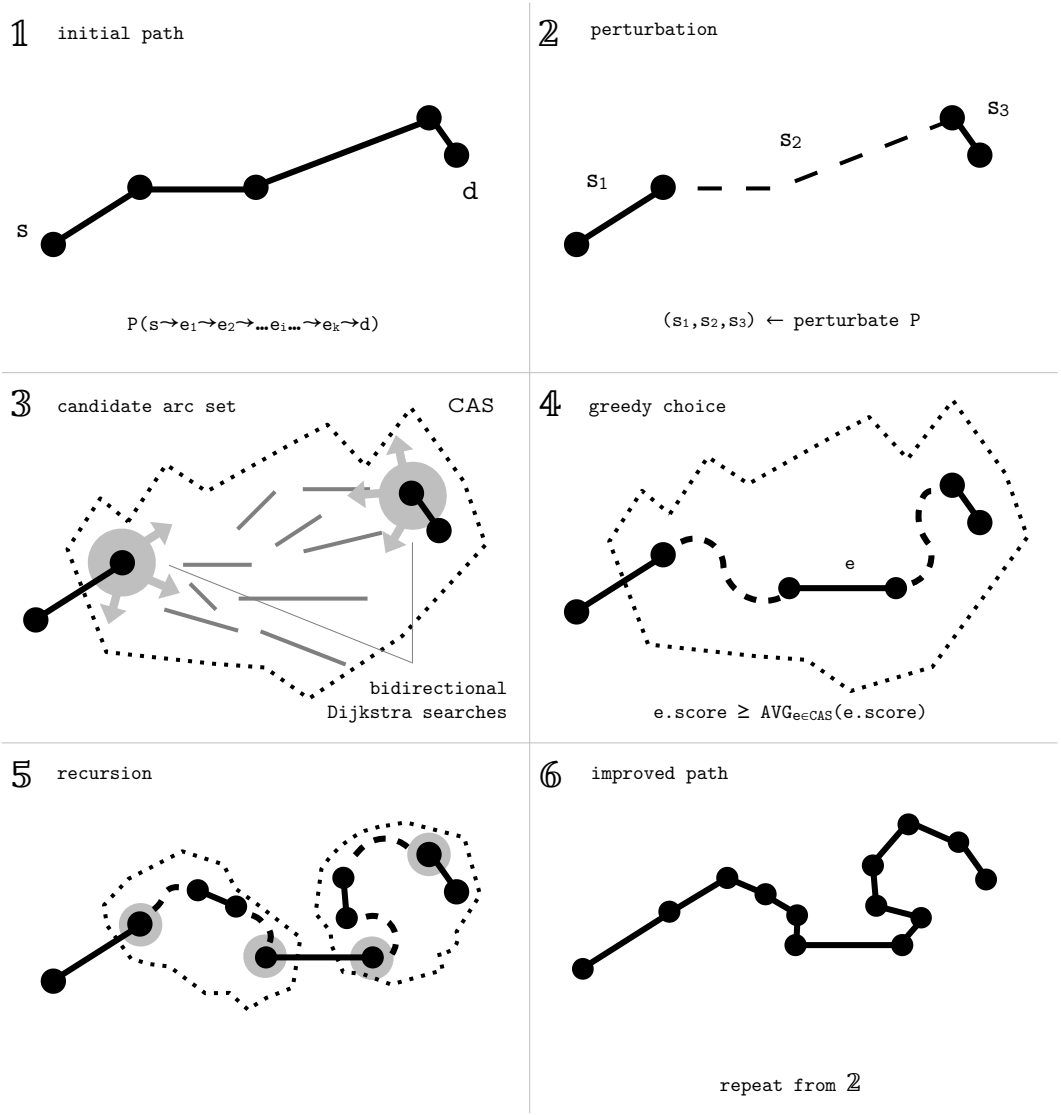


Figure 9: Outline of how algorithm **ScenicRoute** solves the Quality-AOP.

Below the algorithm scenicRoute is given followed by a step by step explanation.

Algorithm 6.1 ScenicRoute

Input: graph $G = (V, E)$, source vertex s, destination vertex d, minimum budget B_{min}, maximum budget B_{max}, limited number of iterations I, quality measure $\mathcal{Q} \in \{\text{SUM}, \text{AVG}\}$.

Output: Non-empty sequence of arcs $\langle e_1, e_2, .., e_k \rangle$ if a path exists.

1: **function** scenicRoute($G, s, d, B_{min}, B_{max}, I, \mathcal{Q}$)
2: **if** $P \leftarrow$ generatePath(G, s, d, B_{max}) **then**
3: **for** $i \in 1, 2, .., I$ **do**
4: $k \leftarrow |P|$
5: $r_1 \leftarrow$ random number from $[1, k]$
6: $r_2 \leftarrow$ random number from $[1, k]$
7: $i \leftarrow \text{MIN}(r_1, r_2)$
8: $j \leftarrow \text{MAX}(r_1, r_2)$
9: $(s_1, s_2, s_3) \leftarrow$ split P at vertex $v_i \in P$ and vertex $v_{j+1} \in P$
10: $B_{rest} \leftarrow B_{max} - s_1.cost - s_3.cost$
11: $G' \leftarrow G \setminus (\bigcup \{s_1, \bar{s}_1, s_3, \bar{s}_3\})$
12: **if** $s_2' \leftarrow$ generatePath($G', v_i, v_{j+1}, B_{rest}, \mathcal{Q}$) for $v \in P$ **then**
13: $P' \leftarrow s_1 +\!\!+ s_2' +\!\!+ s_3$
14: $accept \leftarrow \begin{cases} P'.score > P.score & \textbf{if } \mathcal{Q} = \text{SUM} \vee P.cost < B_{min} \\ \frac{P'.score}{P'.cost} > \frac{P.score}{P.cost} & \textbf{if } \mathcal{Q} = \text{AVG} \wedge P'.cost \geq B_{min} \\ \bot & \textbf{otherwise} \end{cases}$
15: **if** *accept* **then**
16: $P \leftarrow P'$
17: **end if**
18: **end if**
19: **end for**
20: **end if**
21: **return** P
22: **end function**

(line 2) First we do a call to the subprocedure generatePath to calculate an initial path P between source vertex s and destination vertex d, which can have any costs between $0 < P.cost \leq B_{max}$ based on the greedy choice. If not such path exists we directly return without an answer to the problem.

(line 3) Then a loop is started, whose body will be executed I times: the maximum number of iteration provided to the input of the algorithm.

(line 4) Determine the length of P and call it k, which is the number of arcs in the path.

(line 5-6) Generate two random numbers r_1 and r_2 in the interval $[1, k]$ (including 1 and k) which corresponds with an arc from P, namely $e_{r_1} \in P$ and $e_{r_2} \in P$. They will be used later on to split the path at a vertex location between 1 and $k + 1$.

(line 7-8) Because further on we want to split the path at two locations in order to extract a path segment s_2, we order r_1 and r_2 and calling them i and j such that $i \leq j$. Now the first split on the cutting location i comes before the second cutting location that depends on j. The reason to explicitly order the random numbers is to ensure that s_2 will contain at least one arc. Furthermore, arcs in the middle of the path are more likely the be chosen to be extracted, which is preferable because the search space is larger there.

(line 9) Now we have determined two numbers i and j to indicate at which arc to split, namely at arc $e_i \in P$ and $e_j \in P$. For i we split at the beginning of arc e_i (vertex $v_i \in P$) and for j we split at the end of arc e_j (which is vertex $v_{j+1} \in P$). After splitting P at these two locations we obtain three path segments s_1, s_2, s_3. In the case $i = j$, we split at the beginning and end of the same arc e_i, resulting in the extracting of the path segment s_2 that only contains a single arc. In the case $i = 0$ the segment s_1 contains no arcs, like as s_3 will not containing any arcs if $j = k$.

(line 10) We calculate the remaining budget that may be used to find a path that connects s_1 and s_3. The budget is based on the maximum budget minus the costs of the paths s_1 and s_3 because they are part of the solution already at this moment.

(line 11) Omit arcs and twin arcs from the graph before passing it on to GENERATEPATH, to ensure the arcs that are part of the solution will not be used in further searches.

(line 12) Find a path that connects s_1 and s_3. The cost of this path together with the cost of s_1 and the cost of s_3 may not exceed B_{max}. If not such path is found we can not improve P and therefore directly continue with another iteration.

(line 13) Connect path segments s_1 and s_3 together by putting the new path s_2' in between, which gives us a new path P' containing the arcs in sequence (i.e. $P' = \bigcup \{s_1, s_2', s_3\}$).

(line 14) We determine if the new path P' is better than P and would be accepted as replacement for P. In the case the quality measure $\mathcal{Q} = SUM$, the scores of the arcs in P' are summed and must improve upon the score of P. For the quality measure $\mathcal{Q} = AVG$, we use the same method as SUM as long the minimum budget is not reached yet. Once above the minimum budget we ensure it may not drop to below this minimum again. In AVG we only accept the new path P' if its average score improves upon P.

(line 15-16) If the quality of the new path P' is higher than the quality of the previous path P and we accept the replacement, replace the path P by P' and move on to the next iteration. If no improvement is found we will also move on to the next iteration with the same P as at the beginning of the current iteration, but because new random values will be chosen it will possibly find improvement next.

Below the algorithm GENERATEPATH is given followed by a step by step explanation.

Algorithm 6.2 GeneratePath

Input: graph $G = (V, E)$, source vertex s, destination vertex d, cost budget B, flag $R \in \{\top, \bot\}$ indicating (true or false) whether or not to recurse (default set to $\top$).
Output: Non-empty sequence of arcs $\langle e_1, e_2, .., e_k \rangle$ if a path exists.

1: **function** GENERATEPATH(G, s, d, B, R)
2: $P \leftarrow \varnothing$
3: $\mathbb{S} \leftarrow$ SHORTESTPATHS(G, s, d, B)
4: $CAS \leftarrow \{(e, P) \mid \forall e \in E, \exists P(s \rightsquigarrow e \rightsquigarrow d) \in \mathbb{S}, P.cost \leq B\}$
5: **if** $CAS \neq \emptyset$ **then**
6: $avg \leftarrow$ AVERAGE($\{P.quality \mid (e, P) \in CAS\}$) where $P.quality = \frac{P.score}{P.cost}$
7: $CAS \leftarrow \{(e, P) \mid (e, P) \in CAS, P.quality \geq avg\}$ where $P.quality = \frac{P.score}{P.cost}$
8: $(e, P) \leftarrow$ randomly choose an element from CAS
9: $i \leftarrow$ the position of arc e in P
10: $(s_1, s_2, s_3) \leftarrow$ split P at $v_i \in P$ and $v_{i+1} \in P$.
11: **if** R **then**
12: $b_1 \leftarrow (B - e.cost) \cdot \frac{s_1.cost}{P.cost - e.cost}$
13: $G'_1 \leftarrow G \setminus (\bigcup \{s_2, \bar{s}_2, s_3, \bar{s}_3\})$
14: **if** $s'_1 \leftarrow$ GENERATEPATH($G'_1, s, v_i, b_1, \bot$) **then**
15: $s_1 \leftarrow s'_1$
16: **end if**
17: $b_3 \leftarrow (B - e.cost) \cdot \frac{s_3.cost}{P.cost - e.cost}$
18: $G'_3 \leftarrow G \setminus (\bigcup \{s_1, \bar{s}_1, s_2, \bar{s}_2\})$
19: **if** $s'_3 \leftarrow$ GENERATEPATH($G'_3, v_{i+1}, d, b_3, \bot$) **then**
20: $s_3 \leftarrow s'_3$
21: **end if**
22: **end if**
23: $P \leftarrow s_1 +\!\!+ s_2 +\!\!+ s_3$
24: **end if**
25: **return** P
26: **end function**

(line 3) For the current graph $G = (V, E)$ and cost budget B we make a call to SHORTESTPATHS which will return a set of paths $\mathbb{S}$ containing the shortest paths from source vertex s to destination vertex d that traverses through specific arcs e_i that are connecting the forward search from s to the backward search from d. It is ensured that these paths contain arcs at most once, i.e. for a path $p(s \overset{s_1}{\rightsquigarrow} e_i \overset{s_3}{\rightsquigarrow} d)$ in $\mathbb{S}$ is ensured $s_1 \cap s_3 = \emptyset$.

(line 4) We will use $\mathbb{S}$ to query all paths from s to d that go through some arc e from E and do not have arcs twice in the path, if such path exists. Only paths that are within cost budget B would be considered as a candidate path. This gives the candidate arc set CAS that besides arcs also contains a reference to the path containing the arc.

(line 5) Only if CAS is non-empty, there is a path from s to d.

(line 6) For all candidate paths in CAS we determine the quality and then determine the average quality of all paths in this set.

(line 7) We will now reduce the candidate arc set CAS by filtering the arcs with high quality. Arcs that have a higher quality than the average quality avg are considered as high quality arcs. This will help us making a greedy choice. To use the terminology of the Greedy Randomined Adaptive Search Procedure (GRASP), this set is considered to be the restricted candidate list (RCL) [15].

(line 8) The greedy choice is made by randomly selecting an arc from the restricted candidate list.

(line 9-10) Now an arc e has been chosen as part of a shortest path $P(s \overset{s_1}{\rightsquigarrow} e \overset{s_3}{\rightsquigarrow} d)$, we can start preparing a recursion step to replace the shortest path to arc e (segment s_1) and the shortest path from e (segment s_3) with a greedy path. Call GENERATEPATH to generate a path from source s to the beginning of e, and call it another time to generate a path from the end of e to our destination d. This means that we are replacing s_1 and s_3 in P by a greedy chosen path. To make this possible we will split P at the beginning and end of edge e to retrieve three path segments s_1, s_2, s_3.

(line 11) Only if the recursion flag R is true ($\top$) we actually go into recursion. If it is false ($\bot$) the path P will be returned directly, which has already a higher quality than the average quality of the paths initially in CAS.

(line 12 and 17) For the recursive call to GENERATEPATH we must determine how much cost budget is remaining and how much we give to both recursive calls. The remaining budget consists of the initial budget B minus the budget of the arc e, because that arc is already part of the solution path P. We split the remaining budget based on the proportion of the costs between the currently found paths s_1 and s_3, thus if for example s_1 has twice the costs of s_3 it will also get twice as much of the remaining budget.

(line 13 and 18) Omit arcs and twin arcs from the graph before passing it on to GENERATEPATH, to ensure the arcs that are part of the solution will not be used in further searches.

(line 14 and 19) In the recursive call to GENERATEPATH we could find a path s_1' (respectively s_3') that is of an equal or higher quality than the current path segments s_1 (respectively s_3).

(line 15 and 20) If in the recursive call such path is found, we will replace s_1 (respectively s_3) by the new path s_1' (respectively s_3'), otherwise we ignore the result of the recursive call.

(line 23) We connect the three path segments together (i.e. $\bigcup\{s_1, s_2, s_3\}$) to obtain a (new) path P.

Below the algorithm SHORTESTPATHS is given followed by a step by step explanation.

Algorithm 6.3 ShortestPaths

Input: graph $G = (V, E)$, source vertex s, destination vertex d, cost budget B.

Output: a possibly empty set $\mathbb{S}$ of paths $\{P \mid \forall e \in E, \exists P(s \overset{s_1}{\rightsquigarrow} e \overset{s_3}{\rightsquigarrow} d), s_1 \cap s_3 = \emptyset\}$.

1: **function** SHORTESTPATHS(G, s, d, B)
2: $\overrightarrow{G} \leftarrow G'(\emptyset, E)$ ▷ forward graph
3: $\overleftarrow{G} \leftarrow G'(\emptyset, E)^T$ ▷ backward graph
4: Let s be part of $\overrightarrow{G}$ with $s.cost \leftarrow 0$ and $s.\pi \leftarrow \varnothing$
5: Let d be part of $\overleftarrow{G}$ with $d.cost \leftarrow 0$ and $d.\pi \leftarrow \varnothing$
6: $Q \leftarrow$ PRIORITYQUEUE(sorted: $\lambda uv \mapsto u.cost < v.cost$)
7: ENQUEUE(Q, s), ENQUEUE(Q, d)
8: **while** $u \leftarrow$ DEQUEUE(Q) **do if** $u.cost < \frac{1}{2}B$ **then** ▷ bidirectional Dijkstra
9: $(G_\bullet, G_\circ) \leftarrow \begin{cases} (\overrightarrow{G}, \overleftarrow{G}) & \text{if } u \in \overrightarrow{G} \\ (\overleftarrow{G}, \overrightarrow{G}) & \text{otherwise} \end{cases}$
10: **for all** $(u, v) \leftarrow e \in \{(u, v) \mid (u, v) \in \delta^-(u), v \notin G_\circ\}$ **do** ▷ inspect outgoing arcs
11: **if** $v \notin G_\bullet \vee u.cost + e.cost < v.cost$ **then**
12: $v.cost \leftarrow u.cost + e.cost$
13: $v.\pi \leftarrow e$
14: Let v be part of $G_\bullet$
15: ENQUEUE(Q, v)
16: **end if**
17: **end for**
18: **end while**
19: $\mathbb{S} \leftarrow \emptyset$ ▷ build result set
20: **for all** $(u, v) \leftarrow e \in \{(u, v) \mid (u, v) \in E, u \in \overrightarrow{G} \wedge v \in \overleftarrow{G}\}$ **do**
21: $s_1 \leftarrow$ reconstruct path by following the chain from $u.\pi$ to s
22: $s_3 \leftarrow$ reconstruct path (in reverse) by following the chain from $v.\pi$ to d
23: $\mathbb{S} \leftarrow \mathbb{S} \cup \{P(s \overset{s_1}{\rightsquigarrow} e \overset{s_3}{\rightsquigarrow} d)\}$
24: **end for**
25: **return** $\mathbb{S}$
26: **end function**

(line 2) Creates $\overrightarrow{G}$ that will be used in the forward Dijkstra search, which is just a copy of the graph G. This data structure maintains the least cost path (from s) to every vertex, only if such path is found by the algorithm. If a shorter path to a vertex will be found, the vertex will be updated with the details of the new least cost path. Initially all vertices get the value undefined ($\varnothing$). Each vertex would be updated only by either $\overrightarrow{G}$ or $\overleftarrow{G}$, depending on which one ($\overrightarrow{G}$ or $\overleftarrow{G}$) find the vertex first.

(line 3) Creates $\overleftarrow{G}$ with the same purpose as $\overrightarrow{G}$ (line 2) but with the difference that this data structure maintains the least cost paths to every vertex from destination vertex d. In this data structure all arcs are reversed so that it finds paths in a backward order.

(line 4-5) Initially assign the costs 0 to the start and end vertex to indicate that these vertices are discovered. Their least cost paths from s (respectively d) will be 0. For every vertex v we will refer to its previous vertex $v.\pi$ in the least cost path to v so that we can reconstruct the path later. Because s and d are the first vertices in the least cost paths from s (respectively d), they have no $v.\pi$.

(line 6-7) A priority queue Q will maintain the discovered vertices for both $\overrightarrow{G}$ and $\overleftarrow{G}$. They are ordered by the least costs, the same as what is done in Dijkstra. Initially fill the queue with s and d.

(line 8) Gets the least cost vertex u from Q. Because Q maintains vertices from both $\overrightarrow{G}$ and $\overleftarrow{G}$, the forward and backward searches go at once. Therefore if we exceeds half the budget $B/2$ from a search in one direction it could never find a path between s and d within B, so we can skip that iteration.

(line 9) The body of the while loop depends on the search direction of the element u from Q. We refer to that direction by $G_\bullet$ and the opposite search direction is referred by $G_\circ$.

(line 10) Look at the adjacent arcs from vertex u, only the outgoing arcs $\delta^-(u)$ are taken into account. Only proceed processing that arc if the vertex v at the end of the arc is not already found by the opposite search $G_\circ$.

(line 11) If vertex v at the end of the adjacent arc is newly discovered or this path to v has lower cost than a previously found path to v in $G_\bullet$, we need to update the least cost path.

(line 12-14) Updates the least cost path to vertex v.

(line 15) Add v to the queue such that the adjacent arcs from v can be discovered in one of the following iterations.

(line 19) Initiate an empty set of solution paths.

(line 20) Take into account the intersection set of $\overrightarrow{G}$ and $\overleftarrow{G}$ that is determined by the arcs that are connecting a path from the forward search $\overrightarrow{G}$ to a path from the backward search $\overleftarrow{G}$.

(line 21-23) Reconstruct the path between s and d and add this path to the set of solutions.

6.2 Example

We give an example of running the algorithm that will find an optimal path for a call to scenicRoute with parameters $B_{min} = 5, B_{max} = 15, I = 1, \mathcal{Q} = AVG$. The result can be viewed in the last figure of the current section (Figure 17).

The algorithm needs four steps to converge. We will discuss every step in more detail, but first we give a short summary. The first two steps are the initial steps. **Step 1** calls generatePath and **step 2** calls generatePath recursively on the front and tail of the result. Then the loop of scenicRoute is entered that starts with removing a part of the path at random. **Step 3** and **step 4** try to find a new path that replaces the removed segment, again by calling and recursively calling generatePath. At the end of iteration 1 an optimal path is found. Every call to generatePath starts with performing a call to shortestPaths to initiate a bidirectional search between source and destination that returns a set of paths $\mathbb{S}$ between source and destination that traverse an arc from the intersection $\overrightarrow{G} \cap \overleftarrow{G}$. To make a clear distinction between the bidirectional searches and the construction of paths, we have put the bidirectional searches in a light gray box in the example below.

step 1: initial path Initially generate a path between s and d by starting a forward search $\overrightarrow{G}$ from s and a backward search $\overleftarrow{G}$ from d (see Figure 10). The arcs in which both searches met each other are forming a path from s to d. In this example (Figure 11) we see the randomly chosen arc (u, v) which gives us the path $P(s \rightsquigarrow e_4 \rightsquigarrow d)$ with score $2+4 = 6$, costs 5, and quality $\frac{6}{5}$.

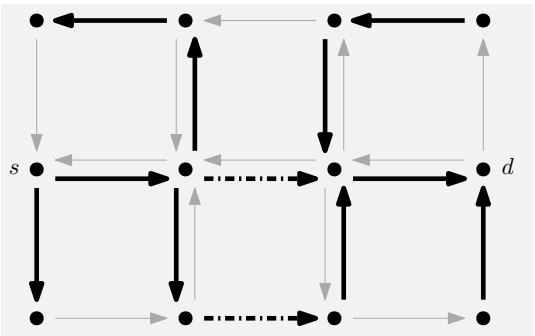


Figure 10: Searches $\overrightarrow{G}$ and $\overleftarrow{G}$ from s and d find two paths connecting $\overrightarrow{G}$ and $\overleftarrow{G}$ through intersection arcs (dashed).

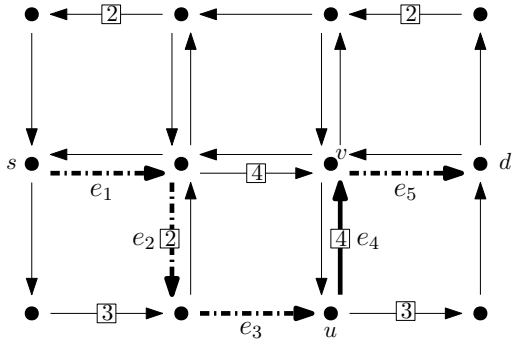
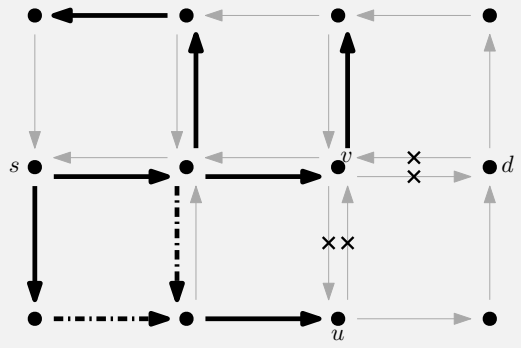
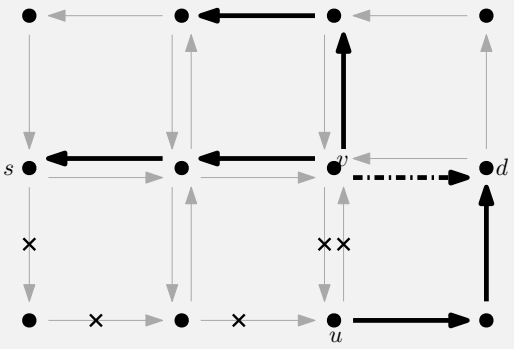


Figure 11: Initial path $P(s \rightsquigarrow e_4 \rightsquigarrow d)$ based on the chosen arc e_4 from $\overrightarrow{G} \cap \overleftarrow{G}$ which is part of a high quality path in $\mathbb{S}$ with score $2 + 4 = 6$, costs 5, and quality $\frac{6}{5}$.

step 2: initial path (recursion) Recurse on $p_1(s \rightsquigarrow u)$ and $p_2(v \rightsquigarrow d)$. In Figure 12 we can see that a few arcs and their twins are omitted in the bidirectional search, because these arcs are part of the solution. The budget is split and given to both searches. Each search gets a proportion of the budget that is determined by the cost ratio of p_1 and p_2. In Figure 13 we can see that p_1 is updated with $p_1'(s \rightsquigarrow e_2 \rightsquigarrow u)$ because it improves quality. For p_2 no improvement is found. The final initial path P has score $3 + 4 = 7$, cost 5, and quality $\frac{7}{5}$.



a: Search between s and u with budget $\frac{3}{4}(B-1)$. $\overrightarrow{G}$ and $\overleftarrow{G}$ proceed until half of the budget $\frac{3}{8}(B-1) = \frac{3}{8}(15-1) = 5\frac{1}{4}$ is used.



b: Search between v and d with budget $\frac{1}{4}(B-1)$. $\overrightarrow{G}$ and $\overleftarrow{G}$ proceed until half of the budget $\frac{1}{8}(B-1) = \frac{1}{8}(15-1) = 1\frac{3}{4}$ is used, so it stops early.

Figure 12: Two bidirectional searches. One between s and u, the other between v and d. Cross signs indicate the omitted arcs. Searches $\overrightarrow{G}$ and $\overleftarrow{G}$ can proceed until half of the budget is used.

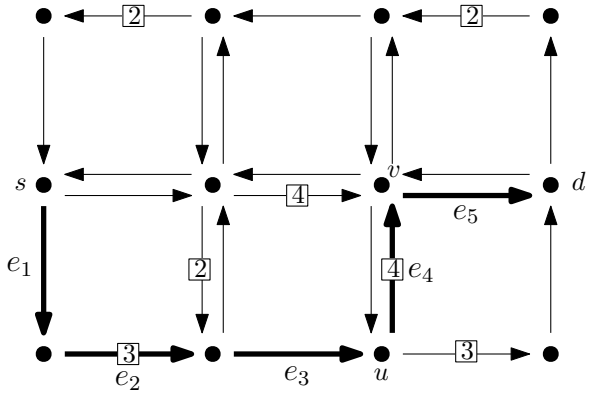


Figure 13: $p'_1(s \rightsquigarrow e_2 \rightsquigarrow u)$ replaces p_1, which gives us the path $P(s \rightsquigarrow e_2 \rightsquigarrow e_4 \rightsquigarrow d)$ with score $3+4=7$, cost 5, and quality $\frac{7}{5}$.

step 3: iteration 1 An iteration starts with perturbation. The path from the previous result is randomly split between arc e_2 and e_3 (call it vertex i) and at vertex d (call it $j+1$, to be conform to the notation used in the algorithm), which removes three arcs wherefore a new path will be found. In Figure 14 a new bidirectional search starts between i and $j+1$. Two arcs are omitted because they are part of the solution. In Figure 15 the chosen arc (u, v) gives us the path $P(s \rightsquigarrow e_2 \rightsquigarrow e_4 \rightsquigarrow d)$ with score $3+4=7$, costs 5, and quality $\frac{7}{5}$.

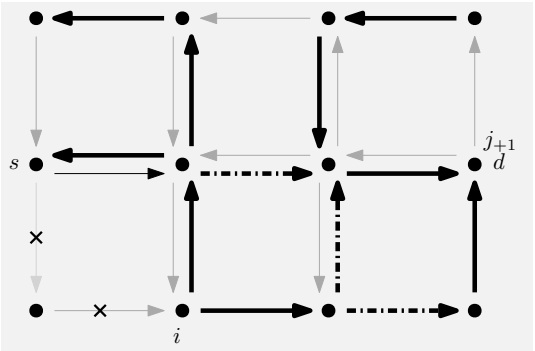


Figure 14: bidirectional search between split points i and $j+1$ with budget $B-2$ (excluding 2 arcs from solution).

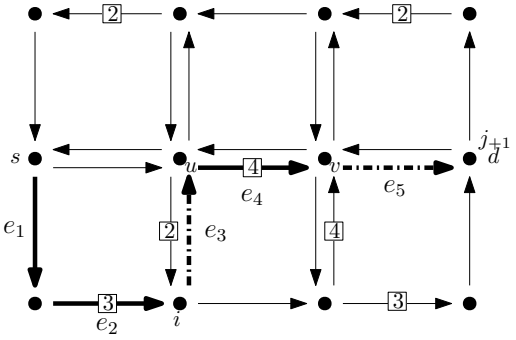
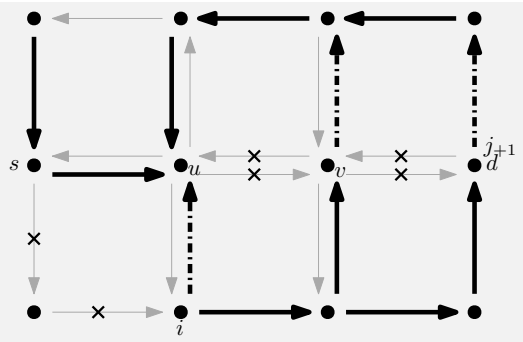
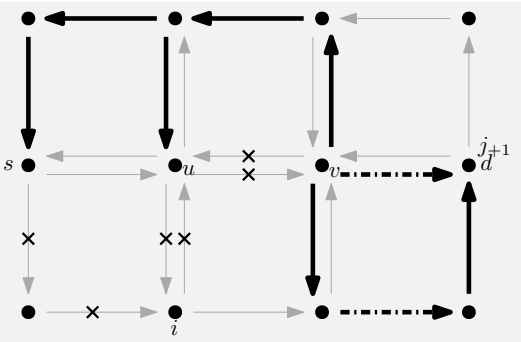


Figure 15: Path $p(i \rightsquigarrow e_4 \rightsquigarrow j+1)$ is based on the chosen arc e_4 from $\overrightarrow{G} \cap \overleftarrow{G}$ that is part of a high quality path in $\mathbb{S}$. Together with the existing solution gives us the path $P(s \rightsquigarrow e_2 \rightsquigarrow e_4 \rightsquigarrow d)$.

step 4: iteration 1 (recursion) Recurse on $p_1(i \rightsquigarrow u)$ and $p_2(v \rightsquigarrow j+1)$. In Figure 16 both bidirectional searches are performed with limited budget that is proportional to p_1 and p_2, and arcs that are already part of the solution are omitted. p_1 can not find any update, but p_2 can be updated with $p_2'(v \rightsquigarrow e_6 \rightsquigarrow j+1)$ that improves quality (see Figure 17). The final (and optimal) path P has score $3+4+3=10$, costs 7, and quality $\frac{10}{7}$.



a: Search between i and u with budget $\frac{1}{2}(B-3)$.



b: Search between v and $j+1$ with budget $\frac{1}{2}(B-3)$.

Figure 16: Two bidirectional searches. One between i and u, the other between v and $j+1$.

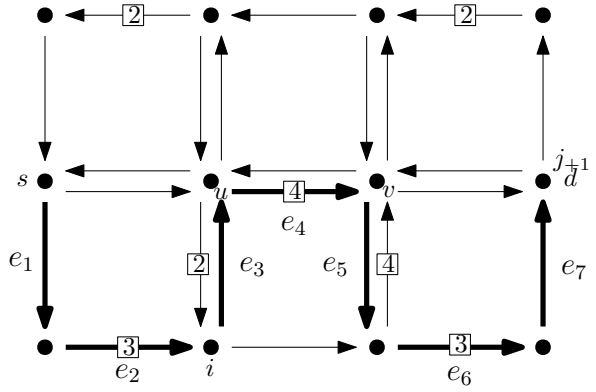


Figure 17: The final and optimal path $P(s \rightsquigarrow e_2 \rightsquigarrow e_4 \rightsquigarrow e_6 \rightsquigarrow d)$ with quality $\frac{10}{7}$ that is found by a call to **scenicRoute** with parameters $B_{min} = 5, B_{max} = 15, I = 1, \mathcal{Q} = AVG$. The algorithm performed four steps in total which were part of the initialisation and the first iteration.

6.3 Design choices

Minimum budget The minimum budget in the Quality-AOP ensures that we do not find high quality routes that are too short or whose travel time is too low. The algorithm scenicRoute with the average score as quality measure (i.e. $AVG \in \mathcal{Q}$), would only accept a route P' if its quality is better than the current route P. A problem arises, because the highest quality route could have travel costs lower than the minimum budget, which is most likely the case as we can see in our experiment (Figure 20). Therefore we modified the algorithm so it forces to update the route until it exceeds the minimum budget, even if the quality becomes lower. After it has exceeded the minimum budget we ensure it stays above this minimum.

Dijkstra's shortest path The reason to use a shortest path as part of the bidirectional search, is because in this way we have control over the amount of detour and the running time of the algorithm. A scenic path should balance scenicness and detour, therefore we use the shortest path to minimize detour, but add a small detour to traverse the greedily chosen high quality arc that improves the scenicness.

No subtours The AOP has the restriction that profit can not be collected twice from the same arc, which avoids subtours but not necessarily excludes all of them. In the Quality-AOP we use the stronger restriction that does not allow to traverse arcs twice, because from the traveler's perspective this is not desirable. Furthermore we also forbid to traverse an arc another time in opposite direction.

Our algorithm solves the Quality-AOP using Dijkstra's shortest path. Shortest paths will never contain cycles (see Cormen *et al.* [9]), but this is no longer garantueed if we connect shortest paths together. To ensure that a connection does not contain arcs multiple times we can think of two methods. The first method checks for duplicates at the moment such path is created from two shortest paths. The second option is to make sure beforehand that these two paths do not share arcs. The first case is rather simple to check, but comes with the expense of recalculation if a path contains duplicates. For the second method we must

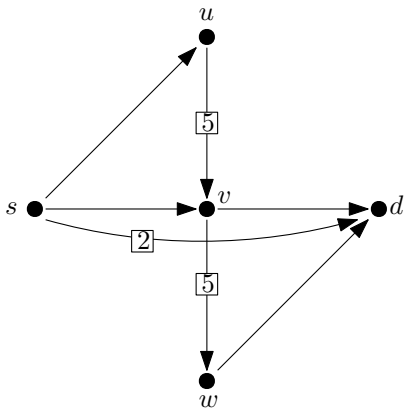
change the way how shortest paths are calculated. We choose the second method, where we calculate paths based on a bidirectional Dijkstra that searches from the start and end vertex simultaneously. Because we make sure that these searches do not cover the same vertices, it will also not cover the same arcs. Then we determine arcs that are bridges between the first and second search. Constructing a path between source and destination through such a bridge will never contain arcs and their twins twice in the path.

Thus our bidirectional search will not contain subtours, but if we combine the result of the bidirectional search with our existing solution, it possibly contains arcs multiple times. We decide to remove the arcs and twin arcs from the graph that are part of the solution, and then initiate a bidirectional search on this new graph.

Recursive path generation For each generated path we will go in recursion twice, one time by calling GENERATEPATH on the first half of the path and another recursive call to GENERATEPATH is performed on the second half. There are two reasons to include this recursion step. The first reason is to generate quickly an initial path. Paths generated based on shortest paths would require several iterations to make fully use of the budget. By adding a recursive step the remaining budget will be used directly to extend the path and ensures that we obtain the optimal path more quickly. The second reason is that it could find paths that otherwise would not be found because the recursive call constructs a path of multiple segments that together can improve the quality, see Observation 3

Observation 3. *Recursively calling* GENERATEPATH *could find higher quality routes between s and d than a single call could find.*

Consider the graph $G = (V, E)$ in which each arc $e \in E$ has cost 1. Arcs (u, v) and (v, w) have score 5 and (s, d) has score 2. There are 5 possible paths between s and d of which only the path $\langle s, u, v, w, d\rangle$ is not part of a shortest path $p_i(s \rightsquigarrow e_i \rightsquigarrow d)$ for $i = 1, 2, ..., |E|$. We find the highest quality of $\frac{2}{1}$ for a path $\langle s, d\rangle$, but higher quality could be obtained if we find the path $\langle s, u, v, w, d\rangle$ with quality $\frac{10}{4}$, by either recurse on the second half of path $p_1(s \rightsquigarrow (u, v) \rightsquigarrow d)$ or recurse on the first half of path $p_2(s \rightsquigarrow (v, w) \rightsquigarrow d)$.



6.4 Running time reduction

When doing scenic route planning on a real road network we want to use speed up techniques because there are many roads and many possible routes. In the first place we use a heuristic approach to reduce the number of routes that are taken into account. We also want to reduce the number of roads that we have to consider because when we find a path in a certain area it is not necessary to look at the whole graph that contains many arcs that are too far away to be part of the solution.

Ellipse property (distance) We can use the geometric aspects of a road network to exclude arcs that are out of range. For example, Lu and Shahabi [23] use the ellipse property. Consider an ellipse that includes the source and destination vertices and which size depends on the budget. More precisely, a line from the source vertex to a point on the ellipse and another line from the destination to that point, will have together a distance equal to the travel budget. Therefore, arcs that are intersecting or lie outside the ellipse can never be part of a solution because the ellipse property states that the costs to travel to such an arc and travel from that arc to the destination will exceed the budget. So we have to consider only the arcs within the ellipse specified by the available budget.

Ellipse property (time) Although the ellipse property can be used if spatial information is available and the travel costs are measured in terms of distance, it can not be used for the cost measure travel time. In order to make the speed up technique compatible with other cost measures we must perform a transformation. In the case of travel time budgets we can do a conservative transformation from time to distance on the basis of maximum speed allowed on roads (e.g. 130 km/h in The Netherlands). That means that when we have a budget of 60 minutes, the reach will be 130km maximally. Therefore our ellipse can be set based on 130km travel distance budget.

The reason to use a conservative transformation and therefore using the maximum speed on the road network is because we do not want to exclude arcs of our solution. In contrast, when we use a non-conservative transformation using average speed (e.g. 60km/h), the speed up will have more effect, excluding more arcs, but then we lose quality. Imagine a area with high quality attractive roads, but in order to get there you must travel 20 minutes over the motorway and another 20 minutes to return and there are 20 minutes left to travel the high quality arcs, there is a high chance the motorway arcs are getting pruned because they intersect the (relative small) ellipse of 60km, such that we will never reach the high quality area. Therefore we must always be conservative to avoid pruning arcs that give access to other areas.

The downside of this conservative transformation is that we only gain a small or even no speedup at all. In the worst case the calculation takes more time than without the speedup technique.

Bidirectional search We showed that the ellipse property can be used for the distance cost measure or a cost measure that can be transformed to a distance measure. To make it work for other cost measures as well we decided to design the algorithm around bidirectional search. It will search in any direction until the budget is fully used in all directions, and the graph then expresses the reach exactly. This is more expensive to calculate, but with the benefit that it works for all cost measures and for certain cost measures it computes a smaller reach than an ellipse will do. Also we know directly the shortest path to every arc in

reach, while using an ellipse this should be still computed for every arc within reach, which is slow for graphs that are not optimized for this purpose. Furthermore we gain performance profit by making greedy choices, so fewer iterations and fewer bidirectional searches are required. The running time complexity is given in Section 6.5. Together with the benefit that bidirectional search also avoids duplicated arcs in our path, it will be a valid choice.

6.5 Running time analysis

We will give a worst-case running time analyses in big-$\mathcal{O}$ notation for the algorithm. Let n the number of vertices $|V|$ and m the number of arcs $|E|$ in the graph $G = (V, E)$.

Worst-case In scenicRoute the main loop is executed I times. All operations in the loop cost at most $\mathcal{O}(m)$ time, except the call to generatePath, whose function body is recursively defined and contains an expensive call to shortestPaths. The costs of shortestPaths depends on a bidirectional Dijkstra search.

The bidirectional Dijkstra search contains two monodirectional searches, each one covering half of the graph. The running time of Dijkstra depends on the running time of the priority queue that sorts vertices. Using a self-balancing binary search tree or binary heap as priority queue to extract a minimum cost vertex takes $\mathcal{O}(\lg(n))$ time. Then Dijkstra needs $\mathcal{O}(m \lg(n))$ time. Constructing a path for every arc in the bidirectional search between source and destination can be done at once for all arcs and costs at most $\mathcal{O}(m)$ time. Hence, shortestPaths runs in the worst-case in $\mathcal{O}(m \lg(n))$ time.

Because the recursion in generatePath is at most one level deep, it makes at most three calls to shortestPaths in every iteration of the loop in scenicRoute. The worst-case running time of the algorithm can therefore be bounded by

$$\mathcal{O}(I \cdot m \lg(n))$$

Recursion We decided to perform only one level of recursion in generatePath, but one can decide to perform multiple levels of recursion. The number of recursion levels determines the balance between greedy choices and randomness. Performing recursion until we reach the base case will utilize the greedy choices maximally, but it comes at the expense of additional running time. The recursion in generatePath follows the divide and conquer structure, whose worst-case running time is specified as

$$T(m) = \begin{cases} \mathcal{O}(1) & \text{if } m = 1 \\ 2T(m/2) + \mathcal{O}(m \lg(m)) & \text{if } m > 1 \end{cases}$$

for which $\mathcal{O}(m \lg(n)) \subseteq \mathcal{O}(m \lg(m))$ because of $m \geq n + 1$ for connected graphs. Using the Master Theorem to solve the recursion, we will find that the second case applies under the condition $\mathcal{O}(m \lg(m)) = \mathcal{O}(m^c \lg^k(m))$ for $c = 1, k = 1$, for which the running time is specified as $\mathcal{O}(m^c \lg^{k+1}(m)) = \mathcal{O}(m \lg^2(m))$. The result is that the algorithm as a whole takes $\mathcal{O}(I \cdot m \lg^2(m))$ time.

Quality measure In the case an advanced quality measure is used as part of the generalized Quality Orienteering Problem, and which takes $\mathcal{O}(Q)$ time to determine the quality of a route, the running time is $\mathcal{O}(I \cdot Q \cdot m \lg(n))$.

7 Experiment

We have presented the Quality-AOP in Section 4.1 and have given an algorithm in Section 6 to solve that problem. Now we will establish whether we actually find better routes for the Quality-AOP than for the AOP. In this section we present our experiment that is performed on a real road network. First we will describe the research questions in 7.1, the setup of the experiment in Section 7.2, and then present and discuss the results in Section 7.3.

7.1 Research questions

We have argued that in our definition of the Quality-AOP we are better able to find routes that match the definition of the most scenic route than we will find for the AOP. It should search for routes of high scenic average score rather than the maximum score available in the road network. We already showed that this is true in theory. The experiment will show that in practice this is the case and how much the impact is on the quality of the route.

Main Question Do we find with the Quality-AOP higher quality routes than we find with the AOP, on a real road network given a desired travel cost budget?

We will answer this question by researching the following topics and questions.

- Optimizing objectives (Section 7.3.1)
- Quality measure (Section 7.3.2)
  - What is the difference between measuring the score and the quality?
  - What is the influence on the quality of a route if we introduce a minimum cost budget?
- Budget and quality (Section 7.3.3)
  - How much quality improvement can we get by using a cost budget interval?
  - Can the use of a budget interval, to search also beyond the desired travel costs, lead to higher quality routes?
  - Does the size of a budget interval influence the possibility to find high quality routes?
  - Does the amount of travel costs influence the possibility to find high quality routes?
- Detours and artifacts (Section 7.3.4)
  - Do route properties match our expectation if we use the objective 'speed'?
  - Will the Quality-AOP improve speed related properties so that it better matches our expectations and avoids artifacts?
- Route properties and artifacts (Section 7.3.5)
  - Do route properties match our expectation if we use the less predictable but more useful objective 'curviness'?
  - Will the Quality-AOP improve the curviness related properties so that it better matches our expectations and avoid artifacts?

Note In this thesis we have used two types of quality measures, SUM (the measure of the AOP in which scores are summed) and AVG (the measure of the Quality-AOP, which is the average score relative to the costs). From now on we use the word **quality** to refer to the quality measure AVG and just use the word **score** to refer to summed scores.

7.2 Setup

First we describe in Section 7.2.1 how we obtained the road network. Then we describe how to measure specific route properties to determine the quality of routes in Section 7.2.2. In Section 7.2.3 we describe the configuration of our experiment.

7.2.1 Road network

We use the data from OpenStreetMap contributors [30] to model a graph that represents a real road network. We create an extract that includes a part of The Netherlands, namely the province Utrecht. The road network of The Netherlands is almost fully covered in OpenStreetMap (OSM) because the Dutch government has integrated their data into that database. OSM does not only contain geographic information about roads, but also data that describes properties of roads such as maximum speed, the number of lanes, type of road, and the accessibility. We will use these properties in our routing algorithm. A full list of road properties is given on the OpenStreetMap wiki[2]. Furthermore it contains information about the environment such as land usage, villages and towns, points of interests, etc.[3], which can be used for scenic routing as well.

Query 7.1 Overpass query of region Utrecht

```
1: [out:json][timeout:3600][maxsize:1073741824];
2: ( area[name="Utrecht"][admin_level="4"];
3: way(area)["highway"~"motorway|trunk|primary|secondary|tertiary];
4: way(area)["highway"~"motorway_link|trunk_link|primary_link|secondary_link|tertiary_link"];
5: >; );
6: out body geom qt;
```

Routable graph We use an OSM extract of The Netherlands provided by Geofabrik[4] who created extracts of many areas from the Full Planet[5]. Using Overpass API[6] we are able to query[7] the network using the Overpass Query Language[8]. From Query 7.1 we get 12342 roadways (and 49232 nodes) within the area Utrecht (Figure 18). Roadways consist of a sequence of nodes to describe the geometry. We only include roads of type motorway, trunk, primary, secondary, and tertiary. So we exclude residental and service roads. We obtain as well the geometric data and road properties. We will give the output to our preprocessor that will construct a connected and directed graph, by determining the intersection points

[2] http://wiki.openstreetmap.org/wiki/Highways
[3] http://wiki.openstreetmap.org/wiki/Map_Features
[4] `http://download.geofabrik.de/osm`
[5] `http://planet.osm.org/`
[6] `https://github.com/drolbr/Overpass-API`
[7] `http://overpass-turbo.eu/s/oEA`
[8] `http://wiki.openstreetmap.org/wiki/Overpass_API/Language_Guide`

and contracting line segments, so we end up with 19631 arcs, each representing a roadway in one direction between crossings.

Figure 18: The road network of the province Utrecht of The Netherlands.

Properties Each arc e in the graph gets assigned the following properties. A *distance* d_e in kilometers determined by the Euclidean distance between the points that describe the geometry of this arc. A *speed* s_e in kilometers per hour, directly obtained from the *maxspeed* property from OSM. The *time* t_e in minutes, determined by the properties distance and speed. A *waytype* ψ_e, which is either motorway, trunk, primary, secondary, or tertiary. The number of *lanes* η_e. The *curviness* ξ_e which is the amount of degrees changed in direction while traversing the geometry of the arc. The *isDike* flag describes whether the road is on a raised bank or not.

7.2.2 Measured properties

In our experiment we will use several score measures. So we can optimize different objectives such as speed or angular change for example. The score measure uses the properties of an arc to determine the score. We use the structure of F-algebras to implement the measure in a generic way, so that it can easily fit different data structures that store a path. The algebra describes how the scores get summed while folding the data structure. In our case a path is stored in a list data structure, so we define a catamorphism function that could fold a list (the path) and use the algebra (score measure) to return a result which is in our case a non-negative score.

Function 7.1 Catamorphism for traversing elements of a list

$\mathcal{C}_{list} :: \mathcal{A}\ \tau \mapsto ([\tau] \mapsto \mathbb{R}_{\geq 0})$
$\mathcal{C}_{list}$ (nil, merge) = foldl merge nil

We also add a score measure that not only determines the score based on the arc properties but also looks at the route as whole. It wil calculate the number of turns. Therefore it requires a more complex fold function that takes into account consecutive arcs. We created an alternative catamorphism to fold over consecutive arcs.

Function 7.2 Catamorphism for traversing pairs of consecutive elements of a list

$\mathcal{C}'_{list} :: \mathcal{A}'\ \tau \mapsto ([\tau] \mapsto \mathbb{R}_{\geq 0})$
$\mathcal{C}'_{list}\ \mathcal{A}'\ (\text{x:xs}) = \mathcal{C}_{list}\ \mathcal{A}'\ (\text{zip (x:xs) xs})$

Now we have a generic fold definition, when it is provided with an algebra $\boldsymbol{\mathcal{A}_{lgebra}}$ it produces a score function $\boldsymbol{f_{score}}$ which can be used to determine the score of a route based on its properties.

$\boldsymbol{\mathcal{A}_{lgebra}}$

$$\mathcal{A}_{distance} = (0, \beta + d_e) \tag{1}$$
$$\mathcal{A}_{time} = (0, \beta + t_e) \tag{2}$$
$$\mathcal{A}_{curviness} = (0, \beta + \xi_e) \tag{3}$$
$$\mathcal{A}_{speed} = (0, \beta + s_e c_e) \tag{4}$$
$$\mathcal{A}_{lanes} = (0, \beta + \eta_e c_e) \tag{5}$$
$$\mathcal{A}_{dike} = (0, \beta + y_e c_e) \quad \text{for } y_e \begin{cases} 1 & \textbf{if } isDike(e) \\ 0 & \textbf{if } \neg isDike(e) \end{cases} \tag{6}$$
$$\mathcal{A}_{way}(\psi) = (0, \beta + y_e c_e) \quad \text{for } y_e \begin{cases} 1 & \textbf{if } \psi_e = \psi \\ 0 & \textbf{if } \psi_e \neq \psi \end{cases} \tag{7}$$
$$\mathcal{A}_{vertices} = (1, \beta + 1) \tag{8}$$
$$\mathcal{A}'_{turns} = (0, \beta + y_{e_1,e_2}) \quad \text{for } y_{e_1,e_2} \begin{cases} 1 & \textbf{if } cos^{-1}(\hat{e}_1 \bullet \hat{e}_2) > 1 \\ 0 & \textbf{otherwise} \end{cases} \tag{9}$$

$\boldsymbol{f_{score}}$

$\sum_{e \in P} d_e$	$\sum_{e \in P} t_e$	$\sum_{e \in P} \xi_e$	$\sum_{e \in P} s_e c_e$	$\sum_{e \in P} \eta_e c_e$	$\sum_{e \in P} y_e c_e$	$\sum_{e \in P} y_e c_e$	$\sum_{e \in P} 1$	$\sum_{i=0}^{\lvert P \rvert - 1} y_{e_i, e_{i+1}}$
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)

In the algebras, the variable β is used to describe the score of the partial sum result while folding the data structure. $\mathcal{A}_{distance}$, $\mathcal{A}_{time}$, and $\mathcal{A}_{curviness}$ sum the distances, times and curviness of arcs in a path in which d_e, t_e, and ξ_e stands for the distance (in kilometers), time (in minutes) and angular change (in degrees) of an arc e. $\mathcal{A}_{speed}$ and $\mathcal{A}_{lanes}$ sum arcs respectively by the property speed s_e and numbers of lanes η_e after they are multiplied by c_e (the time or distance it takes to traverse the arc). In $\mathcal{A}_{dike}$ we determine if the road is a dike or not, if so we use c_e as score. In $\mathcal{A}_{way}(\psi)$ we attach score to an arc if the arc is of

type specified by ψ, where $\psi \in$ {motorway, trunk, primary, secondary, tertiary}. $\mathcal{A}_{vertices}$ or $\mathcal{A}_{arcs}$ just count the number of vertices or arcs encountered while traversing the path. The algebra $\mathcal{A}'_{turns}$ is special, because it is one of the measures that do not fit in the Quality-AOP but in the generalized Quality Orienteering Problem (QOP). This measure does not only take into account arc properties but determines the quality based on properties of the route. This one count the number of turns, which depends on the actual choice of routing. A turn is defined as an angle larger than 1 radian between two consecutive arcs.

7.2.3 Configuration

To answer all research questions from Section 7.1, we have setup and implemented a set of configurations. In each configuration the algorithm will run several times with certain properties. The result is a set of routes with data about the properties of these routes and data about the process of running the algorithm.

We created a configuration for all 9 properties (algebras $\mathcal{A}$) from Section 7.2.2. Each such property is the objective of the configuration and should be optimized in the calls to scenicRoute (Algorithm 6.1). The function scenicRoute is called 3 times, each time with a budget corresponding to the desired travel distance of $40km$, with the same source and destination, and with 250 iterations. In the first call we use the quality measure $SUM \in \mathcal{Q}$ and in the following two calls we use the measure $AVG \in \mathcal{Q}$. For the calls with AVG we will allow 20% deviation from the travel budget. In the first call to AVG, this is 20% below the desired travel distance such that the minimum budget is set to 80% of the desired travel distance ($32km$), while in the second call it searches 10% below and 10% above the desired travel distance (36-44km), which we call AVG^+ from now on, because it extends the search so that it may exceed the budget. Therefore AVG^+ could find routes that SUM and AVG could not find. We will repeat this 30 times, each repeating call with other source and destination vertices, so we can take an average of the routes. In total $|\mathcal{A} \times \mathcal{Q}| \cdot 30 = 810$ routes have been calculated. The results are in Table 1 of Section 7.3.3. In Section 7.3.5 we look in more detail into the results of the objectives $\mathcal{A}_{curviness}$ and $\mathcal{A}_{speed}$.

We decided to use distance as default cost measure because it is easy to analyse these routes on a map and see the result of the optimized objective. If we would use time as cost measure we can not easily see how fast has been travelled on the different roadways.

We also have additional configurations where we change the cost budgets. We would like to see what happens to routes if more cost budget is available. We combine this with time as cost measure. We have 3 configurations with the time budgets $\mathcal{T}$ of 20 min, 50 min, and 80 min. In case of a 80 min budget, the search space of province Utrecht is small in comparison to the budget. Additionally we also created configurations for a varying interval in which AVG and AVG^+ are allowed to deviate from the desired travel costs. The configurations allow deviations Ω of 1%, 5%, 10%, 15% and 20% on a travel distance of again $40km$. In all cases we use the algebra $\mathcal{A}_{curviness}$ to optimize the scenic property that describes the curviness of the road. In total we have calculated $|(\mathcal{T} + \Omega) \times \mathcal{Q}| \cdot 30 = 720$ additional routes. The results are in Table 2 and 3 of Section 7.3.3.

In all 1530 cases we register data about the coordinates of the route, the score of all properties (even if a property is not the current objective), the course of the quality, at which iteration it passes the minimum budget, and if the quality has been converged after 250 iterations. Furthermore, in the calls with $SUM \in \mathcal{Q}$, we store the path that is found in the local optimum of quality just below the budget, i.e. the path that is found in the last iteration in which the quality increases. That would be useful to detect detours and

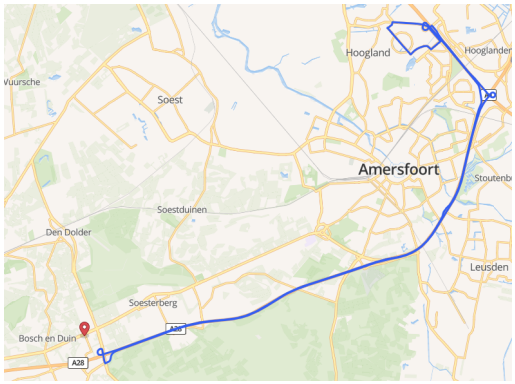
artifacts due to the result of the AOP. The results are in Table 4 of Section 7.3.4.

Besides all configurations we started with getting a picture of the course of the score and quality by calculating routes of increasing budgets from 5 to 45 *km*. We will show the results in Section 7.3.2.

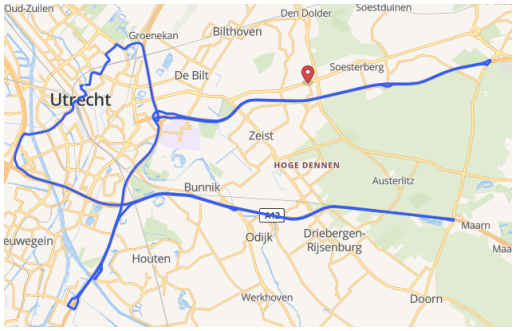
7.3 Results

7.3.1 Optimizing objectives

The result of optimizing different objectives is shown in Figure 19. We can clearly see that each objective results in a different kind of route. In Figures 19a, 19b and 19c we see it mainly travels high speed roads to optimize the objectives *speed, distance* and number of *lanes* respectively. In Figure 19e smaller roads are traversed to increase the travel *time*. In Figure 19f it optimizes *dikes*, although it seems that raised roads are also marked as embankment in OpenStreetMap. In Figure 19d the number of *arcs* in a path is optimized, which results in a route traversing many crossings and making more turns. We also have *curviness* as useful property for scenic routing, in Figure 19g, where we can see that many curved road segments are included in the route.

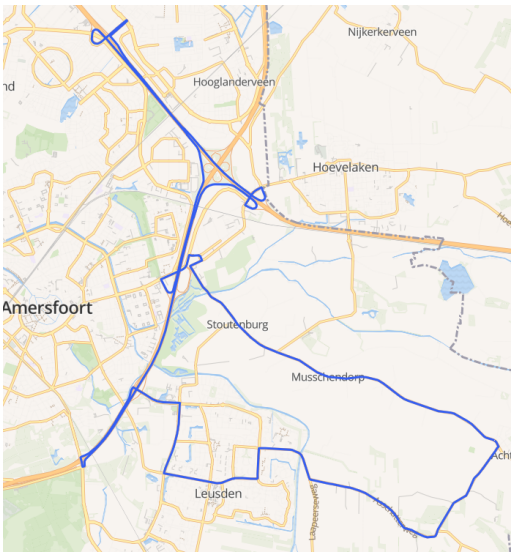


(a) Optimize $\mathcal{A}_{speed}$ to get the highest average speed.

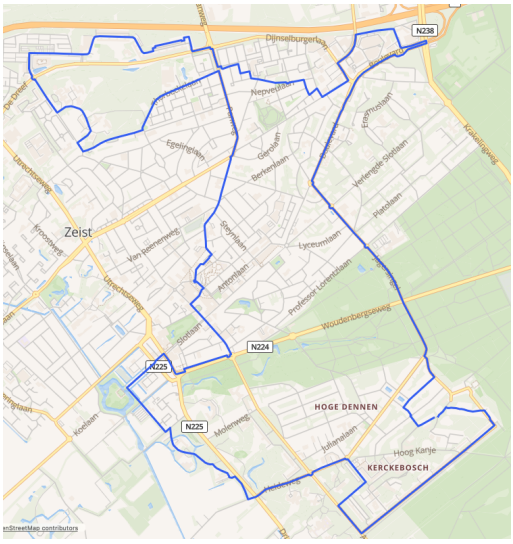


(b) Optimize $\mathcal{A}_{distance}$ to travel as much distance as possible given a time budget.

Figure 19: Different algebras used as objective in the optimization.

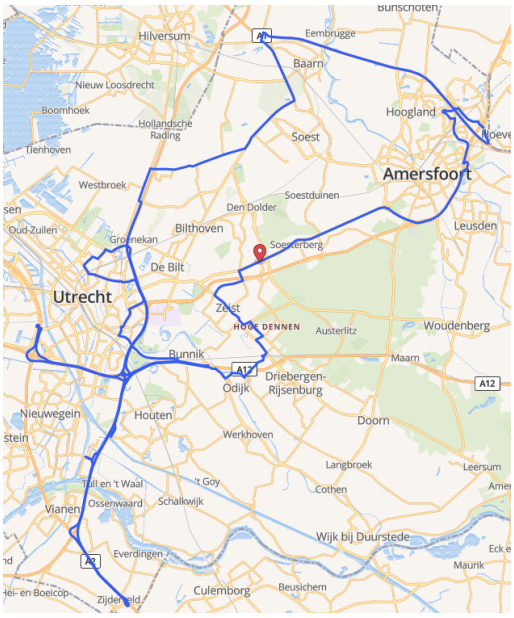


(c) Optimize $\mathcal{A}_{lanes}$ to get on average as many lanes as possible.



(d) Optimize $\mathcal{A}_{arcs}$ to increase the number of traversed road segments.

(e) Optimize $\mathcal{A}_{time}$ to travel slowly and increase travel time for the given distance budget.



(f) Optimize $\mathcal{A}_{dike}$ to include road segments that are on an embankment.

Figure 19: Different algebras used as objective in the optimization.

(g) Optimize $\mathcal{A}_{curviness}$ to include curvy road segments.

Figure 19: Different algebras used as objective in the optimization.

7.3.2 Quality measure

First we show how average collected score is related to summing scores. In Figure 20 we show the course of the two measures for different route calculations with different budget. The budget ranges from 5km to 45km with a step interval of 1km. At every step we take the average of 5 routes.

We see that the summing scores result in an increasing line, however its not monotonically increasing as we discussed in Section 4.1, due to the fact that the amount of score that could be collected depends on the choice of starting location, which we change for every route calculation. Also it is not garantueed that the algorithm will always find the optimal solution. Still, globally we see an increasing line which means that routes are able to collect more score if the budget increases. If we look at the average quality we see a fluctuating graph that globally describes an horizontal line.

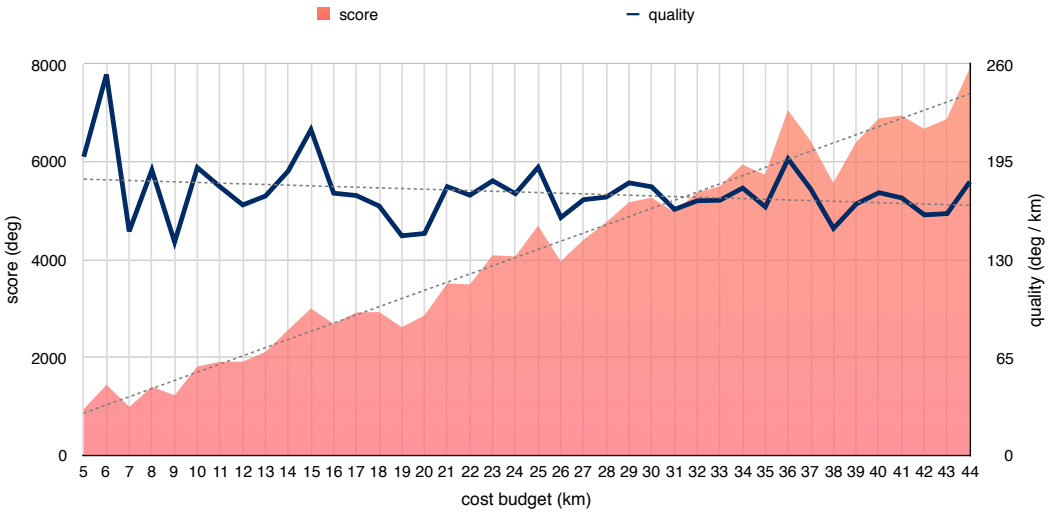


Figure 20: The course of two measures: summing scores (angular change in degrees) and quality ($\frac{deg}{km}$) for different route calculations with different cost budgets (*distance* in kilometers).

Question 1. *What is the difference between measuring the score and the quality?*

If we look at our new measure in Figure 20 we can directly see peaks of high quality routes. Peaks that are followed by a serie of lower values are the most interesting, because it indicates that traveling more distance will not improve the route's quality.

At certain points we can see already that the summing score is able to find more score while the route quality does not increase. Consider the points at budget $24km$ and $30km$ for example. This is already an indication that the Quality-AOP, using the new average quality measure, probably will find higher quality routes.

Question 2. *What is the influence on the quality of a route if we introduce a minimum cost budget?*

We see high quality routes for low budgets, but because a traveler wants to travel at least an amount of costs we introduced the notion of minimum cost budget. Still it is possible to find high quality routes, because there are many peaks due to the fluctuating behavior of the average quality measure. For example if we only look in the range of 20km to 30km, the Quality-AOP gives us the high quality route at 25km, while the AOP gives us a route at the end of the range at 29km for which the quality is worse than the route found in the Quality-AOP.

7.3.3 Budget and quality

		SUM 40 km			AVG 32-40 km				AVG+ 36-44 km			
objective		score	costs	quality	score	costs	quality	Δ	score	costs	quality	Δ
curviness	(deg)	6479	39.56	**163.79**	6585	37.05	**178.65**	**9.1%**	7371	41.38	**178.91**	**9.2%**
dike	(km)	10	38.98	**0.27**	10	37.81	**0.27**	**0.0%**	12	41.72	**0.30**	**11.1%**
lanes	(avg)	101	39.29	**2.58**	97	36.58	**2.66**	**3.1%**	110	40.18	**2.73**	**5.8%**
arcs	(count)	356	39.66	**8.98**	366	38.41	**9.55**	**6.3%**	395	41.15	**9.62**	**7.1%**
speed	(km/h)	3678	39.47	**93.25**	3610	36.43	**99.37**	**6.6%**	4069	40.11	**101.55**	**8.9%**
time	(min)	45	39.84	**1.12**	43	37.16	**1.17**	**4.5%**	50	42.04	**1.18**	**5.4%**
turns	(count)	62	39.17	**1.57**	62	36.56	**1.72**	**9.6%**	69	40.35	**1.73**	**10.2%**

Table 1: Comparison of SUM, AVG, and AVG^+ on the quality of routes when optimizing different objectives with a desired travel distance of $40km$.

In Table 1 we present the results of comparing the Quality-AOP with the AOP. The SUM measure from AOP is used as reference. The columns AVG and AVG^+ show the improvement of using the average quality measure from the Quality-AOP. We perform the test for different objective functions. Each value for every objective is an average of 30 calculated routes with different starting points. The desired travel distance is $40km$. We decide that it may vary at most 20%. Therefore SUM is just provided with a budget of $40km$. AVG has a minimum budget of $32km$ (20% lower than the desired travel distance) and maximum budget of $40km$. In AVG^+ we allow searching 10% below and 10% above the budget, so that its interval is 36-$44km$. Notice that AVG^+ is capable to find routes that SUM could never find.

Question 3. *How much quality improvement can we get by using a cost budget interval?*

In Figure 21 we can see that with AVG routes are found with improved quality relative to SUM. The improvement strongly depends on the objective function, but it can improve up to almost 10% compared to the AOP. The reason less improvement is gained with objectives *dike* and *lanes*, has probably to do with the road network that varies in this properties only a little.

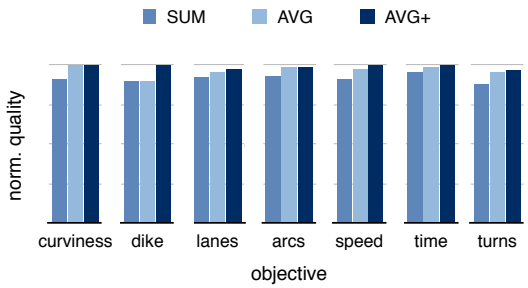


Figure 21: Visual comparison of the data from Table 1 with normalized quality.

Question 4. *Can the use of a budget interval, to search also beyond the desired travel costs, lead to higher quality routes?*

In AVG^+ we search around the desired travel distance and allow 10% less or 10% more travel distance. It could find routes that both SUM and AVG could not find, due to their hard cap on their upper limit that equals the desired travel costs. Therefore we see in all cases AVG^+ improves relative to SUM. It also improves relative to AVG in all cases while adding only $1km$ to the desired travel distance at average to find up to 11.1% improvement in route quality.

Question 5. *Does the size of a budget interval influence the possibility to find high quality routes?*

budget deviation	SUM	40 km		AVG	[40 - Ω, 40] km			AVG+	[40 - Ω/2, 40 + Ω/2] km		
(Ω)	score	cost	quality	score	cost	quality	Δ	score	cost	quality	Δ
1%	6712	39.84	**168.48**	6491	39.85	**162.91**	**-3.3%**	6319	40.06	**157.75**	**-6.4%**
5%	6709	39.72	**168.97**	6545	39.44	**165.98**	**-1.8%**	6319	40.06	**157.75**	**-6.6%**
10%	6702	39.51	**169.71**	6693	38.44	**174.33**	**2.7%**	6925	40.43	**171.25**	**0.9%**
15%	6512	39.59	**164.50**	6578	37.65	**175.20**	**6.5%**	7279	41.05	**177.80**	**8.1%**
20%	6479	39.56	**163.79**	6585	37.05	**178.65**	**9.1%**	7371	41.38	**178.91**	**9.2%**

Table 2: Comparison of SUM, AVG, and AVG^+ on the quality of routes when optimizing the objective *curviness* for different deviations 1%-20% of the desired travel distance of $40km$.

In Table 2 we show the results of changing the size of the budget interval. In the previous example we allowed to deviate from the desired travel costs at most 20%. Here we see what happens for smaller intervals. For small intervals we can see that the Quality-AOP performs worse as a result of our algorithm that only accepts quality improvements if the costs stay above the minimum budget, which is a problem because it can easily get stuck in a local optimum if the budget interval is too small. In addition, if the budget increases the quality becomes higher. The larger the interval of budget, the more chance to find higher quality routes.

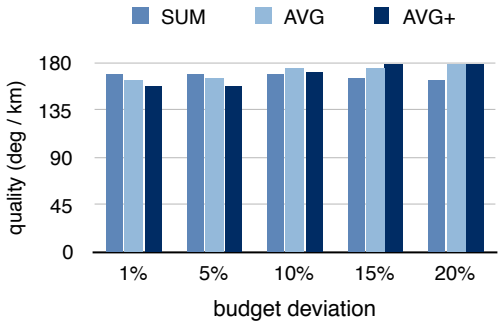


Figure 22: Visual comparison of the data from Table 2.

Question 6. *Does the amount of travel costs influence the possibility to find high quality routes?*

	SUM	B min		AVG	[B - ⅕B, B] min		
budget (B)	score	cost	quality	score	cost	quality	Δ
20 min	3209	19.33	**166.26**	3195	18.29	**175.03**	**5.3%**
50 min	7910	49.44	**159.93**	7983	46.01	**174.05**	**8.8%**
80 min	11789	79.59	**148.14**	12193	74.90	**163.15**	**10.1%**

Table 3: Comparison of SUM and AVG on the quality of routes when optimizing the objective *curviness* with different time budgets of 20, 50, and 80 minutes.

In Table 3 we can see how the cost budget influence the quality of routes. First, notice that now time is used instead of distance as cost measure and we try to optimize the curviness of the route. We run the experiment with budget 20, 50 and 80 minutes of maximum travel time and a minimum budget at 80% for AVG. We see the quality decreases for an increasing budget. An example of this is given in Figure 24, here highly curved road segments on highway intersections are found within a small travel time budget, but we have to travel more time to reach the next place containing curvy roads. Furthermore the road network of province Utrecht is too small for the budget of 80 minutes if the route mainly consists of motorways. This is not what we expected to happen if curviness is optimized.

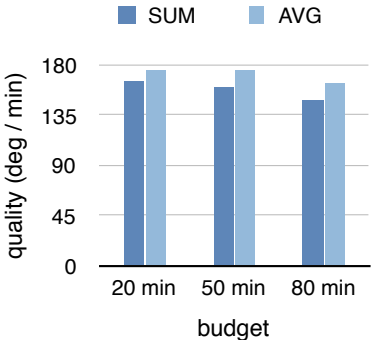


Figure 23: Visual comparison of the data from Table 3.

Figure 24: Example of a highly curved route for a small travel time budget.

Independent of decreasing quality, the Quality-AOP will perform better compared to the AOP if the budget increases. This could be explained by the minimum budget. The minimum budget is set at 80% of the desired travel time. If the budget increases, the amount of allowed deviation from the desired travel time grows and there are more possibilities to find high quality routes.

7.3.4 Detours and artifacts

In Section 4.1.1 we have shown that the AOP would optimize too much. It tries to add score even if it generates a detour, which makes that the quality of the final route is less. We would like to research if this happens in practice. Of course, the goal in the AOP is to maximally utilize the budget so it does not require a minimum budget, but it should not perform the last iterations in which the quality of the route decreases. Therefore we created an variant of SUM which we call SUM'. It gives us the route with the highest score in which the quality still increases. The results are shown in Table 4.

			SUM' ±40 km			SUM 40 km			
objective		route	score	costs	quality	score	costs	quality	Δ
curviness	(deg)	1	6348	39.76	**159.64**	6371	39.92	**159.60**	-0.03%
		2	5934	39.01	**152.14**	5945	39.23	**151.57**	-0.37%
dike	(km)	3	2	38.46	**0.06**	2	39.33	**0.05**	-2.31%
lanes	(avg)	4	172	38.78	**4.45**	172	38.79	**4.45**	-0.03%
		5	127	38.07	**3.33**	127	39.64	**3.20**	-3.91%
		6	84	38.21	**2.19**	84	38.47	**2.18**	-0.36%
		7	115	38.68	**2.98**	116	39.16	**2.96**	-0.58%
		8	74	39.68	**1.86**	74	39.90	**1.86**	-0.25%
		9	93	38.92	**2.40**	95	39.95	**2.38**	-0.70%
		10	73	39.92	**1.84**	73	39.98	**1.84**	-0.07%
arcs	(count)	11	375	39.65	**9.46**	377	39.92	**9.44**	-0.15%
speed	(km/h)	12	3880	39.24	**98.87**	3890	39.37	**98.81**	-0.06%
		13	4110	38.05	**107.99**	4166	38.75	**107.49**	-0.46%
		14	3413	39.69	**85.98**	3414	39.72	**85.96**	-0.03%
		15	2925	39.72	**73.64**	2925	39.73	**73.63**	-0.01%
		16	4001	36.80	**108.71**	4219	39.52	**106.74**	-1.82%
		17	2841	39.26	**72.37**	2868	39.64	**72.35**	-0.03%
time	(min)	18	44	39.88	**1.09**	44	39.93	**1.09**	-0.04%
turns	(count)	19	50	37.51	**1.33**	52	39.12	**1.33**	-0.31%
		20	73	36.69	**1.99**	77	39.36	**1.96**	-1.69%

Table 4: 20 of the 210 routes that show a decrease in quality in SUM as the result of artifacts. SUM' avoids this and stops just before the quality drops.

From the total of 210 calculated routes (30 for each objective) we find 20 cases of which SUM' finds higher quality routes than SUM. These are the cases of which the algorithm for SUM has performed one or multiple iterations at last in which the route's quality decreases. It shows that the AOP is not the correct problem description for scenic route planning. Despite the small decrease of quality, it is exactly here where artifacts grow. Consider the calculated routes in Figure 26 where speed is the objective to optimize. For the AOP we find a route that gets off the motorway in order to collect additional score and fully utilize the budget, while in the same figure we see in the Quality-AOP we stay on the motorway because it will not accept the small detour that lowers the quality.

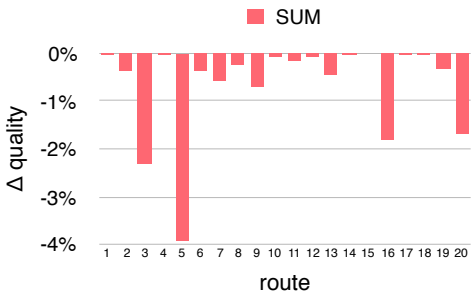


Figure 25: Data from Table 4 shows that too much optimization generates lower quality routes.

Figure 26: The AOP could generate small detours as part of the optimization process, which are considered as artifacts. The objective speed has been optimized here, but the additional score collected from the roundabout does not improve the route's quality, it will only maximally utilize the budget.

7.3.5 Route properties and artifacts

We want to verify if the Quality-AOP will really improve the route's quality, by not only looking at the objective function but also at other route properties. We expect to see an improvement of properties that are related to the objective function. Considering these properties we can analyse if the number of artifacts reduces when using the Quality-AOP instead of the AOP.

Question 7. *Do route properties match our expectation if we use the objective 'speed'?*

We will first show the results of optimizing the objective *speed*. In most cases this property is irrelevant for scenic routing in the sense it is not wanted by recreational travelers, but this property is useful to easily generate a model of expectation what will happen if we try to optimize this objective. For example we expect the route mainly consist of high speed roads, such as motorway or trunk, therefore the travel time will be short, because we are limited by a distance budget. It contains of long straight roads with many lanes, consisting of only a few arcs, almost no curves, no dikes, and we expect only to make a few turns.

After running the experiment we can compare the property values obtained by SUM for the objective *speed*, see Table 5, to route properties of other objective optimizations from Table 1. We find that the properties lanes and dike score above average and curviness, the numbers of arcs, and the number of turns score below average. Thus the property dikes is the only one that does not match our expectation. Furthermore it will indeed use the way types motorway and trunk more than average.

Question 8. *Will the Quality-AOP improve speed related properties so that it better matches our expectations and avoids artifacts?*

Now we focus on the objective *speed*. We will look at the speed related properties in Table 5 and Figure 27 to see if they improve as well. We already saw that the AOP almost matches our expectation of improving the properties related to the objective *speed*. But here we see that AVG and AVG^+ improve all these properties even more. Only the property *dikes* does not match our expectation, probably due to the definition of dike in OpenStreetMap that not matches our definition of dike (embankment along rivers), which makes the property less reliable for our experiment. Also we can clearly see that AVG and AVG^+ increases the number of high speed roads such as motorways, indicating that artifacts such as shown in Figure 26 are eliminated. It is also interesting to see that the number of turns decreases drastically, which is a valuable improvement.

	speed		curviness		time		lanes		dike		arcs		turns		way type				
	km/h	Δ	deg/km	Δ	min/km	Δ	avg-#	Δ	%	Δ	#/km	Δ	#/km	Δ	motorway %	trunk %	primary %	secondary %	tertiary %
SUM	93.2		68.7		0.72		2.29		10.2%		4.3		0.45		64.93%	2.20%	9.31%	13.90%	9.66%
AVG	99.4	6.7%	59.5	-13.4%	0.67	-6.9%	2.46	7.4%	10.8%	5.9%	3.8	-11.6%	0.32	-28.9%	73.64%	0.47%	9.12%	10.60%	6.16%
AVG+	101.5	8.9%	56.1	-18.3%	0.65	-9.7%	2.56	11.8%	7.0%	-31.4%	3.5	-18.6%	0.31	-31.1%	77.07%	1.51%	6.95%	10.12%	4.34%

Table 5: The effect on route properties when SUM optimizes the objective *speed*, and how AVG and AVG^+ affect the properties relative to SUM.

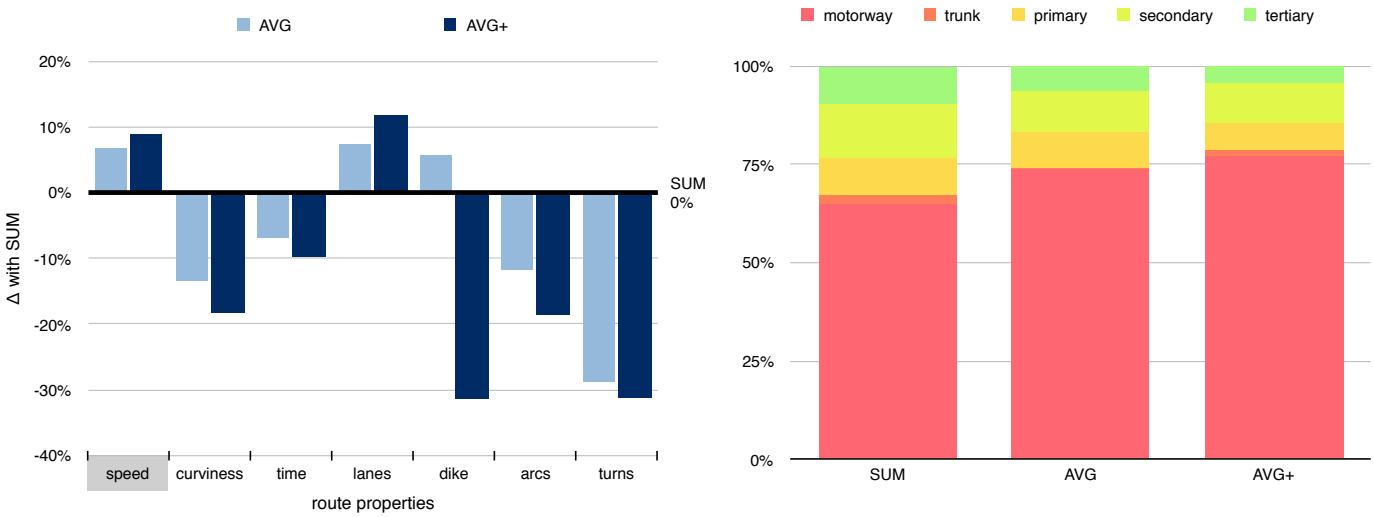


Figure 27: Visualized data from Table 5.

Question 9. *Do route properties match our expectation if we use the less predictable but more useful objective 'curviness'?*

The same as we did for the objective *speed* we will do for the objective *curviness*. We choose this property because in contrast to *speed* this property plays a role in scenic route planning, for example to find highly curved routes for motorcyclists. On purpose we choose to compare this property with *speed* because for *speed* it was easy to predict how properties

behave if we optimize the objective, but for *curviness* this is much harder. Therefore we are interested if we could gain the same quality improvement for *curviness* and its related properties as we get for *speed*.

We expect that curviness is related to smaller roads, where the maximum speed is low, as well as the number of lanes. We also expect that we get more turns, traverse more arcs, and probably travel more on dikes and the type of way will be mainly secondary and tertiary.

The values for SUM in Table 6 show the maximum speed and number of lanes is low, compared to the average of optimizing other objectives from Table 1. The number of turns and the number of arcs are actually above average. Only the property dikes is lower than average. For the type of way we see that secondary and tertiary ways are highly present. So in general the results of the experiment match our expectation.

Question 10. *Will the Quality-AOP improve the curviness related properties so that it better matches our expectations and avoid artifacts?*

Now we have seen that the experiment matches our expectations, we will check if AVG and AVG^+ optimize these properties which indirectly improves the quality of the route. As we can see in Table 6 and Figure 28, all properties improve as we expected except from the property *turns*. If we compare these improvements with the improvement we gained when optimizing *speed*, we see that this improvement is smaller, except for the property dike. Because the improvement is small we can only see a small change in type of ways traveled. So again, the Quality-AOP is able to improve most of the properties.

	speed km/h	Δ	curviness deg/km	Δ	time min/km	Δ	lanes avg-#	Δ	dike %	Δ	arcs #/km	Δ	turns #/km	Δ	way type: motorway %	trunk %	primary %	secondary %	tertiary %
SUM	62.2		163.8		1.06		1.37		1.4%		7.8		1.24		9.69%	2.27%	17.79%	34.13%	36.12%
AVG	62.0	-0.3%	178.6	9.0%	1.06	0.0%	1.32	-3.6%	2.3%	64.3%	7.6	-2.6%	1.22	-1.6%	8.59%	2.19%	17.59%	35.29%	36.33%
AVG$^+$	61.7	-0.8%	178.9	9.2%	1.06	0.0%	1.32	-3.6%	1.5%	7.1%	7.7	-1.3%	1.28	3.2%	9.02%	0.84%	15.78%	37.83%	36.53%

Table 6: The effect on route properties when SUM optimizes the objective *curviness*, and how AVG and AVG^+ effect the properties relative to SUM.

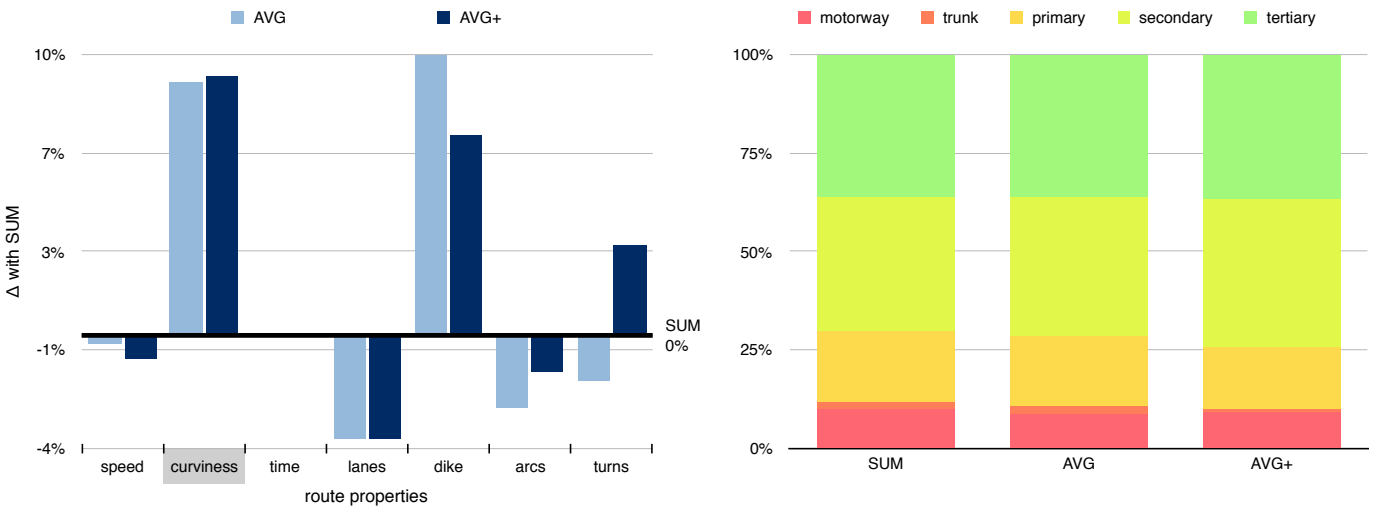


Figure 28: Visualized data from Table 6.

8 Conclusion

The Arc Orienteering Problem (AOP) let us find a scenic route of specific length or travel time in which the scenic score is maximized. We extended the AOP with a few aspects. In the first place we modified the definition to make it suitable for directed graphs. The start and end locations may be the same but they may also be different. Also we explicitly forbid subtours because travellers do not like that. Additionally we forbid to traverse twin arcs, which are the arcs on the path pointing in opposite direction, so we do not travel the same road back and forth. With this definition, it is possible to mark roads as twins even if they are not directly adjacent. We introduced a new quality measure that instead of maximizing the collected score determines the average score, i.e. the amount of score in proportion to the travel costs. The quality measure is fluctuating and therefore we need a minimum budget to exclude high quality routes of too little costs. The new quality measure together with subtour elimination and the introduction of twin arcs gives us a new definition, which we call the Quality-AOP. We also have given a more general definition, the Quality Orienteering Problem (QOP) that determines the score not only based on static information attached to arcs but also takes into account the global properties of a route.

We showed that the use of budget intervals yields higher quality routes with less costs which implies the AOP unnecessarily generates detours. The budget interval could be used as well to search above the desired travel distance or travel time. That causes that routes will be found that the AOP never would find. We can increase the budget in the AOP, but than the desired costs will be exceeded often. Therefore the real problem is better moddelled with Quality-AOP.

We presented an algorithm which solves our definition of the Quality-AOP. The algorithm uses a bidirectional search to ensure the avoidence of multiple occurences of (twin) arcs. It generates a set of high quality routes. To speed up the process the Greedy Randomized Adaptive Search Procedure (GRASP) is used to choose a route and to prevent getting stuck in a local optimum. We showed the algorithm is designed to work with different score and

cost measures, and speed up techniques will work independently of the choice of measure. We implemented it in a generic way to allow a wide range of data structures that are used to store paths.

Our experiment shows that for the Quality-AOP we can find up to 10% higher quality routes if costs may vary 20%. The actual amount of improvement depends on the objective function which has been optimized. Shifting up the interval budget in a way that we search, 10% below budget and 10% above the budget, we find up to 11% improvement. In our case it only adds one kilometer to the desired travel distance at average. For higher budgets and/or larger budget intervals, we see that we can find better routes in the Quality-AOP. Our experiment also shows that the AOP generates detours as a result of too much optimization, just as we have shown theoretically. Although the amount of detour is small, the qualitative analysis shows that exactly this small detour introduces artifacts which a traveler does not want. These artifacts are eliminated when using the Quality-AOP, due to the quality improvement of the objective property as well as the improvement of supported properties related to this objective. This is exactly what we expected.

The elimination of artifacts can be seen in the statistics as well, e.g. if we optimize travel speed, high speed roads are represented to a greater extent in the Quality-AOP compared to the AOP. If we optimize the curviness of the route, which is an interesting property for scenic routing, we also see that the Quality-AOP improves curviness as well as the supported properties. The supported properties are more sensitive to the objective speed than the objective curviness, but in both cases the objective as well as all the supported properties have been improved compared to the AOP.

Further research can be done. For example to find a faster algorithm that works more efficiently on large scale networks, including residental routes, but still avoids subtours and traveling twin arcs. Research on adding a third objective can be done so that routes closer to the traveler's desired travel time are preferred. Also the scenic properties could be expressed as multiple separated objectives. We discovered artifacts by comparing the AOP and the Quality-AOP, one can decide to eliminate these artifacts in an efficient way instead of implementing Quality-AOP.

Concluding, the Quality-AOP fits the scenic routing problem better than the AOP in the sense we can find a route of higher quality and which approximates the desired travel distance or travel time given by an interval.

References

[1] Majid Alivand and Hartwig Hochmair. Extracting scenic routes from VGI data sources. *Proceedings of the Second ACM SIGSPATIAL International Workshop on Crowdsourced and Volunteered Geographic Information - GEOCROWD '13*, pages 23–30, 2013.

[2] 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, 2015.

[3] Julián Aráoz, Elena Fernández, and Cristina Zoltan. Privatized rural postman problems. *Computers & Operations Research*, 33(12):3432–3449, 2006.

[4] Julián Aráoz, Elena Fernández, and Oscar Meza. Solving the prize-collecting rural postman problem. *European Journal of Operational Research*, 196(3):886–896, 2009.

[5] Claudia Archetti and M Grazia Speranza. Arc routing problems with profits. *Arc Routing: Problems, Methods, and Applications, MOS-SIAM Series on Optimization*, pages 257–284, 2013.

[6] Lisa Aultman-Hall, Fred Hall, and Brian Baetz. Analysis of Bicycle Commuter Routes Using Geographic Information Systems: Implications for Bicycle Planning. *Transportation Research Record*, 1578(970168):102–110, 1997.

[7] Egon Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.

[8] Abraham Charnes and William W Cooper. Programming with linear fractional functionals. *Naval Research Logistics (NRL)*, 9(3-4):181–186, 1962.

[9] Thomas H. Cormen, Charles Eric Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[10] George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954.

[11] Mauro Dell'Amico, Francesco Maffioli, and Peter Värbrand. On prize-collecting tours and the asymmetric travelling salesman problem. *International Transactions in Operational Research*, 2(3):297–308, 1995.

[12] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[13] Dominique Feillet, Pierre Dejax, and Michel Gendreau. The profitable arc tour problem: solution with a branch-and-price algorithm. *Transportation Science*, 39(4):539–552, 2005.

[14] Dominique Feillet, Pierre Dejax, and Michel Gendreau. Traveling salesman problems with profits. *Transportation Science*, 39(2):188–205, 2005.

[15] Thomas A Feo and Mauricio GC Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

[16] Damianos Gavalas, Charalampos Konstantopoulos, Konstantinos Mastakas, Grammati Pantziou, and Nikolaos Vathis. Approximation algorithms for the arc orienteering problem. *Information Processing Letters*, 115(2):313–315, 2015.

[17] Bruce L Golden, Larry Levy, and Rakesh Vohra. The orienteering problem. *Naval Research Logistics*, 34(3):307–318, 1987.

[18] Reginald G Golledge. Path Selection and Route Preference in Human Navigation: A Progress Report. *Spatial Information Theory A Theoretical Basis for GIS International Conference COSIT 95 Semmering Austria Proceedings*, 988/1995(277):207–222, 1995.

[19] H Hochmair. Towards a classification of route selection criteria for route planning tools. *Developments in Spatial Data Handling*, pages 481—-492 676, 2005.

[20] Gilbert Laporte and Silvano Martello. The selective travelling salesman problem. *Discrete Applied Mathematics*, 26(2-3):193–207, 1990.

[21] Bingdong Li, Jinlong Li, Ke Tang, and Xin Yao. Many-Objective Evolutionary Algorithms. *ACM Computing Surveys*, 48(1):1–35, 2015.

[22] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search: Framework and applications. In *Handbook of Metaheuristics*, pages 363–397. Springer, 2010.

[23] Ying Lu and Cyrus Shahabi. An arc orienteering algorithm to find the most scenic path on a large-scale road network. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 46. ACM, 2015.

[24] Joris Maervoet, Pascal Brackman, Katja Verbeeck, Patrick De Causmaecker, and Greet Vanden Berghe. Tour suggestion for outdoor activities. In *International Symposium on Web and Wireless Geographical Information Systems*, pages 54–63. Springer, 2013.

[25] R Mansini, M Pelizzari, and R Wolfer. A granular variable neighbourhood search heuristic for the tour orienteering problem with time windows. Technical report, Technical Report RT 2006-02-52, University of Brescia, Italy, 2006.

[26] Damien Mercier, Pierre Schaus, Michael Saint-Guillain, and Yves Deville. PleasantTourFinder: An application to find the most pleasant tour from a given location. 2016.

[27] Clair E Miller, Albert W Tucker, and Richard A Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.

[28] Gerhard Navratil. Curviness as a Parameter for Route Determination. *GI_Forum 2012_ Geovisualization, Society and Learning*, pages 355–364, 2012.

[29] Abolghasem Sadeghi Niaraki and Kyehyun Kim. Ontology based personalized route planning system using a multi-criteria decision making approach. *Expert Systems with Applications*, 36(2 PART 1):2250–2259, 2009.

[30] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . `https://www.openstreetmap.org`, 2017.

[31] Xiaochun Qin, Michael J Meitner, Brent C Chamberlain, Xiaoning Zhang, and Forest Sciences Centre. Estimating Visual Quality of Scenic Highway using GIS and Landscape Visualizations. 2010.

[32] Ilan. Salomon and Patricia L. Mokhtarian. University of California Transportation University of California. *Transportation Research-D*, 2(2):107–123, 1997.

[33] Wouter Souffriau, Pieter Vansteenwegen, Greet Vanden Berghe, and Dirk Van Oudheusden. The planning of cycle trips in the province of East Flanders. *Omega*, 39(2):209–213, 2011.

[34] Thomas Stützle. Simulated annealing, dynamic local search, grasp, iterated greedy- an overview. *MN Summer School, Tenerife Google Scholar*, 2003.

[35] Theodore Tsiligirides. Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, pages 797–809, 1984.

[36] Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1 – 10, 2011.

[37] C. Verbeeck, P. Vansteenwegen, and E. H. Aghezzaf. An extension of the arc orienteering problem and its application to cycle trip planning. *Transportation Research Part E: Logistics and Transportation Review*, 68:64–78, 2014.

[38] Yan-Tao Zheng, Shuicheng Yan, Zheng-Jun Zha, Yiqun Li, Xiangdong Zhou, Tat-Seng Chua, and Ramesh Jain. GPSView. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 9(1):1–18, 2013.