Alternative Route Graphs in Road Networks*

Roland Bader[1], Jonathan Dees[1,2], Robert Geisberger[2], and Peter Sanders[2]

[1] BMW Group Research and Technology, 80992 Munich, Germany.
[2] Karlsruhe Institute of Technology, 76128 Karlsruhe, Germany.

Abstract. Every human likes choices. But today's fast route planning algorithms usually compute just a single route between source and target. There are beginnings to compute *alternative routes*, but there is a gap between the intuition of humans what makes a good alternative and mathematical definitions needed for grasping these concepts algorithmically. In this paper we make several steps towards closing this gap: Based on the concept of an *alternative graph* that can compactly encode many alternatives, we define and motivate several attributes quantifying the quality of the alternative graph. We show that it is already NP-hard to optimize a simple objective function combining two of these attributes and therefore turn to heuristics. The combination of the refined penalty based iterative shortest path routine and the previously proposed Plateau heuristics yields best results. A user study confirms these results.

1 Introduction

The problem of finding the shortest path between two nodes in a directed graph has been intensively studied and there exist several methods to solve it, e.g. Dijkstra's algorithm [1]. In this work, we focus on graphs of road networks and are interested not only in finding *one* route from start to end but to find *several* good alternatives. Often, there exist several noticeably different paths from start to end which are almost optimal with respect to length (travel time). There are several reasons why it can be advantageous for a human to choose his or her route from a set of alternatives. A person may have personal preferences or knowledge for some routes which are unknown or difficult to obtain, e.g. a lot of potholes. Also, routes can vary in different attributes beside travel time, for example in toll pricing, scenic value, fuel consumption or risk of traffic jams. The trade-off between those attributes depends on the person and the persons situation and is difficult to determine. By computing a set of good alternatives, the person can choose the route which is best for his or her needs.

There are many ways to compute alternative routes, but often with a very different quality. In this work, we propose new ways to measure the quality of a solution of alternative routes by mathematical definitions based on the graph structure. Also, we present several different heuristics for computing alternative routes as determining an optimal solution is NP-hard in general.

* Partially supported by DFG grant SA 933/5-1, and the 'Concept for the Future' of Karlsruhe Institute of Technology within the framework of the German Excellence Initiative.

1.1 Related Work

This paper is based on the MSc thesis of Dees [2]. A preliminary account of some concepts has been published in [3]. Computing the k-shortest paths [4,5] as alternative routes regards sub-optimal paths. The computation of disjoint paths is similar, except that the paths must not overlap. [6] proposes a combination of both methods: The computation of a shortest path, that has at most r edges in common with the shortest path. However, such paths are expensive to compute.

Other researchers have used edge weights to compute Pareto-optimal paths [7,8,9]. Given a set of weights, a path is called Pareto-optimal if it is better than any other paths for respectively at least one criteria. All Pareto-optimal paths can be computed by a generalized Dijkstra's algorithm.

The *penalty* method iteratively computes shortest paths in the graph while increasing certain edge weights [10]. [11] present a speedup technique for shortest path computation including edge weight changes.

Alternatives based on two shortest paths over a single *via node* are considered by the Plateau method [12]. It identifies fast highways (plateaus) which define a fastest route from s to t via the highway (plateau). [13] presents a heuristic to speedup this method using via node selection combined with shortest paths speedup techniques and proposing conservative conditions of an *admissible alternative*. Such a path should have bounded stretch, even for all subpaths, share only little with the shortest path and every subpath up to a certain length should be optimal.

2 Alternative Graphs

Our overall goal is to compute a set of alternative routes. However, in general, they can share nodes and edges, and subpaths of them can be combined to new alternative routes. So we propose the general definition of an *alternative graph* (AG) that is the union of several paths from source to target. More formally, let $G = (V, E)$ be a graph with edge weight function $w : E \to \mathbb{R}_+$. For a given source node s and target node t an AG $H = (V', E')$ is a graph with $V' \subseteq V$ such that for every edge $e \in E'$ there exists a simple s-t-path in H containing e, and no node is isolated. Furthermore, for every edge (u, v) in E' there must be a path from u to v in G; the weight of the edge $w(u, v)$ must be equal to the path's weight.

A *reduced* AG is defined as an AG in which every node has indegree $\neq 1$ or outdegree $\neq 1$ and thus provides a very compact encoding of all alternatives contained in the AG. Here, we focus on the computation of (reduced) AGs. We leave the extraction of actual paths from the AG as a separate problem but note that even expensive algorithms can be used since the AGs will be very small.

3 Attributes to Measure in AGs

For an AG $H = (V', E')$ we measure the following attributes

$$\begin{aligned}
\mathsf{totalDistance} &:= \sum_{e=(u,v)\in E'} \frac{w(e)}{d_H(s,u) + w(e) + d_H(v,t)} \\
\mathsf{averageDistance} &:= \frac{\sum_{e\in E'} w(e)}{d_G(s,t) \cdot \mathsf{totalDistance}} \\
\mathsf{decisionEdges} &:= \sum_{v\in V'\setminus\{t\}} \mathsf{outdegree}(v) - 1
\end{aligned}$$

where d_G denotes the shortest path distance in graph G. The total distance measures the extend to which the routes defined by the AG are nonoverlapping – reaching its maximal value of k when the AG consists of k disjoint paths. Note that the scaling by $d_H(s,u) + w(e) + d_H(v,t)$ is necessary because otherwise, long, nonoptimal paths would be encouraged. The average distance measures the path quality directly as the average stretch of an alternative path. Here, we use a way of averaging that avoids giving a high weight to large numbers of alternative paths that are all very similar. Finally, the decision edges measure the complexity of the AG which should be small to be digestible for a human. Appendix A gives examples explaining why considering only two out of three of these attributes can lead to meaningless results.

Usually, we will limit the number $\mathsf{decisionEdges}$ and $\mathsf{averageDistance}$ and under these constraint maximize $\mathsf{totalDistance} - \alpha(\mathsf{averageDistance} - 1)$ for some parameter α.

Optionally, we suggest a further attribute to measure based on

$$\mathsf{variance} = \int_0^1 (\mathsf{totalDistance} - \#\mathsf{edges}(x))^2 dx$$

where $\#\mathsf{edges}(x)$ denotes the number of edges (u, v) at position x, i.e. for which there is a path in the AG including (u, v) such that

$$\frac{d_H(s,u)}{d_H(s,u) + d_H(u,t)} \le x < \frac{d_H(s,v)}{d_H(s,v) + d_H(v,t)} \quad .$$

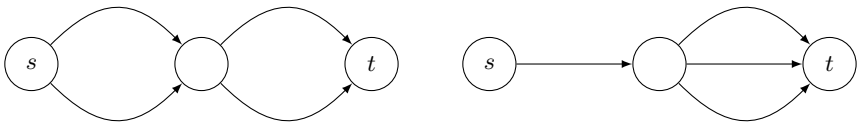
For normalization, we compute the coefficient of variation

$$\mathsf{CoV} = \sqrt{\mathsf{variance}} / \int_0^1 \#\mathsf{edges}(x)\, dx \quad .$$

Fig. 1 gives an example showing that small variance can distinguish between AGs that would otherwise be indistinguishable.

There are also other attributes that seem reasonable at the first glance, but they are problematic at a closer look:

– Counting the number of paths overestimates the influence of a large number of variants of the same basic route that only differ in small aspects.



(a) (b)

Fig. 1: Left graph: better distribution of alternatives

– Averaging path lengths over all paths in the AG or looking at the expected length of a random walk in the AG similarly overemphasizes small regions in the AG with a large number of variants.
– The *area* of the alternative graph considering the geographical embedding of nodes and edges within the plane is interesting because a larger area might indicate more independent paths, e.g., with respect to the spread of traffic jams. However, this requires additional data not always available.

It is also instructive to compare our attributes with the criteria for admissible alternative paths used in [13]. Both methods limit the length of alternative paths as some multiple of the optimal path length. The overlap between paths considered in [13] has a similar goal as our total distance attribute. An important difference is that we consider entire AGs while [13] considers one alternative path at a time. This has the disadvantage that the admissibility of a sequence of alternative paths may depend on the order in which they are inserted. We do not directly impose a limitation on the suboptimality of subpaths which plays an important role in [13]. The reason is that it is not clear how to check such a limitation efficiently – [13] develops approximations for paths of the form PP' where both P and P' are shortest paths but this is not the case for most of the methods we consider. Instead, we have developed postprocessing routines that remove edges from the AG that represent overly long subpaths, see Section 4.6. Results using the quality measures of [13] can be found in Appendix C.

4 Methods to Compute Alternatives

A meaningful combination of measurements is NP hard to optimize (see Appendix B for details). Therefore, we restrict ourselves to heuristics to compute an AG. These heuristics all start with the shortest path and then gradually add paths to the AG. We present several known methods and some new ones.

4.1 k-Shortest Paths

A widely used approach [4,5] is to compute the k shortest paths between s and t. This follows the idea that also slightly suboptimal paths are good. However, the computed routes are usually so similar to each other that they are not

considered as distinct alternatives by humans. Computing all shortest paths up to a number k produces many paths that are almost equal and do not "look good". Good alternatives occur often only for k being very large. Consider the following situation: There exist two long different highways from s to t, where the travel time on one highway is 5 minutes longer. To reach the highways we need to drive through a city. For the number of different paths through the city to the faster highway which travel time is not more than 5 minutes longer than the fastest path, we have a combinatorial explosion. The number of different paths is exponential in the number of nodes and edges in the city as we can independently combine short detours (around a block) within the city. It is not feasible to compute all shortest paths until we discover the alternative path on the slightly longer highway. Furthermore, there are no practically fast algorithms to compute the k shortest path. We consider this method rather impractical for computing alternatives.

4.2 Pareto

A classical approach to compute alternatives is Pareto optimality. In general, we can consider several weight functions for the edges like travel time, fuel consumption or scenic value. But even if we restrict ourselves to a single primary weight function, we can find alternatives by adding a secondary weight function that is zero for edges outside the current AG and the identical to the primary edge weight for edges inside the AG. Now a path is Pareto-optimal if there is no other path which is better with respect to both weight functions. Computing all Pareto-optimal paths now yields all sensible compromises between primary weight function and overlap with the current AG. All Pareto-optimal paths in a graph can be computed by a generalized Dijkstra algorithm [7,8] where instead of a single tentative distance, each node stores a set of Pareto-optimal distance vectors. The number of Pareto-optimal paths can be quite large (we observe up to ≈ 5000 for one s-t-relation in our Europe graph). We decrease the number of computed paths by tightening the domination criteria to keep only paths that are sufficiently different. We suggest two methods for tightening described in [9]. All paths that are $1 + \varepsilon$ times longer than the shortest path are dominated. Furthermore, all paths whose product of primary and secondary weight is $1/\gamma$ times larger than another path are dominated. This keeps longer paths only if they have less sharing. ε and γ are tuning parameters. We compute fewer paths for smaller ε and larger γ. But still we do not find suboptimal paths, as non-dominant paths are ignored. Note that the Pareto-method subsumes a special case where we look for completely disjoint paths.

As there may be too many Pareto-optimal alternatives, resulting in a large decisionEdges variable, we select an interesting subset. We do this greedily by iteratively adding that path which optimizes our objective function for the AG when this path is added.

4.3 Plateau

The Plateau method [12] identifies fast highways (plateaus) and selects the best routes based on the full path length and the highway length. In more detail, we perform one regular Dijkstra [1] from s to all nodes and one backward Dijkstra from t which uses all directed edges in the other direction. Then, we intersect the shortest path tree edges of both Dijkstra's. The resulting set consists of simple paths. We call each of those simple paths a *plateau*. All nodes not represented in a simple path form each an plateau of length 0. As there are too many plateaus, we efficiently need to select the best alternative paths derived from the plateau. Therefore, we rank them by the length of the corresponding s-t-path and the length of the plateau, i.e. rank = (path length − plateau length). A plateau reaching from s to t would be 0, the best value. To ensure that the shortest path in the base graph is always the first path, we can prefer edges in the shortest path tree rooted at s during the backward Dijkstra of t on a tie.

Plateau routes look good at first glance, although they may contain severe detours. In general, a plateau alternative can be described by a single via node. This is the biggest limitation of this method.

4.4 Penalty

We extend the iterative Penalty approach of [10]. The basic idea is to compute a shortest path, add it to our solution, increase the edge weights on this path and start from the beginning until we are satisfied with our solution.

The new shortest path is likely to be different from the last one, but not completely different, as some subpaths may still be shorter than a full detour (depending on the increase). The crucial point of this method is how we adjust the edge weights after each shortest path computation. We present an assortment of possibilities with which the combination results in meaningful alternatives.

First, we want to increase the edge weights of the last computed shortest path. We can add an absolute value on each edge of the shortest path [10], but this depends on the assembly and structure of the graph and penalizes short paths with many edges. We by-pass this by adding a fraction *penalty-factor* of the initial edge weight to the weight of the edge. The higher the factor (penalty), the more the new shortest path deviates from the last one.

Beside directly adding a computed shortest path to the solution, we can also first analyse the path. If the path provides us with a good alternative (e.g. is different and short enough), we add it to our solution. If not, we adjust the edge weights accordingly and recompute another shortest path.

Consider the following case: The first part of the route has no meaningful alternative but the second part has 5. That means that the first part of the route is likely to be increased several times during the iterations (*multiple-increase*). In this case, we can get a shortest path with a very long detour on the first part of the route. To circumvent this problem, we can limit the number of increases of a single edge or just lower successive increases. We are finished when a new

shortest path does not increase the weight of at least one edge. This provides us with a natural saturation of the number of alternatives.

The main limitation of the previous Penalty algorithm [10] is that the new shortest path can have many small detours (hops) along the route compared to the last path. Consider the following example: The last path is a long motorway and the new shortest path is *almost* equal to the last one, but at the middle of the motorway, it contains a very short detour (hop) from the long motorway on a less important road (due to the increase). There can occur many of those small hops; those look unpleasant for humans and contain no real alternative. In the AG, this increases the number of decision edges while having no substantial positive effect on other attributes. To alleviate this problem, we propose several methods: First, we cannot only increase the weights of edges on the path, but also of edges around the path (a tube). This avoids small hops, as edges on potential hops are increased and are therefore probably not shorter. The increase of the edges around the path should be decreasing with the distance to the path. Still, we penalize routes that are close to the shortest path, although there can be a long, meaningful alternative close to the shortest path. To avoid this, we can increase only the weights of the edges, which leave and join edges of the current AG. We call this increase *rejoin-penalty*. It should be additive and dependent on the general increase factor k and the distance from s to t, e.g. *rejoin-penalty* $\in [0..(\textit{penalty-factor}) \cdot 0.5 \cdot d(s,t)]$. This avoids small hops and reduces the number of decision edges in the AG. The higher the *rejoin-penalty*, the less decision edges in the alternative graph. In some cases, we want more decision edges at the beginning or the end of the route, for example to find all spur routes to the highways. Therefore, we can grade the *rejoin-penalty* according to the current position (cf. variance in Section 3). Another possibility to get rid of small hops is to allow them in the first place, but remove them later in the AG (Section 4.6).

A straightforward implementation of the Penalty method iteratively computes shortest paths using the Dijkstra algorithm. However, there are more sophisticated speedup techniques that can handle a reasonable number of increased edge weights [11]. Therefore we hope that we can efficiently implement the Penalty method.

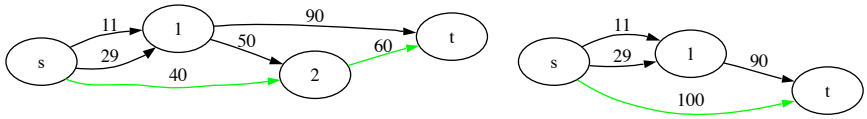
4.5 Combinations

In general, the Penalty method operates on a preexisting set of alternative routes and computes a new one. Therefore, a preprocess based on any other method is possible. Furthermore, the greedy selection strategy developed for the Pareto method could be applied to a set of paths computed by several methods. For example, the combination of the Plateau and Penalty method can produce an algorithm that is superior to a single one.

4.6 Refinements / Post Processing

The heuristics above often produce *reduced* alternative graphs that can be easily improved by local refinements that remove useless edges. We propose two

methods: Global Thinout focuses at the whole path from s to t, and Local Thinout only looks at the path between the edges. *Global Thinout* identifies useless edges (u, v) in the reduced alternative graph $G = (V, E)$ by checking for $d_G(s, u) + w(u, v) + d_G(v, t) \leq \delta \cdot d_G(s, t)$ for some $\delta \geq 1$. *Local Thinout* identifies useless edges in the reduced alternative graph $G = (V, E)$ by checking for $w(u, v) > \delta \cdot d_G(u, v)$ for some $\delta \geq 1$. After having removed edges with Local Thinout, we may further reduce G and find new locally useless edges. In contrast, Global Thinout finds all globally useless edges in the first pass. Also, we can perform Global Thinout efficiently by computing $d_G(s, \cdot)$ and $d_G(\cdot, t)$ using two runs of Dijkstra's algorithm. Fig. 2 illustrates Global Thinout by example.



(a) Base graph, shortest path is $\langle s, 2, t \rangle$ (b) Global thinout with $\delta = 1.2$

Fig. 2: Global Thinout: The only, and therefore the shortest, s-t-path including edge $(1, 2)$ has length 121, which is greater than $1.2 \cdot 100$. Therefore, edge $(1, 2)$ is removed. Every other edge is included in a s-t-path with weight below 120.

5 Different Edge Weights

The methods to compute an AG depend only on a single edge weight function (except Pareto). Therefore, we can use several different edge weight functions to independently compute AGs. The different edge weights are potentially orthogonal to the alternatives and can greatly enhance the quality of our computed alternatives. When we combine the different AGs into a single one, and want to compute its attributes of Section 3, we need to specify a main edge weight function, as the attributes also depend on the edge weights.

6 Experiments

We tested the proposed methods on a road network of Western Europe[3] with 18 029 721 nodes and 42 199 587 directed edges, which has been made available for scientific use by the company PTV AG. For each edge, its length and one out

[3] Austria, Belgium, Denmark, France, Germany, Italy, Luxembourg, the Netherlands, Norway, Portugal, Spain, Sweden, Switzerland, and the UK

of 13 road categories (e.g., motorway, national road, regional road, urban street) is provided so that an expected travel time can be derived. As k-Shortest Paths and normal Pareto are not feasible on this large graph, we also provide results just on the network of Luxembourg (30 732 nodes, 71 655 edges).

Hardware/Software. Two Intel Xeon X5345 processors (Quad-Core) clocked at 2.33 GHz with 16 GiB of RAM and 2x4MB of Cache running SUSE Linux 11.1. GCC 4.3.2 compiler using optimization level 3. For k-shortest path, we use the implementation from `http://code.google.com/p/k-shortest-paths/` based on [14], all other methods are new implementations.

Our experiments evaluate the introduced methods to compute AGs. We evaluate them by our base target function

$$\mathsf{totalDistance} - (\mathsf{averageDistance})$$

with constraints

$$\mathsf{averageDistance} \leq 1.1 \text{ and } \mathsf{decisionEdges} \leq 10 \ .$$

So we want the average distance to be at most 10% larger than the shortest path, providing us with short alternatives. Furthermore, there should not be more than 10 decision edges resulting in an clearly representable AG.

To compute an AG for a source/target pair, each method iteratively computes a new path until the constraints are violated and adds it to the AG. From this evolving set of AGs, the one with the best target function is chosen. As the Pareto method computes several paths at once, we use the greedy method to select the next path to add to the AG: We iteratively add the path which maximizes the target function while still satisfying the constraints. In our experiments with a few different *penalty-factors*, a factor of 0.4 without multi-increase and a factor of 0.3 with infinite multi-increase showed best performance. As rejoin-penalty, we use $0.005 \cdot$*penalty-factor*. We further combine the Penalty + Pareto method using the greedy selection strategy.

We use Global Thinout for refinement; Local Thinout has similar effects but is not as effective. As value for δ we choose 1.2 as it showed best performance. Our experiments showed that Global Thinout only improves the Penalty method with multi-increase, and the Plateau method. We will only report the best results.

The results based on 100 randomly selected source/target pairs are presented in Tab. 1. We see that the Penalty and Plateau method are clearly superior to the other methods. On Europe, Penalty is slightly better, as the Plateau method is limited to a single via node. We observe that the *rejoin-penalty* is a necessary ingredient of the Penalty method, as it increases the target function value by up to 48% on Europe. The best results are achieved when we combine Penalty and Plateau. We counted the number of paths contributed by both methods, showing that the Penalty method contributes 65% to the average AG and Plateau only 35%. The other tested methods are clearly dominated by these two methods. The Pareto method is slightly better than Disjoint and k-Shortest Paths, but the tightened domination criteria significantly reduces quality.

Table 1: Mean target function values.

Method	Thinout	Luxembourg	Europe
Penalty 0.4 rejoin + Plateau	∞	**3.29**	**3.70**
Penalty 0.3 rejoin multi-increase	1.2	2.85	3.34
Penalty 0.4 rejoin	∞	2.91	3.21
Penalty 0.3 multi-increase	1.2	2.77	2.25
Penalty 0.4	∞	2.75	2.47
Plateau	1.2	3.05	3.08
Pareto	∞	2.39	-
Pareto ($\varepsilon = 0.1, \gamma = 1.05$)	∞	1.69	2.02
Disjoint Paths	∞	1.10	1.12
k-Shortest Paths	∞	1.07	-

6.1 User Study

The experiments from above show that the Penalty and the Plateau method produce good results for our target function. However, we want to corroborate more objectively that the graphs which perform well at our target function are meaningful for humans. Every participant of the survey had to describe several (at least 2) meaningful motor vehicle routes for a start and destination pair. The described routes and the region should be known to the participant so that hopefully the given routes are meaningful. There were no restrictions on the length or on region of the routes. Given those routes, we assay whether our methods find most of the alternative routes, i.e. whether most of the routes are included in the (reasonable large) alternative graph. A methods perform well if it finds the alternative paths given by the survey participants.

The survey includes 79 alternatives for 26 different start and destination pairs (≈ 3 paths each), most of the routes are located in southern Germany. The distance of the pairs varies from 5 km up to 150 km.

Table 2: Reason for alternative paths (Survey)

Reason	Count
Faster at specific times	20
Route around (risk of) traffic jam	11
Proposed by route planer	12
Fast(er than proposed by route planer)	12
Relaxed driving/Easy route	10
n/a	14

Reason for alternatives. Tab. 2 shows a summary of the different reasons why a path is considered as a meaningful alternative route by the participant. The categories “Faster at specific times” and “Route around (risk of) traffic jam” are

very similar and are the most occurring reason (20+11), i.e. the alternative paths are dynamically chosen based on the time of day (or weekday) or the current traffic situation (sometimes even based on the current state of traffic lights).

Survey Evaluation. In order to compare the routes given by the survey participants to the routes of our methods, we convert them to routes of our graph data. For each start and destination pair, we obtain an alternative graph (called user graph) by merging the edges of the routes. After that, we compute an alternative graph for each $s-t$-pair with our method (called method graph). Note that the edge weights are not given by the survey. We use edge weights based on the travel time as it is the main reason to select an alternative. The Penalty method uses a *penalty-factor* of 0.4, with *rejoin-penalty*, and without *multiple-increase*, due to the best performance on Luxembourg (cf. Tab. 1).

Table 3: Penalty and Plateau Match Factor. The column "Matched" describes the mean fraction of the edge weights in the user graph, which are covered by the method graph. "Weight Factor" is the mean of the ratio, weight of the method graph to weight of the user graph.

# Iterations	Method	Matched	Weight Factor
2	Penalty	69%	0.91
2	Plateau	65%	0.88
3	Penalty	76%	1.21
3	Plateau	73%	1.18
3	Penalty+Plateau	81%	1.47

Results. In Tab. 3 we illustrate results for all of our test cases. The matching rate is around 70% for 2 iterations and and around 75% for 3. Penalty has a slightly better mean match factor, but the weight of the method graph is also slightly higher. The union of the method graphs increases the match factor to 81%. We consider this matching rates as indication of the usefulness of both methods. Appendix D shows example graphs of the survey.

7 Conclusion and Outlook

Our main contribution is a new way to characterize alternative routes that may look more natural to humans. The attributes defined for an alternative graph allow to measure the quality of a set of alternative routes. Furthermore, we compare methods to compute such AGs. The Plateau method and our improved version of the Penalty method showed best performance and clearly dominate the other tested methods.

The Penalty method has to be integrated with the dynamic speedup technique of [11]. There may be potential for further improvements compared to [11], as we we know that we have to consider all weight changes in the next query.

Also the Penalty method itself can be improved. The user often wants a choice of highways, but also a choice to reach these highways. Further improvements to the Penalty method can help to compute meaningful spur routes to the highways.

A geographic embedding of the AG allows a clearly representation of several alternative paths. To further improve the user experience, highlighting the differences between the alternatives could be added, e.g. showing points of interest along the routes. This allows the user to make a sound choice for the path along she will actually drive. The choice can be further supported by including previous choices and recommendations of other users.

References

1. Dijkstra, E.: A note on two problems in connexion with graphs. Numerische mathematik **1**(1) (1959) 269–271
2. Dees, J.: Computing Alternative Routes in Road Networks. Master's thesis, Karlsruhe Institut für Technologie, Fakultät für Informatik (April 2010)
3. Dees, J., Geisberger, R., Sanders, P., Bader, R.: Defining and Computing Alternative Routes in Road Networks. Technical report, ITI Sanders, Faculty of Informatics, Karlsruhe Institute of Technology (2010)
4. Eppstein, D.: Finding the k shortest paths. In: Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science (FOCS'94). (1994) 154–165
5. Yen, J.Y.: Finding the K Shortest Loopless Paths in a Network. Management Science **17**(11) (1971) 712–716
6. Scott, K.: Finding alternatives to the best path (1997)
7. Hansen, P.: Bricriteria Path Problems. In Fandel, G., Gal, T., eds.: Multiple Criteria Decision Making – Theory and Application –. Springer (1979) 109–127
8. Martins, E.Q.: On a Multicriteria Shortest Path Problem. European Journal of Operational Research **26**(3) (1984) 236–245
9. Delling, D., Wagner, D.: Pareto Paths with SHARC. In Vahrenhold, J., ed.: Proceedings of the 8th International Symposium on Experimental Algorithms (SEA'09). Volume 5526 of Lecture Notes in Computer Science., Springer (June 2009) 125–136
10. Chen, Y., Bell, M.G.H., Bogenberger, K.: Reliable pre-trip multi-path planning and dynamic adaptation for a centralized road navigation system. In: ITSC 2005 - 8th International
    IEEE Conference on Intelligent Transportation Systems, Vienna, IEEE (2007) 14–20
11. Schultes, D., Sanders, P.: Dynamic Highway-Node Routing. In Demetrescu, C., ed.: Proceedings of the 6th Workshop on Experimental Algorithms (WEA'07). Volume 4525 of Lecture Notes in Computer Science., Springer (June 2007) 66–79
12. CAMVIT: Choice routing (2009) http://www.camvit.com.
13. Abraham, I., Delling, D., Goldberg, A.V., Werneck, R.F.: Alternative Routes in Road Networks. In Festa, P., ed.: Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10). Volume 6049 of Lecture Notes in Computer Science., Springer (May 2010) 23–34
14. de Queiros Vieira Martins, E., Queir, E., Martins, V., Margarida, M., Pascoal, M.M.B.: A new implementation of yen's ranking loopless paths algorithm (2000)
15. Garey, M.R., Johnson, D.S.: Computers and intractability. Freeman (1979)

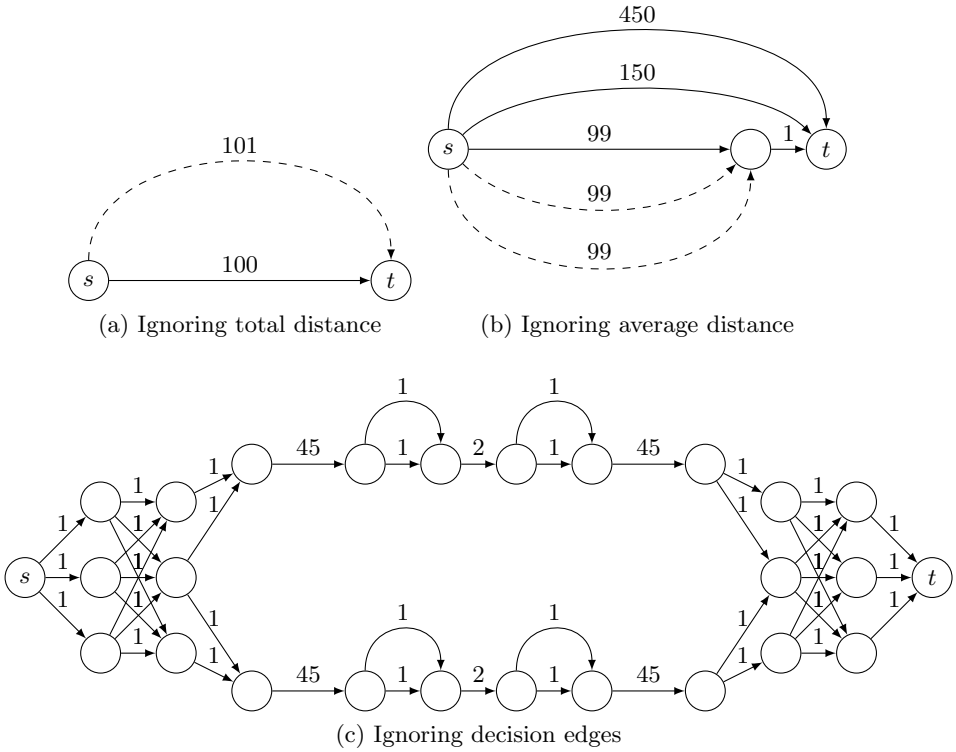
A Three Necessary Attributes

It is not meaningful to observe and optimize only two of the three attributes: total distance, average distance, and decision edges. We illustrate this for the absence of each of those attributes, see Fig. 3. The solid edges within the graphs build up the alternative graph.

If we optimize our alternative graph solely on average distance and decision edges, the best alternative graph consists of a shortest path from s to t, see Fig. 3a. The number of decision edges is 0 and the average distance is 1, which is optimal. However, the resulting graph does not include any alternative routes, although the dashed edge with weight 101 is a meaningful alternative.

If we optimize our alternative graph for total distance and decision edges, the best alternative graph consists of the union of a set of disjoint paths from s to t, see Fig. 3b. The total distance is one more (+1) than the decision edges, which is the best possible. The length of the single paths is not relevant and therefore long. However, this is not acceptable since there exist two considerably shorter alternatives (dashed edges with weight 99) resulting in almost the same total distance.

Fig. 3c illustrates an alternative graph which is optimized for total distance and average distance, but not for decision edges. This seems to be the best from all graphs in Fig. 3. The average distance is 1.0 (optimal) and the total distance is 2.24. However, there are some problems. There are many decision edges (21), most of them are almost useless. At the beginning and at the end, there is a grid of edges. In road networks, this can be due to the fact that s and t are located at positions with minor street classes (e.g. in a residential area). Grids of streets exist at these locations from which there are many junctions to the next higher class streets. The length of the paths through these grids are often similar. However, these paths are no real alternative and are not interesting for users (at least not all of the paths). If we list all of them, the alternative graph is quite similar to the original graph and this is not a useful proposal for alternative routes. Beside the grids, there are two alternative edges in the middle of the graph with weight 1, close to the edges, with weight 45. In road networks, this can happen at higher class streets having an intersection with smaller streets (e.g. a higher class street leading through a small town). At the intersection, there is a very small detour leaving the higher class street and immediately joining it again. This detour is normally no "real alternative" and not interesting. To get rid of the unwanted parts, we must also optimize decision edges. When optimizing, the number of split-ups should be low and thus every split should have advantages (in terms of total distance or average distance). With this (hopefully) only good branches will remain.



(a) Ignoring total distance

(b) Ignoring average distance

(c) Ignoring decision edges

Fig. 3: Optimizing two of: total distance, average distance or decision edges

B NP-Hardness of Objective Function

In this section we show that it is NP-hard to compute an alternative graph $\mathcal{H} = (V', E')$ for a given Graph $G = (V, E)$ and $s, t \in V$ which is optimal for (a simplified version of) the target function with constraints from Section 3. The target function is

$$\text{total distance}$$

with constraints

$$\text{average distance} \leq C^* \quad (C^* \in \mathbb{R}_+)$$
$$\text{decision edges} \leq N \quad (N \in \mathbb{N})$$

Actually, this is the same target function proposed for evaluating the quality of an alternative graph in Section 3 without the average distance in the target function (i.e. using $\alpha = 0$).

To show that the problem is NP-hard (i.e. its decision problem is NP-complete), we perform a reduction from KNAPSACK. KNAPSACK is a well known NP-complete problem [**?**]. It is defined as follows.

KNAPSACK	
INSTANCE:	n Elements with costs $c_i \in \mathbb{N}$ and profit $p_i \in \mathbb{N}$ with $i \in \{1, \ldots, n\}$, maximal cost $C \in \mathbb{N}$ and goal value $K \in \mathbb{N}$.
QUESTION:	Is there a set $B \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in B} c_i \leq C$ and $\sum_{i \in B} p_i \geq K$?

Our Problem to compute an alternative graph is stated as follows:

ALTERNATIVE GRAPH	
INSTANCE:	A graph $G = (V, E)$, $w : E \to \mathbb{R}_+$, $s, t \in V$ and $C^*, K^* \in R_+$, $N \in \mathbb{N}$.
QUESTION:	Is there an alternative Graph $\mathcal{H}$ such that total distance $\geq K^*$ and average distance $\leq C^*$ and decision edges $\leq N$?

Lemma 1. ALTERNATIVE GRAPH $\in NP$.

Proof (Proof). Given an instance of ALTERNATIVE GRAPH, we non-deterministically guess a solution. We are able to verify if this solution is a "Yes"-instance by computing the attributes for the alternative graph and check whether they meet the constraints in polynomial time, hence, ALTERNATIVE GRAPH is in NP.

Lemma 2. *Given a* KNAPSACK *instance I with n elements and N in $\{1 \ldots n\}$, we can build an* ALTERNATIVE GRAPH *instance I^*, such that, if and only if I is a "Yes"-instance which can be solved with N elements, I^* is a "Yes"-instance, too.*

Proof (Proof). We are given an KNAPSACK instance: n Elements with costs $c_i \in \mathbb{N}$ and profit $p_i \in \mathbb{N}$ with $i \in \{1, \ldots, n\}$, maximal cost $C \in \mathbb{N}$ and goal value $K \in \mathbb{N}$. Additionally, let N be a value $N \in \{1, \ldots, n\}$. Our goal is to transform this

instance into an Alternative Graph instance, where the transformed instance is a "Yes"-instance if and only if the Knapsack instance can be solved with N elements, i.e. solution B contains N elements. The idea is to transform each element of the knapsack instance into a branch in G of the Alternative Graph instance. A branch to the alternative graph increases the total distance (profit) and decision edges but also increases the average distance (costs). If a branch is added to the alternative graph, this means, we add the corresponding item to our Knapsack solution. The total distance models the profit constraint and the average distance the cost constraint of Knapsack. As the average distance does not increase by a constant additive value when adding a branch, we have to design the weights of the branches in G carefully. This is also the reason for limiting the solution to exactly N elements.

First, we scale the cost and the profits of the Knapsack instance:

$$p'_i := \frac{p_i}{\max_j p_j} \cdot \delta + 0.5$$

$$c'_i := \frac{c_i}{\max_j c_j} \cdot 0.1 + 0.6$$

where $0 < \delta < 0.1$ (defined later). With that, $p'_i \in [0.5, 0.6) \cap [0.5, \delta]$ and $c'_i \in [0.6, 0.7]$.

Without loss of generality, let p'_i be in non-decreasing order, i.e. $p'_i \leq p'_{i+1}$. Now we build G as follows: Let P be a simple path from s to t with length 1 consisting of several nodes. For every element i of the Knapsack instance there is a node i on P with distance ℓ_i from s. Beside the edges on the path P, for each i, we insert an edge from s to i with length $\ell_i + x_i$. This edge is called branch i and is a detour for a part of path p with length ℓ_i, the detour is x_i longer. To prevent multiple edges, we insert a node s' directly after s on P.

More formally, we build $G = (V, E)$, with

$$V = \{s, s', t\} \cup \{1, 2, \ldots, n\}$$

and

$$E = \{(s, s'), (s', 1)\} \cup \{(1, 2), \ldots, (n-1, n))\} \cup \{(n, t)\} \cup \{(s', 1), \ldots, (s', n)\}$$

and edge weights

$$w(s, s') = 0 \quad w(n, t) = 1 - \ell_n \quad w(i, i+1) = \ell_{i+1} - \ell_i \quad w(s', i) = \ell_i + x_i$$

with $0 \leq \ell_i \leq 1$ and $0 \leq x_i$.

Fig. 4 shows an example graph with three branches (so there are three corresponding items in Knapsack).

Note that every solution not including path p can easily be modified to include p, while all satisfied constraints remain satisfied. That means, we can always improve an alternative graphs and include path p, then only the included

branches differ. We define $A := \{i \mid \text{branch } (s,i) \text{ is in the alternative graph}\}$. Therefore our target function (total distance) can be written as

$$1 + \sum_{i \in A} p_i = 1 + \sum_{i \in A} \frac{\ell_i + x_i}{1 + x_i} \geq K^*$$

with the constraint (average distance)

$$\frac{1 + \sum_{i \in A} (\ell_i + x_i)}{1 + \sum_{i \in A} \frac{\ell_i + x_i}{1 + x_i}} \leq C^*$$

and constraint (decision edges)

$$|A| \leq N$$

If we add branch i to our alternative graph, the target function (total distance) is increased by $\frac{\ell_i + x_i}{1 + x_i}$ and the enumerator in the average distance formula is increased by $\ell_i + x_i$. We set their value dependent on the scaled profit and cost of the KNAPSACK instance:

$$\frac{\ell_i + x_i}{1 + x_i} := p'_i \; (= \frac{p_i}{\max_j p_j} \cdot \delta + 0.5)$$

$$\ell_i + x_i := c'_i \; (= \frac{c_i}{\max_j c_j} \cdot 0.1 + 0.6)$$

This is valid since there always exists a single solution for ℓ_i and x_i satisfying the equations and $0 \leq \ell_i \leq 1$ and $0 \leq x_i$, remembering that $p'_i \in [0.5, 0.6)$ and $c'_i \in [0.6, 0.7]$ (We have $x_i = \frac{c'_i}{p'_i} - 1$ and $\ell_i = c'_i - \frac{c'_i}{p'_i} + 1$).

With that, we set our constraints total distance

$$K^* := 1 + 0.5 \cdot N + \delta \cdot \frac{K}{\max_j p_j}$$

and average distance

$$C^* := (\frac{C}{\max_j c_j} \cdot 0.1 + N \cdot 0.6 + 1)/(1 + N \cdot 0.5)$$

To assure that a valid solution contains exactly N elements we set $\delta \cdot N < 0.5$. With that, the alternative graph must contain N branches to satisfy the total distance constraint. As already $|A| \leq N$ we have $|A| = N$ for all "Yes"-instances.

The total distance constraint of the ALTERNATIVE GRAPH instance is satisfied with N branches if and only if the profit constraint of the KNAPSACK instance is satisfied with the corresponding N elements, as the weights of the elements and the constraints are just linear scaled. However, the average distance constraint is more complex. Our goal is that each element adds up additively to the average distance similar to the KNAPSACK instance. However, the denominator in the average distance formula $(1 + \sum_{i \in A} \frac{\ell_i + x_i}{1 + x_i})$ is not constant. We first assume that all items in the denominator are equal to 0.5. In reality, the items

in the denominator can be larger, each up to $0.5 + \delta$. This weakens the average distance constraint. To be sure that this does not create additional solutions, a solution of the KNAPSACK instance violating the cost constraint must also violate the average distance constraint of the corresponding ALTERNATIVE GRAPH solution. We set

$$\delta := \min(\frac{1}{200N \max_j c_j}, 0.09, 0.5/(N+1))$$

We now show that, if the constraint in the KNAPSACK instance is *not* satisfied, neither is the constraint in the ALTERNATIVE GRAPH instance. Assume that the costs in the KNAPSACK instance are too high ($\sum c > C$), then $\sum c \geq C + 1$ and we need to show that

$$\begin{aligned}
&\frac{1 + N \cdot 0.6 + \frac{\sum c}{max_j c_j} 0.1}{1 + N \cdot (0.5 + \delta)} > C^* && \overset{(\sum c \geq C+1)}{\Leftarrow} \\
&\frac{1 + N \cdot 0.6 + \frac{C+1}{max_j c_j} 0.1}{1 + N \cdot (0.5 + \delta)} > \frac{\frac{C}{\max_j c_j} 0.1 + N \cdot 0.6 + 1}{1 + N \cdot 0.5} && \Leftrightarrow \\
&(\frac{1}{\max_j c_j} 0.1)(1 + N \cdot 0.5) > N\delta(\frac{C}{\max_j c_j} 0.1 + N \cdot 0.6 + 1) && \overset{(\delta \leq \frac{1}{200N \max_j c_j})}{\Leftarrow} \\
&1 + N \cdot 0.5 > \frac{C}{20 \max_j c_j} 0.1 + \frac{N \cdot 0.6}{20 \max_j c_j} + \frac{1}{20} && \Leftarrow \\
&\frac{1}{N} + 0.5 > \frac{C}{20 \max_j c_j \cdot N} + \frac{1}{20} + \frac{1}{20} && \overset{(C \leq N \cdot \max_j c_j)}{\Leftarrow}[4] \\
&0.5 > \frac{1}{20} + \frac{1}{20} + \frac{1}{20}
\end{aligned}$$

which is true.

Hence, every solution for the KNAPSACK instance satisfying the cost and profit constraint with N elements ("Yes"-instance) has a corresponding solution for the corresponding ALTERNATIVE GRAPH instance satisfying the average distance, total distance and decision edges constraint ("Yes"-instance). If there is no solution for the KNAPSACK instance with N elements, there is also no solution for the corresponding ALTERNATIVE GRAPH instance, since if there was a solution, there would be a corresponding KNAPSACK solution with N elements. Contradiction.

Lemma 3. KNAPSACK *is reducible to* ALTERNATIVE GRAPH

Proof (Proof). We are given an KNAPSACK instance with n elements. In Lemma 2, we have shown that, for every $N \in \{1, \ldots, n\}$, we can build a ALTERNATIVE GRAPH instance, such that if a solution for KNAPSACK with N elements exists, so

[4] or else $\sum c > C$ would not be possible

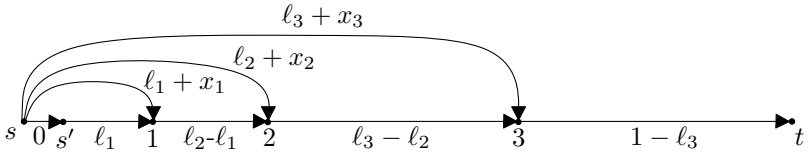


Fig. 4: Example transformation graph G, $d(s,t) = 1$

there exists a solution for Alternative Graph. Also, for every $N \in \{1, \ldots, n\}$, if no solution for Knapsack with N elements exits, there is also no solution for the build Alternative Graph instance. We observe that n is polynomial in the input size of the Knapsack instance. Therefore, we just build n Alternative Graph instances, one instance for every $N \in \{1, \ldots, n\}$. If any of those instances is a "Yes"-instance, the Knapsack instance is a "Yes"-instance, too. If none of those instances is a "No"-instance, the Knapsack instance is a "No"-instance, too. There is only a polynomial number of Alternative Graph instances. Hence, Knapsack is reducible to Alternative Graph in polynomial time.

Theorem 1. Alternative Graph *is NP-complete.*

Proof (Proof). Using Lemma 1 and Lemma 3, it directly follows that Alternative Graph is NP-complete.

C Path Quality Measures

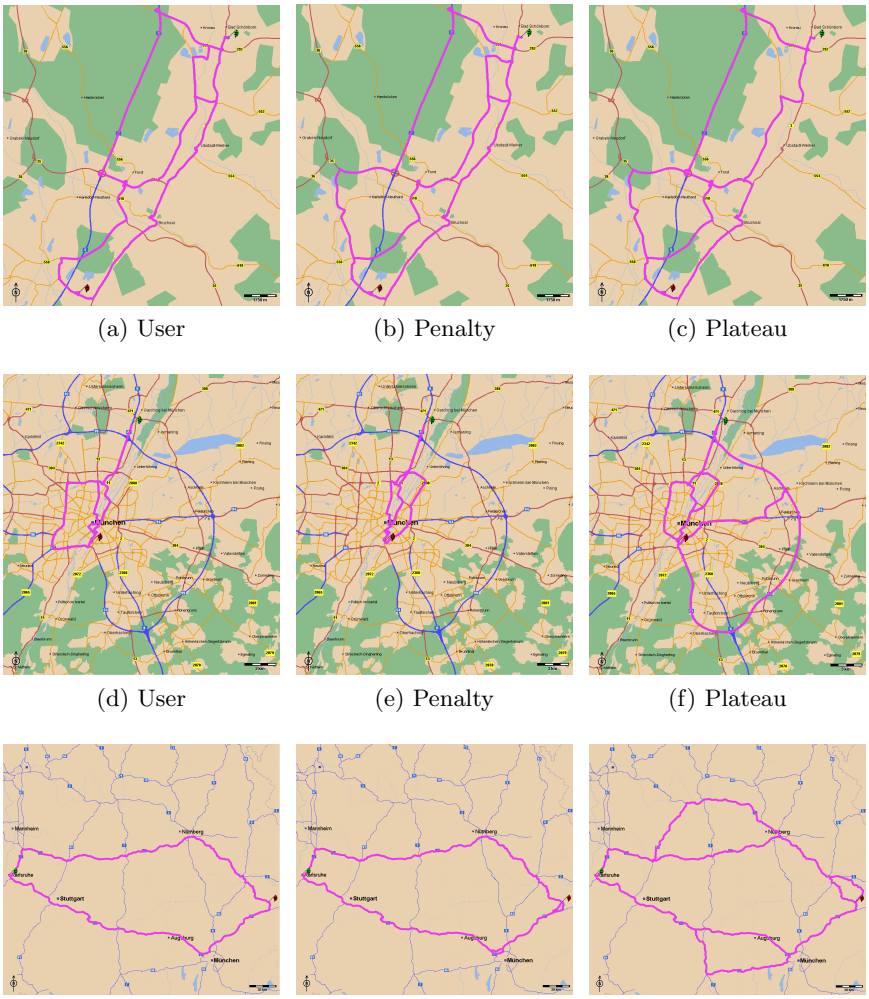
In Tab. 4 we present results using the different path quality measures of [13]. 'UBS' is the maximum stretch over all subpaths, 'sharing' is the percentage that are shared with the shortest path, and 'locality' is the percentage of the path length, such that every path of this length is optimal. We just measured the first path added during the iterative construction of the AG, although there are indications that not the first path we add is the best one with these quality measures. The values for Europe were based on just 100 paths, at it was very time-consuming to compute these values. The first three lines present the methods of [13]. We see that Penalty has worse UBS and locality, but the sharing is much smaller. So our alternatives are more different but at certain parts of the route we have a higher detour. The overall detour is not that large, as we restrict this to 10%. Plateau has similar results as the methods of [13], as they are both based on single via nodes. As we prefer long plateaus, locality is good. But we use a different selection function, resulting in larger UBS but lower sharing. We could not measure results with Global Thinout, as this refinement is applied to the AG and not on a specific path. Most likely, the results with Global Thinout are better.

Table 4: Different path quality measures.

Method	Europe		
	UBS[%]	sharing[%]	locality[%]
X-BDV	9.4	47.2	73.1
X-REV	9.9	46.9	71.8
X-CHV	10.8	42.9	72.3
Penalty 0.4 rejoin	21.1	16.5	11.3
Plateau	24.1	41.9	54.9

D Example Graphs of the Survey

Fig. 5 shows example graphs for three start and destination pairs of the survey. The user graphs on the left are given by survey participants. The graphs in the middle are computed by the Penalty method and the graphs on the right are computed by Plateau method with edge weight travel time. In the examples, both the Penalty and the Plateau graphs cover a significant part of the user graph. Still, some parts are not covered and some parts in the Penalty or Plateau graph do not occur in the user graph.



(a) User (b) Penalty (c) Plateau

(d) User (e) Penalty (f) Plateau

(g) User (h) Penalty (i) Plateau

Fig. 5: Sample Alternative Graphs of the Survey