Improved Alternative Route Planning

Andreas Paraskevopoulos, Christos Zaroliagis

▶ To cite this version:

Andreas Paraskevopoulos, Christos Zaroliagis. Improved Alternative Route Planning. ATMOS - 13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems - 2013, Sep 2013, Sophia Antipolis, France. pp.108–122, 10.4230/OASIcs.ATMOS.2013.108 . hal-00871739

HAL Id: hal-00871739
https://inria.hal.science/hal-00871739

Submitted on 10 Oct 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Improved Alternative Route Planning *

Andreas Paraskevopoulos[1,2] and Christos Zaroliagis[1,2]

1 **Computer Technology Institute & Press "Diophantus"**
Patras University Campus, 26504 Patras, Greece
2 **Department of Computer Engineering & Informatics**
University of Patras, 26504 Patras, Greece
{paraskevop,zaro}@ceid.upatras.gr

Abstract

We present improved methods for computing a set of alternative source-to-destination routes in road networks in the form of an alternative graph. The resulting alternative graphs are characterized by minimum path overlap, small stretch factor, as well as low size and complexity. Our approach improves upon a previous one by introducing a new pruning stage preceding any other heuristic method and by introducing a new filtering and fine-tuning of two existing methods. Our accompanying experimental study shows that the entire alternative graph can be computed pretty fast even in continental size networks.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G2.2.2 Graph Theory, G.4 Mathematical Software

Keywords and phrases Alternative route, stretch factor, shortest path, non-overlapping path, Penalty, Plateau

Digital Object Identifier 10.4230/OASIcs.ATMOS.2013.108

1 Introduction

Route planning services – offered by web-based, hand-held, or in-car navigation systems – are heavily used by more and more people. Typically, such systems (as well as the vast majority of route planning algorithms) offer a best route from an origin s to a destination t, under a single criterion (distance or time). Quite often, however, computing only one such s-t route may not be sufficient, since humans would like to have choices and every human has also his/her own preferences. These preferences may well vary and depend on specialized knowledge or subjective criteria (like or dislike certain part of a road), which are not always practical or easy to obtain and/or estimate (on a daily basis). Therefore, a route planning system offering a set of good/reasonable alternatives can hope that (at least) one of them can satisfy the user, and vice versa, the user can have them as back-up choices for altering his route in case of emergent traffic conditions. In all cases, the essential task is to compute reasonable alternatives to an s-t optimal route and this has to be done fast.

In this context, we are witnessing some recent research which investigates the computation of alternative routes under two approaches. The first approach, initiated in [5] and further extended in [19, 16], computes a few (2-3) alternative s-t routes that pass through specific nodes (called *via nodes*). The second approach [6] creates a set of reasonable alternative routes in the form of a graph, called *alternative graph*. Moreover, there are proprietary algorithms used by commercial systems (e.g., by Google and TomTom) that suggest alternative routes.

* This work was supported by the EU FP7/2007-2013 (DG CONNECT.H5-Smart Cities & Sustainability), under grant agreement no. 288094 (project eCOMPASS).

© Andreas Paraskevopoulos and Christos Zaroliagis;
licensed under Creative Commons License CC-BY
13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'13).
Editors: Daniele Frigioni, Sebastian Stiller; pp. 108–122

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this work, we focus on computing alternative graphs, which appear to be more suitable for practical navigation systems [4, 18], since the approach with via-nodes may create higher (than required) overlapping and may not be always successful. The study in [6] quantified the quality characteristics of an alternative graph (AG), captured by three criteria. These concern the non-overlappingness and the stretch of the routes, as well as the number of decision edges (sum of node out-degrees) in AG. As it is shown in [6], all of them together are important in order to produce a high-quality AG. However, optimizing a simple objective function combining just any two of them is already an NP-hard problem [6]. Hence, one has to concentrate on heuristics. Four heuristic approaches were investigated in [6] with those based on Plateau [3], Penalty [7], and a combination of them to be the best.

In this paper, we extend the approach in [6] for building AGs in two directions. First, we introduce a pruning stage that precedes the execution (and it is independent) of any heuristic method, thus reducing the search space and hence detecting the nodes on shortest routes much faster. Second, we provide several improvements on both the Plateau and Penalty methods. In particular, we use a different approach for filtering plateaus in order to identify the best plateaus that will eventually produce the most qualitative alternative routes, in terms of minimum overlapping and stretch. We also introduce a practical and well-performed combination of the Plateau and Penalty methods with tighter lower-bounding based heuristics. This has the additional advantage that the lower bounds remain valid for use even when the edge costs are increased (without requiring new preprocessing), and hence are useful in dynamic environments where the travel time may be increased, for instance, due to traffic jams. Finally, we conducted an experimental study for verifying our methods on several road networks of Western Europe. Our experiments showed that our methods can produce AGs of high quality pretty fast.

This paper is organized as follows. Section 2 provides background information, including notation, formal problem definitions, and classic algorithms for the *single pair shortest path problem*. Section 3 surveys the methods for building alternative graphs, presented in [6]. Section 4 presents our proposed improvements for producing AGs of better quality. Section 5 presents a thorough experimental evaluation of our improved methods. Conclusions are offered in Section 6.

Recent related work. During preparation of our camera-ready version, we have been informed about a different approach on reducing the running time of the Penalty method [17], which is is based on Customizable Route Planning [8] and includes an iterative updating of shortest path heuristics through a multi-level partition, in order to accommodate the adjustments on the edge weights by the Penalty method.

2 Preliminaries

A *road network* can be modeled as a *directed graph* $G = (V, E)$, where each node $v \in V$ represents intersection points along roads, and each edge $e \in E$ represents road segments between pairs of nodes. Let $|V| = n$ and $|E| = m$.

We consider the problem of tracing alternative paths from a source node s to a target node t in G, with edge weight or cost function $w : E \to \mathbb{R}^+$. The essential goal is to obtain sufficiently different paths with optimal or near optimal cost.

2.1 Alternative Graphs

The aggregation of alternative paths between a source s and a target t can be captured by the concept of the *Alternative Graph*, a notion first introduced in [6]. An Alternative Graph

(AG) is defined as the union of several s-t paths. Formally, an AG $H = (V', E')$ is a graph, with $V' \subseteq V$, and such that $\forall e = (u, v) \in E'$, there is a P_{uv} path in G and a P_{st} path in H, so that $e \in P_{st}$ and $w(e) = w(P_{uv})$, where $w(P_{uv})$ denotes the weight or cost of path P_{uv}. Let $d(u, v) \equiv d_G(u, v)$ be the shortest distance from u to v in graph G, and $d_H(u, v)$ be the shortest distance from u to v in graph H.

Storing paths in an alternative graph AG makes sense, because in general alternative paths may share common nodes (including s and t) and edges. Furthermore, their subpaths may be combined to form new alternative paths.

In the general case, there may be several alternative paths from s to t. Hence, there is a need of filtering and rating all alternatives, based on certain quality criteria. The main idea of the quality criteria is to discard routes with poor rates. For this task, the following quality indicators were used in [6]:

$$totalDistance = \sum_{e=(u,v)\in E'} \frac{w(e)}{d_H(s,u) + w(e) + d_H(v,t)} \qquad (overlapping)$$

$$averageDistance = \frac{\sum_{e\in E'} w(e)}{d_G(s,t) \cdot totalDistance} \qquad (stretch)$$

$$decisionEdges = \sum_{v\in V'\setminus\{t\}} (outdegree(v) - 1) \qquad (size\ of\ AG)$$

In the above definitions, the *totalDistance* measures the extend to which the paths in AG are non-overlapping. Its maximum value is *decisionEdges*+1. This is equal to the number of all s-t paths in AG, when these are disjoint, i.e. not sharing common edges. The *averageDistance* measures the average cost of the alternatives compared with the shortest one (i.e. the stretch). Its minimum value is 1. This occurs, when every s-t path in AG has the minimum cost. Consequently, to compute a qualitative AG, one aims at high *totalDistance* and low *averageDistance*. The *decisionEdges* measures the size complexity of AG. In particular, the number of the alternative paths in AG, depend on the "decision branches" are in AG. For this reason, the higher the *decisionEdges*, the more confusion is created to a typical user, when he tries to decide his route. Therefore, it should be bounded.

2.2 Shortest path Heuristics

We review now some shortest path heuristics that will be used throughout the paper.

Forward Dijkstra. Recall that Dijkstra's algorithm [11] grows a full shortest path tree rooted at a source node s, by performing a breadth-first based search, exploring the nodes in G in increasing order of distance from s. More specifically, for every node u, the algorithm maintains a tentative distance from s of the current known s-u shortest path and the predecessor node *pred* of u on this path. The exploring and processing order of the nodes can be controlled and guided by a priority queue Q. In each iteration, the node u with the minimum tentative distance $d(s, u)$ is removed from Q and its outgoing edges are *relaxed*. More specifically, for any outgoing edge of u, $e = (u, v) \in E$, if $d(s, u) + w(e) < d(s, v)$ then it sets $d(s, v) = d(s, u) + w(e)$ and $pred(v) = u$. Because the distance from s is monotonically increasing ($w : E \to \mathbb{R}^+$) a node dequeued from Q becomes *settled*, receiving the minimum possible distance $d(s, v)$ from s. The algorithm terminates when the queue becomes empty or when a target t is settled (for single-pair shortest path queries). In the latter case the produced shortest path tree consists of nodes with $d(s, v) \leqslant d(s, t)$. We refer to Dijkstra's algorithm also as *forward Dijkstra*.

Backward Dijkstra. To discover the shortest paths from all nodes in G to a target node t we can use a backward version of Dijkstra's algorithm. The *backward Dijkstra* explores the nodes in G in increasing order of their distance to t, traversing the incoming edges of the nodes by the reverse direction. In this variant of Dijkstra's algorithm, the successors ($succ$) nodes are stored instead of the predecessor ones in order to orientate the direction of the built shortest path tree towards the target node t.

The following bidirectional and A* variants of Dijkstra's algorithm are used to reduce the expensive and worthless exploration on nodes that do not belong to a shortest s-t path.

Bidirectional Dijkstra. This bidirectional variant runs forward Dijkstra from s and backward Dijkstra from t, as two simultaneously auxiliary searches. Specifically, the algorithm alternates the forward search from s and the backward (reverse) search from t, until they meet each other. In this way, the full s-t shortest path is formed by combining a s-v shortest path computed by the forward search and a v-t shortest path computed by the backward search. Because two s-v and v-t shortest paths cannot necessarily build an entire shortest s-t path, additionally, there is a need to keep the minimum cost $w(s,v)+w(v,t)$ and the via node v from all current traced paths in the meeting points of the two searches. Let $d_s(u)=d(s,u)$ and $d_t(u)=d(u,t)$. The algorithm terminates after acquiring the correct s-t shortest path. This is ensured only when the current minimum distance in the priority queue of forward search Q_f and the minimum distance in the priority queue Q_b of backward search are such that $\min_{u\in Q_f}\{d_s(u)\}+\min_{v\in Q_b}\{d_t(v)\} > d_s(t)$, meaning that the algorithm cannot anymore provide shorter s-t paths than the previous discovered ones.

A* search. Given a source node s and a target node t, the A^* variant [15] is similar to Dijkstra's algorithm with the difference that the priority of a node in Q is modified according to a heuristic function $h_t : V \to \mathbb{R}$ which gives a lower bound estimate $h_t(u)$ for the cost of a path from a node u to a target node t. By adding this heuristic function to the priority of each node, the search becomes goal-directed pulling faster towards the target. The tighter the lower bound is, the faster the target is reached. The only requirement is that the h_t function must be monotone: $h_t(u) \le w(u,v)+h_t(v)$, $\forall(u,v)\in E$. One such heuristic function is the Euclidean distance between two nodes. But in general, Euclidean lower bounds are not the best approximations of shortest distances in road networks, because the majority of road routes do not follow a strict straight course from s to t.

ALT. The *ALT* technique, that introduced in [13], provides a highly effective heuristic function for the A^* algorithm, using triangle inequality and precomputed shortest path distances between all nodes and few important nodes, the so-called *landmarks*. Those shortest distances can be computed and stored in a preprocessing stage. Then, during a query, the lower bounds can be estimated in constant time. In particular, for a node v and a landmark L, it holds that $d(v,t) \geqslant \max_L\{d(L,t)-d(L,v), d(v,L)-d(t,L)\} = h_t(v)$ and $d(s,v) \geqslant max_L\{d(L,v)-d(L,s), d(s,L)-d(v,L)\} = h_s(v)$. Obviously, these lower bounds contain an important part of the information of the shortest path trees in G.

The efficiency of ALT depends on the number and the initial selection of landmarks. In order to have good query times, peripheral landmarks as far away from each other as possible must be selected, taking advantage of the fact that the road networks are (almost) planar. The nodes in these positions can cover more shortest path trees in G and hence provide more valuable heuristics.

We refer to the consistent bidirectional *ALT* algorithm, with the average heuristic function [13], as *BLA*. In this variant, the forward and backward search use H_s and H_t as heuristic functions, where $H_t(v) = -H_s(v) = \frac{h_t(v)-h_s(v)}{2}$.

3 Approaches for Computing Alternative Graphs

We briefly review the approaches considered in [6] for computing alternative graphs.

k-Shortest Paths. The k-shortest path routing algorithm [12, 22] finds k shortest paths in order of increasing cost. The disadvantage of this approach is that the computed alternative paths share many edges, which makes them difficult to be distinguished by humans. Good alternatives could be revealed for very large values of k, but at the expense of a rather high computational cost.

Pareto. The Pareto algorithm [14, 21, 10] computes an AG by iteratively finding *Pareto-optimal* paths on a suitably defined objective cost vector. The idea is to use as first edge cost the one of the single criterion problem, while the second edge cost is defined as follows: all edges belonging to AG (initially the AG is the shortest s-t path) set their second cost function to their initial edge cost and all edges not belonging to AG set their second cost function to zero.

Plateau. The Plateau method [3] provides alternative P_{st} paths by connecting pairs of s-v and v-t shortest paths, via a specific node v. In this matter, v is selected on the basis of whether it belongs to a plateau (to be defined shortly).

In particular, the s-v and v-t paths that are required to form the P_{st} paths can be found on a forward T_f shortest path tree, with root s, and a backward T_b shortest path tree, with root t. On this, a classical approach for finding T_f and T_b is by performing forward and backward Dijkstra. Apparently, from this process, connecting shortest subpaths does not necessarily ensure the optimality of the resulted P_{st} paths, so there is a need to evaluate them. In order to provide low overlapping alternative P_{st} paths, the connection-node v of s-v and v-t paths should belong to a plateau. The plateaus are simple paths, $\overline{P} \subseteq P_{st}$, consisting of more than one successive nodes, with the property that $\forall u, v \in \overline{P} : d_s(u) + d_t(u) = d_s(v) + d_t(v)$. The plateaus can be traced on the intersection of T_f and T_b. In this way, a node in a plateau following the predecessor nodes in T_f and the successor nodes in T_b can build a complete

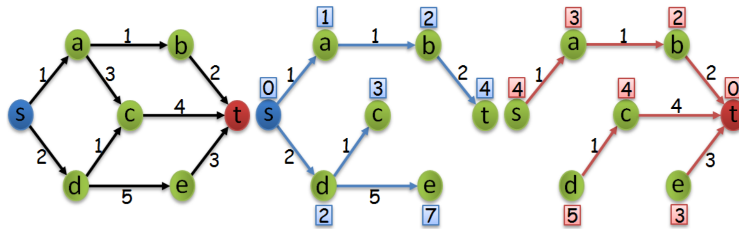


Figure 1 Graph G. Forward T_f shortest path tree with root s. Backward T_b shortest path tree with root t.

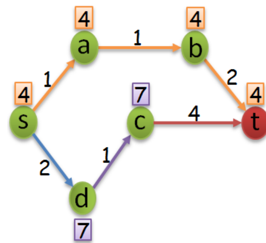


Figure 2 The combination of T_f and T_b trees. The resulted graph reveals two plateaus. The first one is s-a-b-t and the second one is d-c.

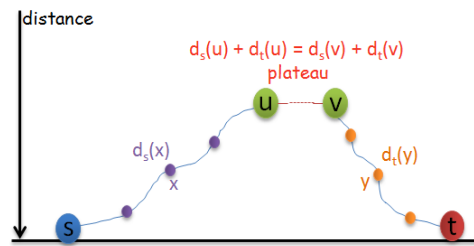


Figure 3 A Plateau.

P_{st} path. As the plateaus are usually too many, a filtering stage is used to select the best of them. In [6], this is implemented by gathering plateaus $\overline{P}$ in a non-decreasing order of $rank = w(P_{st}) - w(\overline{P})$, where P_{st} is the resulted path via $\overline{P}$. Therefore, a plateau that corresponds to a shortest path from s to t has rank zero, which is the best value.

Penalty. The Penalty method [7] provides alternative paths by iteratively running shortest path queries and adjusting the weight of the edges on the resulted P_{st} paths. The basic steps are the following. A shortest P_{st} path is computed with Dijkstra's algorithm or a speedup variation of it. Then, P_{st} is penalized by increasing the weight of its edges. Next, a new s-t query is executed. If the new computed P'_{st} path is short and different enough from the previously discovered P_{st} paths, then it is added to the solution set. The same process is repeated until a sufficient number of alternative paths (with the desired characteristics) is found, or the weight adjustments of s-t paths bring no better results.

In order to offer the best results, an efficient and safe way on weight increases should be adopted. A weight adjustment policy, also considered in [6], is as follows:

- The increase on the weights should be of a small magnitude in order to keep the resulted $averageDistance$ low. When an edge of a P_{st} path is about to be penalized, only a small fraction (penalty factor) $0.1 \leq p \leq 1$ of its initial weight is added, i.e., $w_{new}(e) = w(e) + p \cdot w_{old}(e)$. Note that the use of constant values is avoided, because they do not always guarantee a balanced adjustment, since in some cases longer edges may be favored over shortest ones. In general, the higher the penalty factor is, the more the new shortest path may differ from the last one. On the other hand, the lower the p penalty factor is, more shortest path queries can be performed and less alternative paths may be lost.
- The weight adjustment is restricted when it could lead to the loss of good alternatives. Notice that, an unbounded penalty leads to multiple increases on the edge weights and is risky. For example, suppose that there is only one fast highway into a city, whereas there are many alternatives through the city center. If we allow multiple increases on the weights of the highway then its cost will be increased several times during the iterations. This for new s-t shortest path queries may result to new computed paths that now begin from a detour longer than the highway. Therefore, because of the high cost any possible alternative inside the city will be lost, and the algorithm will terminate with poor results. To overcome this problem the number of the increases or the magnitude of p is limited for edges already included in AG.
- In order to avoid the overlapping between the computed alternative paths is useful to extend the weight adjustment to their neighborhood. This is reasonable, because in some cases the new computed alternative paths may share many small detours with the previous ones. For example, it is possible that the first path is a fast highway and the new computed paths are in the same course with the highway but having one or

many outgoing and incoming small detours distributed along the highway. This increases the *decisionEdges* and offers meaningless (non-discrete) alternatives. Therefore, when increasing the weight of the edges in a shortest P_{st} path, the weights of edges around P_{st} that leave and join the current AG should be additionally penalized (rejoin-penalty) by a factor $0.1 \leq r \leq 1$. Consequently, the rejoin-penalty r contributes to high *totalDistance*.

Thinout. A major issue is the optimality regarding the cost. In [6], the optimality is ensured by bounding the *averageDistance*, and further in a post-processing phase by setting tighter bounds to the local optimality of the edges or the subpaths in AG. In Plateau method, the local optimality of the s-v and v-t paths is guaranteed because these are selected from the T_f and T_b shortest path trees. In the penalty method, however, the adjustment of the weights may insert non optimal paths. A way in [6] to overcome this issue, when considering alternative paths globally, is by performing a *global refinement* (focusing on the entire s-t paths), and an iterative *local refinement* (focusing on individual edges). In more detail, for some $\delta > 1$, in global refinement, an edge $e = (u, v) \in E'$ is removed from AG if $d_H(s, u) + w(u, v) + d_H(v, t) > \delta \cdot d_H(s, t)$. In local refinement, an edge $e = (u, v) \in E'$ is removed from AG if $w(u, v) > \delta \cdot d_H(u, v)$.

4 Our improvements

As the experimental study in [6] showed, the k-Shortest Paths and the Pareto approaches generate alternative graphs of low quality and hence we shall not investigate them. On the other hand, the Plateau and Penalty methods are the most promising ones and thus we focus on extending and enhancing them. Our improvements are twofold :

A. We introduce a pruning stage that precedes the Plateau and Penalty methods in order to a-priori reduce their search space without sacrificing the quality of the resulted alternative graphs.

B. We use a different approach for filtering plateaus in order to obtain the ones that generate the best alternative paths. In addition, we fine tune the penalty method, by carefully choosing the penalizing factors on the so far computed P_{st} paths, in order to trace the next best alternatives.

4.1 Pruning

We present two bidirectional Dijkstra-based *pruners*. The purpose of both of them, is to identify the nodes that are in P_{st} shortest paths. We refer to such nodes, as the useful search space, and the rest ones, as the useless search space. Our goal, through the use of search pruners, is to ensure: (a) a more quality-oriented AG construction and (b) a reduced dependency of the time computation complexity from the graph size. The latter is necessary, in order to acquire fast response in queries. We note that the benefits are notably for the Penalty method. This is because, the Penalty method needs to run iteratively several s-t shortest path queries. Thus, having put aside the useless nodes and focussing only on the useful ones, we can get faster processing. We also note that, over the P_{st} paths with the minimum cost, may be desired as well to let in AG paths with near optimal cost, say $\tau \cdot d_s(t)$, which will be the maximum acceptable cost $w(P_{st})$. Indicatively, $1 \leqslant \tau \leqslant 1.4$. Obviously, nodes far away from both s and t, with $d_s(v) + d_t(v) > \tau \cdot d_s(t)$, belong to P_{st} paths with prohibitively high cost. In the following we provide the detailed description of both pruners, which is illustrated by Figures 4 and 5.

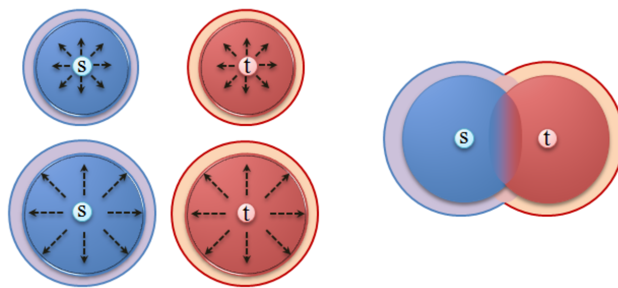


Figure 4 The forward and backward searches meet each other. In this phase the minimum distance $d_s(t)$ is traced.

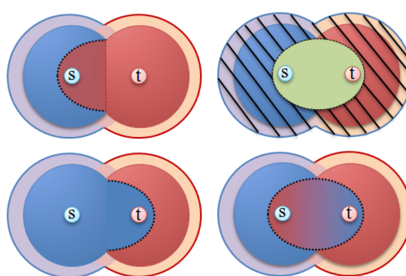


Figure 5 The forward and backward settles only the nodes in the shortest paths, taking account the overall $d_s(v) + d_t(v)$.

Uninformed Bidirectional Pruner. In this pruner, there is no preprocessing stage. Instead, the used heuristics are obtained from the minimum distances of the nodes enqueued in Q_f and Q_b, i.e. $Q_f.minKey() = \min_{u \in Q_f}\{d_s(u)\}$ and $Q_b.minKey() = \min_{v \in Q_b}\{d_t(v)\}$.

We extend the regular bidirectional Dijkstra, by adding one extra phase. First, for computing the minimum distance $d_s(t)$, we let the expansion of forward and backward search until $Q_f.minKey() + Q_b.minKey() \geq d_s(t)$. At this step, the current forward T_f and backward T_b shortest path trees produced by the bidirectional algorithm will have crossed each other and so the minimum distance $d_s(t)$ will be determined. Second, at the new extra phase, we continue the expansion of T_f and T_b in order to include the remaining useful nodes, such that $d_s(v) + d_t(v) \leq \tau \cdot d_s(t)$, but with a different mode. This time, we do not allow the two searches to continue their exploration at nodes v that have $d_s(v) + h_t(v)$ or $h_s(v) + d_t(v)$ greater than $\tau \cdot d_s(t)$. We use the fact that Q_f and Q_b can provide lower-bound estimates for $h_s(v)$ and $h_t(v)$. Specifically, a node that is not settled or explored from backward search has as a lower bound to its distance to t, $h_t(v) = Q_b.minKey()$. This is because the backward search settles the nodes in increasing order of their distance to t, and if u has not been settled then it must have $d_t(u) \geq Q_b.minKey()$. Similarly, a node that is not settled or explored from forward search has a lower bound $h_s(v) = Q_f.minKey()$. Furthermore, when a search settles a node that is also settled from the other search we can calculate exactly the sum $d_s(u) + d_t(u)$. In this case, the higher the expansion of forward and backward search is, the more tight the lower bounds become. The pruning is ended, when Q_f and Q_b are empty. Before the termination, we exclude the remaining useless nodes that both searches settled during the pruning, that is all nodes v with $d_s(v) + d_t(v) > \tau \cdot d_s(t)$.

Informed ALT bidirectional pruner. In the second pruner, our steps are similar, except that we use tighter lower bounds. We acquire them in an one-time preprocessing stage, using the *ALT* approach. In this case, the lower bounds that are yielded can guide faster and more accurately the pruning of the search space. We compute the shortest distances between the nodes in G and a small set of landmarks. For tracing the minimum distance $d_s(t)$, we use *BLA* as base algorithm, which achieves the lowest waste exploration, as experimental results showed in [13, 20]. During the pruning, we skip the nodes that have $d_s(v) + h_t(v)$ or $h_s(v) + d_t(v)$ greater than $\tau \cdot d_s(t)$.

The use of lower-bounding heuristics can be advantageous. In general, a heuristic stops being valid when a change in the weight of the edges is occurred. But note that in the penalty method, we consider only increases on the edge weights and therefore this does not affect the lower bounds on the shortest distances. Therefore, the combination of the *ALT* speedup [20, 13] with Penalty is suitable. However, depending on the number and the magnitude of the increases the lower bounds can become less tight for the new shortest distances, leading to a reduced performance on computing the shortest paths.

4.2 Filtering and Fine-tuning

Over the standard processing operations of Penalty and Plateau, we introduce new ones for obtaining better results. In particular:

Plateau. We use a different approach on filtering plateaus. Specifically, over the cost of a plateau path we take account also its non-overlapping with others. In this case, the difficulty is that the candidate paths may share common edges or subpaths, so the *totalDistance* is not fixed. Since at each step an insertion of the current best alternative path in AG may lead to a reduced *totalDistance* for the rest candidate alternatives, primarily we focus only on their unoccupied parts, i.e., those that are not in AG. We rank a x-y plateau $\overline{P}$ with $rank = totalDistance - averageDistance$, where $totalDistance = \frac{w(\overline{P})}{d_s(x)+w(\overline{P})+d_t(y)}$ is its definite non-overlapping degree, and $averageDistance = \frac{w(\overline{P})+d_s(t)}{(1+totalDistance)\cdot d_s(t)}$ is its stretch over the shortest s-t path in G. During the collection of plateaus, we insert the highest in rank of them via its node-connectors $v \in \overline{P}$ in T_f and T_b to a min heap with fixed size equal to *decisionEdges* plus an offset. The offset increases the number of the candidate plateaus, when there are available, and it is required only as a way out, in the case, where several P_{st} paths via the occupied plateaus in AG lead to low *totalDistance* for the rest P_{st} paths via the unoccupied plateaus.

Penalty. When we "penalize" the last computed P_{st} path, we adjust the increases on the weights of its outgoing and incoming edges, as follows:

$$w_{new}(e) = w(e) + (0.1 + r \cdot d_s(u)/d_s(t)) \cdot w_{old}(e), \quad \forall e = (u,v) \in E: \ u \in P_{st}, \ v \notin P_{st}$$
$$w_{new}(e) = w(e) + (0.1 + r \cdot d_t(v)/d_t(s)) \cdot w_{old}(e), \quad \forall e = (u,v) \in E: \ u \notin P_{st}, \ v \in P_{st}$$

The first adjustment puts heavier weights on those outgoing edges that are closer to the target t. The second adjustment puts heavier weights on those incoming edges that are closer to the source s. The purpose of both is to reduce the possibility of recomputing alternative paths that tend to rejoin directly with the previous one traced.

An additional care is given also for the nodes u in P_{st}, having $outdegree(u) > 1$. Note that their outgoing edges can form different branches. Since the edge-branches in G constitute generators for alternative paths, they are important. These edges are being inserted to AG with a greater magnitude of weight increase than the rest of the edges.

The insertion of the discovered alternative paths in G and the maintenance of the overall quality of AG should be controlled online. Therefore, we establish an online interaction with the AG's quality indicators, described in Section 2, for both Plateau and Penalty. This is also necessary because, at each step an insertion of the current best alternative may lead to a reduced value of *totalDistance* for the next candidate alternative paths that share common edges with the already computed AG.

In order to get the best alternatives, we seek to maximize the $targetfunction = totalDistance - \alpha \cdot averageDistance$, where α is a balance factor that adjusts the stretch magnitude rather than the overlapping magnitude. Maximization of the target function leads to select the best set of low overlapping and shortest alternative paths.

Since the penalty method can work on any pre-computed AG, it can be combined with Plateau. In this way, we collect the best alternatives from Penalty and Plateau, so that the resulting set of alternatives maximizes the target function. In this matter, we can extend the number of decision edges and after the gathering of all alternatives, we end by performing thinout in AG. Moreover, in order to guide the Penalty method to the remaining alternatives, we set a penalty on the paths stored by Plateau in AG, by increasing their weights. We also use the same pruning stage to accommodate both of them.

5 Experimental Results

The experiments were conducted on an Inte(R) Xeon(R) Processor X3430 @ 2.40GHz, with a cache size of 8Mb and 32Gb of RAM. Our implementations were written in C++ and compiled by GCC version 4.6.3 with optimization level 3.

The data sets of the road networks in our experiments were acquired from OSM [1] and TomTom [2]. The weight function is the travel time along the edges. In the case of OSM, for each edge, we calculated the travel time based on the length and category of the roads (residential street, tertiary, secondary, primary road, trunk, motorway, etc). The data set of the Greater Berlin area was kindly provided by TomTom in the frame of the eCOMPASS project [4]. The size of the data sets are reported in Table 1.

For our implementations, we used the packed-memory graph (PMG) structure [20]. This is a highly optimized graph structure, part of a larger algorithmic framework, specifically suited for very large scale networks. It provides dynamic memory management of the graph and thus the ability to control the storing scheme of nodes and edges in memory for optimization purposes. It supports almost optimal scanning of consecutive nodes and edges and can incorporate dynamic changes in the graph layout in a matter of μs. The ordering of the

Table 1 The size of road networks.

map		n	m
B	Berlin	117,839	310,152
LU	Luxembourg	51,576	119,711
BE	Belgium	576,465	1,376,142
IT	Italy	2,425,667	5,551,700
GB	GreatBritain	3,233,096	7,151,300
FR	France	4,773,488	11,269,569
GE	Germany	7,782,773	18,983,043
WE	WesternEurope	26,498,732	62,348,328

nodes and edges in memory is in such a way that increases the locality of references, causing as few memory misses as possible and thus a reduced running time for the used algorithms.

We tested our implementations in the road network of the Greater Berlin area, the Western Europe (Austria, Belgium, Denmark, France, Germany, Italy, Luxembourg, Netherlands, Norway, Portugal, Spain, Sweden, Switzerland, and Great Britain), as well as in the network of each individual West European country. In the experiments, we considered 100 queries, where the source s and the destination t were selected uniformly at random among all nodes. For the case of the entire Western Europe's road network, the only limitation is that the s-t queries are selected, such that their geographical distance is at most 300 kilometers. This was due to the fact that although modern car navigation systems may store the entire maps, they are mostly used for distances up to a few hundred kilometers.

For far apart source and destination, the search space of the alternative P_{st} paths gets too large. In such cases, it is more likely that many non-overlapping long (in number of edges) paths exist between s and t. Therefore, this has a major effect on the computation cost of the overall alternative route planning. In general, the number of non-overlapping shortest paths depends on the density of the road networks as well on the edge weights.

There is a trade-off between the quality of AG and the computation cost. Thus, we can sacrifice a bit of the overall quality to reduce the running time. Consequently, in order to deal with the high computation cost of the alternative route planning for far apart sources and destinations we can decrease the parameter τ (max stretch). A dynamic and online adjustment of τ based on the geographical distance between source and target can be used too. For instance, at distance larger than 200km, we can set a smaller value to τ, e.g., close to 1, to reduce the stretch and thereby the number of the alternatives. We adopted this arrangement on large networks (Germany, Western Europe). In the rest, we set $\tau = 1.2$, which means that any traced path has cost at most 20% larger than the minimum one. To all road networks, we also set $averageDistance \leq 1.1$ to ensure that, in the filtering stage, the average cost of the collected paths is at most 10% larger than the minimum one.

In order to fulfill the ordinary human requirements and deliver an easily representable AG, we have bounded the *decisionEdges* to 10. In this way, the resulted AG has small size, $|V'| \ll |V|$ and $|E'| \ll |E|$, thus making it easy to store or process. Our experiments showed that the size of an AG is at most 2 to 3 times the size of a shortest s-t path, which we consider as a rather acceptable solution.

Our base *target function* [1] in Plateau and Penalty is $totalDistance - averageDistance + 1$. Regarding the pruning stage of Plateau and Penalty, we have used the ALT-based informed bidirectional pruner with at most 24 landmarks for Western Europe.

In Tables 2, 3, and 4, we report the results of our experiments on the various quality indicators: targetFunction (*TargFun*), totalDistance (*TotDist*), averageDistance (*AvgDist*) and decisionEdges (*DecEdges*). The values in parentheses in the header columns provide only the theoretically maximum or minimum values per quality indicator, which may be far away from the optimal values (that are based on the road network and the s-t queries).

In Tables 2, 3, and 4, we report the average value per indicator. The overall execution time for computing the entire AG is given in milliseconds. As we see, we can achieve a high-quality AG in less than a second even for continental size networks. The produced alternative paths in AG are directly-accessible for use (e.g., they are not stored in any compressed form).

[1] We have been very recently informed [9] that this is the same target function as the one used in [6] and not the erroneously stated $totalDistance - averageDistance$ in that paper.

Table 2 The average quality of the resulted *AG* via Plateau method.

map	TargFun (max:11)	TargFun in [6]	TotDist (max:11)	AvgDist (min:1)	DecEdges (max:10)	Time (ms)
B	3.82	-	3.91	1.09	9.95	45.61
LU	4.44	3.05	4.49	1.05	9.73	37.05
BE	4.83	-	4.87	1.04	10.00	85.08
IT	4.10	-	4.14	1.04	9.92	114.29
GB	4.36	-	4.40	1.04	9.93	180.12
FR	4.22	-	4.26	1.04	9.97	159.93
GE	4.88	-	4.92	1.04	10.00	286.40
WE	4.35	3.08	4.37	1.02	9.88	717.57

Table 3 The average quality of the resulted *AG* via Penalty method.

map	TargFun (max:11)	TargFun in [6]	TotDist (max:11)	AvgDist (min:1)	DecEdges (max:10)	Time (ms)
B	4.16	-	4.23	1.07	9.92	49.34
LU	5.14	2.91	5.19	1.05	9.23	41.56
BE	5.29	-	5.33	1.04	9.54	159.71
IT	4.11	-	4.14	1.03	9.47	105.84
GB	4.38	-	4.41	1.03	9.87	210.94
FR	4.11	-	4.16	1.05	9.32	192.44
GE	5.42	-	5.46	1.04	9.91	388.97
WE	5.21	3.34	5.24	1.03	9.67	776.97

Table 4 The average quality of the resulted *AG* via the combined Penalty and Plateau method.

map	TargFun (max:11)	TargFun in [6]	TotDist (max:11)	AvgDist (min:1)	DecEdges (max:10)	Time (ms)
B	4.55	-	4.61	1.06	9.97	54.12
LU	5.25	3.29	5.30	1.05	9.81	43.69
BE	5.36	-	5.41	1.05	9.89	163.75
IT	4.37	-	4.41	1.04	9.79	178.08
GB	4.67	-	4.71	1.04	9.86	284.38
FR	4.56	-	4.60	1.04	9.86	217.30
GE	5.50	-	5.54	1.04	9.89	446.38
WE	5.49	3.70	5.52	1.03	9.94	987.42

Table 5 Alternative route queries in the road network of Western Europe, with geographical distance up to 500km and τ value of up to 1.2.

map WE	TargFun	TotDist	AvgDist	DecEdges
Plateau	4.71	4.73	1.02	10.00
Penalty	6.46	6.48	1.02	9.97
Plateau & Penalty	6.82	6.84	1.02	9.98

Due to the limitation on the number of the decision edges in AG and the low upper bound in stretch, we have chosen in the Penalty method small penalty factors, $p = 0.1$ and $r = 0.1$. In addition, this serves in getting better low-stretch results, see Table 3. In contrast, the *averageDistance* in Plateau gets slightly closer to the 1.1 upper bound.

In our experiments, the Penalty method clearly outperforms Plateau on finding more qualitative results. However it has higher computation cost. This is reasonable because it needs to perform around to 10 shortest s-t path queries. The combination of Penalty and Plateau is used to extract the best results of both of the methods. Therefore in this way the resulted AG has better quality than the one provided by any individual method. In Tables 2, 3, and 4, we also report on the *TargFun* quality indicator of the study in [6]. The experiments in that study were run only on the LU and WE networks, and on data provided by PTV, which concerned smaller in size networks and which may be somehow different from those we use here [1]. Nevertheless, we put the *TargFun* values in [6] as a kind of reference for comparison.

We would like to note that if we allow a larger value of τ (up to 1.2) for large networks (e.g., WE) and for s-t distances larger than 300km, then we can achieve higher quality indicators (intuitively, this happens due to the much more alternatives in such a case). Indicative values of quality indicators for WE are reported in Table 5.

6 Conclusion

We have extended the Penalty and Plateau based methods in [6] as well as their combination in several ways. We can generate a large number of qualitative alternatives with high non-overlappingness and low stretch in time less than 1 second on continental size networks. The new heuristics can tolerate edge cost increases without requiring new preprocessing.

References

1. Openstreetmap. http://www.openstreetmap.org.
2. Tomtom. http://www.tomtom.com.
3. Camvit: Choice routing, 2009. http://www.camvit.com.
4. eCOMPASS project, 2011-2014. http://www.ecompass-project.eu.
5. Ittai Abraham, Daniel Delling, Andrew V Goldberg, and Renato F Werneck. Alternative routes in road networks. In *Experimental Algorithms (SEA)*, pages 23–34. Springer, 2010.
6. Roland Bader, Jonathan Dees, Robert Geisberger, and Peter Sanders. Alternative route graphs in road networks. In *Theory and Practice of Algorithms in (Computer) Systems*, pages 21–32. Springer, 2011.
7. Yanyan Chen, Michael GH Bell, and Klaus Bogenberger. Reliable pretrip multipath planning and dynamic adaptation for a centralized road navigation system. *Intelligent Transportation Systems, IEEE Transactions on*, 8(1):14–20, 2007.

8 Daniel Delling, Andrew V Goldberg, Thomas Pajor, and Renato F Werneck. Customizable route planning. In *Experimental Algorithms*, pages 376–387. Springer, 2011.

9 Daniel Delling and Moritz Kobitzsch. Personal commnication, July 2013.

10 Daniel Delling and Dorothea Wagner. Pareto paths with SHARC. In *Experimental Algorithms*, pages 125–136. Springer, 2009.

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

12 David Eppstein. Finding the k shortest paths. *SIAM Journal on computing*, 28(2):652–673, 1998.

13 Andrew V Goldberg and Chris Harrelson. Computing the shortest path: A* search meets graph theory. In *Proc. 16th ACM-SIAM symposium on Discrete algorithms*, pages 156–165. Society for Industrial and Applied Mathematics, 2005.

14 Pierre Hansen. Bicriterion path problems. In *Multiple criteria decision making theory and application*, pages 109–127. Springer, 1980.

15 Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

16 Moritz Kobitzsch. An alternative approach to alternative routes: HiDAR. In *Euroepan Symposium on Algorithms (ESA)*. Springer, 2013. to appear.

17 Moritz Kobitzsch, Dennis Schieferdecker, and Marcel Radermacher. Evolution and evaluation of the penalty method for alternative routes. In *ATMOS*, 2013.

18 Felix Koenig. Future challenges in real-life routing. In *Workshop on New Prospects in Car Navigation*. February 2012. TU Berlin.

19 Dennis Luxen and Dennis Schieferdecker. Candidate sets for alternative routes in road networks. In *Experimental Algorithms*, pages 260–270. Springer, 2012.

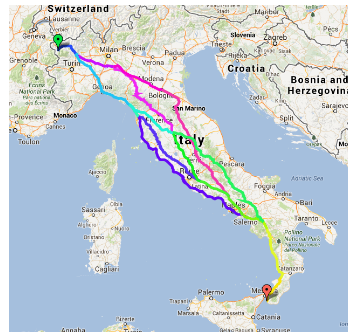
20 Georgia Mali, Panagiotis Michail, Andreas Paraskevopoulos, and Christos Zaroliagis. A new dynamic graph structure for large-scale transportation networks. In *Algorithms and Complexity*, volume 7878, pages 312–323. Springer, 2013.

21 Ernesto Queiros Vieira Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16(2):236–245, 1984.

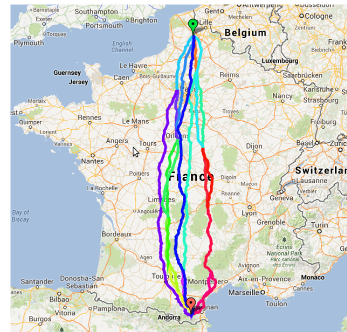
22 Jin Y Yen. Finding the k shortest loopless paths in a network. *management Science*, 17(11):712–716, 1971.

A Appendix

Figure 6 shows visualized AGs for a few representative cases.

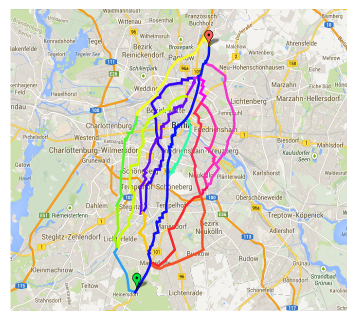


(a) Penalty



(b) Penalty and Plateau

(c) Plateau



(d) Penalty and Plateau

Figure 6 Shape of AG: (a) Italy, (b) France, (c) Spain, (d) Berlin.