Route Planning in Road Networks with Turn Costs

Lars Volker

July 22, 2008

Abstract

In this paper, we will describe the use of edge-based graphs to integrate turn penalties into algorithms used for route planning in road networks. We will give a model of these turn costs and describe a way to obtain reasonable estimates. We also will describe different ways to obtain edge-based graph prior and during the runtime of a shortest path search and also present a hybrid approach. These ways come with characteristic impacts on time and space consumption. We will analyze those drawbacks theoretically and experimentally and show techniques, which can be used to reduce their impact.

Contents

1 Introduction

Route planning in road networks has been discussed quite extensively and even for the largest road networks available, like the United States or Europe, the average execution times of queries are very fast. Those fast query times in turn allow for consideration of operations and graph-algorithms, which increase the level of detail of the model representing the road network. This includes the integration of banned turns and the consideration of the time delays introduced by turning on road crossings.

In this work, we will focus on the integration of turn costs, in order to obtain a finer grained model of the road networks used for route planning.

First we describe the concept of edge-based graphs, their use and limitations in Section 2. Section 3 is focused on how to integrate arbitrary turn costs into these edge-based graphs. An alternative approach which integrates turn cost information into the searchgraph **during** the execution of DIJKSTRA's algorithm and a comparison between different approaches is presented in Section 5. How to obtain reasonable turn delays for different kinds of roads and intersections is shown in Section 6. Section 4 gives an overview of the algorithms used for our experiments while results of the experiments are presented in Sections 7, 8 and 9.

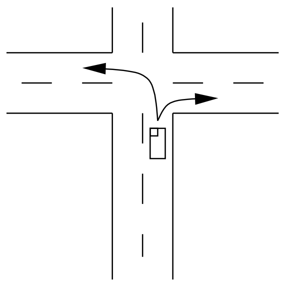

Figure 1: *Which of those turns takes longer? Clearly there is a difference.*

1.1 Motivation

Including the cost of turning on a crossing leads to a refined model of the road network. This way better results in matters of travel time, fuel consumption etc. can be achieved. There are even minimalistic solutions which already implement a partial optimization based on turn costs: The parcel service UPS eliminated left-hand-turns for their delivery trucks yielding a reduction of fuel consumption [4]. To unveal the full potential of turn cost integration, it is necessary to develop a model finer than this left-turn penalty. At least three types of turn costs can be observed:

Static costs: These are turn costs, which depend on geometry and type of roads and junctions involved. The most important delay is introduced by the turns angle, which can only be taken at a certain maximum velocity and by the need to watch out for other traffic participants. Easy to handle, those have a significant impact on travel times.

Dynamic costs: Dynamic costs can be best described as the time dependant part of a turns total cost. They include turning restrictions changing over time, traffic lights etc. It is possible to include them based on the probability of their occurrence. For time-dependent restrictions one could also obtain a timetable and add delays as they occur in reality. This however would necessitate a time-dependent route planning approach as described in [1].

Statistic costs: Those are events delaying a turn, which are more or less likely to occur, like pedestrians and other traffic participants. They can be accounted for by using probabilities for their occurance.

During this work, we will focus on the first category, static costs. Although the second and third categories would not be too hard to implement, the lack of statistical information prevents us from anything but guessing probabilities.

1.2 Previous Work

Studies regarding banned turns have already been conducted. Most important, Kirill Müller in [5] already described many of the basic concepts used to integrate turn-cost information into searchgraphs, however they are only applied to banned turns. Those concepts were in turn partially derived from [8]. Also the concept of a generalized interface to Dijkstra's algorithm is described in [5].

2 Edgebased Graphs

2.1 Preliminaries

In our work, we consider *directed graph*s. A graph G is denoted as a tuple $G = (V, E)$ consisting of a list of nodes $V = \langle v_1, \dots, v_n\rangle$, $|V| = n$ and a list of edges $E = \langle e_1, \dots, e_m\rangle$, $|E| = m$. Where unambiguous, lists will be denoted using simple parentheses, e.g. $V = (v_1, \dots, v_n)$. Each edge $e = (s, t)$ consists of a *source* node and a *target* node.

The *weight function* $w(e) : E \to \mathbb{R}_0^+$ specifies a weight for each edge. Note, that in contrast to other works we need the weight function to allow zero-cost edges as well. We define the following mappings for edges:

$$\begin{aligned} source((s,t)) &:= s \\ target((s,t)) &:= t \end{aligned}$$

For each node we define sets of outgoing and incoming edges:

$$\begin{aligned} out(v) &:= \{e \in E : source(e) = v;\} \\ in(v) &:= \{e \in E : target(e) = v;\} \end{aligned}$$

A *path* is a list of edges $P = \langle e_1, \ldots, e_p \rangle$ such that $source(e_{i+1}) = target(e_i)$ for all $i = 1..p-1$. The paths weight is $w(P) = \sum_{e_i \in P} w(e_i)$. Similar to edges, we define $source(P) := source(e_1)$ and $target(P) := target(e_p)$. A *turn* is a path consisting of 2 edges.

For our analysis, we assume a graph without parallel edges. Therefore we can write a path of p edges as the list of connected nodes: $P = (v_1, \ldots, v_{p+1})$. In the same way, a turn will be denoted as $T = (u, v, w) \in V^3$.

Also we require each node in the graph to have at least one outgoing edge[1]. This means:

$$\forall v \in V : |out(v)| \geq 1$$

Our primary goal will be to find shortest paths for given source and target nodes.

2.2 Conversion

In order to use the algorithms described in Section 4 for turn-cost aware route planning without significant changes, we need a way to store information about turn-costs inside the searchgraph. This is accomplished by converting the original graph into a new searchgraph while integrating available information on banned turns and turn costs into the generated graph structure.

In order to be able to store information for each turn (u, v, w) inside the graph, such a turn should at least be represented by a single edge. The nodes connected by this edge will be named (u, v) and (v, w). This observation leads to the conceptual idea of an *edge-based graph*.

When converting an original, node-based graph into an edge-based graph, each node-based edge becomes an edge-based node. Formally this means:

[1]This requirement is later weakened by the introduction of a so-called interface-graph. See Subsection 2.4

Let $G(V, E)$ be a strongly connected searchgraph with a weight function $w(e), e \in E$. Then

$$G_{eb} := (E, E')$$

with

$$E' := \{((s,t),(u,v)) \in E \times E : t = u\}$$

and

$$w'(e_1, e_2) := w(e_1), e_1, e_2 \in E, (e_1, e_2) \in E'$$

defines the edge-based graph G_{eb} to the original graph G. An example for this conversion is shown in figure 2.

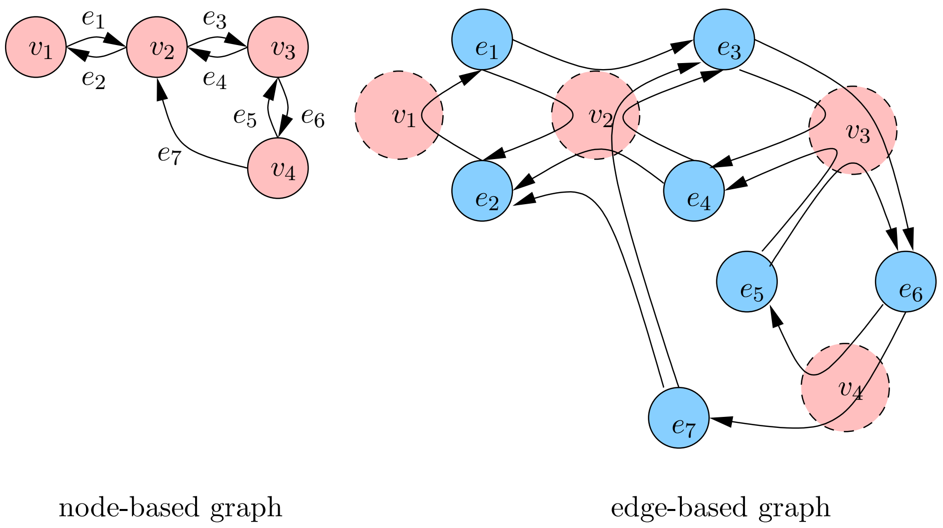

Figure 2: *The conversion of an node-based graph on the left to an edge-based graph on the right. The edges on the node-based graph G become nodes on the edge-based graph G_{eb}.*

To minimize the confusion about edge-based nodes, node-based edges and the like, we use the following denotation through the rest of the document:

A node-based graph consists of *junctions* connected by *roads*[2]. An edge-based graph however consists of *vertices*, which are connected by *connections*.

2.3 Analysis

From an informal point of view, the conversion introduced above creates a new vertex for all roads $e \in E$. By chosing the cost function $c'(e_1, e_2) = c(e_1)$

[2]Think of “node” and “road” having a similar sound when pronounced.

to only include costs of the first road e_1, each connection inserted accounts for the costs from the source of e_1 to the source of e_2.

Junctions are required to have an outgoing road for the following reason: As roads become vertices in the edge-based graph, it is not further possible to route between what was a junction in the node-based graph. Instead, routing is only possible between roads. If a junction has no outgoing roads, it will not be represented by a vertex in the edge-based graph and therefore will not be usable as a target for queries.

2.4 Interface Graphs

While junctions need to have at least one incoming and one outgoing road, another problem arises with junctions, that have more than one outgoing road. A shortest-path search from s to t in the original graph would now have to be conducted as an *Any to Any Shortest Path* (*AtoA*) query from $out(s)$ to $out(t)$ while the latter might be empty if s is a dead end, e.g. a international ferry terminal's ramp. During this query, the shortest distance from one set to the other set is computed.

To cope with those problems during the implementation, we add a so-called *interface graph* to the edge-based graph G_{eb}. First, all n junctions $v_{in_i} \in V$, $i = 1..n$ are appended to V'. Then for each junction v_{in_i} and all roads $e \in out(v_{in_i})$, a shortcut-connection (v_{in_i}, e) with $w'(v_{in_i}, e) = 0$ is added to E'. The new vertices formed by the appended junctions are called *entry-vertices*. Each represents a junction in the node-based graph and has adjacent connections to those vertices, which represent the outgoing roads of the corresponding junction.

In the same way n junctions $v_{out_i} \in V$, $i = 1..n$, are appended to V', together with shortcut-connections $(e, target(e))$, $w'(e, target(e) := w(e)$ for all roads $e \in out(v_{out_i})$. [replaced $target(v_i)$ by $target(e)$.] These vertices are called *exit-vertices*. By adding entry- and exit-vertices separately, zero-cost-loops inside the generated graph are avoided. $\Longleftarrow$

An example for an interface graph is given in Figure 3. [Changed from "nodes" to "vertices" inside Figure 3.] $\Longleftarrow$

Using this interface graph enables us to easily compare results of searches conducted in node-based and edge-based graphs, as the node ids of test queries can be translated easily from node-based junction ids to edge-based vertex ids. Figure 4 shows, how the interface vertices are added to the edge-based graph.

2.5 Changes in Size

By converting a graph from node-based to edge-based, the size increases: let $G = (V, E)$ be a graph with $|V| = n$ and $|E| = m$. Then the resulting graph $G_{eb} = (V', E')$ obviously has $|V'| = |E| = m$ vertices. The number of

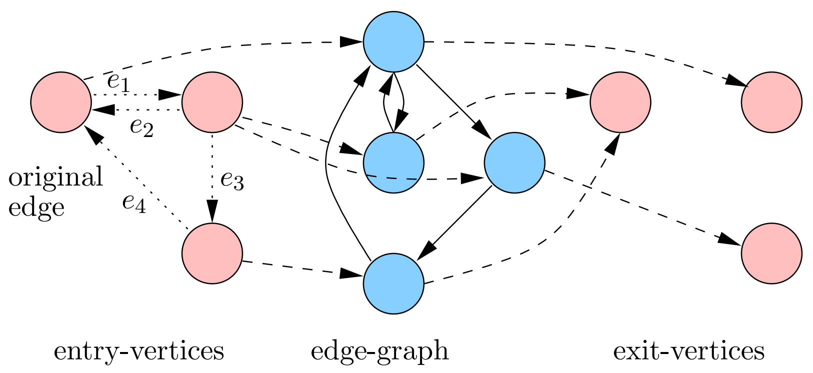

Figure 3: *An interface graph consisting of entry- and exit-vertices. The dotted roads represent the original graph, the dashed connections belong to the interface graph.*

connections in G_{eb} not only depends on n and m but also on the number of roads adjacent to each junction v.

During the conversion a connection e_{new} is added to E' for each pair of roads $(e_1, e_2) \in in(v) \times out(v)$. Therefore the total number of connections in E' can be calculated as:

$$|E'| = \sum_{v \in V} |in(v)| \cdot |out(v)|$$

Note
The number of connections cannot be computed from only n and m. This fact is illustrated in Figure 5. ■

If the graph is converted including the interface, the number of vertices is increased by twice the number of junctions: $|V'| = |E| + 2|V|$. The number of connections increases by $2|E|$ because for each road $v \in V$ $|out(v)|$ connections from the entry-vertex and $|in(v)|$ to the exit-vertex are added to E'.

3 Integration of tc-data

The information, which is to be integrated into the edge-based searchgraphs consists of two kinds, which will be handled separately.

Banned turns: A *banned turn* is a turn in the road network, that one is not allowed to take. It will be denoted as $T = (e_{in}, e_{out})$ or as a triple of junctions $T = (s, t, u)$ where unambiguous.

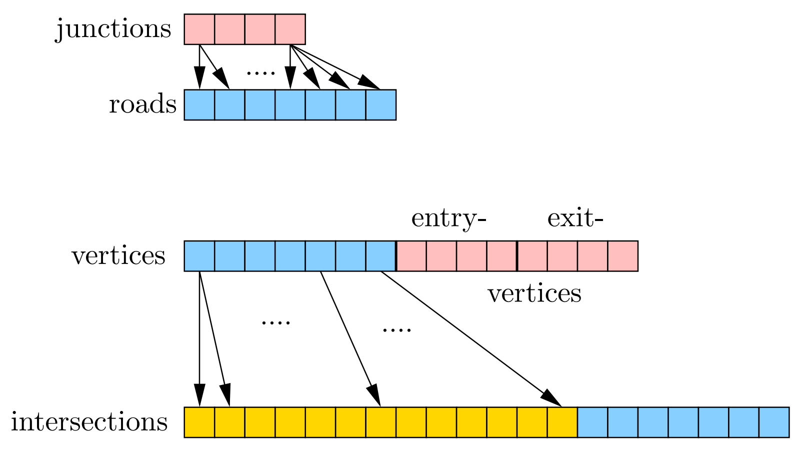

Figure 4: *The conversion of the node-based graph (top) to its counterpart edge-based graph. The graphs are schematically depicted as adjacency arrays. Also shown are the entry- and exit-vertices to give an understanding, how they were added to the graph data structure. To reduce the figures' complexity, connections of the interface graph were omitted, as they are shown in detail in Figure 3.*

Turn costs: If one is allowed to take a turn, this comes at a certain cost, which has to be added to the weight of the incoming road. A detailed description of how to obtain reasonable turn costs for different turns is given in chapter 6.

Whenever both kinds of additional turn cost information are addressed, the term *tc-data* will be used.

3.1 Integration of Banned Turns

The integration of banned turns into the edge-based graph G_{eb} is straightforward. It will be accomplished by **not** adding a connection $e = ((s,t),(t,u))$ to the generated graph, iff the corresponding turn $T = (s,t,u)$ is banned.

3.2 Integration of Turn Costs

If the connection $e = ((s,t),(t,u))$ to be added to the graph is not omitted, the cost of the corresponding turn $T = (s,t,u)$ is obtained and added to the weight of e. The weight function w' of G_{eb} thereby results in:

$$w'((s,t),(t,u)) := w(s,t) + \textsc{turnCost}(s,t,u)$$

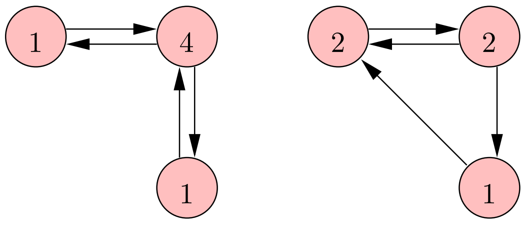

Figure 5: *Two node-based graphs: Each node contains the number of connections $|in(v)| \cdot |out(v)|$ which will be inserted during the conversion. Note, that both graphs fulfill $|V| = 3$ and $|E| = 4$. However the resulting edge-based graphs will have 6 connections for the example on the lhs, compared to 5 connections for the one on the rhs.*

Chapter 6 shows in detail, how turn costs are calculated inside TURNCOST(). A complete algorithm for the offline conversion including banned turns and turn costs is given in Section 5.

4 Important Algorithms

4.1 Dijkstra's Algorithm

DIJKSTRA's algorithm described in [2] can be used to compute the shortest path from one source node s to all other nodes of a graph G which is known as the *single source shortest path* problem (*SSSP*). It also is a fundamental technique for the speedup concepts referenced below. For a comprehensive and contemporary introduction to DIJKSTRA's algorithm see [9].

4.2 Highway Hierarchies

The first out of two speedup concepts used for our experiments is *Highway Hierarchies* (*HH*). HH constructs a hierarchy of levels, consisting of so called highway nodes. Searches are started on the lowest level and as soon as some local area around the start nodes is covered, a switch to the next higher level is performed. The whole concept is described in [9].

4.3 Contraction Hierarchies

Contraction Hierarchies CH is the second concept we use and is presented in [3]. The main idea is to construct a hierarchy of nodes, where each node creates its own level representing the node's importance. Then the nodes are contracted in ascending order: shortcuts for bypassed nodes are added, iff the bypassed nodes are used on a shortest path between two adjacent nodes. Contracting the nodes in this order ensures, that forward edges only lead from lower to higher levels, fulfilling $\forall e = (s,t) \in E : \mathit{level}(s) < \mathit{level}(t)$. Bidirectional searches from s to t meet at an intermediate node v, which has the highest level of all nodes on the shortest path from s to t.

5 Offline versus Online Conversion

The intuitive approach of converting a graph described earlier is based on the idea of generating an edge-based searchgraph **before** preprocessing and execution of queries take place. However, very large working graphs will be obtained. We call this approach *offline conversion*, as the construction happens before and separated from subsequent processings.

In a second approach we push the conversion logic towards the search algorithm, thus being able to store only the node-based graph and some additional information, while converting those parts of the graph included in a query's searchspace to an edge-based *view* on-the-fly, as the search is conducted.

A detailed analysis of both approaches as well as their benefits and drawbacks is presented in this section.

5.1 Offline Conversion

The basic algorithm for offline conversion is pretty simple and illustrated using pseudocode below:

Algorithm 1 Offline Conversion

```
1: procedure CONVERT-OFFLINE($G = (V, E)$)
2:     $V' := E$
3:     $E' := \emptyset$
4:     Mapping $w' := \emptyset$                          ▷ new weight function
5:     for all $(s,t) \in E$ do
6:         for all $(t,u) \in E$ do
7:             if not BANNEDTURN($s,t,u$) then
8:                 EdgeWeight $\mathit{cost} = $ TURNCOST($s,t,u$)
9:                 $E'$.append($(s,t),(t,u)$)
10:                $w'[(s,t),(t,u)] := G.$WEIGHT($(s,t)$) $+ \mathit{cost}$
11:            end if
12:        end for
13:    end for
14:    $G_{eb}$.WEIGHT $:= w'$
15:    return $G_{eb} = (V', E')$
16: end procedure
```

For each triple (s,t,u) a new connection is added to G_{eb}. The algorithm has runtime $O(|E'|)$ which is acceptable and not of our further concern, as the offline conversion accounts to preprocessing times and does not affect query times.

As a benefit for the increased size of the generated graph we can now apply preprocessing and execute test queries using the speedup algorithms described before. Results of the experiments conducted are presented in Section 7.

5.2 Online Conversion

In contrast to the offline conversion portrayed above we introduce an alternative approach: *online conversion*. Hereby an edge-based *view* on the node-based graph is generated during the execution of a query. The kind of access to the edge-based graph-structure however depends on the algorithms used, and in particular on the way DIJKSTRA's algorithm obtains information about the graph. Therefore we first depict a standard-version of DIJKSTRA's algorithm to identify the operations accessing the graph:

5.2.1 Dijkstras algorithm

Algorithm 2 Dijkstra's Algorithm

```
1: procedure DIJKSTRA($G = (V, E)$, Node s, Node t)
2:     PriorityQueue $Q := \emptyset$
3:     List $d = \langle \infty, \ldots, \infty \rangle$ of tentative distances
4:     $Q$.APPEND(s)
5:     while $Q \neq \emptyset$ do
6:         Node $u = Q$.EXTRACTMIN
7:         if u = t then
8:             return $d[u]$
9:         end if
10:        for all Edge $e \in out(u)$ do                ▷ Edges are obtained here
11:            Node $v := target(e)$
12:            $d[v] := min(d[v], d[u] + G$.WEIGHT$(e))$      ▷ relax edge $e$
13:        end for
14:    end while
15:    return $\infty$
16: end procedure
```

5.2.2 Analysis

Basically there is only **one** crucial functionality we have to implement for any graph on which an adapted version of DIJKSTRA's algorithm is supposed to operate: we must provide a way to retrieve all outgoing edges $out(u)$ for a node u.

This requirement can be implemented by a method GETTARGETS(Node v), which returns all vertices adjacent to vertex v together with the distance between v and each vertex. The method GETTARGETS(v) is shown as pseudocode in Listing 3.

Algorithm 3 Online Conversion

```
procedure GETTARGETS(node-based Graph : $G = (V, E)$, Edge : $v$)
                              ▷ Note, that $v$ is a node in the edge-based graph
    List $T$ of Pair⟨ Node, Weight⟩ := $\emptyset$
    EdgeWeight $w := G$.WEIGHT($v$)
    Node $t = v$.TARGET(())
    for all $(t, u, w_1) \in E$ do
        if not BANNEDTURN($s, t, u$) then
            EdgeWeight cost = TURNCOST($s, t, u$)
            T.PUSH_BACK($(\langle u, w + cost \rangle)$)
        end if
    end for
    return $T$
end procedure
```

When obtaining the targets, v is a road in the node-based graph and a vertex in the edge-based graph. It has the weight $w(v)$, and as the edge $v = (s, t)$ is the first half of the triple (s, t, u), its weight plus the cost of the particular turn (s, t, u) is returned as the weight to reach the targets in T. The check for banned turns prevents those turns from being added to the target list.

Figure 6 shows, how GETTARGETS() interacts with both graphs.

The benefit of having no space overhead for storing the edge-based graph on disk however comes at quite a cost:

Adaptation of algorithms: All algorithms accessing the graph have to be adapted. Most of the algorithms use more methods to access the searchgraph like NOOFNODES() etc. There might even be cases, where such an adaptation is not possible, for example if an algorithm's need to determine the number of edges can't be circumvented.

No preprocessing: In combination with algorithms using preprocessing of the whole graph, like the ones introduced in Section 4, the space overhead is reintroduced. By implementing the interface described above, those algorithms could obtain the whole converted graph G_{eb} however. As this is possible without any modifications by using offline-conversion, the benefit is lost.

No edges: Specifically the concept of an edge id is lost when using online-conversion. For vertices, an id can be predicted based on the id of

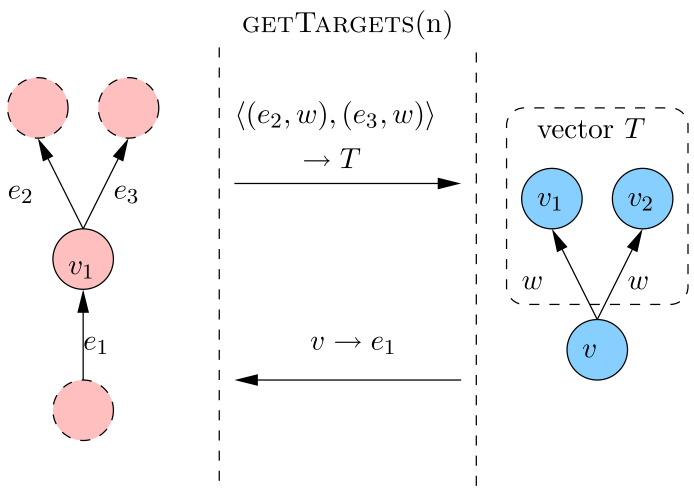

Figure 6: *The* GETTARGETS*() method interacting with both, the node-based and the edge-based graphs.*

the corresponding road. However it cannot be predicted, how many connections the new graph will contain and therefore a numbering cannot be introduced without additional data structures and a previous examination of all roads in the node-based graph.

6 Calculating Turn Costs

In order to estimate the time needed to take a turn on a certain junction, numerous calculations will be done. The turn itself shall be the travelling sequence of junctions $s{\rightarrow}t{\rightarrow}u$ with roads $in = (s{\rightarrow}t)$ and $out = (t{\rightarrow}u)$ where v_{in} and v_{out} are the travelling speeds of the particular roads. Note, that v is used in physics to denote a velocity. Therefore we will stick to this notation and name nodes s, t, u during this section.

6.1 Turning Speed

We have to calculate a turning velocity v_{turn} at which a vehicle of length len can take the turn. This velocity is affected by the following constraints:

- v_{angle}: The turns angle ψ with

$$-\pi \leq \psi \leq \pi$$

. $\psi = 0$ means a straight road with no change of direction.

- v_{cars}: Roads on which other cars could arrive at the junction with a higher precedence.

- $v_{pedestrian}$: Pedestrians crossing the *out*-edge after leaving the crossing.
- v_{size}: The vehicle's size (trucks take right turns much slower).

All those factors limit the maximum velocity, at which the turn can be taken. As a result, $v_{turn} = \min\{v_{angle}, v_{cars}, v_{pedestrian}, v_{size}\}$. Also some of the factors might not only induce a maximum velocity of 0, but also require the vehicle to wait for a certain time $t_{additional}$, which should be later added to the resulting delay. This feature is currently not used in our model. The turns total delay can be obtained by adding the time it takes to decelerate, the waiting time on the crossing, and the time to accelerate to the travelling speed of the *out*-edge.

We will now give detailed information on how each v can be calculated:

- v_{angle}: The velocity, at which a curve with angle ψ can be taken is estimated as
$$(1 - \frac{|\psi|}{\pi}) \cdot \min\{v_{in}, v_{out}\}$$
.

- v_{cars}: To estimate v_{cars}, a list of all incominging edges of node t is created and sorted counterclockwise by the angle they form with the road on which the vehicle enters the crossing. Then those edges with a higher or equal road category are counted. If there is at least one of those edges, v_{cars} is set to 20 km/h and for each edge $i = 1..n$, $\frac{10}{i^2}$ km/h are subtracted from v_{cars}. This respects, that more incoming edges require more looking and therefore further slow a vehicle down.

- $v_{pedestrian}$: If the crossing has more than two streets connected, there might be a pedestrian to watch for. Therefore, $v_{pedestrian}$ will be limited to 4 km/h.

- v_{size}: If the vehicle is of extraordinary size, like a heavy truck, then it will have difficulties with sharp turns like turning right. If the vehicle's length *len* exceeds 4.5 m – which means it is larger than a car – the size penalty factor for turning is
$$p(\psi) = \begin{cases} 1 - \frac{2}{3} \cdot \frac{|\psi|}{\pi} & \text{turn to the right}, \psi < 0, \\ 1 & \text{turn to the left.} \end{cases}$$
The result is
$$v_{size} = v_{angle} \cdot \frac{4.5}{len} \cdot p(\psi)$$

After the maximum velocity $v_{turn} = min\{v_{angle}, v_{cars}, v_{pedestrian}, v_{size}\}$ and the additional delay for the turn $t_{additional}$ are given, the delay induced by de- and accelerating before and after the turn can be determined as

$$\begin{aligned} t_{decelerate} &= decelerationFactor \cdot (v_{in} - v_{turn}) \\ t_{accelerate} &= accelerationFactor \cdot (v_{out} - v_{turn}) \end{aligned}$$

Finally, the total delay results in

$$t_{turn} = t_{decelerate} + t_{additional} + t_{accelerate}.$$

For those calculations, the following variable parameters can be chosen:

Parameter	Meaning
Long vehicle	The vehicle's length exceeds 4.5 m.
Vehicle size *len*	The length of the vehicle
Maximum velocity v_{max}	The vehicle's maximum velocity v_{max} km/h
accelerationFactor	The vehicle's maximum acceleration
decelerationFactor	The vehicle's maximum deceleration

6.2 Road Categories and Corresponding Limits

The road categories range from 1 to 12 for usual roads, 13 is used for ferries and 15 is used for forest roads. As turn costs do not apply to the latter – one does not turn on a ferry and driving on forest roads is limited to 10 km/h anyways – we will focus on motorways (cat. $1 - 3$), national ($4 - 6$) and regional ($7 - 9$) roads as well as urban streets ($10 - 12$). Roads of the same type will be treated alike.

The following observations shall be taken into account:

- On motorways, turn costs are of no concern. The angle of a so called turn is almost always near 0, and if the roads category specifies a certain average velocity, the turns can be taken at that speed as well. Also there are no other cars or pedestrians involved in turning.
- National roads are usually closed for pedestrians as well, but there can be traffic lights and sharp bends or crossings requiring to take the angle of a turn into account. There are usually no other cars involved, as such a case is prevented by traffic lights.
- On regional roads there can be other cars as well as angle-dependant turns. However one might take into consideration, that between villages and crossings a road may take several turns, causing the actual angles between connected roads to differ from the ones obtained by triangulation.
- Inside built-up areas, there can be other cars and pedestrians delaying a turn. Also the angle between adjacent roads can be estimated by triangulation, as roads usually do not have unmodelled turns when surrounded by buildings.

- A vehicle's size only matters on regional roads and urban streets with sharp bends.
- Only when watching for other cars, the category of the entering road is important, in all other cases the category of the exit road is considered in the following mapping. For the "other cars" decision this means, that you don't have to wait for motorways if you enter a crossing on a national road, which makes sense as such a crossing has a specific local regulation anyways.

Those observations lead to a matrix, which maps road types to the considered maximum velocities:

road type	turn angle	other cars	pedestrians	vehicle size
motorway	o	o	o	o
national	x	o	o	o
regional	c	x	o	x
urban	x	x	x	x

Table 1: *"x": used, "o": unused, "c": configurable*

6.3 Possibilities of Further Improvement

Here we present a list of further improvements, which could be applied to refine our current model.

Left-hand driving: In most European countries, traffic flows on the right hand side of the road. However there are countries with left-hand traffic like Great Britain. For those regions, one would probably swap "left" and "right" in most of the above considerations.

Local colorations and habits: Most of the above considerations reflect the situation in Germany and many of them should be further generalized. Also local habits should be taken into account: e.g. if you turn right into a road of lower category, in Germany you would have to let a pedestrian cross the street taking precedence over your turn. In other European countries like France or Italy such a rule might not be known.

Stochastic events: Before, we only accounted for the time it takes to slow down and look for other cars or pedestrians (cf. v_{cars} and $v_{pedestrian}$). If however another car or pedestrian in fact occurs, we would not only have to slow down, but to stop and wait to let them pass. This additional delay is of stochastic nature and could further be analyzed and modelled.

Traffic lights: They can be regarded as stochastic events as well, but as they are of a different nature and can be predicted more precisely, given the necessary information, they could be treated specifically.

Analyze junction-form: Given the categories of adjacent roads, one could implement a way to guess the type of a crossing. This way for example junctions with reflected right of way could be determined. A visualization is given in Figure 7.

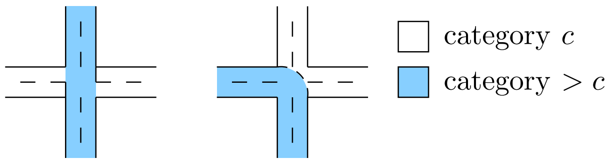

Figure 7: *This example shows, how the type of junction could be guessed, given the categories of adjacent roads.*

7 Offline Conversion: Experiments

The offline conversion of a graph to its edge-based counterpart G_{eb} increases the overall size of the searchgraph and therefore also has an impact on preprocessing and query times. In this section, we analyze this impact on both, Highway Hierarchies (HH) and Contraction Hierarchies (CH).

7.1 Graphs used

For our experiments described in the following sections, we used a graph representing the road network of Germany. The node-based graph has a total of 4 378 446 nodes and 5 519 553 edges, most of which are bidirectional (open in both directions).

7.2 Hardware used

All experiments were conducted using one core of a Dual Core AMD Opteron 270 Processor running at 2GHz. The machine has 8GB RAM and is running openSuSE version 10.3 with kernel 2.6.22.17 x86_64. All programs were compiled using gcc version 4.2.1 with optimization level 3. The CPU was running in 64bit mode. The same hardware was used for the experiments in [9] and [3]. Therefore the results depicted there can be compared to those obtained during our experiments.

Graph	#nodes	#edges	ratio	conversion time [sec]
G	4 378 446	10 736 842	2.452	–
G_{eb} w/o tc-data	10 736 425	30 085 141	2.802	2.8548
G_{eb} with tc-data	10 736 425	29 878 929	2.783	68.6179

Table 2: *Sizes of generated graphs and the size ratio between node-based and edge-based graphs. The edge-based graph without tc-data is given for comparability. Note, that the number of nodes increases by the average degree of the node-based graph, and the number of edges increases by the average degree of the edge-based graph. The small difference between #edges of G and #nodes of G_{eb} is due to parallel edges of same weights and self-loops.*

7.3 Conversion

In a first step, raw graph data is converted to a node-based graph which has 10 736 842 unidirectional roads. Also the geographic locations used for turn cost calculations and a list of banned turns are generated. We choose to open roads, which are marked as "closed" in the raw data files, in both directions, as removing them from the node-based graph would render the graph severly unconnected. These are mainly forest roads, marked as closed because one is not allowed to drive inside these forests. As they are not contained in the shortest path for most test queries, they only increase the graph-size artifically, while the experiments still show fast results.

Then the actual conversion is conducted combining the node-based graph, the banned turns and using the geographic locations. This takes 68.6179 seconds to complete. During the process, 206 212 banned turns are embedded into the edge-based graph. The resulting graph G_{eb} has 10 736 425 vertices and 29 878 929 connections. The comparison of the generated graphs is shown in table 2.

7.4 Experiments: HH

Table 3 shows the preprocessing and query times when using HH on the generated graphs. After the preprocessing was done, 10 000 queries were performed. As the number of nodes and edges increases when converting the graphs, the neighbourhood size H should be increased as well. If H is chosen too small, lots of the improvement gained by HH is lost as too many levels of hierarchy are needed to cover the whole searchgraph. This effect is shown in figure 8.

Graph	H	preprocessing [min]	query time [msec]
node-based	50	3.23	0.963
edge-based	50	42.64	22.698
edge-based	130	43.88	4.636
edge-based	200	57.78	3.451

Table 3: *Shown are the results of HH running on the generated graphs. The edge-based graph was generated* ***with*** *tc-data. A selection of neighbourhood sizes H is given. The full range of sizes tested is shown in figure 8.*

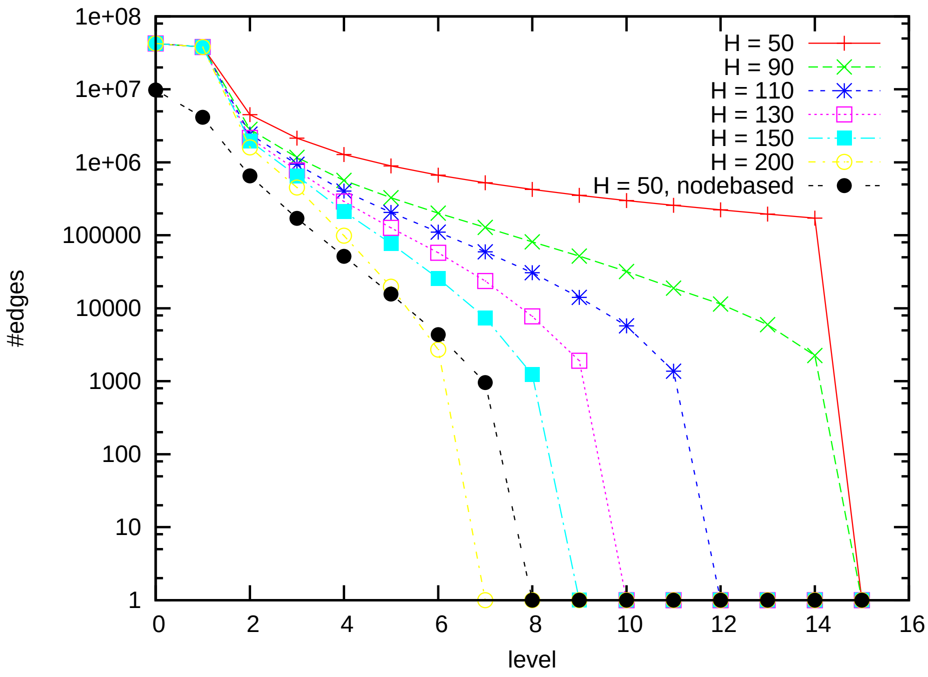

Figure 8: *Different neighbourhood sizes H have a significant impact on the number of levels. The experiments were conducted using an offline-converted version of the graph representing the road network of Germany. Turn-costs were included. For comparison, the diagram contains results for the node-based graph as well, where experiments have shown a neighbourhood size of 50 to be a good choice.*

Graph	node sorting [sec]	hierarchy construction [sec]	query time [msec]
node-based	105.935	11.934	0.200
edge-based w/o tc-data	718.217	58.683	0.422
edge-based w/ tc-data	813.776	64.5905	0.374

Table 4: *Results for running* 100 000 *random queries on the generated graphs using Contraction Hierarchies. Edge-based graphs without tc-data show slightly faster time for preprocessing, as opposite edges often have the same weight and therefore can be combined into one bidirectional edge. However, the graph with tc-data is slightly smaller and therefore shows faster query times.*

7.5 Experiments: CH

Table 4 shows the results for Contraction Hierarchies running on the generated graphs.

8 Online Conversion: Dijkstra Modification

In order to conduct experiments with online conversion, we modify our implementation of DIJKSTRA's algorithm. Instead of relaxing all outgoing edges for a node v, the algorithm obtains a list of targets, reachable from v. Each target is a pair of $\langle \mathit{NodeID}, \mathit{EdgeWeight} \rangle$. By this modification we eliminate the need for the algorithm to work on edge ids, which are not available during online conversion. For further explanations consider Section 5.2.

8.1 Experiments

Our modified Dijkstra implementation executes 100 random queries on the node-based graph of Germany. The node-based graph again has 4 378 446 junctions and 10 736 842 roads. The same queries are performed on different views on the graph: node-based, edge-based **without** and edge-based **with** tc-data. For each view, unidirectional and bidirectional mode of operation is tested. The results are shown in table 5.

8.2 Interpretation

The most interesting observations are the following:

1. Unidirectional searches on the edge-based views have far more deleteMin operations per query than bidirectional searches. This is due to the exit-nodes of the interface graph. For each vertex v removed from the

mode	query time [sec]	#deleteMin	$\frac{\#\text{deleteMin}}{\text{time}}$	#relaxed edges
nb 1	1.750	2 235 222	1 277 160	5 504 037
nb 2	1.113	1 203 825	1 081 448	2 965 243
eb 1	6.442	7 739 259	1 201 373	20 955 837
eb 2	3.511	2 947 047	839 464	8 268 102
eb-tc 1	56.348	7 748 083	137 505	20 875 608
eb-tc 2	27.534	2 634 966	95 698	7 340 290

Table 5: *Results from running our modified Version of Dijkstra's algorithm. Node-based mode is labeled "nb", "eb" means "edge-based" and "eb-tc" stands for "edge-based with tc-data".*

head of the priority queue, the reachable targets are obtained from GETTARGET(v). This list contains all vertices reachable from vertex v through junction $target(v)$ **plus** the exit-node for junction $target(v)$. When running in **bidirectional mode**, only the exit-node of the target is added to the priority queue and the interface graph is left after both searches have settled their start node.

One would expect a ratio of 2 for the number of deleteMin operations when comparing unidirectional and bidirectional search:

$$d_{expected} \approx 2 * 2\,947\,047 = 5\,894\,094$$

However the increase factor is $\frac{7\,748\,083}{2\,634\,966} \approx 2.940$.

2. The number of deleteMin operations per second significantly decreases when switching from unidirectional search to bidirectional search on the edge-based views. The reason is, that the backward search has to operate on forward connections, in order for the searchspaces to meet. Therefore, all backward connections reachable from node v have to be matched to their corresponding forward connections during GETTARGETS(v). This could be further optimized though and is mainly due to our implementation of the graph datastructure.

9 Hybrid Approach

Both approaches, which we have introduced and experimentally examined before have specific drawbacks, rendering them less useful for practical applications.

The first, *offline conversion*, increases the graph's size by adding a significant amount of new vertices and connections. Thereby the file size of the graph on disk increases from 247 megabytes for the node-based graph to 623 megabytes for the edge-based graph. Most of the roads in the node-based graph are of the same weight and of opposite directions and can therefore be combined into a bidirectional road instead of two unidirecitonal ones, reducing the size of the node-based graph to 130 megabytes. In the edge-based graph parallel, antidirectional connections exist as well, due to the added turn costs however, these can't be combined into bidirectional connections as they differ in weight.

The second approach, online conversion, has the obvious drawback of increased query times. Compared to the fastest method used during the experiments, Contraction Hierarchies, we face query times increasing by the magnitude of 10^5. However, the space consumption is significantly better. The only additional data needed is a list of banned turns as well as the geophysical coordinates for each junction, both of which are application specific and needed nonetheless.

To combine benefits of both approaches – little space overhead and fast query times – we therefore propose the following hybrid approach:

First we perform an offline conversion to obtain the full edge-based searchgraph G_{eb}. Afterwards we construct a contraction hierarchy of this graph. As the vertices inside the CH are ordered by their level, which in turn corresponds to their importance inside the road network, we then cut off all vertices below a certain threshold t leaving only the upper part of the CH. The parameter t can be used to tune the ratio between space overhead and average query times. Having this upper CH at hand the hybrid query algorithm can perform searches by conducting the following two subqueries.

- Perform a *local search* in the node-based graph, using online conversion to enable an edge-based view onto the graph.
- As soon as the whole search space is covered by vertices inside the previously generated upper CH, continue the search there, using the covering vertices determined as source and target vertex sets.

To determine the moment, when to switch from local online search to offline CH search, it is necessary to monitor, how the vertices contained in the upper CH cover the search space. For this purpose, a technique called stalling is utilized, which is described in [9]. In the next section we perform experiments to compare three different stalling techniques: conservative- and aggressive stalling as well as stall-on-demand.

9.1 Stalling Methods

We give a short description of the different stalling techniques used in the next subsection. Before a search is conducted, all nodes contained in the upper CH are marked.

conservative stalling: The search is stopped, as soon as every path in the shortest path tree constructed by DIJKSTRA's algorithm is covered by

Stalling method	avg query time [sec]	#deleteMin	$\frac{\#\text{deleteMin}}{\text{time}}$	#relaxed edges
conservative	0.87388	90 506	103 568	252 235
aggressive	0.00806	1 143	141 766	3 065
on-demand	0.01269	925	72 871	2 460

Table 6: *Results for each stalling method.*

a marked node. If there are ferry connections included in the shortest path tree, the search space can become very large before the last path is covered, thus taking long to complete. However good covering sets are computed.

aggressive stalling: This is the most eradicative approach. As soon as a path in the search tree hits a marked node, it is settled without relaxing adjacent edges, thus pruning the search on marked nodes. However nodes "behind" the marked node can be reached on a circuitous road, thus adding more nodes to the covering set than neccessary.

stall-on-demand: In this mode, the circumventions described before, are detected, by searching backwards while considering nodes, that were reached before. If a circumvention is detected, a marked node v will mark all nodes as stalled, which have v on their shortest path. The search is then pruned at marked **and** stalled nodes.

A complete description of these stalling concepts is given in [9].

9.2 Stalling: Experiments

For the experiments, we first determine a threshold t so that the upper part of the edge-based contraction hierarchy has the same size as the node-based graph when storing bidirectional roads. For the road network of Germany, we choose both to be approximately 130 megabytes large. Keeping all nodes above level t = 10 000 000 results in a CH with the 736 425 most important nodes left. Then we mark all edges in the node-based graph, which are contained in the remaining CH and run 1 000 test queries bidirectional on the edge-based view while using tc-data, for all of the three stalling methods. We measure the average time until a query is stalled and therefore stopped. The results are presented in table 6.

The following observations are made:

1. The number of relaxed edges massively decreases from conservative mode to the other modes. The conservative approach continues until the whole search space is covered. A path, which is not covered keeps

the whole search space growing, which causes increased query times. This can happen for example when ferries are involved.

2. The number of deleteMin operations decreases from aggressive mode to stall-on-demand. The stalling operation used to implement the latter has to take backward connections into account, which are therefore explicitly calculated by getTargets() together with the forward connections to relax next. This is why the amount of work inside getTargets() effectively doubles.

Afterwards we determine the average query time for random queries in the upper CH, giving a rough estimate of how long the remainder of the search would take. The average query time using CH was 0.354791 ms compared to 0.375957 ms when using the full hierarchy generated from the edge-based graph.

The searches conducted inside the upper CH only were performed as SSSP queries. Actually those would have to be searches for the shortest path between two sets of nodes: the marked nodes covering the forward search space on the one hand and those covering the backward search space on the other hand. We already found this to be an AtoA search (see Section 2.4). Details are given in Section 10.

The question arises, if the time it takes to process a query in the full CH serves as an upper bound for the remainder AtoA search inside the upper CH.

Theorem 1
Let $T := \{v \in V : level(v) \geq t\}$ be the upper CH nodes left by the cutoff at threshold t. Let $V' \subset V$ be an optimal, minimal covering set for the current search space of a search from source node s. Let the target z of the search be sufficiently far away from s, so the search spaces do not meet at a node with level below t.

Then all nodes in V' will be settled by a CH-search from s.

Proof
Each path of the shortest path tree, which is covered by V', is ended by a vertex $v \in V'$. Also, all vertices u of each path $P = \langle s, \ldots, u, \ldots v \rangle$ fulfill $u \notin V'$. Hence v has the highest level of all vertices on the path.

During the construction of the CH, shortcuts are only introduced for two nodes bypassing nodes of **lower** levels. As v has the highest level on the path, there is no shortcut bypassing v from any $u \prec v$, $u \in P = \langle s, \ldots, u, \ldots v \rangle$. The search has to settle v. ■

The above theorem can be generalized to the following:

Corollary 1
If a path in the shortest path tree described above is covered by more than one node from V' and if $v_1 \prec v_2$ implies $level(v_1) < level(v_2)$, the CH-search will settle all marked nodes on the path in order.

As the conservative stalling approach finds optimal[3] solutions, the following holds as well:

Corollary 2
When using the **conservative stalling method** to switch from online search to upper CH search, the remaining search time in the upper CH is bounded by the total query time in the full CH.

If stall on-demand fulfills the requirement in Corollary 1, the following will hold as well:

Conjecture 1
When using the **stall on-demand** to switch from online search to upper CH search, the remaining search time in the upper CH is bounded by the total query time in the full CH.

For the third stalling technique – aggressive stalling – the presented bound does not hold. The example in Figure 9 illustrates this.

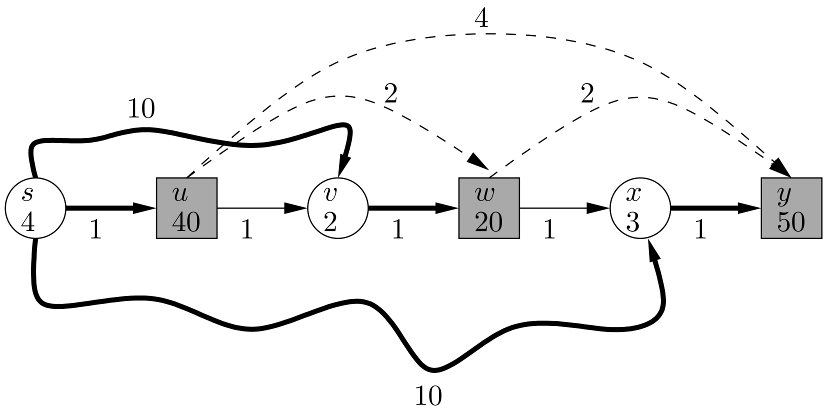

Figure 9: *The above image shows a simple example which illustrates the stalling process and the vertices from which the search in the upper CH would be started. Vertices contained in the upper CH are depicted as rectangles. The vertices used by aggressive stalling to stall the search are colored grey. Inside each vertex, its level is given. The construction of the CH will insert shortcuts (u, w), (w, y) and (u, y) depicted as dashed lines. The CH-search in the upward graph would therefore not settle w.*

[3][9] gives a hint how to make sure to find an optimal solution for cases with ambigous shortest paths.

10 Limitations and Future Work

In this section we focus on the work which is left on the algorithmic side of the problem. The model used to calculate turn costs is given in Section 6 and further modifications are already presented there. However the current implementation of the turn cost calculation also allows for possible improvements:

Lookup table for turn costs: One could apply a lookup table of precalculated turn costs for different types of junctions, to speed up the process of turn cost calculation. Each node's junction could be categorized and turn costs would then be looked up in this table.

A major limitation of the hybrid approach is the inability to unpack shortest paths. As the CH is cut at a threshold t, all shortcuts bypassing nodes with a level less than t can't be unpacked. The following improvements will be necessary:

Use Dijkstra's algorithm: To resolve the shortcuts, a number of local searches using DIJKSTRA's algorithm on the online view could be performed.

Store unpacked shortcuts: Another approach would be to include all shortcuts in the upper CH, only cutting non-shortcut edges. The missing edges and their weights would have to be extracted using the online view.

Unrelated to path unpacking, the following improvements should be made:

Integration of hybrid searches: The integration of the online search and the upper CH search should be implemented. To run the already introduced AtoA search, not only source and target nodes would be put into the priority queues, but the whole sets would be inserted at the beginning.

getTargets(): The current implementation of GETTARGETS() populates a vector with targets, which is passed by the calling function. This should be replaced by an iterator, thus preventing the materialization of the target list. At the moment, bidirectional searches only operate on forward connections. Therefore, backward edges found in GETTARGETS() have to be reverted to their corresponding forward edges. This takes time and is a limitation, that should be addressed in future works.

11 Conclusion

During this work, we examined the method of including turn costs in route planning algorithms using edge-based graphs. We found offline generated graphs containing all information about turn costs to perform well using current speedup technologies. However, due to the increasing graph size, the preprocessing and query times increased as well. This came partially unexpected, as all vertices created out of a junction are located close to each other and one would have expected, that contraction based techniques eliminated those additional vertices during the first steps of their preprocessing.

The online approach shows far better space consumption, however DIJKSTRA's algorithm has to be used without speedup technologies. Here the bidirectional search performs best and could be further improved. Especially, little of the conceptual idea of interface graphs is needed: only start and end nodes need to be modelled explicitely.

Most promising looks the hybrid approach while also being unfinished and limited to a conceptual idea. We already implemented stalling techniques. We also showed, where further improvements are necessary.

References

[1] G. Veit Batz, Robert Geisberger, and Peter Sanders. Time Dependent Contraction Hierarchies - Basic Algorithmic Ideas. Technical report, ITI Sanders, Faculty of Informatics, Universität Karlsruhe (TH), `http://algo2.iti.uni-karlsruhe.de/documents/tdch.pdf`, 2008. arXiv:0804.3947.

[2] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik 1*, pages 269 – 271, 1959.

[3] Robert Geisberger. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. Master's thesis, Universität Karlsruhe, 2008.

[4] Joel Lovell. Left-Hand-Turn Elimination. `http://www.nytimes.com/2007/12/09/magazine/09left-handturn.html?pagewanted=all`, Dec 2007.

[5] Kirill Müller. Studienarbeit: Berechnung kürzester Pfade unter Beachtung von Abbiegeverboten. `http://i11www.iti.uni-karlsruhe.de/teaching/theses/files/studienarbeit-kmueller-05.pdf`, 2005.

[6] P. Sanders and D. Schultes. Highway hierarchies hasten exact shortest path queries. In *Proceedings of the 13th Annual European Symposium on Algorithms (ESA '05)*, volume 3669 of *LNCS*, pages 568–579. Springer, 2005.

[7] P. Sanders and D. Schultes. Engineering highway hierarchies. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA '06)*, volume 4168 of *LNCS*, pages 804–816. Springer, 2006.

[8] Wolfgang Schmid. *Berechnung kürzester Wege in Straßennetzen mit Wegeverboten.* PhD thesis, Universität Stuttgart, 2000.

[9] Dominik Schultes. *Route Planning in Road Networks.* PhD thesis, Universität Karlsruhe, 2008.