Julius-Maximilians-Universität Würzburg
Institut für Informatik
Lehrstuhl für Informatik I
Algorithmen, Komplexität und wissensbasierte Systeme

**Master's Thesis**

# Scalability of Route Planning Techniques and Empirical Growth of Road Network Parameters

Johannes Blum

submitted on September 6, 2017

Supervisor: Prof. Dr. Sabine Storandt

## Abstract

To accelerate the computation of shortest paths in road networks, several preprocessing-based route planning techniques have been developed, as contraction hierarchies (CH), transit nodes (TN) or hub labels (HL). On real-world instances, these techniques outperform the algorithm of Dijkstra by several orders of magnitude. To explain this, network parameters as the highway dimension and the skeleton dimension were introduced. While these parameters are conjectured to grow polylogarithmically in the size of the networks, their true nature was not thoroughly investigated before. Furthermore, the resulting theoretical bounds obtained for route planning techniques are asymptotic, which rises the question whether they describe real networks with a finite number of nodes and edges sufficiently.

In this thesis, we empirically analyze the growth of important network parameters and also the scaling behavior of several state-of-the-art route planning techniques. We describe efficient algorithms to lower bound the highway dimension and to compute the skeleton dimension, even in huge networks. This allows us to formulate new conjectures about the scaling behavior of these parameters, which could be the starting point for new theoretical investigations.

## Zusammenfassung

Um die Berechnung kürzester Wege in Straßennetzwerken zu beschleunigen wurden zahlreiche Routenplanungstechniken entwickelt, welche sich eines Vorverarbeitungsschrittes bedienen, wie etwa Contraction Hierarchies (CH), Transit Nodes (TN) oder Hub Labels (HL). In der Praxis sind solche Verfahren um mehrere Größenordnungen schneller als der Algorithmus von Dijkstra. Zur Erklärung dieses Verhaltens wurden Graphparameter wie die Highway Dimension oder die Skeleton Dimension eingeführt. Obwohl für diese Parameter ein polylogarithmisches Wachstum bezüglich der Größe des Netzwerks vermutet wird, ist ihr tatsächliches Verhalten nicht bekannt. Des Weiteren sind die resultierenden Schranken für Laufzeit und Speicherbedarf von asymptotischer Natur, was die Frage aufwirft, ob sich diese für existierende Netzwerke mit endlicher Knoten- und Kantenanzahl überhaupt gut eignen.

In dieser Arbeit analysieren wir das Verhalten wichtiger Netzwerkparameter empirisch und untersuchen das Skalierungsverhalten mehrerer aktueller Routenplanungstechniken. Wir stellen effiziente Algorithmen vor, welche die Berechnung unterer Schranken für die Highway Dimension sowie die Berechnung der Skeleton Dimension selbst in großen Netzwerken ermöglichen. Darauf aufbauend schlagen wir neue Modelle für das Wachstum beider Parameter vor, welche die Grundlage weiterer Forschung sein könnten.

## Acknowledgement

I would like to thank my supervisor Sabine Storandt for her support and many illuminating discussions. Moreover, I would like to thank everyone at Chair I for making work fun.

# Contents

# 1 Introduction

Finding the shortest path in a road network is a very natural problem that gained a lot of interest in the last decade. In general graphs, shortest paths can be computed with the well-known algorithm of Dijkstra (assuming non-negative edge weights), which has runtime $\mathcal{O}(m + n \log n)$ where $n$ and $m$ denote the number of nodes and edges, respectively [Dij59]. On large networks, this takes however several seconds. As a consequence, several preprocessing-based techniques have been developed, as contraction hierarchies (CH [GSSV12]), transit nodes (TN [BFSS07]) or hub labels (HL [ADGW11]), among others [BDG$^+$16]. These acceleration techniques include a preprocessing step where auxiliary data is computed that then enables the answering of shortest path queries within milliseconds or even microseconds. The required preprocessing time and space consumption is usually moderate.

The empirical justification stems from the investigation of real-world road networks, extracted, e.g., from OpenStreetMap (Germany about 20 million, Europe 174 million nodes), TIGER data (USA 24 million nodes), or PTV graphs (Europe about 19 million nodes) and other networks provided for the corresponding DIMACS challenge[1]. But for all of the above listed acceleration techniques, one can construct artificial input networks on which they perform unsatisfactorily. This inspired the question what characteristics of real road networks enable their great performance. As a result, new network parameters have been designed which try to capture the essence of road networks:

Abraham et al. [AFGW10] introduced the notion of highway dimension. Intuitively, a small highway dimension $h$ implies the existence of small local hitting sets for the shortest paths around every node. It was shown that, with a special preprocessing, one can obtain query times of $\mathcal{O}(h \log D)$ and a space consumption of $\mathcal{O}(nh \log D)$ for CH, TN and HL where $D$ denotes the network diameter, i.e., the length of the longest shortest path. For TN, there is also a preprocessing variant with a query time of $\mathcal{O}(h^2)$ and a space consumption of $\mathcal{O}(hn + m)$. The proposed preprocessing is however NP-hard as it is based on computing optimal hitting sets. Using the standard greedy approach to solve the underlying hitting set problem leads to a polynomial preprocessing algorithm with query time and space consumption increasing by a factor of $\log h$. On large networks, this approach is not practical either.

An alternative parameter for the analysis of HL is the skeleton dimension $k$, which bounds the width of small subgraphs of the shortest path trees in the network (the *skeletons*) [KV17]. It was shown that with a randomized polynomial time preprocessing algorithm, one can achieve query times of $\mathcal{O}(k \log D)$ and a space consumption of $\mathcal{O}(nk \log D)$ in expectation.

[1] `http://www.dis.uniroma1.it/challenge9/download.shtml`

The nature of the above-mentioned parameters is however unknown. While they are conjectured to grow at most polylogarithmically in the size $n$ of the road network, grid instances with $h, k \in \Theta(\sqrt{n})$ are known. As grid substructures are ubiquitous in real-world road networks, computing $h$ and $k$ on a variety of instances is necessary to gain insights in their real dependency on $n$. Preliminary results for $h$ and $k$ are available for New York (about 200,000 nodes), where $h > 173$ and $k = 73$ is known [KV17], while for the US road network $h > 1000$ was reported[2]. These $h$-values are certainly larger than $\log_2 n$ (18 and 25 respectively). But of course, the conjecture about polylogarithmic dependency is not invalidated by this, as it could be that $h = a \log^b n$ for suitable constants $a$ and $b$. In this thesis we present scaling experiments performed on both real-world and artificially generated road networks in order to gain more insight into the growth behavior of the mentioned network parameters.

Moreover, the analysis of CH, TN and HL in dependency of $h$ reveals similar asymptotic search space sizes and space consumption bounds for all three of them. In contrast, empirical observations indicate that wrt query time, we have CH $>$ TN $>$ HL and wrt space consumption we have CH $<$ TN $<$ HL (see [BDG$^+$16], Table 1). So far, it remains unclear if this difference between theory and practice can be explained by asymptotic bounds versus finite networks (or in other words, is there some $n$ for which our current observations would no longer be true?), or if the notion of highway dimension is not fine-grained enough to differentiate between the scaling behavior of these techniques sufficiently. Hence another goal of this thesis is to shed some light on this question.

## 1.1 Contribution and Outline

- We first give an overview over some shortest path algorithms, namely Dijkstra's algorithm, CH, TN and HL; see Chapter 2.
- We then provide new and efficient algorithms for lower bounding the highway dimension and for exact computation of the skeleton dimension in large networks; see Chapter 3.
- We review different approaches for the generation of large road networks as required for our scaling experiments; see Chapter 4.
- The scaling behavior of the discussed route planning techniques and road network parameters (on networks with $10^3$ to more than $10^8$ nodes) is investigated experimentally in Chapter 5. As our core results we provide empirical evidence that $h$ and $k$ grow indeed differently in $n$, and that the route planning techniques exhibit growth functions that are surprisingly different from the predictions made in dependency of $h$.
- Finally, we draw some conclusions and provide directions for future work; see Chapter 6.

[2] `https://www.slideshare.net/csclub/andrew-goldberg-highway-dimension-and-provably-efficient-shortest-path-algorithms`

# 2 Route Planning Techniques

In this chapter we outline the classic algorithm of Dijkstra as well as the preprocessing-based techniques contraction hierarchies, transit nodes and hub labels.

In the following we assume that we are given a road network as an undirected graph $G = (V, E)$ with positive edge weights $\ell \colon E \to \mathbb{N}$ where shortest paths are unique; the latter can be achieved, e.g., by symbolic perturbation. We abbreviate $\ell(\{u, v\})$ by $\ell(u, v)$. A *path* is a sequence of nodes $v_1, \dots, v_k$ where $\{v_i, v_{i+1}\} \in E$ for all $i = 1, \dots, k-1$.

Given a source node $s \in V$ and a target node $t \in V$, the shortest path from $s \in V$ to $t \in V$ is denoted by $\pi(s, t)$. We consider the problem of finding the length $d_s(t)$ of this shortest path. Note, however, that all presented route planning techniques can easily be adapted to compute actual shortest paths instead of shortest path distances.

## 2.1 Dijkstra's Algorithm

The probably best-known method to compute shortest paths is the algorithm of Dijkstra [Dij59]. Starting from a start node $s$, it keeps a distance label for every node. In every step it considers a node $v$ with minimal label that has not been chosen so far and relaxes all edges $\{v, w\}$ incident to $v$. If such an edge leads to a new shortest path to $w$, its distance label is updated accordingly. Using a Fibonacci heap as the central data structure, one obtains a runtime of $\mathcal{O}(m + n \log n)$.

## 2.2 Preprocessing-based Algorithms

For many applications, however, Dijkstra's algorithm is too slow. A common concept to overcome this is the idea of preprocessing-based algorithms. There, in an initial step auxiliary data is precomputed, which is subsequently used to quickly answer shortest path queries. For instance, precomputing all pairwise shortest path distances reduces the query time to $\mathcal{O}(1)$, but also requires $\Omega(n^2)$ space, which renders this approach impracticable. Hence, any reasonable preprocessing-based acceleration technique has to provide some trade-off between query time and space consumption. CH, TN and HL, the algorithms described in the following, are empirically proven to provide sensible trade-offs on road networks. Furthermore, experiments indicate that they all hit Pareto points: On the Western Europe network (PTV, 18 million nodes, 42.5 million directed edges), the space consumption of CH is about 0.4 GB; for TN it's 6 times higher, for HL 47 times higher. Regarding query times, HL needs 0.56 $\mu$s; query times for TN are higher by a factor of 4 and for CH by a factor of 196 [BDG$^+$16]. A natural question is, whether these relations are generally true (invariant of the scale of the network); or if the results are artifacts of evaluation on a 'small' network or sensitive to the chosen implementation.

The performance of shortest path algorithms is usually measured in terms of space consumption and search space size, i.e. the number of nodes and edges considered during a query. The $h$-dependent analysis indicates that, up to constant factors, the mentioned techniques exhibit the same scaling behavior: the space consumption is in $\mathcal{O}(nh \log D)$ and the search spaces are in $\mathcal{O}(h \log D)$. For TN, there is also a variant with a query time of $\mathcal{O}(h^2)$ and a space consumption of $\mathcal{O}(hn)$ [ADF+13]. For CH, it depends whether the search space is just defined as the number of settled nodes or if also edge relaxations are counted. In the latter case, the query time rises to $\mathcal{O}((h \log D)^2)$. But the preprocessing schemes used to prove these bounds differ significantly from the preprocessing schemes used in practice, as we will discuss more thoroughly in the following.

## Contraction Hierarchies

The CH algorithm is based on the notion of node contraction [GSSV12]. During the preprocessing, nodes are successively removed from the network. Whenever a node $v$ to be deleted lies on the shortest path between two of its neighbors $u$ and $w$, a shortcut edge $\{u, w\}$ of length $\ell(u, w) = \ell(u, v) + \ell(v, w)$ is inserted (cf. Figure 2.1). Eventually, all nodes have been removed and we obtain a set $E^+$ consisting of all shortcuts created during this process. The query algorithm operates on the resulting shortcut graph $G^+ = (V, E \cup E^+)$, so the space consumption is determined by $|E^+|$.

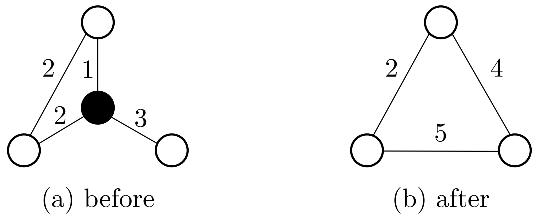

Figure 2.1: Example of a node contraction. The graph is shown before and after contracting the black node.

Let $rank(v)$ be the rank of a node $v$ in the contraction order. The upward graph $G^{\uparrow}(v)$ consists of all paths $v_1, \ldots, v_k$ in $G^+$ originating from $v$ that point upwards in the hierarchy, i.e. that satisfy $rank(v_i) < rank(v_{i+1})$ for $i = 1, \ldots, k-1$. Given two nodes $s, t \in V$, the shortest path distance $d_s(t)$ is computed by performing Dijkstra runs from the nodes $s$ and $t$ in the upward graphs $G^{\uparrow}(s)$ and $G^{\uparrow}(t)$, respectively. Both runs settle the node that was contracted last on the original shortest path from $s$ to $t$ in $G$. Hence the shortest path distance can be computed by identifying the node $p$ that minimizes $d_s(p) + d_t(p)$. An example of a CH query is illustrated in Figure 2.2.

It was shown that any contraction order leads to correct query answering, but the space consumption and the search space sizes depend on the chosen permutation. In practice, a simple greedy preprocessing scheme produces good results. In every step

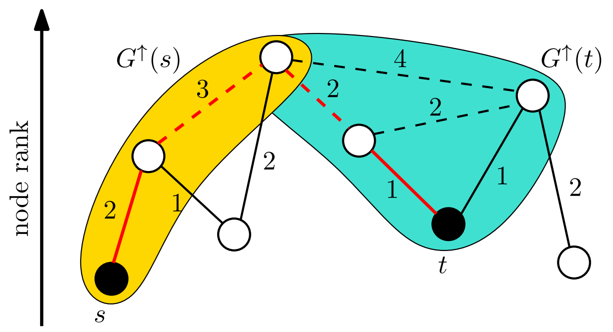

Figure 2.2: Example of a CH query. The upward graphs of the source $s$ and the target $t$ and the shortest path are highlighted in yellow, green and red, resp. Solid lines denote edges in the original network, dashed lines denote shortcuts.

it contracts the node that currently minimizes the number of shortcuts to be inserted. More complicated selection functions were also tested [GSSV12].

A treewidth-$t$-based analysis of CH [BCRW13] showed that query times are in the order of $\mathcal{O}(t \log n)$ and the space consumption is bounded by $\mathcal{O}(nt \log n)$. This analysis inspired a different CH construction scheme based on nested dissections, which also performs very well in practice [DSW14]. The preprocessing scheme used in the highway-dimension-based analysis involves the enumeration of all shortest paths in the network and computing (approximate) hitting sets. This takes at least superquadratic time and uses quadratic space in $n$. Therefore, this scheme cannot be applied to large real-world networks [ADF+13].

### Transit Nodes

The TN algorithm is based on the idea that all shortest paths from a node $v$ to all 'far away' destinations pass through some small set $AN(v)$ of so-called access nodes (e.g. slip roads of nearby interstates) [BFSS07]. The union of all access nodes forms the transit node set $T$. For every pair of transit nodes, the shortest path distance is precomputed and stored in a look-up table. In addition, every node stores the distances to all its access nodes. So the total space consumption can be expressed as $|T|^2 + \sum_{v \in V} |AN(v)|$. Therefore, to compute the distance between 'far away' node $s$ and $t$, it suffices to check all access node distances of $s$ and $t$ and the respective distances between them in the look-up table, i.e. to compute

$$\min_{u \in AN(s), v \in AN(t)} d_s(u) + d_u(v) + d_v(t).$$

All this distances are precomputed, so the query time is in $\mathcal{O}(|AN(s)| \cdot |AN(t)|)$.

If $s$ and $t$ are not 'far away', the resulting distance is however not necessarily correct. Hence, a fallback algorithm needs to be used for 'short' queries, e.g. the algorithm of Dijkstra. So strictly speaking, the performance of TN does not solely depend on the number of access nodes, but also on the maximum length of all shortest paths not containing any transit node, as the runtime of the fallback algorithm usually depends on this distance.

To distinguish between 'long' and 'short' queries, a so called *locality filter* is used. Formally, a locality filter for a fixed distance $r$ is an oracle which answers if the shortest path distance between two nodes $s$ and $t$ is at most $r$ (false positives being allowed) [ALS13]. One possibility for such a locality filter is to treat every query as a 'long' query first and to check whether the returned distance is at least $3r'$, where $r'$ is the maximum distance between all nodes and their furthest access node [EF12]. If this is the case, the result is correct (i.e. the oracle returns false), otherwise we perform a 'short' query that is pruned at radius $3r'$ and return the minimum of both results.

There are two different paradigms for the construction of the transit node set, bottom-up and top-down.

In the *bottom-up* approach, the radius $r$ for a 'long' query is fixed a priori. Then, for each $v \in V$ access nodes are selected such that all shortest paths longer than $r$ emerging from $v$ contain an access node. The size of the transit node set is only known afterwards. This paradigm was applied in [EF12] by extracting all shortest paths of length $r$ and then computing a greedy hitting set for those. With the help of instance based lower bounds, it was shown that the computed transit node set sizes are close to the possible minimum for the given $r$. But it was also noted, that this approach scales badly with the network size, as extracting and storing all shortest paths of length $r$ is time and space consuming, especially for larger $r$.

In the *top-down* approach, the set of transit nodes of given size $c$ is fixed first. Then access nodes for each $v \in V$ are computed by running Dijkstra until each active shortest path contains a transit node. The radius $r$ can then only be determined a posteriori as it is the maximum length of all shortest paths not containing any transit node. In [ADF$^+$13], this paradigm was followed to prove theoretical bounds for TN in dependency of $h$. There, first $|T| \leq \sqrt{m}$ got fixed and hitting sets $T_r$ for $r = D, D/2, \cdots$ were computed to choose the smallest $r$ such that $|T_r| \leq \sqrt{m}$ holds. As this approach invokes multiple hitting set computations and also has to consider large $r$, it scales even worse. In practice, an efficient method for TN computation also follows the top-down paradigm, but is based on CH [ALS13]. Here, the transit node set is chosen as the $c$ nodes of highest *rank* for some $c$. Usually, $c \in \Theta(\sqrt{n})$ provides a reasonable trade-off between space consumption and query times. To determine the access nodes $AN(v)$ of a node $v$, a Dijkstra run is performed in the upward graph $G^{\uparrow}(v)$ where on every branch of the shortest-path tree the first encountered transit node is selected.

### Hub Labels

In the HL algorithm, a set $L(v)$ of hub labels is assigned to every node $v$. A hub label is a node $w$, together with its distance $d_v(w)$ from $v$. The label sets are required to fulfill the *cover property*, that is, for every pair $s, t \in V$, the intersection $L(s) \cap L(t)$ of the label sets contains a node $w$ on the shortest path from $s$ to $t$. Then queries can be answered by simply summing up $d_s(w) + d_t(w)$ for all $w \in L(s) \cap L(t)$ and keeping track of the minimum, i.e. computing

$$\min_{w \in L(s) \cap L(t)} d_s(w) + d_t(w).$$

This can be done by a merging-like step if the label sets are presorted by node IDs. Hence the query time is in $\mathcal{O}(|L(s)| + |L(t)|)$, while the space consumption is in $\mathcal{O}(\sum_{v \in V} |L(v)|)$. The goal is therefore to find small hub label sets.

Abraham et al. [ADF+13] showed that hub labels can be constructed by computing multiple hitting sets $H_r$ for sets of shortest paths with length $r = 1, 2, 4, \ldots, D$. The label set of a single node $v$ is then determined by $L(v) = \bigcup_r (H_r \cap B_{2r}(v))$ where $B_{2r}(v)$ denotes the set of all nodes whose distance from $v$ is at most $2r$. Kosowski and Viennot [KV17] presented a more practical algorithm for HL. There, a shortest path tree is computed for each node. Then hub labels are selected on certain subpaths via a randomized process. A thorough analysis shows that this leads on average to the selection of $\mathcal{O}(k \log D)$ labels per node, so the total space consumption is in $\mathcal{O}(nk \log D)$. Abraham et al. [ADF+11] observed that hub labels can also be computed based on CH. More precisely, performing a Dijkstra run in the upward graph $G^+(v)$ of a node $v$ and choosing the nodes in $G^+(v)$ with the computed distances as the hub label set $L(v)$ leads to a correct HL data structure. The resulting label sets can however further be pruned, as there might be nodes for which the distance from $v$ in the upward graph $G^+(v)$ is larger than in the actual network $G$ (cf. Figure 2.3). Such superfluous labels could be identified and subsequently removed e.g. by an additional one-to-all shortest path computation from $v$ that is pruned at the distance of the furthest hub label.

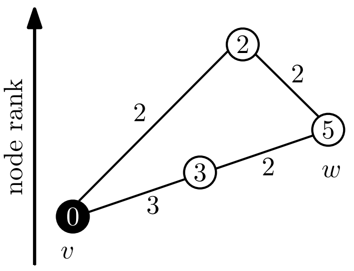

Figure 2.3: An example of a superfluous label in an HL data structure constructed based on CH. In the upward graph $G^{\uparrow}(v)$, the node $w$ has a distance of 5 from $v$. The shortest path distance in $G$ is however 4, so $w$ can be pruned from $L(v)$.

## 2.3 Implementation

In conclusion, the preprocessing schemes that allow to derive theoretical bounds in dependency of $h$ are impractical due to their high complexity and time/space consumption. Note that this even applies to the polynomial time variants (where hitting sets are not computed exactly but via the standard greedy algorithm, at the cost of slightly increased bounds).

So for a fair comparison of the three route planning techniques in our scalability study, we will exploit the fact that for all three schemes there exists a preprocessing variant based on CH. In particular, we use the following approaches:

- For CH we use the greedy preprocessing scheme described of Geisberger et al. [GSSV12] where in every step a node minimizing the number of shortcuts to be inserted is contracted.
- For TN we select the $5\sqrt{n}$ nodes of highest rank in the contraction order as transit nodes as described by Abraham et al. [ADF+11].
- For HL we use the CH-based approach of Abraham et al. [ADF+11] with subsequent pruning.

# 3 Scalable Computation of (Road) Network Parameters

To be able to compare the scaling behavior of route planning techniques to the growth of the network parameters used in theoretical analyses, we describe efficient ways to bound these parameters or to compute them even exactly.

## 3.1 Parameters, Complexity of Computation and Relations

We focus on the highway dimension $h$ and the skeleton dimension $k$, as these parameters were explicitly designed to analyze road networks. But as described above also the treewidth $t$ – a classical network parameter – has been used to prove small search spaces and low space consumption for CH [BCRW13]. It is NP-complete to decide whether a graph has treewidth at most $t$ [ACP87]. Upper bounds for selected road networks were reported by Dibbelt et al. [DSW14], e.g. $t \leq 479$ for the Europe network.

### Highway Dimension

The highway dimension is defined as follows. Let $B_r(v)$ be the ball around a vertex $v$ with radius $r$ which consists of all nodes with a shortest path distance of at most $r$ from $v$. Furthermore, let $S_r(v)$ be the set of all shortest paths that intersect $B_r(v)$ and have a length in the range $(r/2, r]$. The highway dimension is the smallest $h \in \mathbb{N}$ such that, for all radii $r$ and for all nodes $v \in V$, there exists a hitting set $H$ for $S_{2r}(v)$ of size at most $h$ (i.e. $H \subseteq V$ and for every $P \in S_{2r}(v)$, we have $H \cap P \neq \emptyset$) [ADF+11]. Bauer et al. [BCRW13] showed that for a given network $G$, there exist edge lengths such that $h(G) \geq (pw(G) - 1)/(\log_{3/2} n + 2)$ where $pw(G)$ denotes the pathwidth of $G$. As $pw(G) \geq t(G)$, this inequality also relates $h$ and $t$. Deciding whether a network exhibits a highway dimension of at most $h$ is NP-hard [FFKP15].

### Skeleton Dimension

To introduce the skeleton dimension, we establish the following notation. The geometric realization of a graph $G = (V, E)$ is denoted by $\tilde{G} = (\tilde{V}, \tilde{E})$. Intuitively it consists of infinitely many infinitely short edges such that, for every edge $\{u, v\} \in E$ and every $\alpha \in [0, 1]$, there exists a node $w \in \tilde{V}$ satisfying $\ell(u, w) = \alpha\ell(u, v)$ and $\ell(w, v) = (1-\alpha)\ell(u, v)$. The shortest path tree of a node $s$ which is directed from the root $s$ towards the leaves is denoted by $T_s$. For nodes $s, v \in \tilde{V}$, we define $Reach_s(v)$ as the shortest path distance of $v$ to its furthest descendant in $\tilde{T}_s$.

The *skeleton* $T_s^*$ rooted at a node $s \in V$ is the subtree of $\tilde{T}_s$ induced by all nodes $v \in \tilde{V}$ satisfying $d_s(v) \leq 2 \cdot \mathit{Reach}_s(v)$. Intuitively, we obtain $T_s^*$ by cutting every branch of the shortest path tree $T_s$ at two thirds of its length.

The width of a tree $T$ rooted at $s$ is the maximum number of nodes at a certain distance from $s$, i.e. $\mathit{Width}(T) = \max_r |\mathit{Cut}_r(\tilde{T})|$ where $\mathit{Cut}_r(\tilde{T})$ denotes the nodes of $\tilde{T}$ at distance $r$ from $s$. Finally, the *skeleton dimension* $k$ of $G = (V, E)$ is defined as the maximum width of the skeletons of all shortest path trees, so $k = \max_{u \in V} \mathit{Width}(T_u^*)$ [KV17].

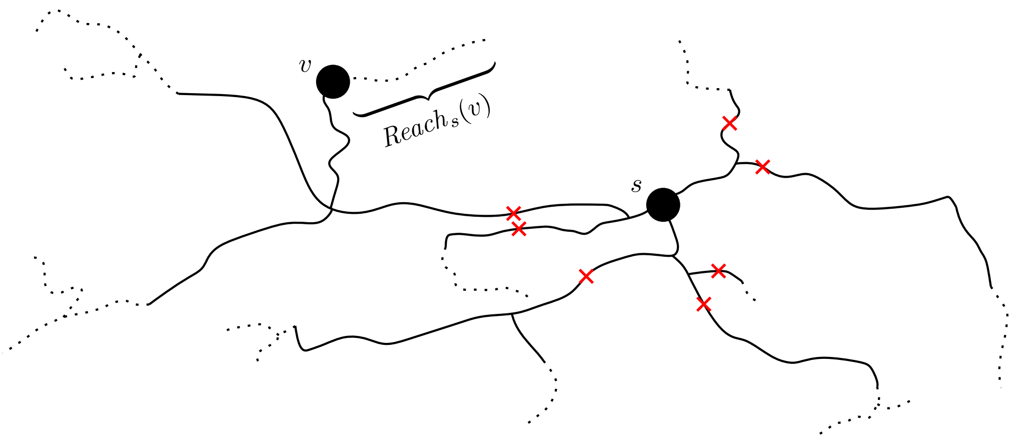

Figure 3.1: An illustration of a shortest path tree $T_s$ and its skeleton $T_s^*$. The red marks indicate a cut $\mathit{Cut}_r(T_s^*)$ of size 7.

For graphs with bounded maximum degree (which is the case in road networks), $k \in \mathcal{O}(h)$ was proven. In contrast to $t$ and $h$, the skeleton dimension $k$ can be computed in polynomial time by inspecting all shortest path trees in $G$. But for large networks, computing all shortest path trees is far too time consuming. Therefore, we will propose a more efficient approach for determining $k$.

## 3.2 Lower Bounds for the Highway Dimension

Computing the skeleton dimension $h$ exactly seems to be impossible, as this would require to extract and solve for each $v \in V$ and for each $r \in [1, D]$ the respective hitting set instance on $S_{2r}(v)$. Even just the extraction of the instances, requiring Dijkstra computations from all $w \in B_{2r}(v)$ and storing all shortest paths in $S_{2r}(v)$, is too demanding for larger $r$ to be practical.

### Path Packings

We observe, however, that for any node $v \in V$, for any radius $r$, and for any subcollection of the paths in $S_{2r}(v)$, any lower bound on the size of the hitting set $H$ for these paths is also a lower bound for $h$. Clearly, for any packing $S$ of disjoint shortest paths from

$S_{2r}(v)$, it holds that $|H| \geq |S|$, as no two paths from $S$ can be hit by one single node. In a greedy approach to generate such a packing, one would iteratively select a path from $S_{2r}(v)$ that intersects the fewest other paths. But as we already pointed out, this idea disqualifies already for moderate radii, as computing and storing $S_{2r}(v)$ is too expensive. To improve this algorithm, we could proceed as follows. Instead of explicitly enumerating all shortest paths $S_{2r}(v)$, we perform a Dijkstra run from every node $w \in B_{2r}$ which is pruned at distance $r + \delta$, where $\delta$ is the maximum length of all edges incident to $w$. From the resulting shortest path tree we select a set of disjoint shortest paths that have length greater than $r$. However, if at some point all active branches of the Dijkstra tree intersect an already chosen path, the current computation can be pruned. A more efficient algorithm to find a packing of disjoint shortest paths from $S_{2r}(v)$ is to simply perform a single Dijkstra run from the node $v$ that is pruned at radius $3r$ and to partition the resulting shortest path tree into disjoint paths of length larger than $r$.

## Hitting Sets

We obtained the best results, however, by directly considering the hitting set problem on a reasonably sized subset $S$ of $S_{2r}(v)$ that is constructed as follows. First we perform a Dijkstra run from $v$ that is pruned at distance $3r$. In the resulting shortest path tree we backtrack the paths of all leaves up to a distance of $r$ (provided they are not too close to the root $v$) and add them to our collection $S$. Then for parameters $c_s$ and $\alpha < 1$, we select $c_s$ other nodes from $B_{\alpha 2r}(v)$ and run Dijkstra computations up to a radius of $r+\delta$, where $\delta$ is the maximum length of all edges incident to the root. Again, we backtrack paths of sufficient lengths from the leaves and add them to $S$.

On the resulting hitting set instance $S$ we run the standard greedy algorithm that always selects the node that hits the largest number of paths. This algorithm has an approximation guarantee of $H_b$ (the $b$-th harmonic number), where $b$ is the maximum number of paths that can be hit with a single node [Joh74, Lov75]. Therefore, a greedy solution $H$ implies a lower bound of $|H|/H_b$, which is however somewhat pessimistic. A more promising (though computationally more expensive) method to compute a lower bound is to solve the relaxation of the following hitting set ILP.

$$\begin{aligned} \text{minimize:} \quad & \sum_{u \in B_{2r}(v)} x_u \\ \text{subject to:} \quad & \sum_{u \in P} x_u \geq 1, \quad \forall P \in S \\ & x_u \in \{0,1\}, \quad \forall u \in B_{2r}(v) \end{aligned}$$

To reduce the size of the ILP, we let $P_u$ be the set of paths hit by a node $u$ and choose at most $c_p$ paths from each $P_i$ for some parameter $c_p$. With examples from each $P_u$ considered in the ILP, we hope to preserve the essence of the instance. To improve the runtime of our algorithm, we construct a greedy solution first and solve the ILP relaxation only if this solution exceeds the best lower bound computed so far by a certain factor $\beta$. If the ILP relaxation improves the current lower bound, we update it accordingly and proceed in any case by selecting a new source node $v$.

### Choice of Parameters

For choosing the radius $r$, we compute the average shortest path distance $d_{\text{avg}}$ in the network by performing random $s$-$t$-queries. Then we set $r = d_{\text{avg}}/10$. This turned out to be a reasonable choice, as for larger radii, the size of $S$ decreases because the boundary of the network is hit while for smaller radii the size of $S$ decreases due to a smaller ball $B_{2r}(v)$ (cf. Section 5.2). For the choice of the parameters $c_s$ and $c_p$ one should note that larger values lead to a larger hitting set instance $S$ (and therefore likely to better bounds), but also increase the runtime. For the parameter $\beta$, the average ratio between the solutions of the greedy algorithm and the ILP relaxation is a good choice. A small parameter $\alpha$ reduces the overlap between the shortest paths chosen in the first and second step of the construction of $S$, but also increases the overlap within the paths added in the second step. We made good experiences with $\alpha = 0.95$.

## 3.3 Computing the Skeleton Dimension

In contrast to the highway dimension, the skeleton dimension can be computed in polynomial time (by determining the skeleton for each vertex and its width). To make large road network instances tractable, however, all these steps need to be done very efficiently. In the following, we provide details of the naive algorithm and our improvements.

### Computing the Width of a Tree

The width of a given rooted tree $T$ can be computed by iterating over the nodes by increasing distance from the root and storing the (target nodes of the) currently cut edges in a priority queue. In every step we pop all nodes from the priority queue that have the same distance label as the top element and push their direct descendants. The width of the tree is the maximal size of the priority queue during this process, which takes $O(n \log n)$ time.

### Naive Algorithm

Naively, one would simply run Dijkstra's algorithm from every node, determine the distance to the furthest descendant in every shortest path tree, construct the skeletons by pruning nodes whose furthest descendant is too close[1], and then compute the widths. In order to compute $Reach_s(v)$ for every node $v$ (the distance of $v$ to its furthest descendant in $T_s$), one can iterate over the nodes in the shortest path tree $T_s$ in reverse topological order (starting with the node of maximum distance from $s$) and propagate the distance of the furthest descendant of every node to its predecessor. But as running a one-to-all Dijkstra computation alone is already expensive on large networks (order of multiple seconds), and the computation of the furthest descendants adds to that, one cannot afford to repeat this procedure for millions of nodes.

[1]As the skeleton is based on the geometric realization, we might also be required to insert an additional node $\tilde{w}$ at the end of every branch that satisfies $d_v(\tilde{w}) = 2Reach_v(\tilde{w})$.

## PHAST

CH, TN and HL are designed to answer point-to-point shortest path queries. For one-to-all queries, the PHAST algorithm was developed, which usually outperforms Dijkstra's algorithm [DGNW11]. Based on CH, it works on a CH data structure $G^+ = (V, E \cup E^+)$. To compute the shortest path distances from a node $s$ to all other nodes, a Dijkstra run is executed in the upward graph $G^\uparrow(s)$ of $s$, before the algorithm iterates over all edges $E \cup E^+$. During this edge sweep, edges $\{u, v\}$ are considered in descending order by $\min\{\mathit{rank}(u), \mathit{rank}(v)\}$. Whenever an edge decreases the shortest path distance of an incident node, its distance and predecessor labels are updated. Eventually, every node is labeled with the correct distance. The predecessor labels correspond however only to a shortest path tree in $G^+$ (which includes shortcuts) and the algorithm does not construct any explicit topological order. This disables the naive propagation of furthest descendant distance labels as described previously for shortest path trees computed by Dijkstra's algorithm.

Therefore we propose the following approach to compute $\mathit{Reach}_s(v)$ for every node $v$ based on PHAST. After running a normal PHAST query, we iterate over all nodes in contraction order (i.e. nodes of low rank first) and propagate the distance of the furthest descendant of every node to its predecessor. When this step is finished, we have determined $\mathit{Reach}_s(v)$ for every node $v$, for which the path to its furthest descendant strictly decreases wrt to node ranks (cf. Figure 3.2b). In the next step, we iterate over the nodes in $G^\uparrow(s)$ in reverse topological order (that is obtained through the Dijkstra run in $G^\uparrow(s)$) and propagate the descendant labels again to the predecessors. Now, the distance of every node to its furthest descendant in the shortest path tree wrt $G^+$ is known (cf. Figure 3.2c). To propagate the correct value also to the nodes, that are shortcut on the shortest path from $s$ to their furthest descendant, we sweep again over all edges $E \cup E^+$ as in the PHAST algorithm. For every edge $\{v, w\}$ we check if it is contained in the actual shortest path tree $T_s$, which is the case if $d_s(v) + \ell(v, w) = d_s(w)$. If this holds, we propagate the descendant label of $v$ to $w$. With this approach, we can compute $\mathit{Reach}_s(v)$ for every node $v$ and some root $s$ in the time required for one Dijkstra run in the upward graph $G^\uparrow(s)$ and two linear sweeps over the edges of $G^+$ (creating only a mild overhead of one edge sweep over PHAST).

Still, running PHAST for every node in the network is demanding. Hence, we now describe several ideas that allow to spare a lot of these computations.

## Upper and Lower Bounds

The width of any skeleton $T_s^*$ is a valid lower bound for $k$. Moreover we can observe that, given a supergraph $T'$ of a tree $T$, the width of $T'$ is an upper bound on the width of $T$. Hence, if for every skeleton $T_s^*$ there is a supergraph $T_s'$ that suffices $\mathit{Width}(T_s') \leq k'$ for some $k'$, it follows that $k \leq k'$. We will now concentrate on the efficient computation of small supergraphs in order to obtain good upper bounds on $k$.

Kosowski and Viennot [KV17] suggested the following method to compute reasonable sized supergraphs of shortest path tree skeletons based on the notion of (global) reach.

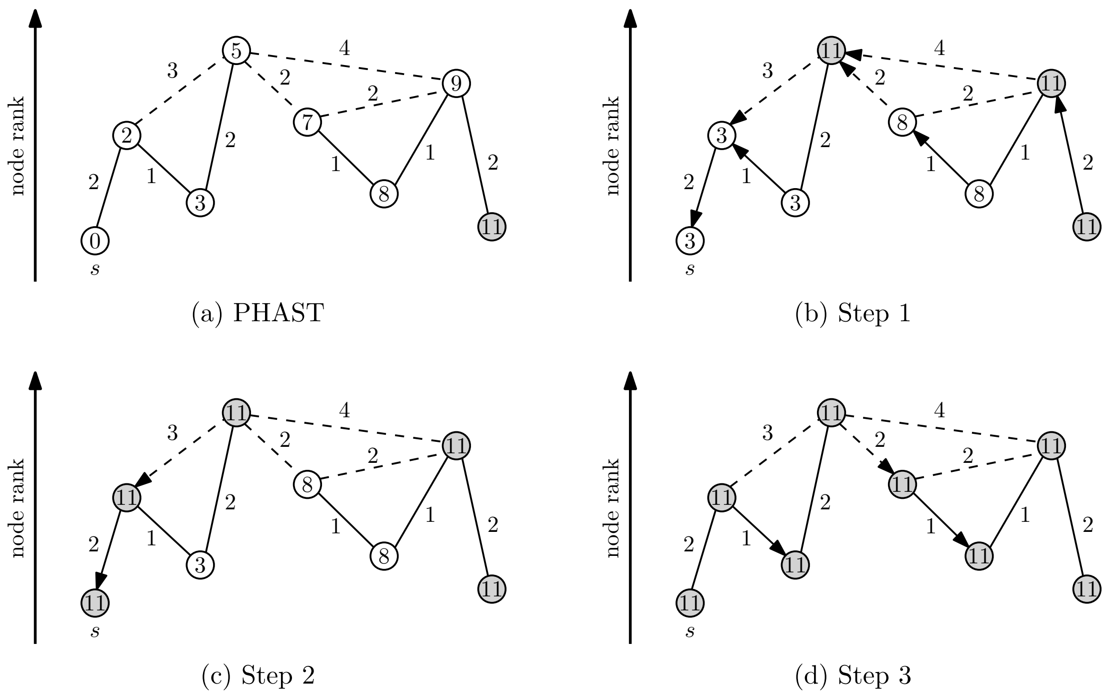

Figure 3.2: Example of the computation of $\mathit{Reach}_s(v)$ based on PHAST. Arrows denote label propagation, gray nodes are labeled correctly.

Given a shortest path $\pi(s,t)$ and a node $v \in \pi(s,t)$, the reach of $v$ wrt $\pi(s,t)$ is defined as the minimum length of the two subpaths $\pi(s,v)$ and $\pi(v,t)$. The (global) reach $\mathit{Reach}(v)$ of a node $v$ is the maximum reach of $v$ wrt to all shortest paths in the network [GKW06].

Assuming that $\mathit{Reach}(v)$ is known for every node $v$, a supergraph of a skeleton $T_s^*$ can now be computed by performing a Dijkstra run from $s$ where every node $v$ gets pruned if $d_s(v) > 2 \cdot \mathit{Reach}(v)$. The resulting trees are however relatively large compared to the skeletons and do not provide too good bounds on $k$ (cf. Section 5.2). Moreover, this algorithm requires a reach computation for all nodes in the network, which is not trivial. On the road network of North America (about 30 million nodes) computing these values takes about 11 hours[GKW06].

Therefore we propose an alternative approach, which is based on the following observation.

**Observation 3.1.** *Let $v$ be a descendant of $u$ in the shortest path tree $T_s$. Then we have $\mathit{Reach}_s(v) \leq \mathit{Reach}_u(v)$.*

*Proof.* Let $v$ be a descendant of $u$ in the shortest path tree $T_s$ and let $x$ be the furthest descendant of $v$ in $T_s$, i.e. $\mathit{Reach}_s(v) = d_v(x)$. As shortest paths are unique, $x$ is also a descendant of $v$ in the shortest path tree of $u$. This means that $\mathit{Reach}_u(v) \geq d_v(x) = \mathit{Reach}_s(v)$. □

Assume now that for some nodes $a_1, \ldots, a_c$ the distances $\mathit{Reach}_{a_i}(u)$ are already known for all $i \in \{1, \dots, c\}$ and all $u \in V$. Then for a node $s$ we can compute a supergraph of the skeleton $T_s^*$ by performing a Dijkstra search from $s$ and keeping track of the first $a_i$ encountered on every branch of the search tree. Whenever we scan a vertex $v$ such that $d_s(v) > 2 \cdot \mathit{Reach}_{a_i}(v)$ for the corresponding preceding $a_i$, it follows from Observation 3.1 that no descendant of $v$ is contained in the skeleton $T_s^*$ an we can prune the current branch. When the algorithm terminates, it has explored a supergraph of $T_s^*$.

But every branch on which none of the nodes $a_1, \dots, a_c$ was encountered has been explored entirely and can still be pruned. Therefore, we iterate over all nodes $v$ in such a branch in reverse topological order and determine the value $\mathit{Reach}_s(v)$ before discarding all nodes $v$ with $d_s(v) > 2 \cdot \mathit{Reach}_s(v)$ and adding boundary nodes $\tilde{v}$ that satisfy $d_s(\tilde{v}) = 2 \cdot \mathit{Reach}_s(\tilde{v})$. After this step, we obtain a smaller supergraph of $T_s^*$ and compute its width $\hat{k}(s)$, a valid upper bound on the width $k(s)$ of $T_s^*$.

## Exploiting Transit Nodes

The idea of our algorithm is that we compute a transit node set of the given network and use the access nodes $AN(v)$ as the nodes $a_1, \dots, a_c$ of every node $v$. As illustrated in Figure 3.3, usually for every node $v \in V$ there is a whole set of nodes $v_1, \dots, v_c$ that use the same access nodes as $v$, i.e. $AN(v) = AN(v_i)$ for $i = 1 \dots, c$. We call such a set $\{v_1, \dots, v_c\}$ also a *cell*. For every cell $\{v_1, \dots, v_c\}$ and every access node $a_j \in AN(v_1)$, we need to compute $\mathit{Reach}_{a_j}(u)$ only once in order to compute $\hat{k}(v_i)$ for all $v_i \in \{v_1, \dots, v_c\}$ as described previously. The whole network is processed by iterating over all cells via a depth-first search. As adjacent cells are very likely to share some of their access nodes, we store the computed values for $\mathit{Reach}_{a_j}(u)$ for the most recent access nodes in a least recently used cache, where the values of the least recently considered access node gets evicted when the cache is full. By doing so we can avoid computing the values of $\mathit{Reach}_{a_j}(u)$ several times for some access nodes.

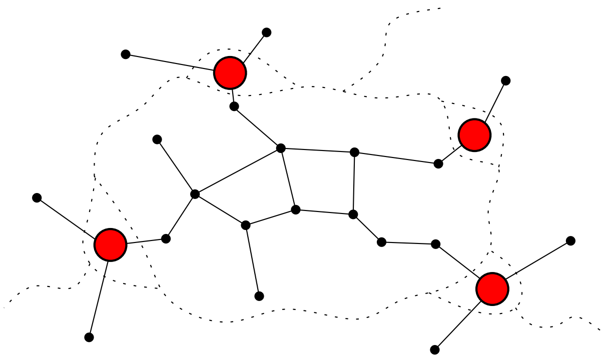

Figure 3.3: An illustration of the TN cell structure. Red nodes denote transit nodes.

## Pruning

We now define a *chain* and show, how certain nodes on a chain can be pruned during the computation.

**Definition 3.1.** *A path $v_1, \ldots, v_p$ is called a* chain *with end nodes $v_1$ and $v_p$, if $v_2, \ldots, v_{p-1}$ have degree 2 and $v_1, v_n$ don't.*

**Observation 3.2.** *Let $v_1, \ldots, v_p$ be a chain. Then*

*(i) We have $k(v_1) \leq k(v_n)$ if $\deg(v_1) = 1$.*

*(ii) For all $i \in \{2, \ldots, p-1\}$ we have $k(v_i) \leq \max\{2, k(v_n)\}$ if $\deg(v_1) = 1$.*

*(iii) For all $i \in \{2, \ldots, p-1\}$ we have $k(v_i) \leq k(v_1) + k(v_n)$.*

*Proof.* Let $v_1, \ldots, v_p$ be a chain and consider the skeleton $T^*_{v_i}$ for some $i \in \{1, \ldots, p\}$. We partition $T^*_{v_i}$ into trees $T_1$ and $T_p$, such that $T_1$ is induced by all descendants and predecessors of $v_1$ in $T^*_{v_i}$ and $T_p$ is defined analogously (cf. Figure 3.4). Let $\pi_j$ denote the geometric realization of the path $v_i, \ldots, v_j$ for $j \in \{1, p\}$. We claim that $T_1 \subseteq T^*_{v_1} \cup \pi_1$.

To prove this, consider a node $x$ of $T_1 \setminus (T^*_{v_1} \cup \pi_1)$. Then we have $d_{v_1}(x) > 2 \cdot \mathit{Reach}_{v_1}(x)$. As $x \notin \pi_1$, the node $x$ must be a descendant of $v_1$ in $T_1$, so $d_{v_i}(x) \geq d_{v_1}(x)$. From Observation 3.1 it follows that $\mathit{Reach}_{v_1}(x) \geq \mathit{Reach}_{v_i}(x)$. This means that $d_{v_i}(x) > 2 \cdot \mathit{Reach}_{v_i}(x)$, so $x$ is not contained in the skeleton $T^*_{v_i}$, which is a contradiction to the choice of $x$. Analogously one can show that $T_p \subseteq T^*_{v_p} \cup \pi_p$.

It follows that the width of $T_1$ is bounded by $k(v_1)$ (the width of $T^*_{v_1}$) and that the width of $T_p$ is bounded by $k(v_p)$. Consider now the following cases.

(i) Let $\deg(v_1) = 1$. Then we have $T^*_{v_1} = T_p$, so $k(v_1) \leq k(v_n)$.

(ii) Let $\deg(v_1) = 1$ and $i \in \{2, \ldots, p-1\}$. We have $k(v_i) = 2$ if $\deg(v_p) = 1$ as $\deg(v_i) = 2$ and $k(v_i) \leq k(v_n)$ otherwise, as $T^*_{v_1} = T_p$.

(iii) From the partitioning $T^*_{v_i} = T_1 \cup T_p$ and the observation following our claim, we obtain that $k(v_i) \leq k(v_1) + k(v_n)$. □

Note that replacing $k(v_1)$ and $k(v_n)$ on the right hand side of the inequalities by some upper bounds $\hat{k}(v_1)$ and $\hat{k}(v_n)$ does not invalidate the same. Provided that the network contains some node of degree at least 3 (which implies $k \geq 3$), we can therefore skip all nodes of degree 1 or less in our algorithm (isolated nodes do not contribute to the skeleton dimension at all). Consider now a node $u$ with $\deg(u) = 2$ that lies on a chain with end nodes $v$ and $v'$. Then we can simply choose $\hat{k}(u) = \max\{2, \hat{k}(v)\}$ if $\deg(v') = 1$, $\hat{k}(u) = \max\{2, \hat{k}(v')\}$ if $\deg(v) = 1$ and $\hat{k}(u) = \hat{k}(v) + \hat{k}(v')$ otherwise.

This requires however that $\hat{k}(v)$ and $\hat{k}(v')$ are already known. In every cell we consider therefore all non degree 2 nodes before processing the degree 2 nodes. It may however happen that the bound $\hat{k}(u) = \hat{k}(v) + \hat{k}(v')$ is not very tight. At the beginning of our algorithm we compute therefore a lower bound $\check{k}$ on the skeleton dimension $k$ by computing the width of a few skeletons and choosing $\check{k}$ as the maximum of this widths. Then we use the bound $\hat{k}(u) \leq \hat{k}(v) + \hat{k}(v')$ for a degree 2 node $u$ only if $\hat{k}(v) + \hat{k}(v') \leq \check{k}$, otherwise we compute a better bound based on the access nodes of $u$.

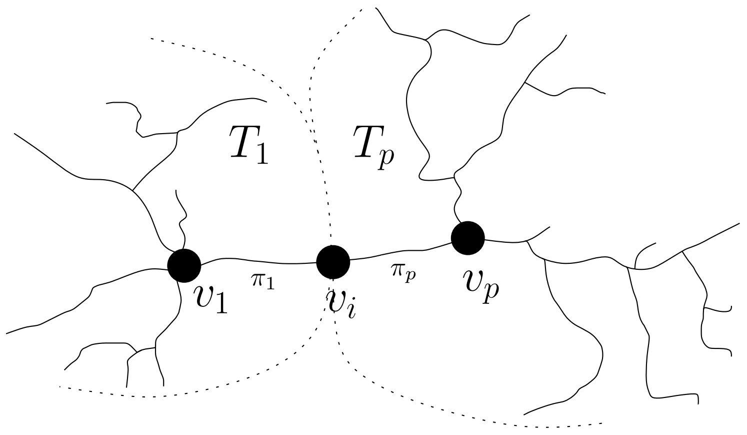

Figure 3.4: An example of the partitioning of a skeleton $T^*_{v_i}$ into trees $T_1$ an $T_p$.

## Computing Exact Values

When all cells have been processed, we have an upper bound $\hat{k}(u)$ on the width of every skeleton $T^*_u$ and it follows that $k \leq \max_{u \in V} \hat{k}(u)$. In order to compute the exact value of $k$, we iterate over all nodes $u$ sorted descending by $\hat{k}(u)$ and compute the actual skeleton $T^*_u$ and its width. During this process we keep updating the lower bound $\check{k}$, which is the maximum width of all skeletons computed so far. If at some point for the upper bound $\hat{k}(u)$ of the currently considered node $u$ we have $\hat{k}(u) \leq \check{k}$, it follows that $k = \check{k}$. Provided that the bounds $\hat{k}(u)$ are not too bad, this last step involves considerably less complete one-to-all shortest path computations than the naive approach.

# 4 Algorithmic Generation of Road Networks

To enable sound scalability studies, we need access to road networks of different sizes. Usually, cutouts of real road networks are used for this purpose. But we argue that this is not sufficient here. First of all, the planet road network as available in OSM[1] contains only about 600 million nodes, and these nodes are distributed over many unconnected or only sparsely connected components. Secondly, due to the silhouette of real road networks, one runs quickly into border affects – which leads to an artificial reduction of search space sizes of route planning techniques and possibly also affects the growth of network parameters. To avoid such distortions and to be able to consider even larger networks than those available at the moment, we want to generate road networks of compact shapes and arbitrary sizes.

## 4.1 Synthetic Road Network Generators

Simple models for generating road networks are e.g. grid graphs or unit disk graphs. But they lack the hierarchy of slow and fast roads usually present in large road networks. Eppstein and Goodrich [EG08] characterized road networks as multi-scale dispersed graphs and modeled them as subgraphs of disk intersection graphs. Eisenstat [Eis11] proposed nested quad trees to model road networks but with the primary goal of analyzing maximum flows. The graph generator presented in [AFGW10] is custom-tailored to produce networks with constant highway dimension. There, the road network is constructed in an online fashion, where a new node is always connected to the so far existing network according to a specific protocol. To model the hierarchy, a speedup parameter is defined and used to make travel times on longer edges proportionally quicker.

Bauer et al. [BKMW10] implemented most of these generators and tested them against a newly designed generator based on recursive Voronoi cell computations. Properties as node degree distribution, distance distributions and speedups for selected route planning techniques were measured with the help of a (small) scalability study (up to half a million nodes). It was experimentally shown that the sophisticated generators perform well for most considered aspects. But the highway dimension related generator produced too dense networks, and the scalability studies revealed too large speed-ups for CH when using the Voronoi-based generator. Furthermore, most of these generators require to fix a large number of parameters manually and are quite resource demanding.

[1]`https://www.openstreetmap.org`

## 4.2 Road Networks as Jigsaw Puzzles

Blum [Blu17] proposed a different approach, whose main idea we sketch in the following. The suggested generator combines tiles cut out from real road networks, providing the possibility to construct road networks larger than that of our planet – with a compact shape, and with similar properties as the original networks.

As input it requires a road network where for every node its coordinates and for every edge the corresponding road type (e.g. motorway or living street) and traversal speed (or distance) are given. All of this information is available for road networks extracted from OSM. A distinction is made between critical and non-critical road types, where critical roads such as motorways are the most important and fastest ones in the network.

From the input network square shaped tiles are cut out at random positions and combined on a grid that is filled bottom-up from left to right. In every tile so called *portals* are inserted at all positions where an edge was cut by the tile boundary, these portals are the places where the whole network will be connected by inserting edges between neighboring tiles on the grid. To smooth the transitions between tiles, they are not placed seamlessly next to each other, but with a small gap (see Figure 4.1).

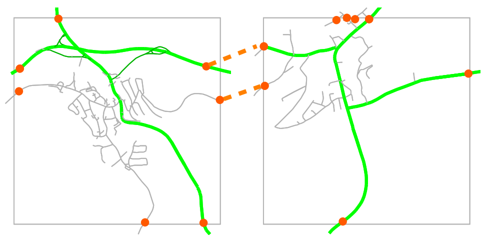

Figure 4.1: Two neighboring tiles that have been connected via some of their portals. Green roads are critical.

To create sensible networks, only similar tiles are placed next to each other. For that, the generator stores a supply of already cut out tiles. When a new tile is to be placed into the grid, the generator considers the portals along the neighboring tiles that have already been placed and selects a tile according to the following criteria. Connecting portals that arose from cutting a critical road to portals of the same road type has highest priority. The remaining portals are connected whenever possible according to their relative positions (avoiding crossings). To quantify the similarity of two tiles, the following measure is used: Connecting two portals of the same type comes with no cost, connecting two portals of different types and turning a portal into a dead-end has a cost of 1. The total cost of connecting two tiles can be computed by means of dynamic programming as in the common algorithm for the computation of the Levenshtein distance of two strings [Vin68]. So when a new tile is to be placed into the grid, the tile exhibiting the least total cost to its neighbors is chosen from the supply. If necessary, ties are broken by the total offset along the boundary (in terms of Euclidean distance) that all portals to be connected exhibit. Finally, reasonable road types and traversal speed of the inserted inter-tile edges are derived from the portals that they connect.

# 5 Experiments

We implemented the route planning techniques CH, TN and HL in C++ as well as the proposed algorithms for computing bounds on the highway dimension and the skeleton dimension. Experiments were conducted on an AMD Opteron 6272 CPU (32 cores clocked at 2.1 GHz) with 264 GB main memory, running Ubuntu 16.04.2 (kernel 4.4.0). We used the GNU C++ compiler 5.4.0 with optimization level 3.

All experiments were based on the OSM road network of Germany (in the following just "Germany") consisting of about 23.9 million nodes and 24.6 million undirected edges. Shortest paths were computed wrt travel time.

## 5.1 Scaling Behavior of Route Planning Techniques

To study the performance of the mentioned route planning techniques on networks of different sizes, we used the generator from [Blu17] to construct a road network with roughly 800 million nodes and recursively cut out squares, reducing the number of nodes in every step by a factor of four until we stopped at about 2000 nodes. To show that the results are truthful, we also include results on cutouts of Germany. There, the largest square we could cut contained about 11 million nodes. For networks with up to $2 \cdot 10^5$ nodes, we report the average over 16 instances (for both Germany and the generated network), to avoid strong distortions by 'unlucky' selection. For networks with $7 \cdot 10^5$ nodes, the results are the averages over 4 instances.

On these networks we measured the sizes of the search spaces and the memory consumption of CH, TN and HL using the preprocessing schemes described in Section 2.3. The results are shown in Table 5.1. We observe that the outcomes on the generated instances are very close to the ones obtained on Germany, there is no overestimation on larger instances, and the respective ratios never exceed 2.2. Hence we deem the generated networks suitable as basis for our analysis.

To evaluate the scalability of the route planning techniques, we tried to fit the data to the model functions $f(x) = a \cdot \log_2(x)^b$ (polylogarithmic growth) and $g(x) = c \cdot x^d$ (polynomial growth). We also tried more complicated models with an additional constant term but obtained very similar results. For this purpose we used the gnuplot implementation of the nonlinear least-squares Marquardt-Levenberg algorithm.[1] Given a parametrized model function $f_\beta$, the objective of this algorithm is to minimize the sum of squared residuals $\sum_i r_i(\beta)^2$, where $r_i(\beta)$ is the residual of the $i$-th data point. Starting from an initial parameter $\beta$, it iteratively improves the fitting by replacing $\beta$ with a new estimate $\beta + \delta$ until the improvement falls below a certain threshold.

[1] `http://gnuplot.sourceforge.net/docs_4.2/node82.html`

| | | | Germany | | | | | | Generated | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | CH | | | TN | | HL | CH | | | TN | | HL |
| $n$ | $\log_2 n$ | $\sqrt{n}$ | $\frac{\lvert E^+\rvert}{\lvert E\rvert}$ | $\lvert V^\uparrow\rvert$ | $\lvert E^\uparrow\rvert$ | $\lvert AN\rvert$ | $r$ | $\lvert HL\rvert$ | $\frac{\lvert E^+\rvert}{\lvert E\rvert}$ | $\lvert V^\uparrow\rvert$ | $\lvert E^\uparrow\rvert$ | $\lvert AN\rvert$ | $r$ | $\lvert HL\rvert$ |
| $2 \times 10^3$ | 11.0 | 45 | 0.76 | 14.5 | 26.6 | 1.8 | 18 | 12.0 | 0.73 | 15.2 | 28.3 | 1.7 | 20 | 12.6 |
| $1 \times 10^4$ | 13.3 | 100 | 0.80 | 23.4 | 57.5 | 2.2 | 37 | 17.8 | 0.78 | 23.7 | 58.4 | 2.0 | 41 | 17.4 |
| $4 \times 10^4$ | 15.3 | 200 | 0.81 | 42.8 | 153.3 | 2.8 | 75 | 26.7 | 0.79 | 41.3 | 146.7 | 2.6 | 83 | 26.1 |
| $2 \times 10^5$ | 17.6 | 447 | 0.82 | 82.8 | 481.7 | 3.8 | 141 | 40.0 | 0.79 | 74.2 | 381.0 | 3.6 | 160 | 37.5 |
| $7 \times 10^5$ | 19.4 | 837 | 0.83 | 176.3 | 1529.2 | 5.2 | 268 | 58.2 | 0.79 | 149.2 | 1103.2 | 5.1 | 319 | 54.6 |
| $2 \times 10^6$ | 21.0 | 1414 | 0.83 | 399.3 | 5069.6 | 7.7 | 518 | 88.9 | 0.80 | 301.0 | 3027.4 | 7.8 | 567 | 74.8 |
| $1 \times 10^7$ | 23.3 | 3162 | 0.82 | 929.8 | 15575.4 | 17.5 | 1049 | 125.2 | 0.80 | 558.6 | 7321.5 | 11.5 | 1071 | 102.9 |
| $5 \times 10^7$ | 25.6 | 7071 | - | - | - | - | - | - | 0.79 | 1196.1 | 22267.1 | 21.1 | 2066 | 150.5 |
| $2 \times 10^8$ | 27.6 | 14142 | - | - | - | - | - | - | 0.79 | 2546.6 | 63223.4 | 44.9 | 4057 | 213.0 |
| $8 \times 10^8$ | 29.6 | 28284 | - | - | - | - | - | - | 0.79 | 5187.9 | 160193.4 | 88.0 | 7459 | 295.0 |

Table 5.1: Average search space sizes (over 1,000 random queries) and space consumption for different route planning techniques: ratio $|E^+|/|E|$, number of nodes in upward graph ($|V^\uparrow|$), number of edges in upward graph ($|E^\uparrow|$), number of access nodes ($|AN|$), radius to furthest access node in seconds ($r$), number of hub labels ($|HL|$)

In the following, we sketch the most important outcomes.

### Contraction Hierarchies

For the space consumption of CH, indicated by the ratio of shortcuts $|E^+|$ and original edges $|E|$ in Table 5.1, we get the clear result that $|E^+| \approx 0.8 \cdot |E|$, so the space consumption is linear in $|E|$ invariant of the size of the network. Consequently, the contraction of every node introduces only a constant number of shortcuts on average, although nodes of high rank usually also have high degree. This is really surprising, given that analysis of CH even on special graph classes (such as planar graphs) never revealed a bound of $o(n \log n)$ [Mil12].

For the size of the search space (column $|V^\uparrow|$) depending on $n$, the best fit for $f(n)$ was $a = 2.4 \cdot 10^{-11}$ (note that $a$ can also be interpreted as a change of the log base) and $b = 9.8$, with a sum of squared residuals being 17 695 and the asymptotic standard error (ASE)[2] for $a$ being over 58%. This renders a polylogarithmic growth unlikely. For $g(n)$ on the other hand, we got $c = 0.18$ and $d = 0.5$ with the sum of squared residuals being 2 421 and the ASE for $c$ about 7% and less than 1% for $d$. Furthermore, $g(n) \approx 0.18 \cdot \sqrt{n}$ was also roughly the result when we fitted only a subset of the available data, while in this case the exponent of the log changed significantly for $f(n)$. Hence we conclude that the search spaces of CH most likely grow linearly in $\sqrt{n}$.

The number of edges in the search space was assumed to be quadratic in $|V^\uparrow|$ in previous analyses [AFGW10], [BCRW13]. Our analysis of $|E^\uparrow|$ depending on $|V^\uparrow|$, however, shows the best fit to be $|E^\uparrow| = 0.5|V^\uparrow|^{1.5}$, hence the quadratic bound seems to be overly pessimistic.

### Transit Nodes

The number of access nodes ($|AN|$) fits the polynomial model $g(n) = 6.0 \cdot 10^{-3} \cdot n^{0.5}$ best with a sum of squared residuals of 41 and an ASE of 45% and 4.8% for $c$ and $d$, respectively. For the polylogarithmic model, the best fit was $a = 5.8 \cdot 10^{-12}$ and $b = 9.0$ with an ASE of 200% and 6.5%, respectively and a sum of squared residuals of 70. Again, the exponent $b$ changed drastically when only sampling over a subset of the data, so we conclude that $|AN|$ most likely grows linearly in $\sqrt{n}$, which would imply a search space size of $\mathcal{O}(n)$.

For the radius $r$ we get very similar results with a polynomial fit of $g(n) = 0.27 \cdot n^{0.5}$ (sum of squared residuals 289 495, ASE 20% for $c$ and 2.2% for $d$) and a polylogarithmic fit of $f(n) = 4.2 \cdot 10^{-9} \cdot \log_2(n)^8$ (sum of squared residuals 145 627, ASE 98% for $a$ and 3.5% for $b$). Actually, the growth of $r$ resembles the behavior of $|V^\uparrow|$ and is hence most likely linear in $\sqrt{n}$.

[2]The ASE is only a very rough estimate for the standard deviation of each parameter that is generally over-optimistic, but useful to compare the quality of different models.

### Hub Labels

For the number of hub labels ($|HL|$), the best fit in the log model is $f(n) = 4.9 \cdot 10^{-4} \log_2(n)^4$. For $g(n)$ we get $2.0 \cdot n^{0.25}$. The sum of squares of residuals are 759 and 388, respectively, the ASE values are 68% and 5.2% for $a, b$ and 15% and 3.5% for $c, d$. So again, the polynomial model seems to fit better.

### Comparison

For CH, we get an estimated space consumption of $\mathcal{O}(n)$, and search space sizes of $\mathcal{O}(\sqrt{n})$. For TN, we obtain that the number of access nodes $|AN|$ and the radius $r$ most likely both grow in $\mathcal{O}(\sqrt{n})$. For HL, we get a space consumption of $\mathcal{O}(n^{1.25})$ and search spaces of $\mathcal{O}(n^{0.25})$.

If our best polynomial fit reflects the growth behavior truthfully, then extrapolations indicate that for instances with $n > 1.3 \cdot 10^{10}$ the number of access nodes $|AN|$ would exceed the estimated number of hub labels $|HL|$. In that case HL would dominate TN wrt space consumption (wrt search spaces, HL is superior to TN already in the tested networks with $n = 1 \cdot 10^7$).

## 5.2 Empirical Growth of Network Parameters

Finally, we applied our algorithms for computing (bounds on) the highway and skeleton dimensions on the networks up to 11 million nodes. The results are shown in Table 5.2, for every order of magnitude the maximum value is reported.

| | | | Germany | | | | Generated | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | $\log_2 n$ | $\sqrt{n}$ | $h$ | $r$ | $k$ | $r$ | $h$ | $r$ | $k$ | $r$ |
| $2 \times 10^3$ | 11.0 | 45 | $\geq 93$ | 39 | 30 | 54 | $\geq 86$ | 15 | 24 | 101 |
| $1 \times 10^4$ | 13.3 | 100 | $\geq 159$ | 48 | 35 | 66 | $\geq 129$ | 25 | 36 | 78 |
| $4 \times 10^4$ | 15.3 | 200 | $\geq 304$ | 61 | 58 | 82 | $\geq 244$ | 41 | 64 | 100 |
| $2 \times 10^5$ | 17.6 | 447 | $\geq 468$ | 76 | 73 | 60 | $\geq 519$ | 131 | 79 | 161 |
| $7 \times 10^5$ | 19.4 | 837 | $\geq 656$ | 163 | 86 | 221 | $\geq 523$ | 266 | 83 | 161 |
| $2 \times 10^6$ | 21.0 | 1414 | $\geq 642$ | 458 | 86 | 221 | $\geq 328$ | 598 | 82 | 165 |
| $1 \times 10^7$ | 23.3 | 3162 | $\geq 775$ | 876 | 114 | 186 | $\geq 320$ | 1145 | 89 | 129 |

Table 5.2: Highway dimension $h$ and skeleton dimension $k$ of different sized networks and the corresponding radii in seconds

### Highway Dimension

To find a radius $r$ that provides good lower bounds on the highway dimension, we sampled 100 balls in the largest cutout of Germany with different radii and computed a lower bound on the corresponding hitting set. The results are shown in Figure 5.1. As we can see, the best bound was found for a radius of about 10 minutes which corresponds approximately to 10% of the average shortest path distance in the network.

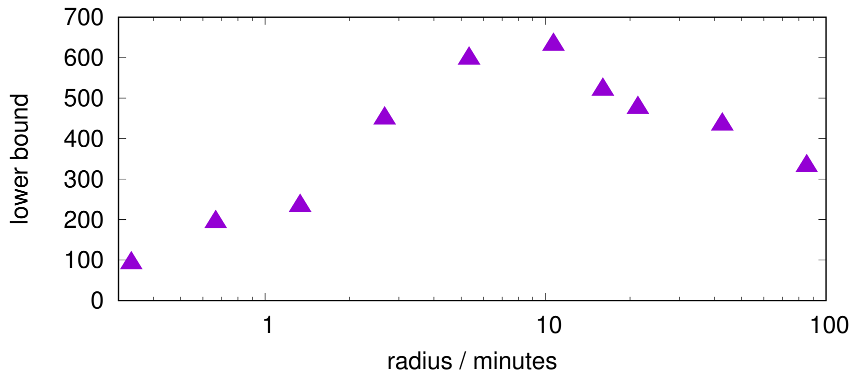

Figure 5.1: The best lower bounds found in a cutout of Germany with 11 million nodes depending of the chosen radius $r$. The x-axis uses a logarithmic scale.

Still, it seems safe to assume that due to the nature of our lower bound construction, the results are far from being tight. Moreover, the quality of our bounds presumably decreases the larger the networks become, as due to runtime and network size we were only able to sample a small fraction (a few thousand) of the nodes. This also explains, why the bounds are not strictly increasing. Still, especially the results obtained for Germany suggest that there is a correlation between the size of the network and $h$.

When fitting these lower bounds to the polylogarithmic model, we obtained $f(n) = 0.52 \log_2(n)^{2.3}$ with a sum of squared residuals of 28 500 and an ASE of 121% and 17% for the factor $a$ and the exponent $b$, respectively. For the polynomial model, we obtained $g(n) = 46 \cdot n^{0.18}$ with a sum of squared residuals of 46 332 and an ASE of 55% and 21% for the factor $c$ and the exponent $d$, respectively. Hence, better bounds for larger networks are required for clearer results, which are however not trivial to obtain.

Taking into account that the search space size of CH is supposed to be in $\mathcal{O}(h \log D)$ according to the $h$-dependent analysis and that our empirical CH analysis revealed a growth of $\mathcal{O}(\sqrt{n})$, we conclude that $h$ should roughly grow linearly in $\sqrt{n}$ as well. But this would result in a space consumption of $\mathcal{O}(n\sqrt{n} \log n)$ which appears to be very loose compared to the linear growth indicated by our empirical results. Furthermore, our results indicate that at least CH and HL scale differently, while the $h$-dependent analysis predicts the same behavior for both (with matching lower bounds as proven in [Whi15]). Of course, it might be that the more complicated CH and HL preprocessing routines analyzed for $h$ lead to a different scaling behavior. But this means that either the $h$-dependent analysis has little relevance for the variants of the route planning techniques used in practice or the $h$ value scales worse than conjectured (and actually close to the proven growth of $\sqrt{n}$ on grid instances).

### Skeleton Dimension

The instances with 11 million nodes were the two largest networks for which we managed to compute an exact value for the skeleton dimension. Both computations required less than 39 hours runtime (using 14 cores on average) and 75 GB RAM. We used $20\sqrt{n}$ transit nodes and dedicated 50 GB memory to simultaneously store the results of 1 200 PHAST runs. For the cutout of Germany and the generated network, in the last step 182 275 and 83 963 exact computations were required to close the gap between upper and lower bounds, respectively. In total 17% and 24% of the degree 2 nodes could be pruned, respectively. On these instances we estimated that the naive algorithm based on Dijkstra computations would have taken more than 500 days on one core, and the PHAST based algorithm without our improvements and pruning strategies about 120 days.

We observe that the radius defining the skeleton dimension is small in all considered networks (only up to about 3 minutes). Figure 5.2 shows the maximum and average size of $Cut_r(T^*)$ (the number of edges cut at radius $r$) of 1 000 randomly sampled skeletons in the largest cutout of Germany. We can see that the radii at which the largest cuts occur are significantly smaller than the radius of the whole network. This might hint that the value of the parameter is indeed not dependent on the total size of the network but rather on the densest cluster therein (e.g. the largest city). In fact, for the networks $7 \times 10^5$ and $2 \times 10^6$ the skeleton of the very same root defined $k$.

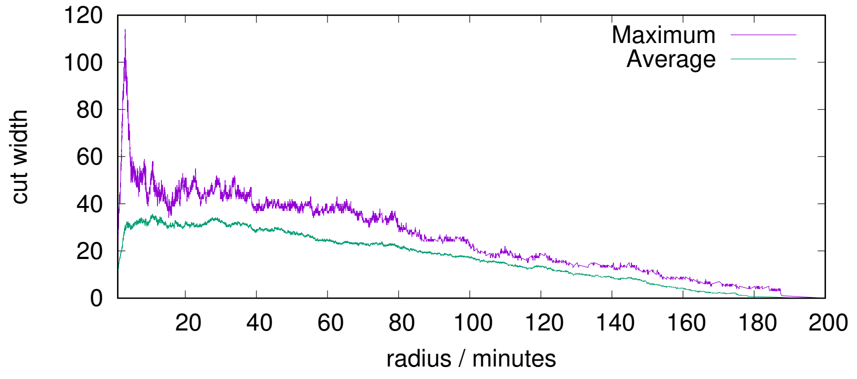

Figure 5.2: The maximum and average size of $Cut_r(T^*)$ in dependency of the radius $r$ in the largest cutout of Germany

The maximum skeleton dimension in all networks is 114, which is the width of a skeleton in the largest cutout of Germany with 11 million nodes (see Figure 5.3). This width is assumed at a radius of 186 seconds, which means that the relevant part of the skeleton is contained in a small cutout consisting only of 16 126 nodes.

Taking into consideration that the search spaces of HL was shown to be in $\mathcal{O}(k \log n)$ and that our experiments provide some evidence for a growth linear in $\sqrt{n}$, this indicates that the skeleton-based preprocessing technique for HL might behave differently than the chosen CH-based variant, which could be worth investigating further.

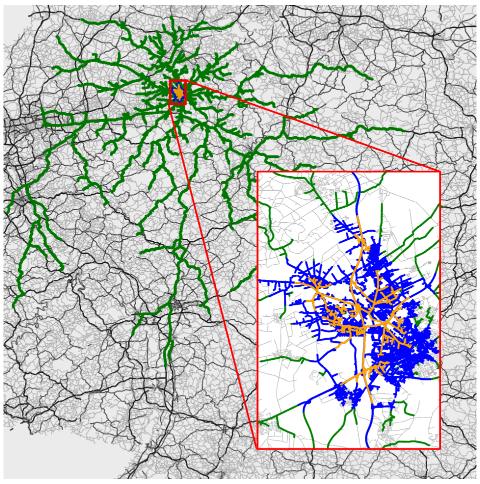

Figure 5.3: A skeleton of width 114 (green), the 114 shortest paths cut at a radius of 186 seconds (yellow) and the relevant ball of radius 279 seconds (blue).

We also computed 1 000 shortest path tree skeletons in a network with about 180 000 nodes and compared the sizes of the supergraphs obtained by the TN approach (using $5\sqrt{n}$ transit nodes) and the reach approach. The former created supergraphs that were on average 5% larger than the actual skeleton, while the graphs resulting from the latter were too large by 171%. The average width of the trees were 23.2, 26.0 and 61.8, respectively. An example can be seen in Figure 5.4. This shows that the TN approach provides bounds that are a lot tighter than the reach approach.

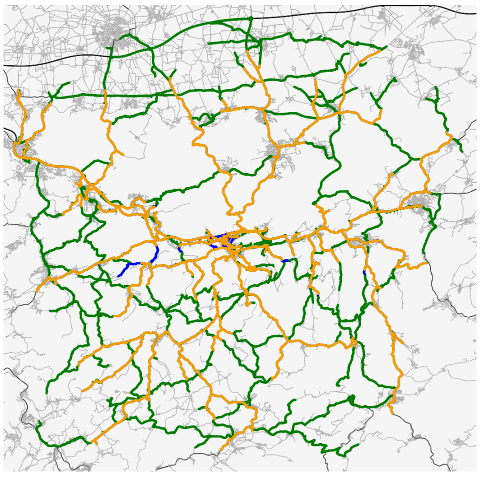

Figure 5.4: A shortest path tree skeleton (yellow, 12 442 nodes, width 29) and two supergraphs based on TN (blue, 12 845 nodes, width 29) and reach (green, 32 932 nodes, width 85).

# 6 Conclusions and Future Work

We have presented the first thorough scalability study of three state-of-the-art route planning techniques and two important (road) network parameters, based on different sized networks with up to more than 750 million nodes. Somehow surprisingly, the obtained empirical results do not comply with existing theoretical bounds and conjectures in many cases. There is strong evidence that, on real-world road networks, CH exhibits a linear space consumption and a search space size of $\mathcal{O}(\sqrt{n})$. For TN we conjecture linear search spaces of $\mathcal{O}(n)$. For HL, search spaces of $\mathcal{O}(n^{0.25})$ and a space consumption of $\mathcal{O}(n^{0.25})$ are most likely. Moreover it seems, that the skeleton dimension $k$ does not depend on the network size. This gives rise to several new research questions and additional analysis tasks.

- As the preprocessing techniques used for the highway dimension based analysis are too demanding to implement, it might be a good alternative to try to come up with efficiently computable (instance based) lower bounds for feasible CH, TN or HL data structures and to study their growth behavior.
- Better and more efficient lower bounds for the highway dimension itself would be helpful to determine its growth in dependency of $n$ with higher confidence. Also, bootstrapping the used generator with other networks (e.g. the US network with more grid structures) is necessary to observe further structural dependencies.
- Our results indicate that the skeleton dimension $k$ grows much slower than the highway dimension (and is possibly even independent of $n$). Only for HL bounds depending on $k$ are known. Therefore it seems worthwile to investigate such bounds also for CH and TN.
- There are several engineering techniques available for CH, TN and HL. For example, stall-on-demand allows to settle fewer nodes in a CH query [GSSV12], compression techniques reduce hub label sizes [DGW13], etc. It would be interesting to investigate whether these techniques influence the scaling behavior.

The most important open question of course is whether our empirical results can be explained by some alternative road network model or suitable road network parameter, or even by a more involved analysis of existing ones. The functions which fitted our data best look natural in many cases, hence it would be interesting to see whether there are sound interpretations and explanations for these results.

# Bibliography

[ACP87] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski: Complexity of finding embeddings in a $k$-tree. *SIAM J. Algebraic Discrete Methods*, 8(2):277–284, 1987.

[ADF$^+$11] Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck: VC-dimension and shortest path algorithms. In *Proceedings of the 38th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 6755 of *Lecture Notes in Computer Science*, pages 690–699. Springer, 2011.

[ADF$^+$13] Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck: Highway dimension and provably efficient shortest path algorithms. Technical Report MSR-TR-2013-91, Microsoft Research, 2013.

[ADGW11] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck: A hub-based labeling algorithm for shortest paths in road networks. In *Proceedings of the 10th International Conference on Experimental Algorithms (SEA)*, volume 6630 of *Lecture Notes in Computer Science*, pages 230–241. Springer, 2011.

[AFGW10] Ittai Abraham, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck: Highway dimension, shortest paths, and provably efficient algorithms. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 782–793. SIAM, 2010.

[ALS13] Julian Arz, Dennis Luxen, and Peter Sanders: Transit node routing reconsidered. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA)*, volume 7933 of *Lecture Notes in Computer Science*, pages 55–66. Springer, 2013.

[BCRW13] Reinhard Bauer, Tobias Columbus, Ignaz Rutter, and Dorothea Wagner: Search-space size in contraction hierarchies. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 7965 of *Lecture Notes in Computer Science*, pages 93–104. Springer, 2013.

[BDG$^+$16] Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck: Route planning in transportation networks. In *Algorithm Engineering*, pages 19–80. Springer, 2016.

[BFSS07] Holger Bast, Stefan Funke, Peter Sanders, and Dominik Schultes: Fast routing in road networks with transit nodes. *Science*, 316(5824):566–566, 2007.

[BKMW10] Reinhard Bauer, Marcus Krug, Sascha Meinert, and Dorothea Wagner: Synthetic road networks. In *Proceedings of the 6th International Conference on Algorithmic Aspects in Information and Management (AAIM)*, volume 6124 of *Lecture Notes in Computer Science*, pages 46–57. Springer, 2010.

[Blu17] Johannes Blum: Generation of large road networks based on real-world data. Praktikumsbericht, University of Würzburg, 2017.

[DGNW11] Daniel Delling, Andrew V. Goldberg, Andreas Nowatzyk, and Renato F. Werneck: PHAST: Hardware-accelerated shortest path trees. In *Proceedings of the 25th IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 921–931. IEEE, 2011.

[DGW13] Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck: Hub label compression. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA)*, volume 7933 of *Lecture Notes in Computer Science*, pages 18–29. Springer, 2013.

[Dij59] Edsger W. Dijkstra: A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[DSW14] Julian Dibbelt, Ben Strasser, and Dorothea Wagner: Customizable contraction hierarchies. In *Proceedings of the 13th International Symposium on Experimental Algorithms (SEA)*, volume 8504 of *Lecture Notes in Computer Science*, pages 271–282. Springer, 2014.

[EF12] Jochen Eisner and Stefan Funke: Transit nodes–lower bounds and refined construction. In *Proceedings of the 14th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 141–149. SIAM, 2012.

[EG08] David Eppstein and Michael T. Goodrich: Studying (non-planar) road networks through an algorithmic lens. In *Proceedings of the 16th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems (ACM-GIS)*, page 16. ACM, 2008.

[Eis11] David Eisenstat: Random road networks: The quadtree model. In *Proceedings of the 8th Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, pages 76–84. SIAM, 2011.

[FFKP15] Andreas Emil Feldmann, Wai Shing Fung, Jochen Könemann, and Ian Post: A $(1+\varepsilon)$-embedding of low highway dimension graphs into bounded treewidth graphs. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 9134 of *Lecture Notes in Computer Science*, pages 469–480. Springer, 2015.

[GKW06] Andrew V. Goldberg, Haim Kaplan, and Renato F. Werneck: Reach for A*: Efficient point-to-point shortest path algorithms. In *Proceedings of the 8th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 129–143. SIAM, 2006.

[GSSV12] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter: Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, 2012.

[Joh74] David S. Johnson: Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3):256–278, 1974.

[KV17] Adrian Kosowski and Laurent Viennot: Beyond highway dimension: Small distance labels using tree skeletons. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1462–1478. SIAM, 2017.

[Lov75] László Lovász: On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13(4):383–390, 1975.

[Mil12] Nikola Milosavljević: On optimal preprocessing for contraction hierarchies. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, pages 33–38. ACM, 2012.

[Vin68] Taras K. Vintsyuk: Speech discrimination by dynamic programming. *Cybernetics*, 4(1):52–57, 1968.

[Whi15] Colin White: Lower bounds in the preprocessing and query phases of routing algorithms. In *Proceedings of the 23rd European Symposium on Algorithms (ESA)*, volume 9294 of *Lecture Notes in Computer Science*, pages 1013–1024. Springer, 2015.

# Erklärung

Hiermit versichere ich die vorliegende Abschlussarbeit selbstständig verfasst zu haben, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben, und die Arbeit bisher oder gleichzeitig keiner anderen Prüfungsbehörde unter Erlangung eines akademischen Grades vorgelegt zu haben.

Würzburg, den 6. September 2017

. . . . . . . . . . . . . . . . . . . . . . . . . . . .
Johannes Blum