Synthetic Road Networks*

Reinhard Bauer, Marcus Krug, Sascha Meinert, and Dorothea Wagner

Karlsruhe Institute of Technology (KIT), Germany
{Reinhard.Bauer,Marcus.Krug,Sascha.Meinert,Dorothea.Wagner}@kit.edu

Abstract. The availability of large graphs that represent huge road networks has led to a vast amount of experimental research that has been custom-tailored for road networks. There are two primary reasons to investigate graph-generators that construct synthetic graphs similar to real-world road-networks: The wish to theoretically explain noticeable experimental results on these networks and to overcome the commercial nature of most datasets that limits scientific use. This is the first work that experimentally evaluates the practical applicability of such generators. To this end we propose a new generator and review the only existing one (which until now has not been tested experimentally). Both generators are examined concerning structural properties and algorithmic behavior. Although both generators prove to be reasonably good models, our new generator outperforms the existing one with respect to both structural properties and algorithmic behavior.

1 Introduction

During the last two decades, advances in information processing led to the availability of large graphs, that accurately represent the road networks of whole continents in full detail. Today, these networks are omnipresent in applications like route planning software, geographical information systems or logistics planning. While there is a vast amount of research on algorithms that work on (and often are custom-tailored for) road networks, the natural structure of these networks is still not fully understood.

Aims. In this work we aim to synthetically generate graphs that replicate real-world road networks. The motivation of doing so is manifold: Firstly, the existing data is often commercial and availability for research is only restricted. In those situations, graph generators are a good and established way to obtain test-data for research purposes. Additionally, it seems likely that datasets that represent the road-network of the entire world will be available in a few years. It will be shown later that the size of the road-network has a crucial (non trivial, non-linear) influence on the performance of algorithms on it. Using graph generators that are able to generate graphs of appropriate size and structure one can then do algorithmic research on such networks.

Secondly, we want to improve the understanding of the structure within road-networks. This may support a theoretical analysis of algorithms that have been

* Partially supported by the DFG (project WAG54/16-1).

custom-tailored for road-networks. A well known example is the development of route-planning techniques during the last decade that yield impressive runtimes by exploiting a special 'hierarchy' in road networks [5]. Many of these algorithms have recently been analyzed in [2]. There, the intuition of hierarchy has been formalized using the notion of 'highway dimension' and a generator for road networks. In [2] no evidence is given that this generator is a good model for road networks. We check that in this work.

Further, there is not only one 'road network'. Hence, we want to compare graphs originating from different sources with each other. This helps practitioners to assess the value of a given experimental work for their specific data and problem. Finally, as generators usually involve some tuning parameters, experimentalists can use them to generate interesting instances and steer the properties these instances have. This can help to understand custom-tailored algorithms better.

Related Work. There is a huge amount of work on point-to-point route-planning techniques. These are mostly custom-tailored for road networks. An overview can be found in [5]. In [2] a graph generator for road-networks is proposed.

The work [7] studies properties of real-world road-networks. Firstly, road networks are characterized as a special class of geometric graphs. This characterization is evaluated on the Tiger/Line road networks. It could be a starting point for a possible graph generator, but too many degrees of freedom are left open to use it directly. Furthermore, road networks are analyzed concerning planarity: The typical number of crossings (of the embedding given by the GPS-coordinates) of an n-vertex road-network is reported to be $\Theta(\sqrt{n})$.

Contribution. This is the first work that experimentally generates synthetic road networks and tests their practical applicability. In Chapter 2 we survey existing networks that are (partly) available to the scientific community. Chapter 3 introduces a new generator for road networks and describes the generator given in [2]. In order to assess the quality of these custom-tailored generators, we also describe two standard graph generators, namely those for Unit-Disk and Grid-graphs.

Chapter 4 evaluates the generators and compares their output with real-world networks. There, structural properties and algorithmic behavior are taken into account. As structural properties we consider connectedness, directedness, degree distribution, density and distance distribution. We found out that the custom-tailored generators are a good model for the real-world data. The generator of Abraham et al. [2] performs also well, with the exception that it produces graphs that are too dense and incorporate nodes of too high degree. For testing the algorithmic behavior we focus on point-to-point shortest path computation as this area uses techniques that have been highly custom-tailored for road-networks. One unexpected outcome is that the real-world graphs significantly differ in their algorithmic behavior. Further, the standard-generators approximate road-networks only to a limited extend (which is not surprising). Finally, the custom-tailored generators seem to be reasonably good approximations for the real-world instances.

2 Real-World Road-Networks

Graphs that represent road networks are typically constructed from digital maps. Digital mapping is expensive and mostly done by companies like Google, Teleatlas and NAVTEQ. Hence, this data is hard or expensive to obtain. We are aware of only three sources for real-world road-networks that are (almost) free for scientific use.

The dataset PTV is commercial and property of the company PTV-AG (`http://www.ptv.de/`). It is based on data of the Company NavTeq and not fully free, but has been provided to participants of the 9th Dimacs Implementation Challenge [6]. We use slightly updated data from the year 2006. The U.S. Census Bureau publishes the Tiger/Line datasets TIGER (`http://www.census.gov/geo/www/tiger/`). We use the version available at the 9th Dimacs Implementation Challenge (`http://www.dis.uniroma1.it/~challenge9/`). OpenStreetMap (`http://www.openstreetmap.org/`) is a collaborative project that uses GPS-data gathered by volunteers to assemble a digital map of the whole world. We use Europe-data of December 2009 and remove all items that do not correspond to roads (more details can be found in the appendix). See Table 1, page 52 for basic information on these three datasets, nomination is as follows: *Density* denotes the number of edges divided by the number of nodes, *directedness* denotes the number of edges (u, v) for which there either is no edge (v, u) or for which (v, u) has a different length than (u, v) divided by the total number of edges. In the experimental evaluation we always use Euclidean distances as edge weights. This results in better comparability and is well justified as changing to other metrics like travel time has only low impact on the considered algorithms [3]. Edges are always counted as directed, i.e. edges (u, v) and (v, u) both contribute to the overall number of edges.

3 Graph Generators

Voronoi-Based Generator. Our generator is based on the following assumptions about road networks which are well motivated from real-world road networks: (1) Road networks are typically built so as to interlink a given set of resources, such as, for instance, natural resources or industrial agglomerations of some sort. (2) Road networks exhibit a hierarchical, nested structure: The top level of this hierarchy is defined by the highways, which form a grid-like pattern. Each cells of this grid-like network is subdivided by a network of smaller roads, which again exhibits a grid-like structure. (3) Shortest paths in road networks do not typically exhibit a large dilation with respect to the Euclidean distance: Although dilation may be very large in the presence of long and thin obstacles, such as rivers and lakes, it is rather low for the larger part of the network. We further assume that in the presence of two or more sites of resources it is best to build roads along the bisectors of these sites, since any two sites incident to a road can access this road at the same cost.

Our generator is a generic framework which works in two phases: At first, we generate a random graph based on recursively defined Voronoi diagrams in the Euclidean plane. In the second phase we then compute a sparser subgraph since the graph computed in the first phase is rather dense as compared to real-world networks.

A *Voronoi diagram of a set of points* P is a subdivision of the plane into convex Voronoi regions $\text{vreg}(p)$ for all $p \in P$. The Voronoi region $\text{vreg}(p)$ contains all points whose distance to p is smaller than their distance to any other point $q \in P \setminus \{p\}$. The union of all points which are equally far from at least two points form a plane graph, which we call the *Voronoi graph* $\mathcal{G}(P)$. The vertices of this graph are exactly the set of points which are equally far from at least three points in P. Each face f of this graph corresponds to the Voronoi region of some point p. By $P(f)$ we denote the simple polygon which forms the boundary of f. The Voronoi diagram for a set of n points can be computed in $\mathcal{O}(n \log n)$ points by a simple sweep line algorithm [9]. In our implementation, we used the CGAL library in order to compute the Voronoi diagram of a set of points [1].

Algorithm 1. Subdivide-Polygon

```
Input: $P, n, \mathcal{D}, \mathcal{R}$
1  $C \leftarrow \emptyset$;
2  for $i = 1$ to $n$ do
3      $x \leftarrow$ choose uniform point inside $P$;
4      $\alpha \leftarrow$ random value chosen according to $\mathcal{D}$;
5      $r \leftarrow$ random value chosen according to $\mathcal{R}$;
6      $m \leftarrow \lceil r^\alpha \rceil$;
7      $C \leftarrow C \cup \{x\}$;
8      for $j = 1$ to $m$ do
9          $p \leftarrow$ choose random point in $R(x, r)$;
10         $C \leftarrow C \cup \{p\}$;
11     compute Voronoi diagram of $C$ in $P$;
```

The first phase of our generator is a recursive procedure whose core is a routine called Subdivide-Polygon$(P, n, \mathcal{D}, \mathcal{R})$. The pseudo-code for this routine is listed in Algorithm 1. Invoked on a simple polygon P this routine computes a Voronoi diagram inside P from a set of points which are chosen as follows: First, we choose a set of n uniformly distributed points in P, which we call center sites. For each of the center sites x we choose a density parameter α according to the distribution $\mathcal{D}$ as well as a radius r according to the distribution $\mathcal{R}$. As distributions we used the uniform distribution with density function $\text{Unif}_{[a,b]}(x) = \frac{1}{b-a}$ for all $x \in [a, b]$ with $a < b \in \mathbb{R}$ as well as the exponential distribution with density function $\text{Exp}_\lambda(x) = \lambda e^{-\lambda x}$ for all $x \in \mathbb{R}_0^+$. Then we choose $\lceil r^\alpha \rceil$ points in the disc centered at x with radius r by choosing radial coordinates uniformly at random. Thus, we create a set of points as agglomerations around uniformly distributed centers.

In our implementation we choose points inside the bounding box of P uniformly at random by rejecting points not in P until we have found the desired number of points.

The pseudo-code of the first phase is listed in Algorithm 2. The input consists of an initial polygon P, a number ℓ of levels for the recursion and for each recursion level $1 \leq i \leq \ell$ a fraction γ_i of cells to be subdivided along with distributions $\mathcal{C}_i$, $\mathcal{R}_i$ and $\mathcal{D}_i$. We then proceed as follows: First, we choose a set of Voronoi regions to subdivide among the regions which were produced in the previous iteration of the algorithm. Let S be the set of Voronoi regions which were produced in the previous iteration, then we subdivide the $\gamma_i|S|$ smallest regions in the current iteration of the algorithm. Hence, the distribution of points will be concentrated in areas with many points. Therefore, we simulate the fact that sites with many resources will attract even more resources.

Algorithm 2. Voronoi-Roadnetwork

```
Input: $\ell$, $\gamma_i, \mathcal{C}_i, \mathcal{R}_i, \mathcal{D}_i$, $1 \leq i \leq \ell$
1  $\ell_0 \leftarrow 1$;
2  $S \leftarrow P$;
3  for $i = 1$ to $\ell$ do
4      $m \leftarrow \gamma_{i-1}|S|$;
5      $S \leftarrow$ smallest $m$ faces in $S$;
6      $S' \leftarrow \emptyset$;
7      for $f \in S$ do
8          $n \leftarrow$ choose according to distribution $\mathcal{C}_i$;
9          $S' \leftarrow S' \cup$ Subdivide-Polygon $(P(f), n, \mathcal{D}_i, \mathcal{R}_i)$;
10     $S \leftarrow S'$
```

For each Voronoi region R which has been chosen to be subdivided, we first choose an associated number of centers n according to the distribution $\mathcal{C}_i$. Then we call Subdivide-Polygon on input R, n and the distributions corresponding to the current level in the recursion.

In the second phase we greedily compute a sparse graph spanner of the graph computed in phase one. Given a graph G a *graph spanner* H of G with stretch t is a subgraph of G such that for each pair of vertices u, v in G we have $\text{dist}_H(u, v) \leq t \cdot \text{dist}_G(u, v)$. We would like H to contain as few edges as possible. Determining if a graph G contains a t spanner with at most m edges is NP-hard [8].

In order to compute a sparse graph spanner greedily, we iterate over the edges sorted by non-increasing length and add only those edges to the graph whose absence would imply a large dilation in the graph constructed so far. Note that the order in which we consider the edges differs from the greedy algorithm for graph spanners discussed, e.g., in [4]. Let H be the graph we have obtained after considering m edges. Then we insert the $(m + 1)$-st edge $\{u, v\}$ if and only if $\text{dist}_H(u, v)$ is larger than $t \cdot \text{len}(u, v)$. We assume $\text{len}(u, v)$ to be infinity if u and v are not in the same component. Hence, at each step H is a t-spanner for all pairs of vertices which are connected in H. At the end we will obtain a connected graph, and therefore, a t-spanner for G.

In order to determine $\text{dist}_H(u, v)$, we use Dijkstra's algorithm for computing shortest paths, which can be pruned whenever we have searched the complete graph-theoretic ball of radius $t \cdot \text{len}(u, v)$. Since we consider the edges in sorted order with non-increasing length, we will heuristically consider only few edges, as long edges are considered at the beginning, when the graph is still very sparse. Since the larger part of the edges in the graph is short compared to the diameter, the running time for this algorithm is not too large. In order to speed up computation for the cases in which u and v are not connected, we use a union-find data structure to keep track of the components of H.

The Generator of Abraham et al. This approach is due to [2] and is based on an arbitrary underlying metric space (M, d) with diameter D. In this work M will always be a rectangle in the Euclidean plane with $d(u, v)$ being the Euclidean distance between points u and v. Further, the generator requires a random generator rand() for points in M. No distribution is given in [2], we will report the distributions used in this work later in this section.

Algorithm 3. Generator of Abraham et al. [2]

```
input  : number of vertices $n$
output: Graph $(V, E)$

1  initialize $V = C_0, \ldots, C_{\log D}, E$ to be $\emptyset$ ;
2  for $t = 1$ to $n$ do
3      $v_t \leftarrow$ rand() ;
4      $V \leftarrow V \cup \{v_t\}$ ;
5      for $i = \log D$ to $1$ do
6          if $d(v_t) > 2^i$ for each $w \in C_i$ then
7              $C_i \leftarrow C_i \cup \{v_t\}$ ;
8              for $w \in C_i$ do
9                  if $d(v_t, w) \leq k \cdot 2^i$ then $E \leftarrow E \cup \{v_t, w\}$
10         $w \leftarrow$ closest point of $v_t$ in $C_{i+1}$ ;
11         if $v_t \neq w$ then $E \leftarrow E \cup \{v_t, w\}$
12 set edge weights such that $\text{len}(u, v) = d(u, v)^{1-\delta}$ for $\delta = 1/8$
```

The generator starts with the empty graph $(V, E) = (\emptyset, \emptyset)$ and iteratively adds new vertices to V. Their location in M is distributed according to rand(). A 2^i-cover is a set C_i of vertices such that for $u, v \in C_i$, $d(u, v) \geq 2^i$ and such that for each $u \in V$ there is a $v \in C_i$ with $d(u, v) \leq 2^i$. During the process of adding vertices, the generator maintains for each i with $1 \leq i \leq \log D$, a 2^i-*cover* C_i: After a vertex v_t has been added, the lowest index i is computed such that there is a $w \in C_i$ with $d(v_t, w) \leq 2^i$. Then v_t is added to all C_j with $0 \leq j < i$. If no such i exists, v_t is added to all sets C_j. Then, given a tuning-parameter k, for each $C_i \ni v_t$ and each $w \in C_i$ an edge (w, v_t) is added if $d(w, v_t) \leq k \cdot 2^i$. Further, for each $C_i \ni v_t$ with $i < \log D$ such that $v_t \notin C_{i+1}$, an edge from v_t to its nearest neighbor in C_{i+1} is added. Finally, edge lengths are fixed such that $\text{len}(u, v) = d(u, v)^{1-\delta}$ for $\delta = 1/8$. Pseudocode of the generator can be found in Algorithm 3. Throughout the rest of the paper, we fill the remaining tuning parameters as follows. We choose the number of levels to be 25 and therefore set $D := 2^{25}$. The aspect ratio of the rectangle representing M is 0.75. The parameter $k = \sqrt{2}$ (deviating from the original description where $k = 6$). We tried two point sources rand(). Firstly, we sampled points uniformly at random (which will not be used later on). Secondly, we used an improved point source that mimics city-like structures: We use a 2-phase approach. In the preparation phase we iteratively compute special points within M called *city centers*. This is done as follows. A new city center c is chosen uniformly at random within M. We assign a value r_c to c which is chosen from an exponential distribution with parameter $\lambda = 1/(0.05 \cdot s)$ where s is the length of the longer border of M.

We then assign a population p_c to c which is $r_c^{1.1}$. The preparation-phase stops, when the overall population exceeds the number of requested nodes n.

Afterwards, a new point x is generated on request as follows: Firstly, a center c with positive population p_c is chosen uniformly at random. We then set $p_c := p_c - 1$. The location of x is determined in polar-coordinates with center c by choosing an angle uniformly at random and by choosing the distance from a normal distribution with mean 0 and standard deviation r_c. Whenever we sampled points not lying in M these get rejected.

Unit Disk. Given a number of nodes n, a unit disk graph is generated by randomly assigning each of the n nodes to a point in the unit square of the Euclidean plain. There is an edge (u, v) in case the Euclidean distance between u and v is below a given radius r. We adjusted r such that the resulting graph has approximately $7n$ edges. We use the Euclidean distances as edge weights.

Grid. These graphs are based on two-dimensional square grids. The nodes of the graph correspond to the crossings in the grid. There is an edge between two nodes if these are neighbors on the grid. Edge weights are randomly chosen integer values between 1 and 1000.

4 Experimental Evaluation

In this section we experimentally assess the proposed generators. To that end we generated the datasets VOR (originating from the generator our Voronoi-based generator) and ABR (originating from the Abraham et al. generator). More information on the generation-process can be found in the appendix.

Graph Properties. We now have a look at some basic structural properties of the networks. We first observe that all real-world graphs are undirected or almost undirected and have an almost equal density of 2.05 to 2.4 (Table 1). By construction the synthetic graphs are undirected, the density of VOR is similar to the real-world graphs, the density of ABR is slightly too high. Further, all three real-world networks consist of one huge and many tiny strongly connected components. The OSM-data deviates in the size of the biggest strongly connected component. An explanation for this is the unfinished state of the underlying

Table 1. Overview of origin, size, density and directedness of the available real-world data and the generated datasets

dataset	origin	represents	#nodes	#edges	density	dir'ness
TIGER	U.S administration	USA	24,412,259	58,596,814	2.40	.0 %
PTV	commercial data	Europe	31,127,235	69,200,809	2.22	4.9 %
OSM	collaborative project	Europe	48,767,450	99,755,206	2.05	3.5 %
ABR	Abraham et al	synthetic	15,000,000	43,573,536	2.90	.0 %
VOR	Voronoi generator	synthetic	42,183,476	93,242,474	2.21	.0 %

map and we expect the deviation to decrease with increasing level-of detail in the OSM-data in the future (Table 2). By construction the synthetic graphs are strongly connected.

The distribution of the node-degrees is very similar for PTV and TIGER but again deviates significantly for OSM. All three graphs have in common that almost no nodes of degree of at least 5 exists. The difference in the number of degree-2 nodes can be explained by a higher level of detail of the OSM-data (Table 3). The distribution of VOR is quite similar to OSM while ABR significantly deviates from all other distributions: 7% of the nodes have degree 6 or higher, the maximum degree is 58.

Table 2. Relative sizes (measured in number of nodes) of the k biggest SCCs

	k					total #
dataset	1	2	5	20	100	of SCCs
OSM	.80	.91	.94	.96	.97	541264
PTV	.97	.97	.97	.97	.97	924561
TIGER	.98	.98	.99	.99	.99	89796

Table 3. Degree distribution of the datasets: Relative frequency according to degree

	degree						
dataset	0	1	2	3	4	5	≥ 6
OSM	.001	.097	.773	.114	.015	0	0
PTV	.030	.247	.245	.429	.049	.001	0
TIGER	0	.205	.301	.386	.106	.001	0
ABR	0	.324	.254	.160	.093	.056	.077
VOR	0	.100	.601	.292	.005	.002	0

Figure 1 shows the distribution of the distances between pairs of nodes in the networks. The distributions are hard to interpret but can be used as a fuzzy fingerprint for the structure within the graphs (for instance to separate them from different graph classes like small-world graphs). We observe that OSM and PTV have a quite similar distribution, differing from the TIGER-data. A possible explanation for that could be the geographical origin of the data. Both OSM and PTV map the European road-network while TIGER maps the U.S. road-network. The VOR-dataset is a good approximation for the TIGER-data, the ABR-dataset is a good approximation for OSM and PTV.

For the sake of completeness, we also report the distribution of the according edge weights (Figure 1). Note that the edge-weight distribution has only small impact on the considered algorithms [3]. For better comparability, we always applied Euclidean distances instead of the original weights.

Algorithmic Behavior. In this section we compare the algorithmic behavior of both real-world and synthetically generated graphs. For this purpose we analyze the speedups which can be achieved using speedup techniques for point-to-point shortest path queries, such as Bidirectional Dijkstra's algorithm, ALT (16 landmarks computed by max-cover strategy), Arc-Flags (128 Voronoi-cells), Reach-Based Routing ($\epsilon = 0.5$, $\delta = 16$) as well as Contraction Hierarchies (CH, original code), as compared to standard Dijkstra's algorithm. See [5] for more information. These techniques have specifically been designed for the task of heuristically speeding up Dijkstra's algorithm on real-world road networks and,

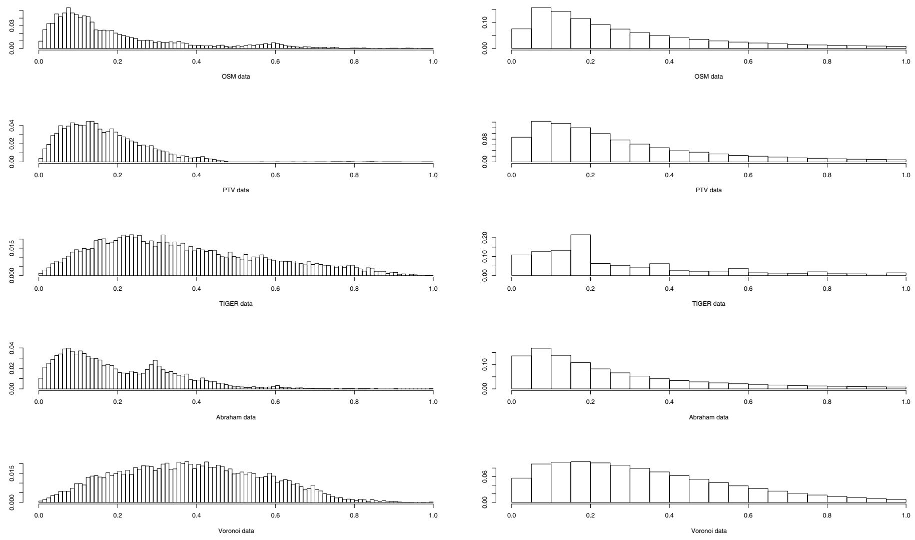

Fig. 1. Left: Distance-distribution sampled by considering 1000 source nodes and, for each source node, 1000 target nodes. Unconnected pairs have been removed, values have been normalized. Right: Edge-weight distributions. Outliers have been removed, values have been normalized. Graphs are from top: OSM, PTV, TIGER, ABR, VOR.

thus, we will use their performance as an indicator for the similarity of the networks.

As a benchmark set we analyze the data from the real-world instances PTV, TIGER and OSM as well as the synthetically generated graphs VOR and ABR. Since the speedup of the various techniques is non-trivially correlated with the size of the graph, we sampled random snapshots with sizes ranging from 1,000 up to 512,000 from our benchmark set in order to be able to capture the underlying correlation. Sampling has been performed exploiting the given geometrical layout and extracting square-sized subgraphs around random points. Thereby, the diameter has been adapted, such that the resulting graph has the desired size. In order to assess the quality of the synthetic data, we also included data from generators that are not custom-tailored for road-networks: Grid-graphs and unit-disk-graphs (a description of the generators is given at the end of the previous section). The respective data-sets are named GRID and UD. The measured speedups are summarized in Table 4.

The speedups we measured using a bidirectional Dijkstra search range between 1.1 and 1.8 and are concentrated around 1.5 for all graph sizes. There does not seem to be any significant trend in this data and, hence, we omit a detailed analysis of this speedup technique. Our results can be summarized as follows: Real-world graphs significantly differ in their algorithmic behavior: Although OSM and TIGER behave similarly with respect to the speedup techniques ALT, Arc-Flags and CH, the two speedup techniques differ by almost a factor

Table 4. Speedups of the tested Point-To-Point Shortest Path Algorithms

technique	#nodes	OSM	TIGER	PTV	VOR	ABR	UD	GRID
ALT	2000	1.9	3.9	5.1	4.9	9.3	5.9	8.1
ALT	32000	5.5	9.6	10.7	11.3	16.4	12.3	12.6
ALT	128000	10.8	10.8	18.2	13.8	31.4	21.7	22.2
ALT	256000	15.3	13.4	26.9	16	35.8	26.1	25.6
ALT	512000	16	17.7	30.6	21.6	37.3	28.8	24.2
Arc-Flags	2000	4.4	14.5	20.7	20.5	48.6	29.1	35.7
Arc-Flags	32000	21.8	55.9	68.5	89.7	131.5	44.1	32.7
Arc-Flags	128000	48.7	65.7	142.2	111.7	219.9	64.4	52.3
Arc-Flags	256000	69.5	76.2	219.9	143.2	245.6	74	72
CH	2000	24.3	16.5	20.7	13.9	13.7	9	7.5
CH	32000	100.4	80.5	102.5	114.9	42.2	38.3	29.1
CH	128000	229	209.2	390.2	346.4	184.6	134.2	74.5
CH	256000	435.2	357.3	626.1	684.7	327.8	195.7	128.2
CH	512000	459.5	497.9	831.4	1565.5	474.2	271.8	175.6
Reach	2000	3.9	3.3	3.8	3.3	4.2	1.8	1.2
Reach	32000	10.1	7.5	9.1	9.9	5.9	2.9	2.1
Reach	128000	28.2	15.7	24.2	22.3	10.1	4.2	2.9
Reach	256000	37.8	22.8	31.8	32.9	10	4.4	4
Reach	512000	34.3	16.3	27.1	49.2	10.7	5	4.7

of two with respect to Reach. Even worse, the PTV-dataset achieves speedups almost twice as large as OSM and TIGER with ALT and CH, and it shows a totally different trend concerning its behavior with Arc-Flags.

GRID and UD approximate streetgraphs only to a limited extent: Both GRID and UD behave similarly for all speedup techniques we considered. Although both graphs seem to be rather good approximations for PTV with respect to ALT and for both OSM and TIGER with respect to Arc-Flags, they seem to be rather bad estimates for CH and Reach. This may be explained by the lack of hierarchy inherent to both generators and the fact that both techniques are based on hierarchical decompositions of the underlying graphs.

ABR and VOR are reasonably good approximations: Contrary to GRID and UD, ABR and VOR seem to capture both the magnitude and the trend exhibited by real-world instances quite well. The speedups measured for ABR are slightly too large for ALT and Arc-Flags and they are slightly too small for Reach. For CH, on the other hand, they are very close to the speedups measured for OSM and TIGER. Except for CH, the speedups measured for Voronoi are well in between the speedups measured for the real-world instances. For CH, the speedups are very close to the speedups for PTV which are larger than those for OSM and TIGER by roughly a factor of two.

Note that parameters of the synthetic data have not been finetuned for approximating the given results. Since there is considerable amount of deviation in the behavior of the real-world data we consider doing so to be over-fitting.

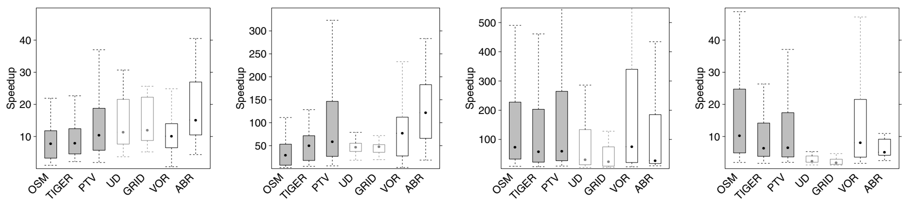

Fig. 2. Speedups observed for different techniques: ALT, Arc-Flags, Contraction Hierarchies and Reach (from left to right)

Summary. Figure 2 shows a summary of the speedups measured for the different techniques. GRID and UD perform poorly for Arc-Flags, CH and Reach, since either the median is far too small as in the case of CH and Reach or the interquartile range is far too small as in the cases of Arc-Flags. Although both may be considered good estimates for the PTV-data with respect to ALT, VOR seems to be the better choice. A similar, albeit slightly better behavior can be observed for ABR: Although it captures the interquartile range rather well for all techniques, except Reach, the medians do not fit too well. VOR, on the other hand, seems to be the best fit with respect to both median and interquartile range. Although the median is slightly too large for Arc-Flags and the interquartile range is too large for CH it seems to capture the behavior of the real-world data with respect to the speedup techniques best.

References

1. Cgal, Computational Geometry Algorithms Library, `http://www.cgal.org`
2. Abraham, I., Fiat, A., Goldberg, A.V., Werneck, R.F.: Highway Dimension, Shortest Paths, and Provably Efficient Algorithms. In: Proceedings of the 21st Annual ACM–SIAM Symposium on Discrete Algorithms, SODA 2010 (2010)
3. Bauer, R., Delling, D., Wagner, D.: Experimental Study on Speed-Up Techniques for Timetable Information Systems. In: Liebchen, C., Ahuja, R.K., Mesa, J.A. (eds.) Proceedings of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2007), Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, pp. 209–225 (2007)
4. Bose, P., Carmi, P., Farshi, M., Maheshwari, A., Smid, M.: Computing the Greedy Spanner in Near-Quadratic Time. In: Gudmundsson, J. (ed.) SWAT 2008. LNCS, vol. 5124, pp. 390–401. Springer, Heidelberg (2008)
5. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Engineering Route Planning Algorithms. In: Lerner, J., Wagner, D., Zweig, K.A. (eds.) Algorithmics of Large and Complex Networks. LNCS, vol. 5515, pp. 117–139. Springer, Heidelberg (2009)
6. Demetrescu, C., Goldberg, A.V., Johnson, D.S. (eds.): The Shortest Path Problem: Ninth DIMACS Implementation Challenge. DIMACS Book, vol. 74. American Mathematical Society, Providence (2009)

7. Eppstein, D., Goodrich, M.T.: Studying (non-planar) road networks through an algorithmic lens. In: Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM Press, New York (2008)
8. Peleg, D., Schäffer, A.A.: Graph spanners. Journal of Graph Theory 13(1), 99–116 (1989)
9. Preparata, F.P., Shamos, M.I.: Computational geometry: an introduction. Springer, New York (1985)

A Additional Technical Information

Extraction of the OSM-Dataset. To distill the OSM-dataset, we used the Europe-data available at `http://download.geofabrik.de/osm/` of December 2009. The graph includes all elements for which the highway-tag equals one of the following values: residential, motorway_link, trunk, trunk_link primary, primary_link, secondary, secondary_link, tertiary, unclassified, road, residential, living_street, service, services.

Generation of the Synthetic Data. The graph VOR is a 4-level graph computed using the following parameters: Level 1 has 1700 centers, density of 0.2 The Voronoi streetgraph was computed using the parameters listed in Table 5. In the second phase we greedily computed a 4-spanner subgraph.

The first phase of the algorithm took 122 minutes and consumed up to 19 GB of space; the second phase of the algorithm took 28 minutes consumed up to 21 GB of space. The ABR streetgraph has 15 mio. nodes and more than 43 mio. edges. It was generated within 63 minutes and consumed up to 8.39 GB of disk space.

Table 5. Parameters of the Voronoi streetgraph

Level i	# of centers $\mathcal{C}_i$	density $\mathcal{D}_i$	radius $\mathcal{R}_i$	fraction γ_i
1	1700	.2	$\text{Exp}_{.01}$	.95
2	$\text{Unif}_{[2,40]}$	.5	$\text{Exp}_{.1}$	.9
3	$\text{Unif}_{[2,70]}$	.9	Exp_{2}	.7
4	$\text{Unif}_{[4,40]}$	.0	0	0

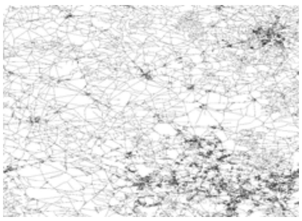
Fig. 3. PTV graph

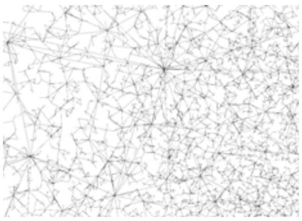
Fig. 4. ABR graph

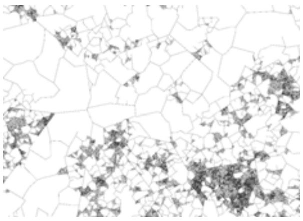
Fig. 5. VOR graph