Flexible Routing with GraphHopper
And how it can be misused for data analysis

Peter Karich SOTM 2019
Co-Founder GraphHopper
Who am I?
GraphHopper company
What is our mission?

Peter Karich
Stefan Schröder
Dr.-Ing. Michael Zilske
Maximilian Sturm
Robin Boldt
Dr. Michal Maciejewski
Dr. Andreas Barth
GraphHopper Routing Engine

• Open Source under Apache License 2.0
• Java library and web service for routing
  No maps, no geocoding
• It is fast and memory efficient
• Works with OpenStreetMap data, GTFS and others
• Algorithms: Dijkstra, A*, Landmarks, CH
• Out of the box: for walking, car, bike, public transit, …
Features

Here is a list of the more detailed features including a link to the documentation:

- **Quick installation and start for users** - just Java necessary! Simple start for developers due to Maven.
- Works out of the box with OpenStreetMap (osm/xml and pb) and can be adapted to custom data
- OpenStreetMap integration: Takes care of the road type, speed limit, the surface, barriers, access restrictions, ferries, conditional access restrictions, ...
- GraphHopper is fast. And with the so called “Contraction Hierarchies” it can be even faster (enabled by default).
- Memory efficient data structures, algorithms and the low and high level API is tuned towards ease of use and efficiency
- Provides a simple web API including JavaScript and Java clients
- Multiple weightings (fastest/shortest/...) and pre-built routing profiles: car, bike, racingbike, mountain bike, foot, motorcycle, ...
- Supports public transit routing and GTFS.
- Offers turn instructions in more than 35 languages, contribute or improve here
- Displays and takes into account elevation data (per default disabled)
- Can apply real time changes to edge weights (flexible and hybrid mode only)
- Customization of vehicle profiles per request are possible (flexible and hybrid mode only)
- Possibility to specify a heading parameter of the vehicle for start, end and via points for navigation applications via pass_through or heading parameters (flexible and hybrid mode only)
- Generate routes (flexible and hybrid mode only)
- Costs and restrictions

**Country specific routing via SpatialRules**

- The core uses only a few dependencies (hppc, jts and slf4j)
- Scales from small indoor-sized to world-wide-sized graphs
- Finds nearest point on street e.g. to get elevation or ‘snap to road’ or being used as spatial index (see #1485)
- Does map matching with GraphHopper
- Calculates isochrones with GraphHopper
- Shows details along a route (“path details”) #1142
- Shows the whole road network in the browser for debugging purposes (“vector tile support”) #1572
Graphhopper

Selected New Features
Vector Tiles Endpoint

#1572
Shortest Path Tree Endpoint

- CSV with lat,lon and previous coordinate
- Feedback from community

- Example in R lang:
Use GraphHopper For Data Analysis

1. Impact of bridge construction on road network. High precise “Isochrones”: draw shortest path tree directly in browser. Simulate “what if” scenarios
2. Level of Traffic Stress & Highlight curly roads
3. Speed limit debate regarding safety
4. Plan location of new fire station
5. Find closest restaurants by driving time & Find closest restaurants from a route
1. Impact of bridge construction
Avoid highway=motorway
Uses /spt endpoint
2. Level of Traffic Stress For Biking

- Avoid biking on dangerous roads
- Prefer bike routes
- Modify render rule
- Demo Link
2. Highlight Curvy Roads

- For some people curly roads are dangerous. For others they are attractive.
- Fetch vector tiles from /mvt endpoint and return curvy factor e.g. <0.6 → red
- Modify render rule
3. Speed limit debate regarding safety

- German crash data 2016 and 2017 from “destatis”
- OpenStreetMap speed limit data
- Use new storage feature for highway tag, maxspeed and crash counter
3. Speed limit debate regarding safety

Results:

- 13500 km highways in Germany
- ~65% highways without speed limit (official source is similar)
- 69.5% of deaths on segments w/o speed limit
- Traffic density required →
- Signs could safe >100 lifes/a
4. Plan Location of new Fire Station

How to find gaps in reachability?
→ Multi-source isochrone
5. Find Closest Restaurants

- Get ~18K restaurants:
  `bzgrep -B 1 restaurant germany.osm.bz2 | grep node`

- Store restaurant count per edge
  → 5s on my old laptop

- Start in “Erfurt” city and explore Germany
  9.3M nodes & 11.8M edges

- Return the list of “driving-time-sorted”
  restaurants
  → <30s
5. Histograms with Restaurants for Fun

- **Berlin**
- **Erfurt**
- **Heidelberg Univ.**
- **Graphhopper HQ in Munich**
5. Find Closest Restaurants along a route

Or same algorithm, different problem: Find shortest path from location to river

Stuttgart → Hamburg (~35s)

Berlin → Mönchengladbach (~30s)
Advantages

- Fast
- Handles massive data well (even on weak computers)
  
  Avoid loading everything into memory via graph.dataaccess=MMAP_STORE
- Perfect for everything that requires road connectivity

Disadvantages

- Need to select properties of the source data that go into the graph
  
  max_speed, distance, avg_speed, max_height, max_width, road_class, surface, road_environment, toll, ...
- Certain use cases still require Java knowledge
Resources

- Different tweaks like curvy roads & find restaurants along a route: [https://github.com/graphhopper/graphhopper/tree/sotm_trials](https://github.com/graphhopper/graphhopper/tree/sotm_trials)
- Crash stats: [https://github.com/karussell/crashstats/](https://github.com/karussell/crashstats/)
- Destatis: [https://unfallatlas.statistikportal.de/](https://unfallatlas.statistikportal.de/)
We are looking for contributors!

Contribute Code & Translations
https://github.com/graphhopper/graphhopper/contribute

Forum
https://discuss.graphhopper.com/
Flexible Routing with GraphHopper

peter.karich@graphhopper.com