```
                                std::shared_ptr<std::vector<EdgeWeight>> result_table) const
{
    const NodeID node = query_heap.DeleteMin();
    const int source_distance = query_heap.GetKey(node);

    // check if each encountered node has an entry
    const auto bucket_iterator = search_space_with_buckets.find(node);
    // iterate bucket if there exists one
    if (bucket_iterator != search_space_with_buckets.end())
    {
        const std::vector<NodeBucket> &bucket_list = bucket_iterator->second;
        for (const NodeBucket &current_bucket : bucket_list)
        {
            // get target id from bucket entry
            const unsigned target_id = current_bucket.target_id;
            const int target_distance = current_bucket.distance;
            const EdgeWeight current_distance =
                (*result_table)[source_id * number_of_locations + target_id];
            // check if new distance is better
            const EdgeWeight new_distance = source_distance + target_distance;
            if (new_distance > 0 && new_distance < current_distance)
            {
                (*result_table)[source_id * number_of_locations + target_id] =
                    (source_distance + target_distance);
            }
        }
    }
    if (StallAtNode<true>(node, source_distance, query_heap))
```

Everything But Directions

All the Other Exciting Things You Can Do With Routing

Dennis Luxen
Mapbox / Project OSRM
dennis@mapbox.com

Todays Agenda

1. Overview of (Route) Labeling Algorithms
2. Applications
   - Closest Dispatch
   - Walkability
   - Distance Tables
3. Vandalism Detection

(Route) Labeling Algorithms

Intuitive Approach to Route Planning

Observation 1:

Almost all long-distance routes enter the arterial network at one point.

Observation 2:

The routes to almost everywhere lead over a few but very important locations in the road network.

Intuitive Approach to Route Planning

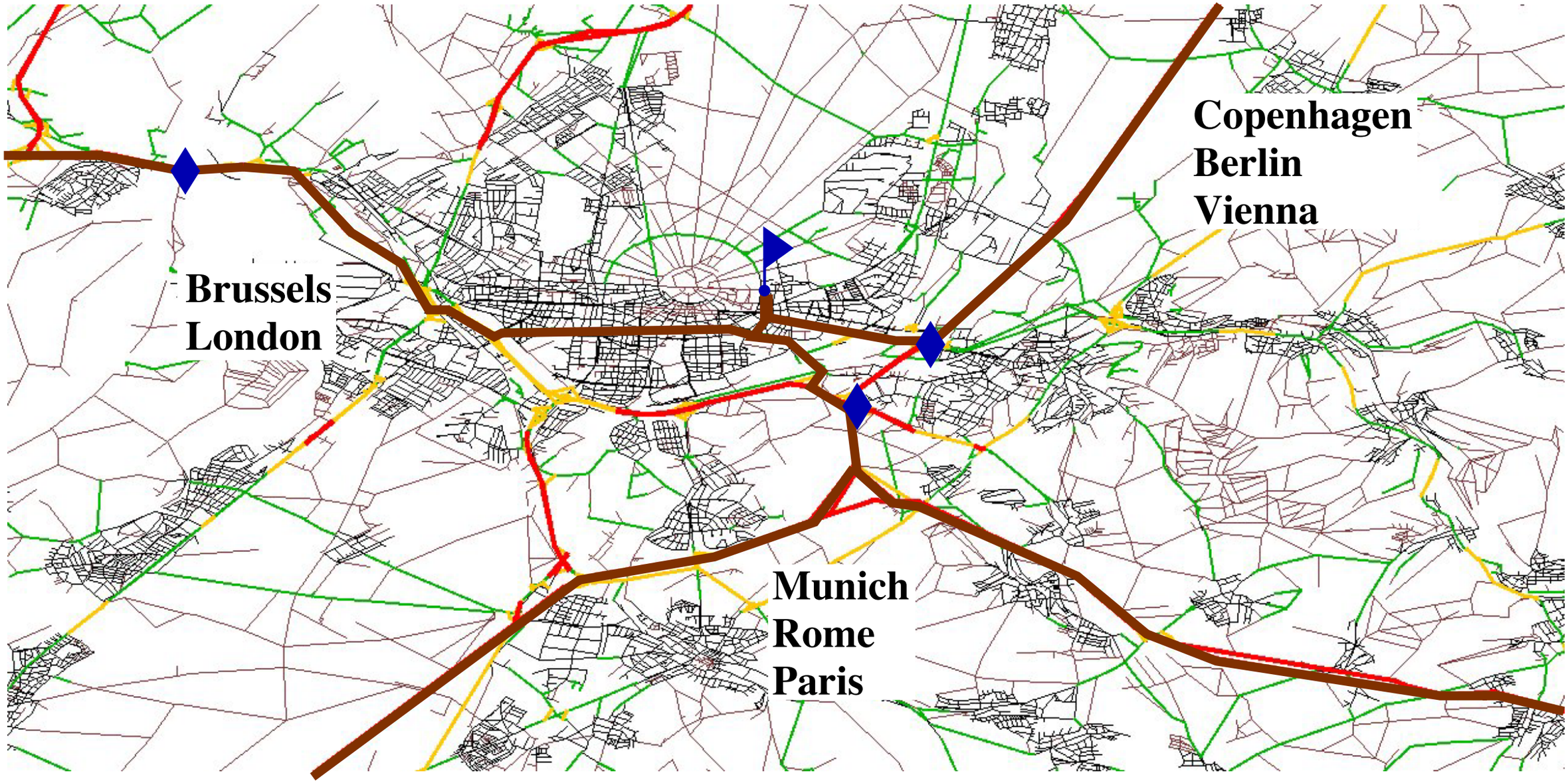
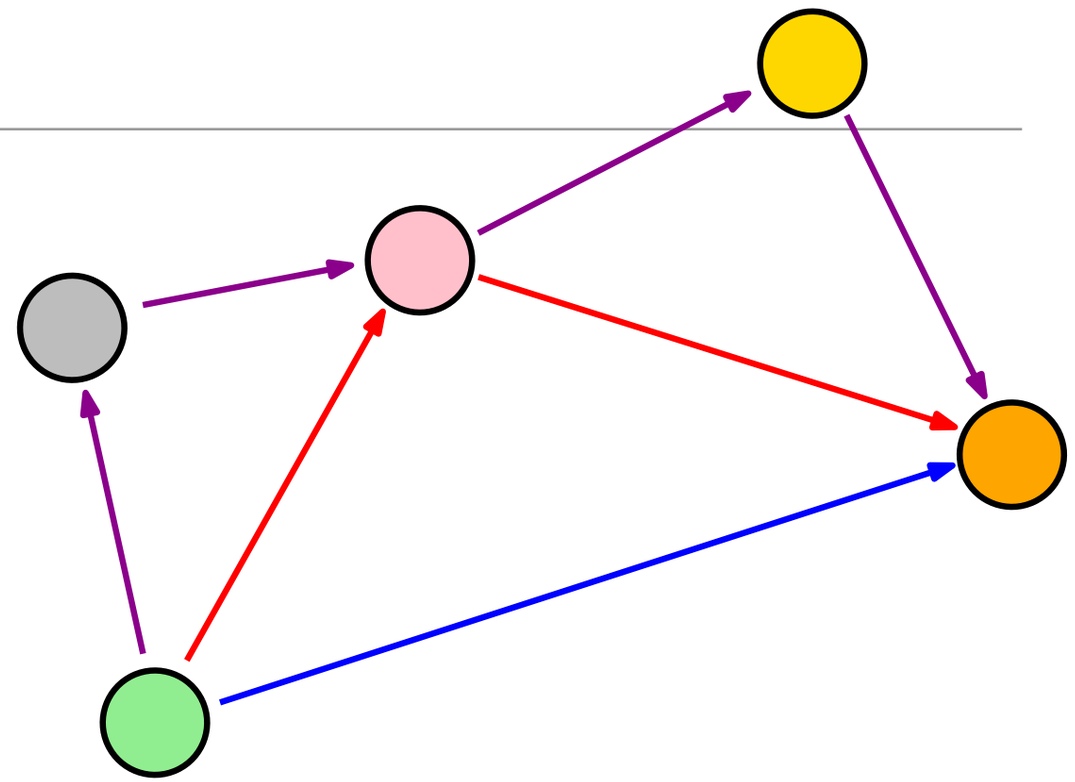


Image from a previous life in academia: http://algo2.iti.kit.edu

(Route) Labeling Algorithms*



Our current approach:

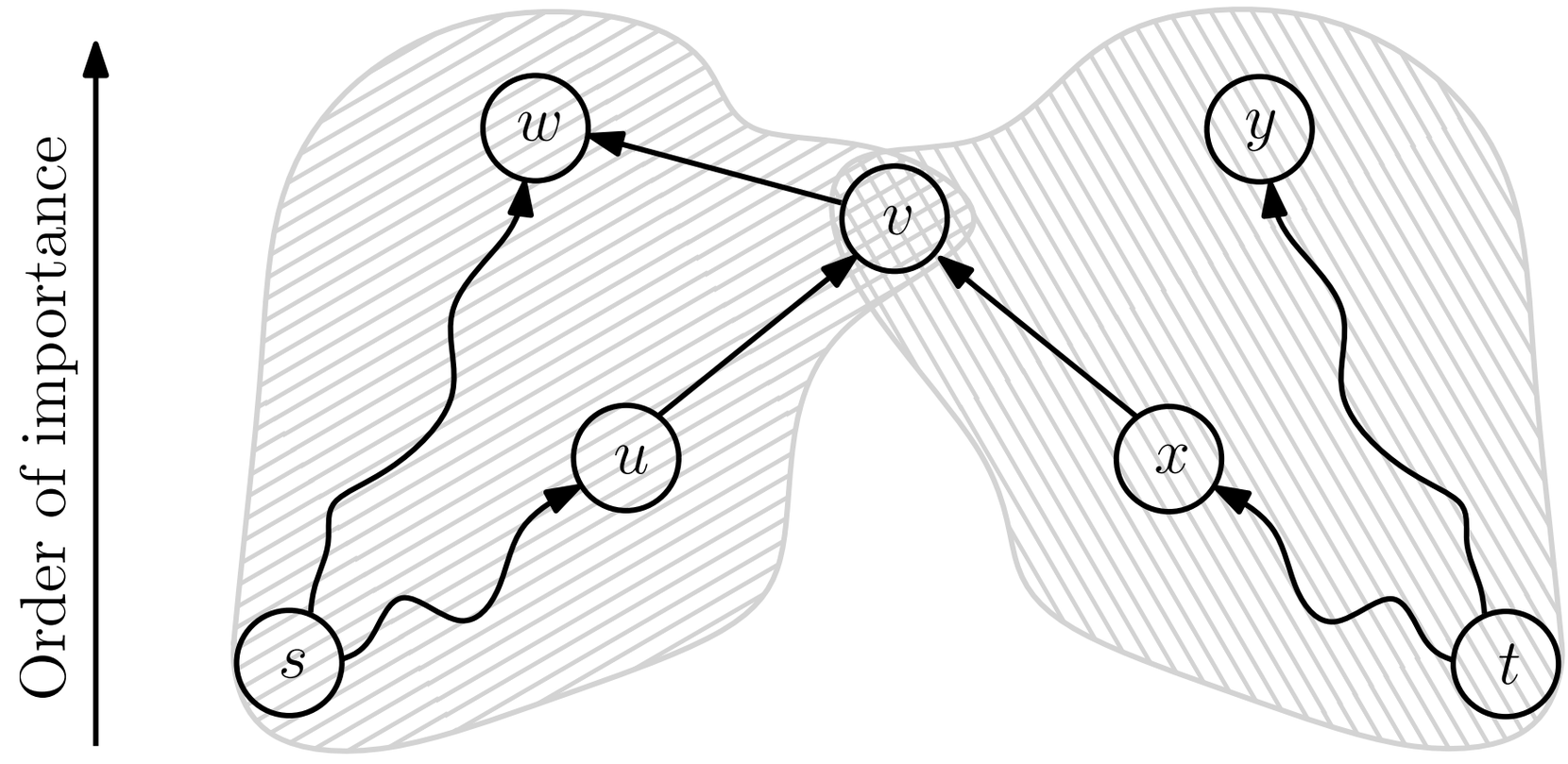
1. Preprocessing the input data:
   - discover layers of *importance*
   - add additional *structure*

2. Leverage this data for fast queries
   - ignore most of the data
   - layaway from one piece of structure to the next
   - search spaces are small

* omitting all the details

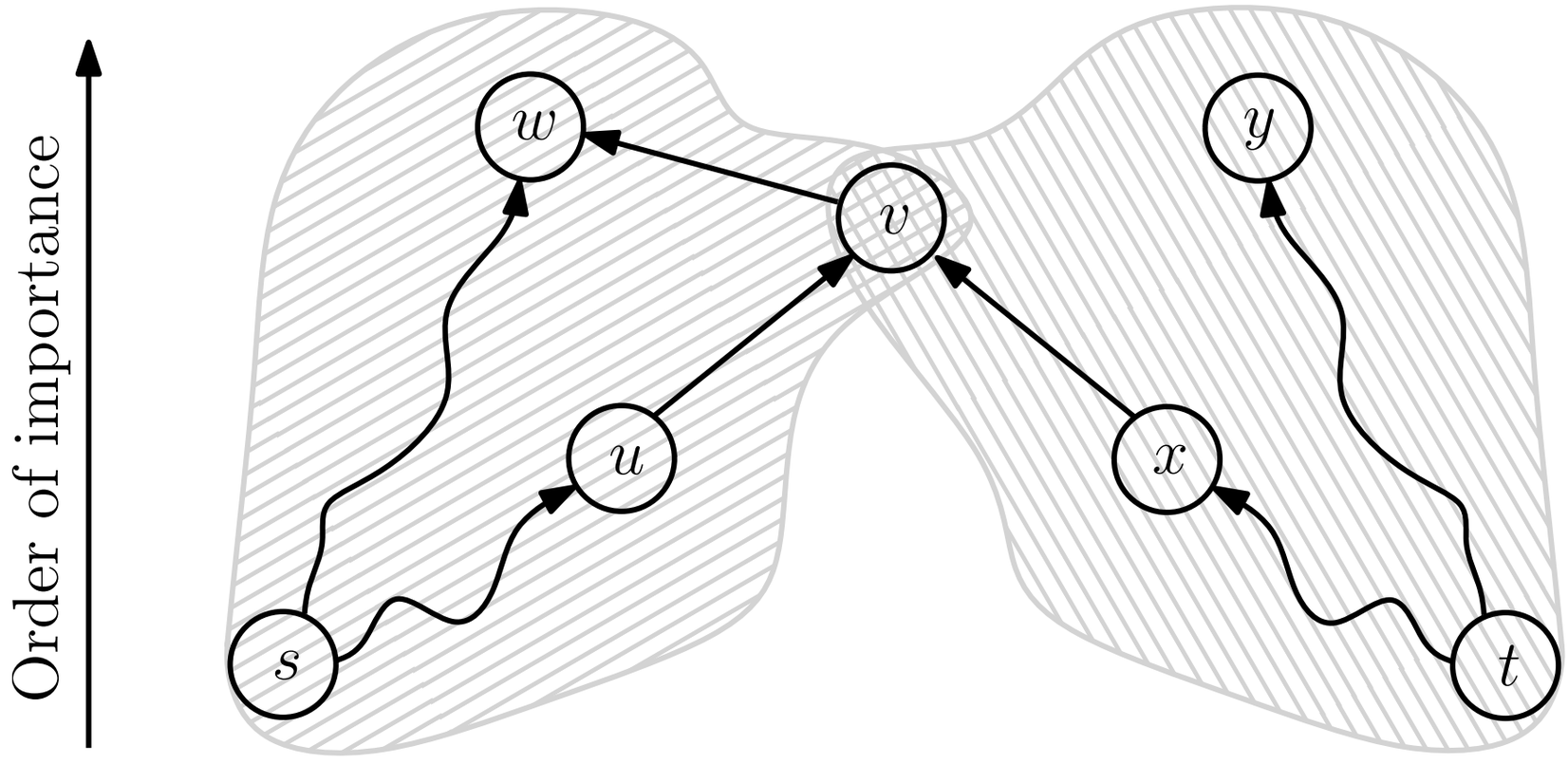
(Route) Labeling Algorithms

Searching simultaneously from source and target location guarantees the **labeling** of one piece of road with the minimum *distances* from both ends.



(Route) Labeling Algorithms

Finding a route is now formulated as looking at the intersection of two sets.

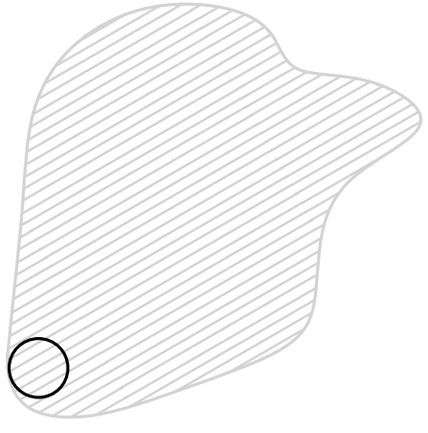
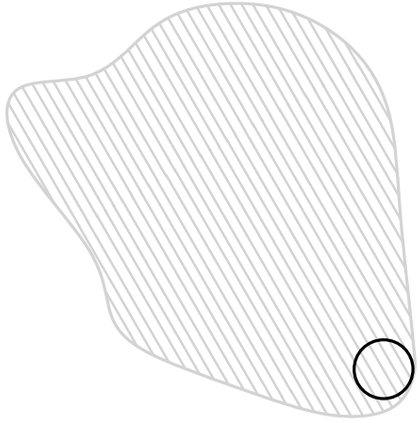


Also note: search spaces of a location always the same

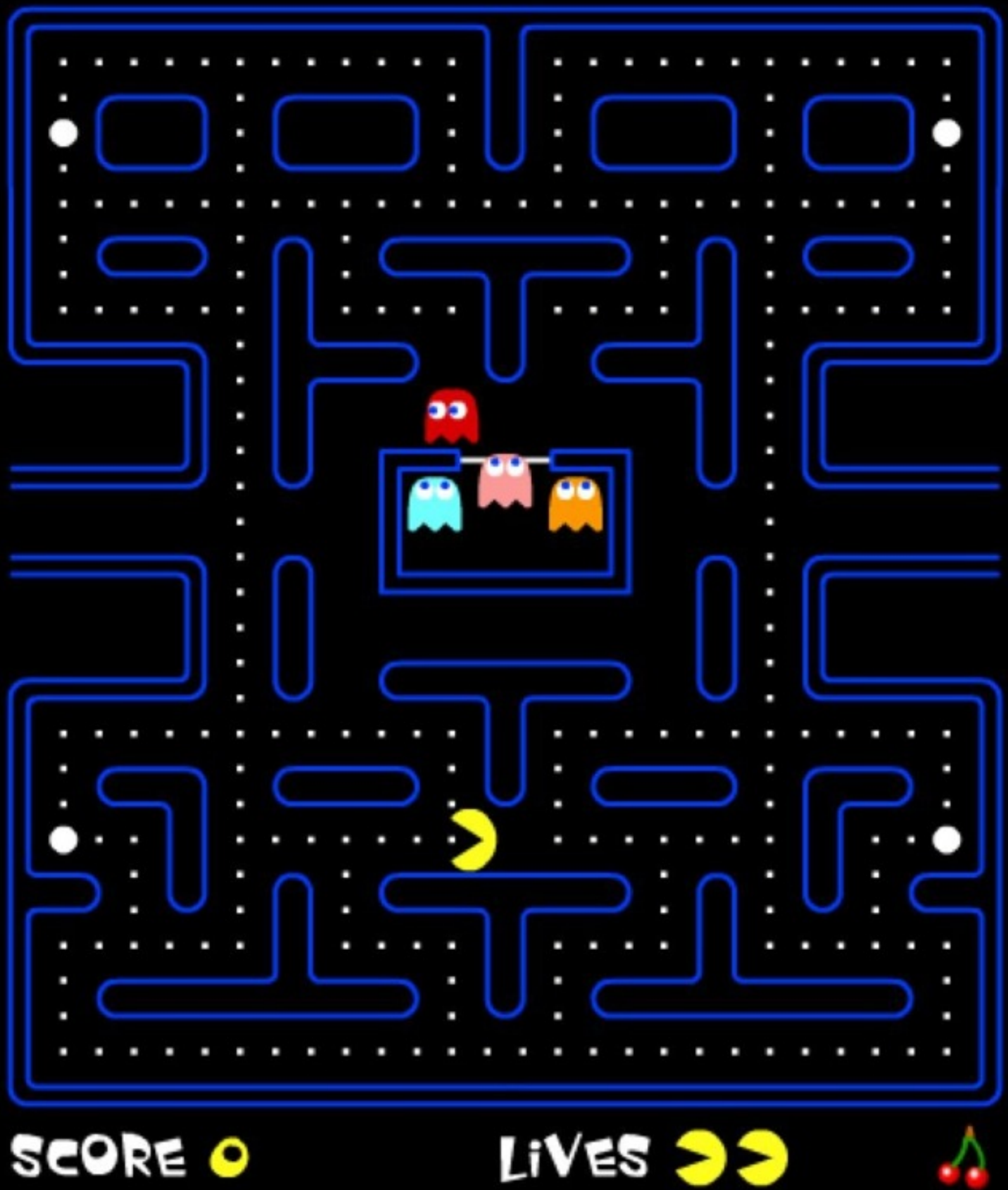
In a nutshell

1. Search space are small subsets of data:
   - forward search from source
   - reverse search from target
2. Paths are found by intersecting both
   - Fast search: Milliseconds even for long-distance routes

<u>Notation:</u>

forward search   reverse search

Application: Closest Dispatch


SCORE 0
LIVES

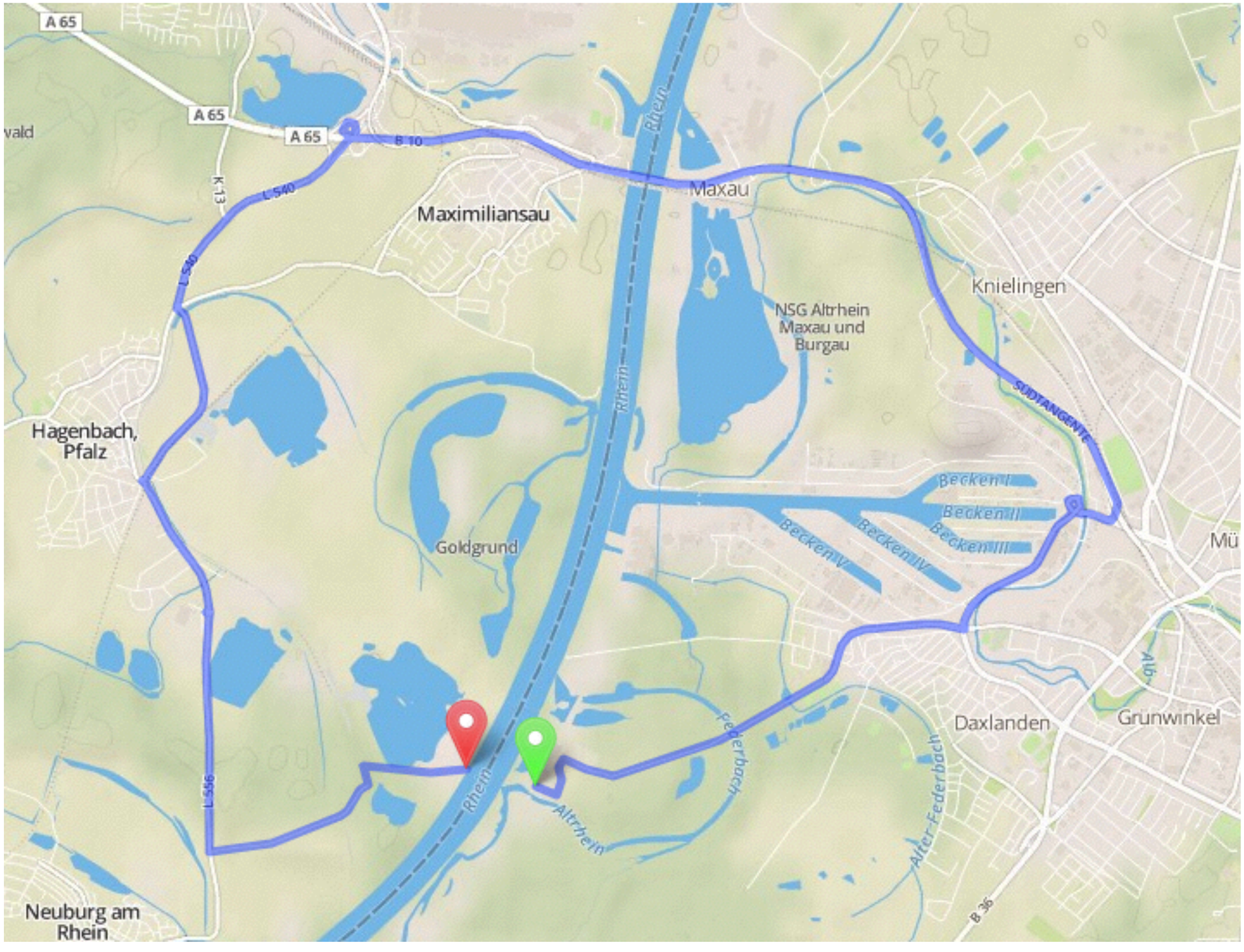
Closest Dispatch

Setting:
- multiple vehicles on patrol
- for a given location find the nearest one to respond

Examples:
- Taxis looking for customers
- Police cars/bikes/peds on patrol

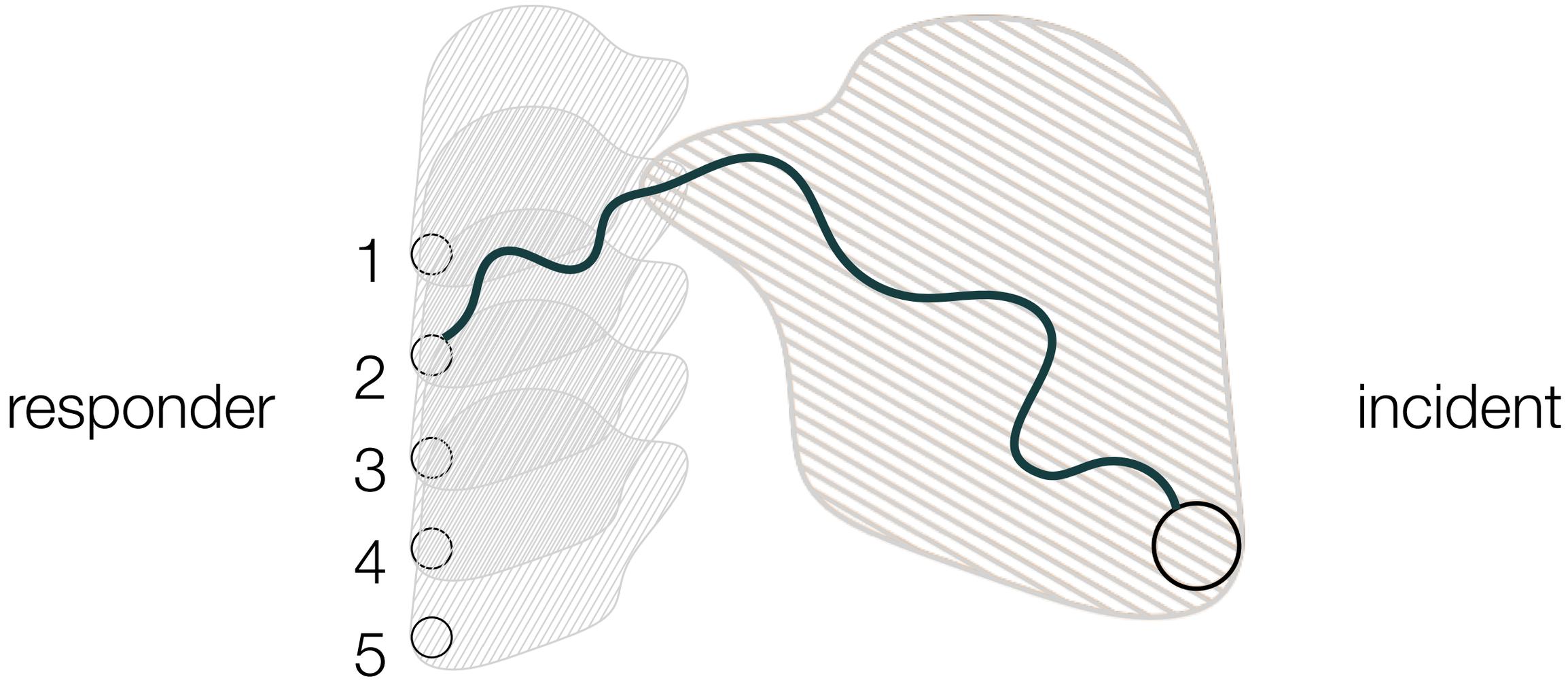
Essentially, this is a many-to-one path problem!

Closest Dispatch - Network vs Crows Fly Distance



Closest Dispatch - Modeling

UPDATE stored *reverse* search spaces w/ GPS ping



Intersect *incident* with the preprocessed data for best responder

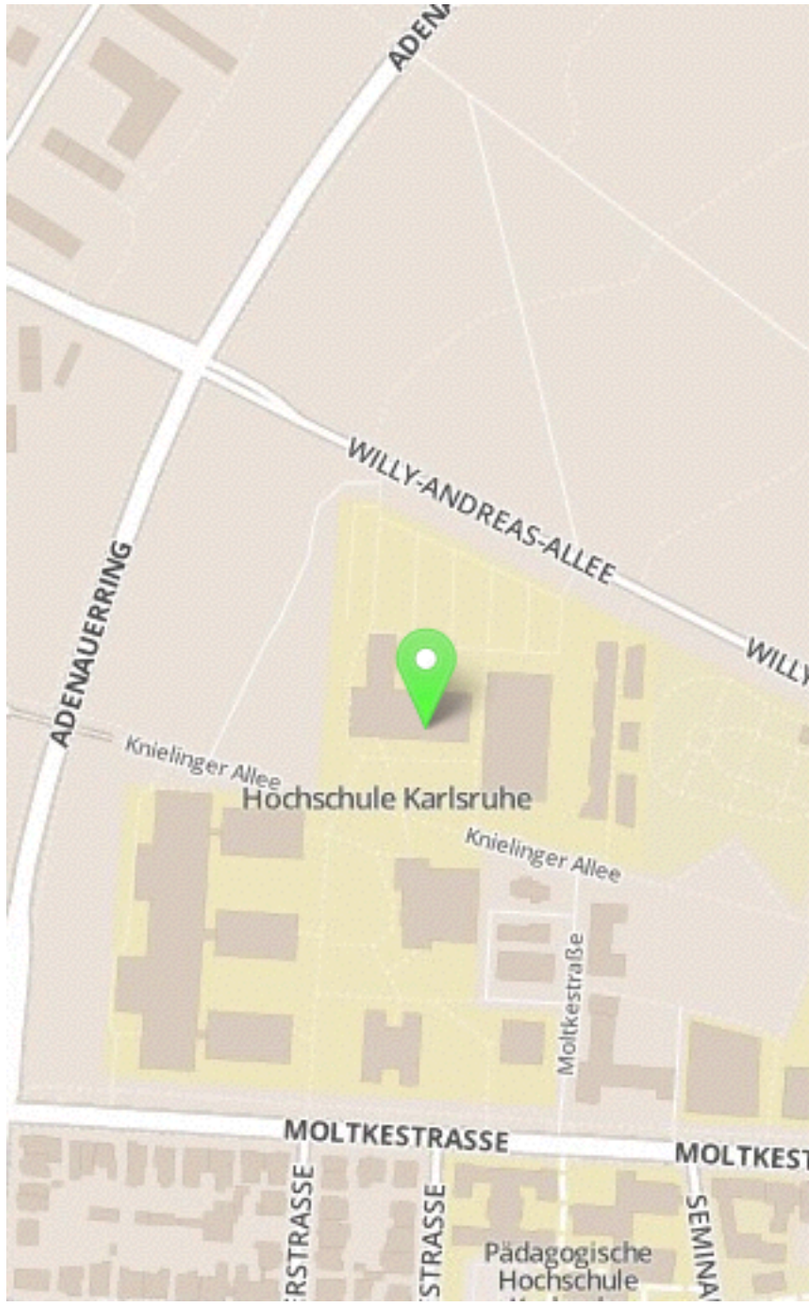
Application: Walkability

How walkable is a location?

We need to look at a location and determine distance to a nearest:

- grocery store
- doctors office
- bar
- italian/chinese/mexican/… restaurant
- bus or tram stop
- ….

Essentially this is a one-to-many problem!

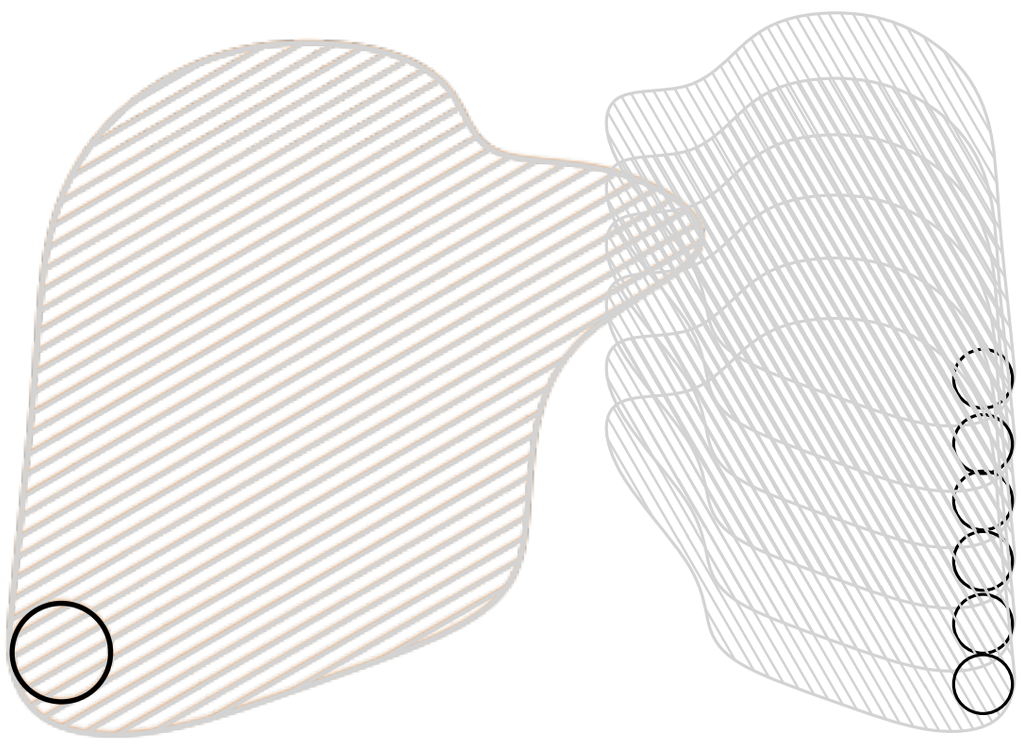


Walkability - Modeling

- Many fixed locations -> targets
- One query location -> one source

Preprocessing simple: reverse search space for each POI

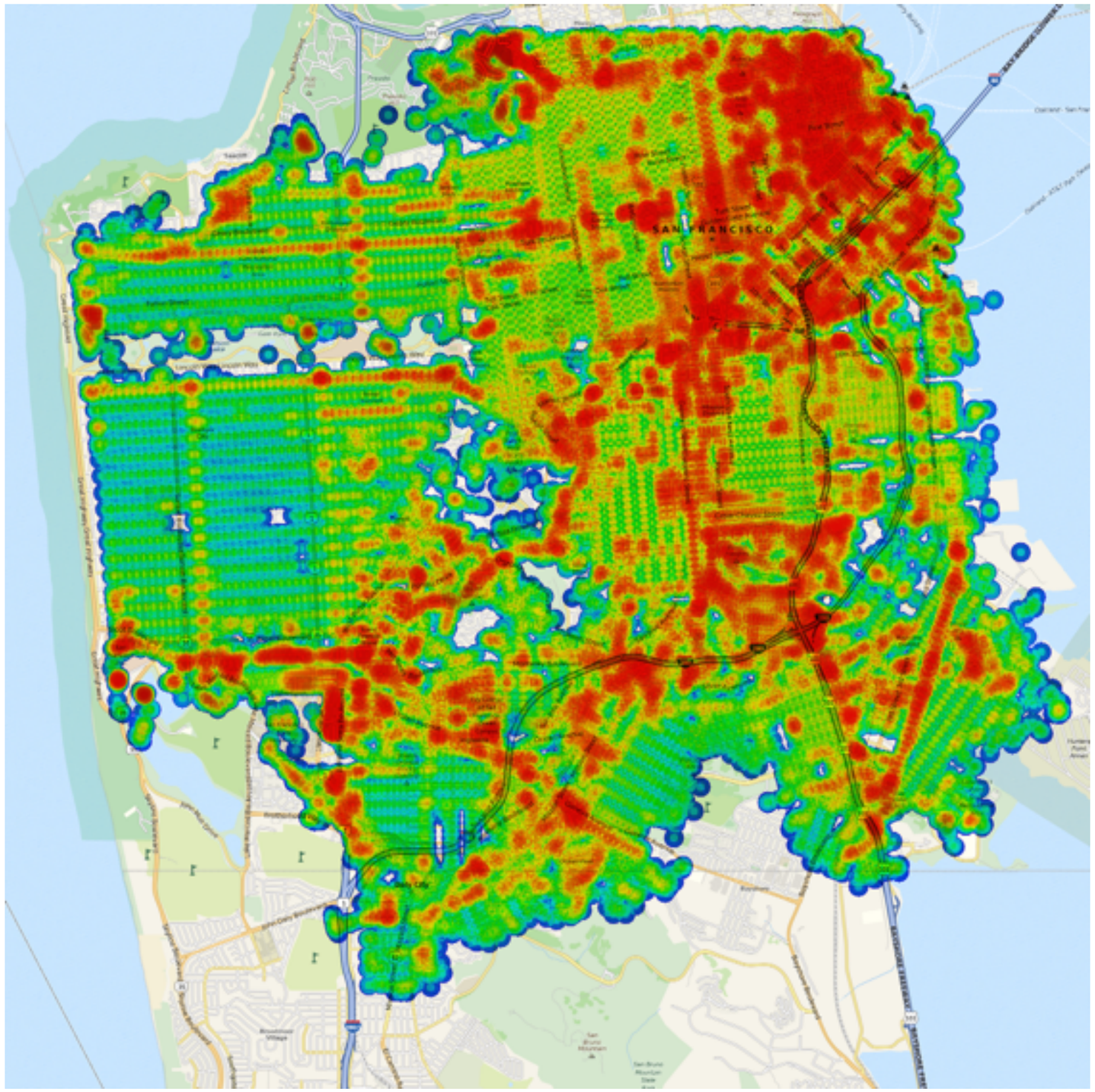
location



POIs

Query: Get one forward search space and aggregate distances

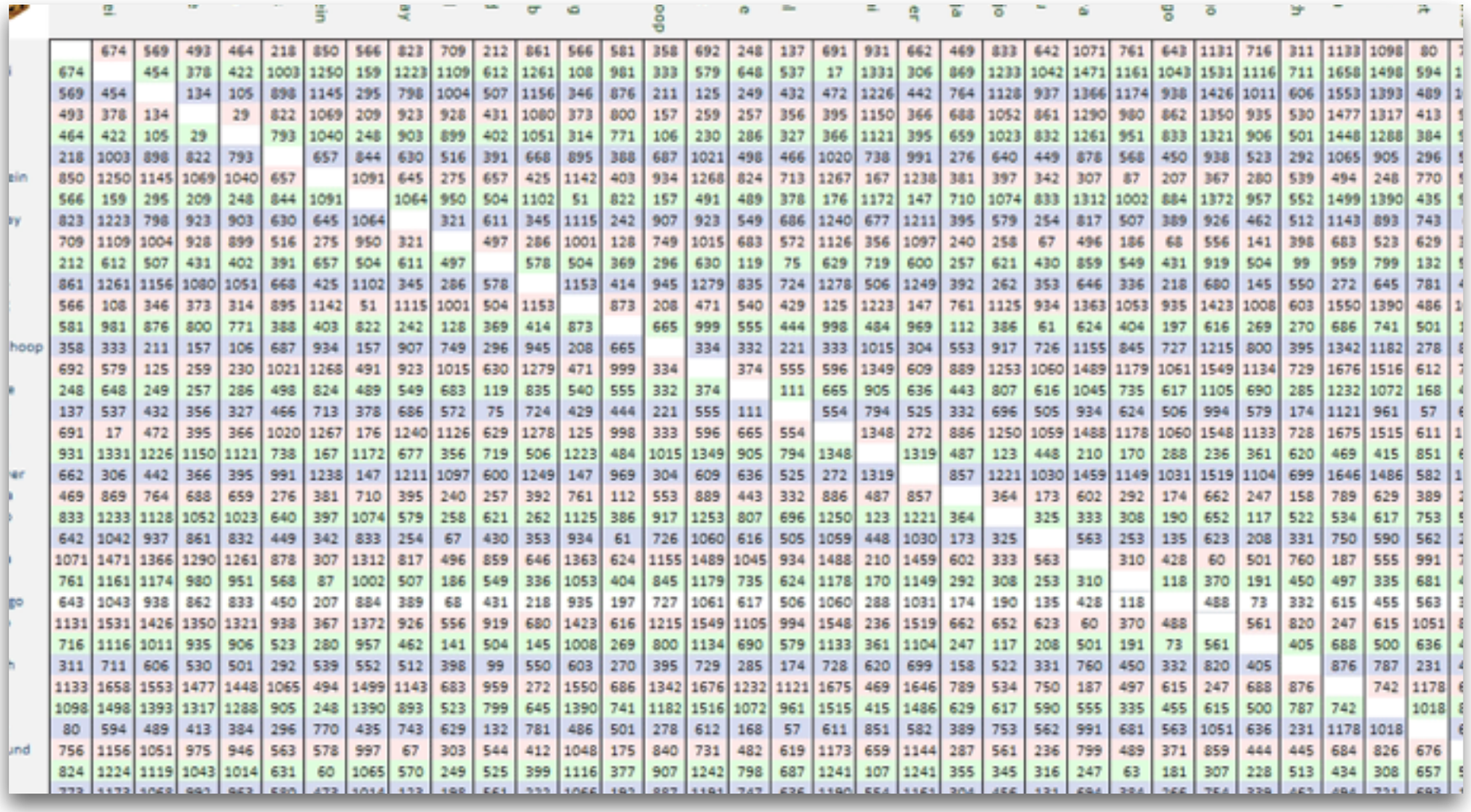
Walkability - Results for San Francisco



Application: Distance Tables

Distance Tables*

Given sets of source and target locations, compute all pair-wise distances.

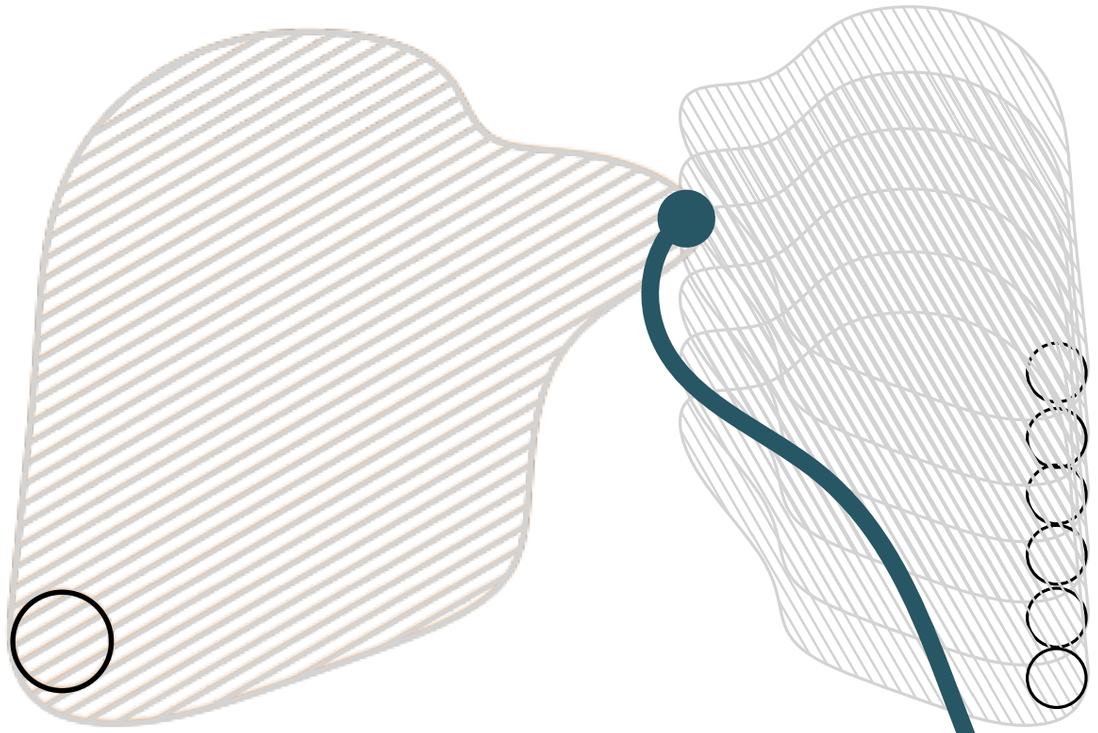


Essentially, this is a many-to-many path problem!

* http://router.project-osrm/table?loc=lat,lon&loc=……&loc=lat,lon

Distance Tables - Approach

1. Preprocess reverse searches for targets
2. Intersect with forward search for each source



	a	b	c	d
a	-	8	6	4
b	23	-	64	4
c	3		-	7
d	35	85	4	-

Search complexity: linear instead of quadratic !

Vandalism Detection

Vandalism Detection

Goal: Detect data issue automatically by looking at change
Here: Detect vandalism in the road network

Remember?

> The routes to almost everywhere lead over a few but very important locations in the road network.

Removing an important street will have an impact on many routes, even between randomly selected locations.

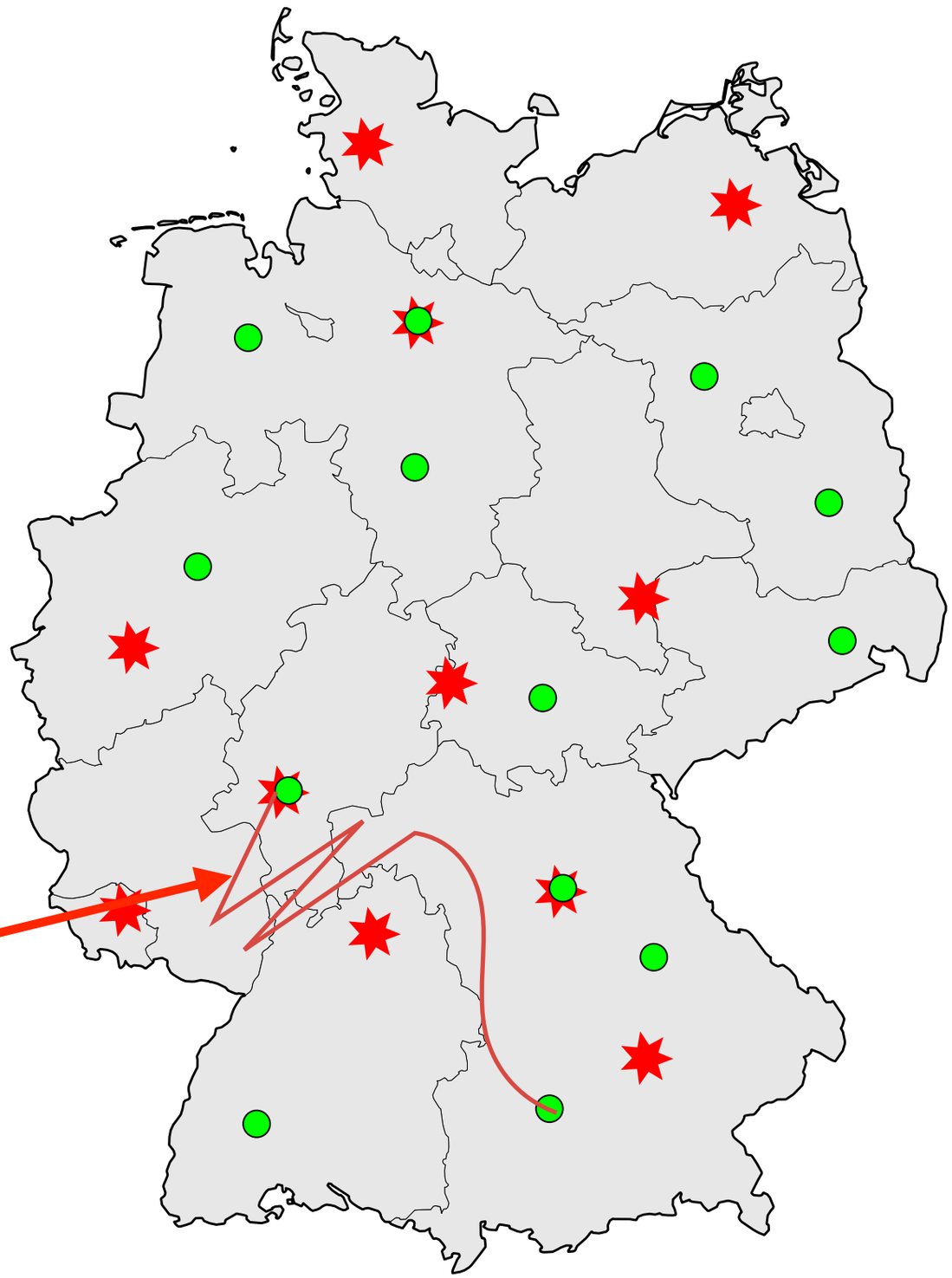
Let’s cover those roads and look for significant change!

Vandalism Detection

1. Select good sets of locations
2. Compute distance tables
3. Do driving times change?

Look into changes

	a	b	c	d
a	-	8	6	4
b	23	-	64	4
c	3	7	-	7
d	35	85	4	-



Questions?

Email: dennis@mapbox.com

Twitter: @ProjectOSRM

Website: http://osrm.at

github: DennisOSRM/Project-OSRM

Yet Another Thing

Ctrl-Alt-Del: Directions from A:\ to Z:\

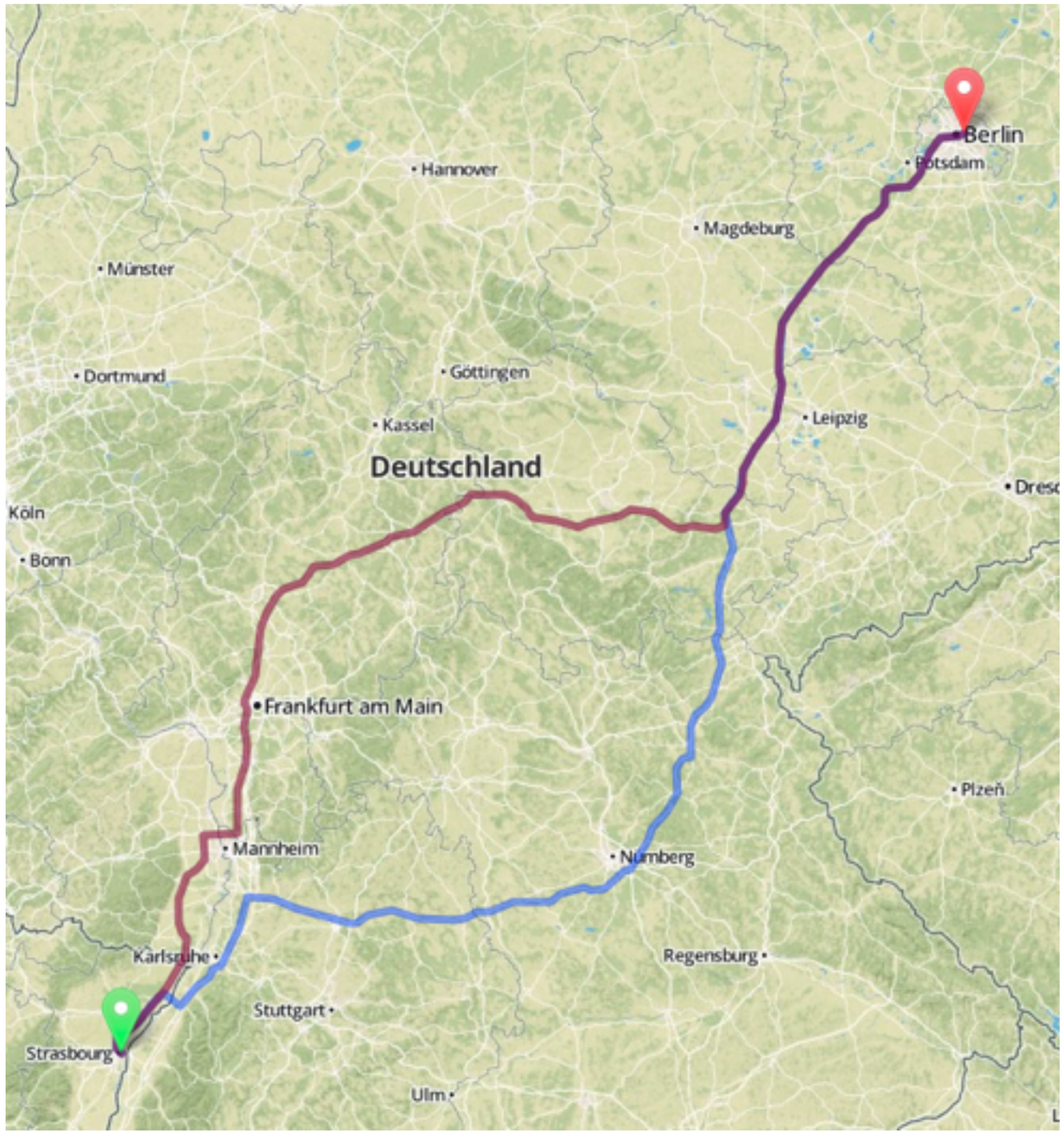
URL: http://build.project-osrm.org

Application: Ride Sharing

Ride Sharing

online ride sharing is dull

only city-to-city?

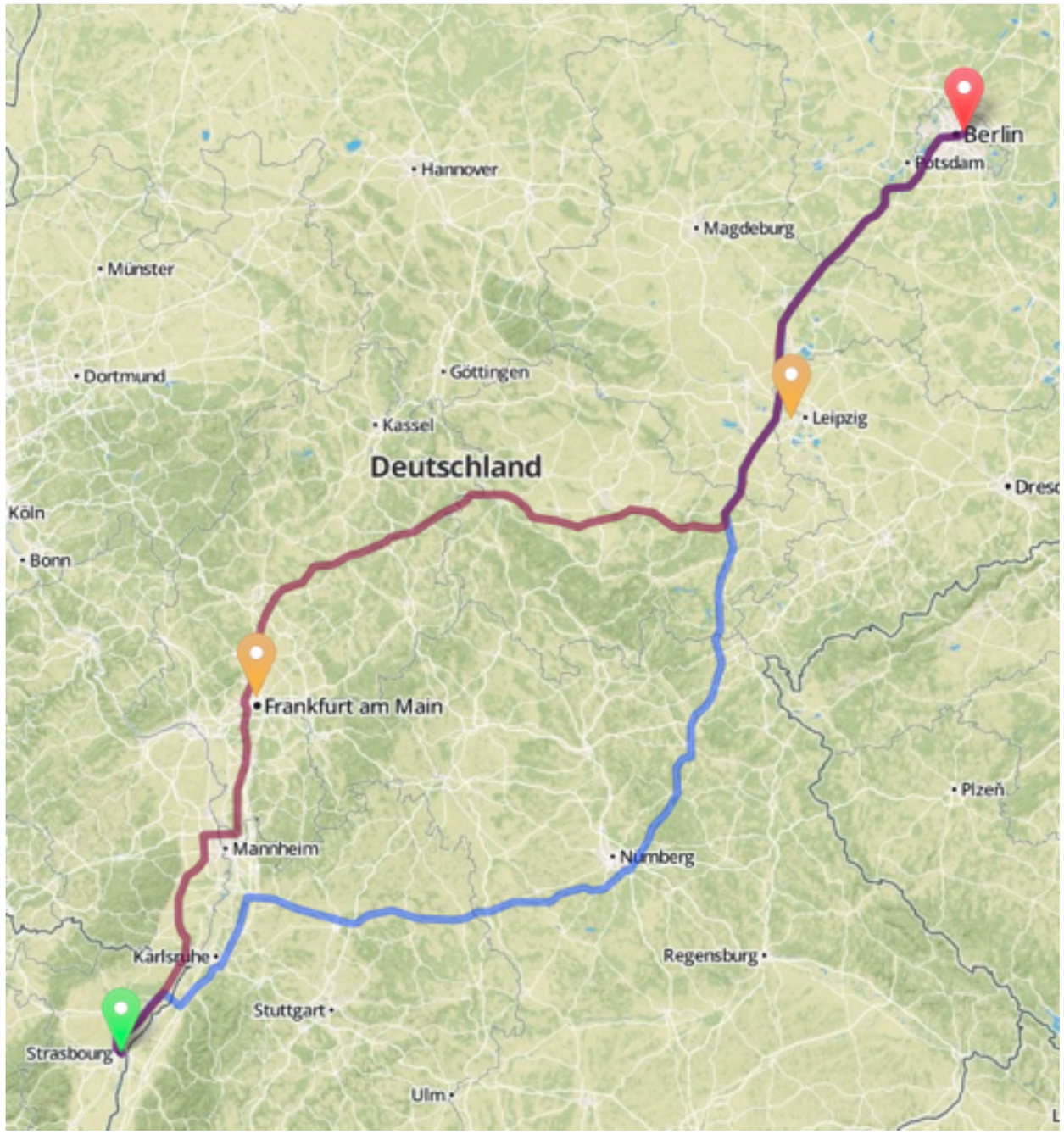


Ride Sharing

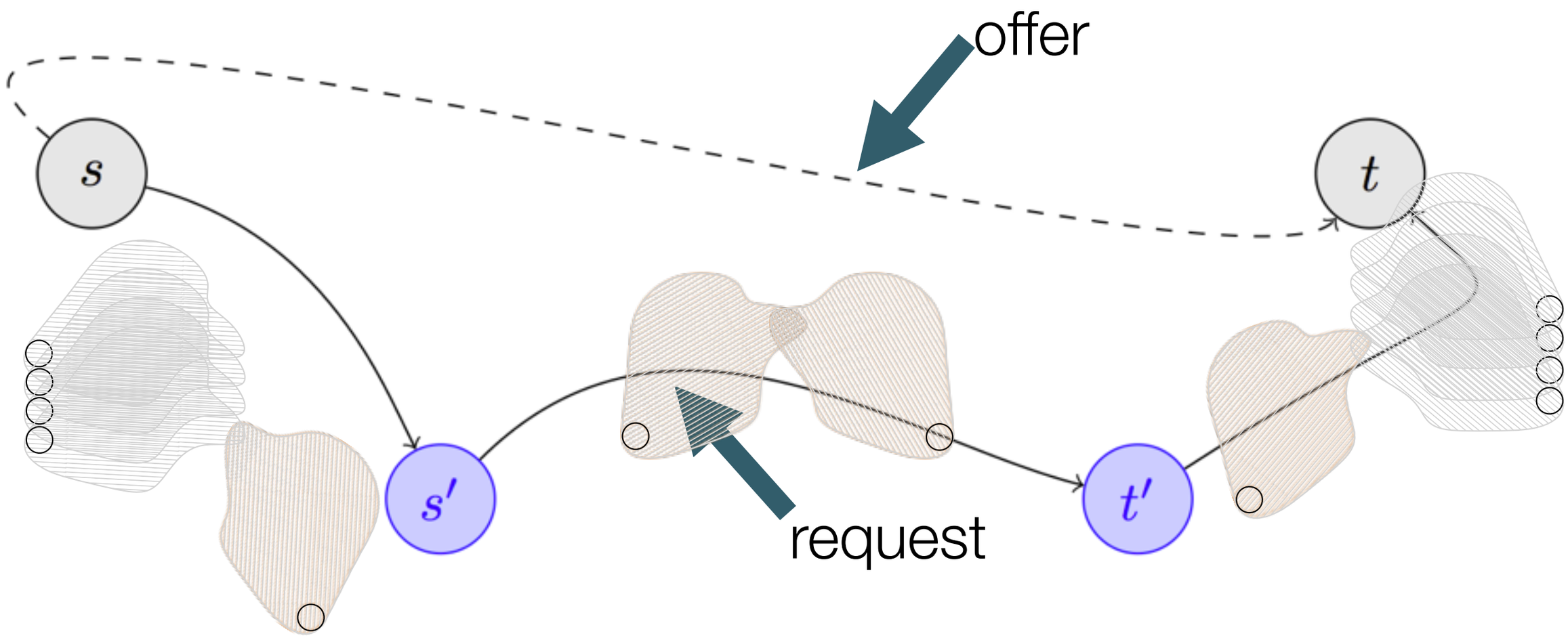
online ride sharing is dull

only city-to-city?

reasonable detours



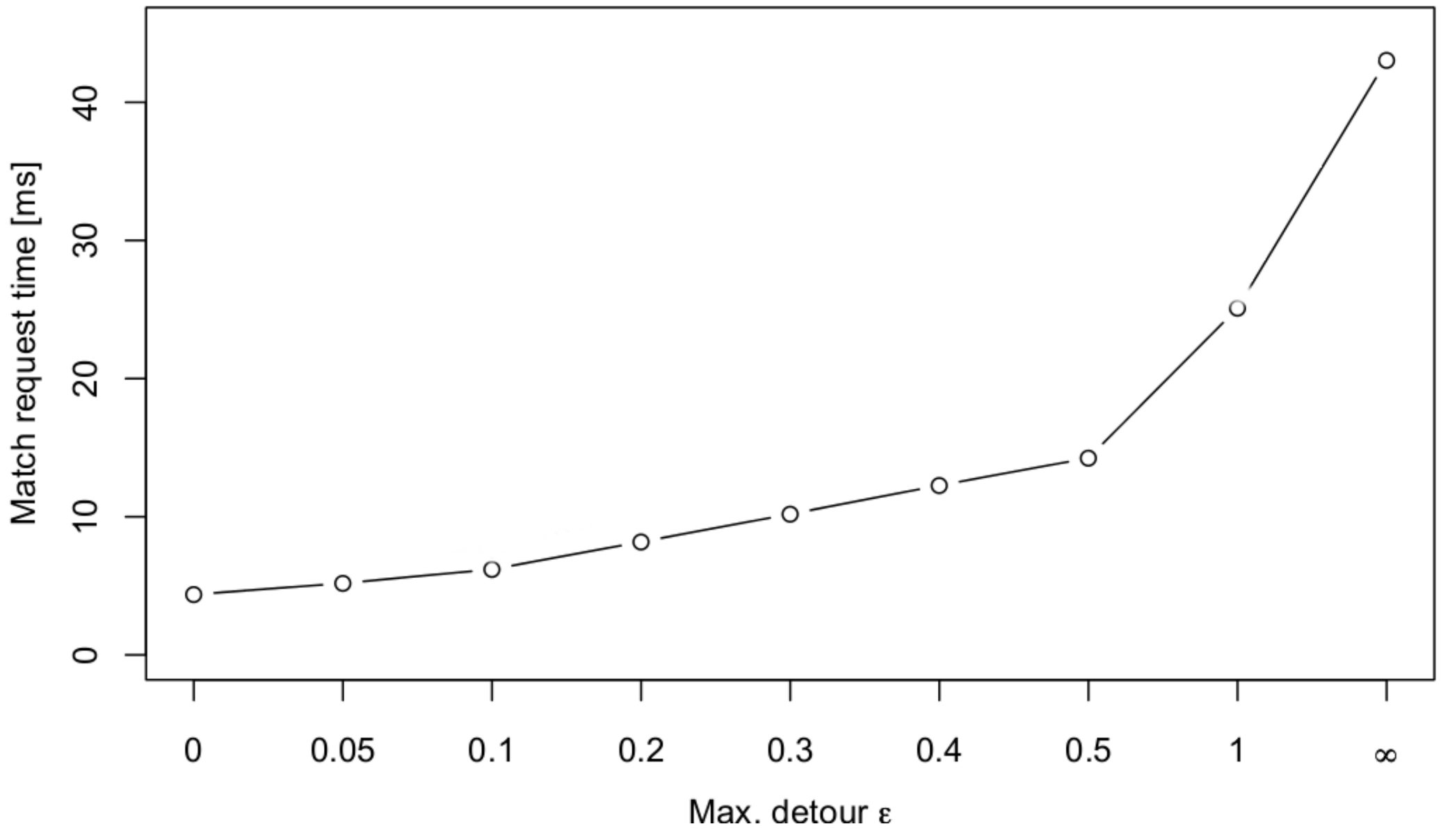
Ride Sharing - modeling



- search spaces for all offers
- matching on the fly

Ride Sharing - Performance

Matching against a set of 100,000 offers



„© OpenStreetMap Contributors“