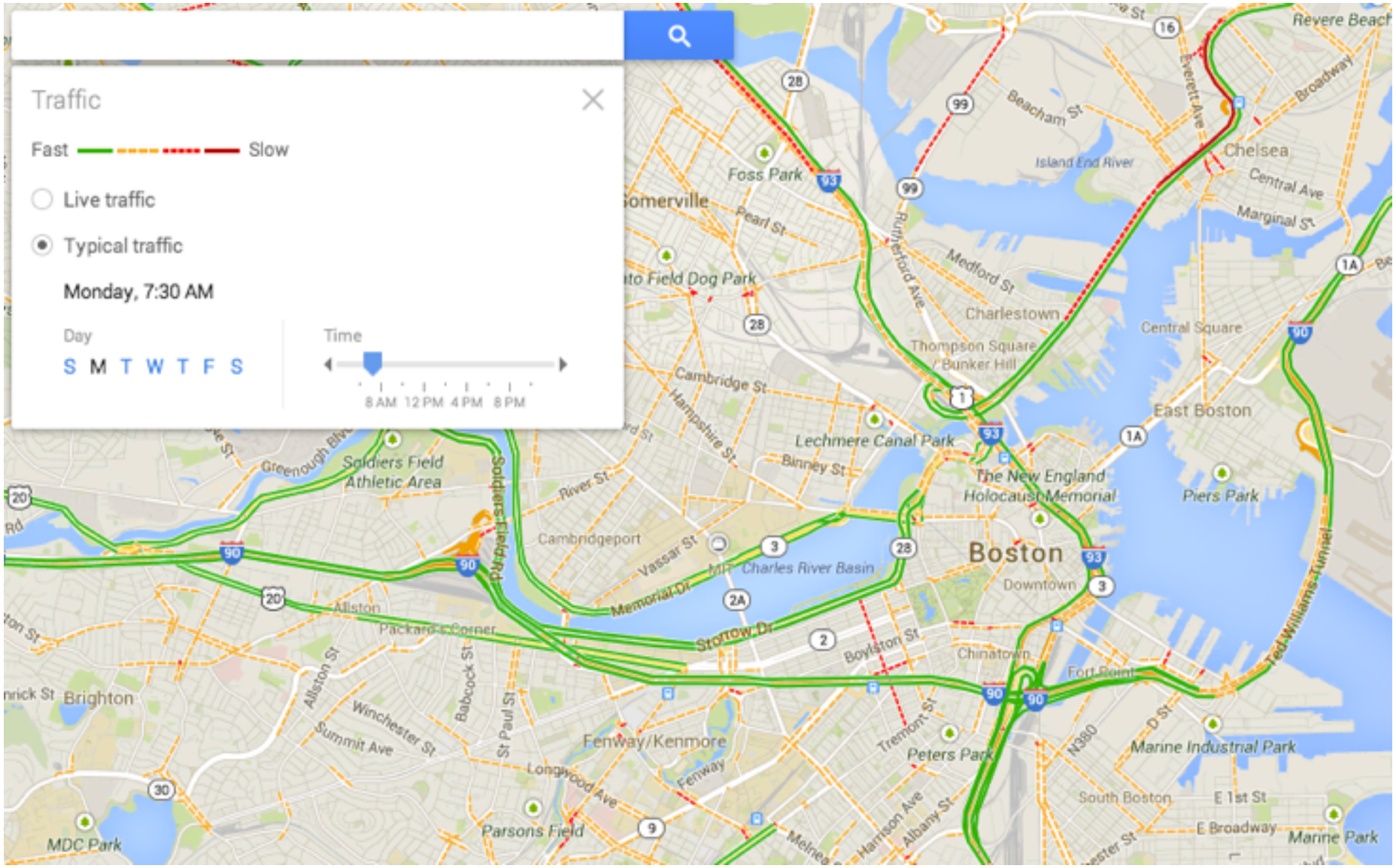
Mining massive geographic data

Jameson Toole & Yingxiang Yang
Human Mobility and Networks Lab
MIT

The question.

- How do you build a richer “Google Maps”?



With data!

- Call Detail Records (CDRs)
  - Every time you make a phone call, the network operator stores:
    - Location (either [lat,lon] or towerID)
    - Timestamp
    - Social Network
    - Duration
    - Transmission type (data, SMS, call, etc.)

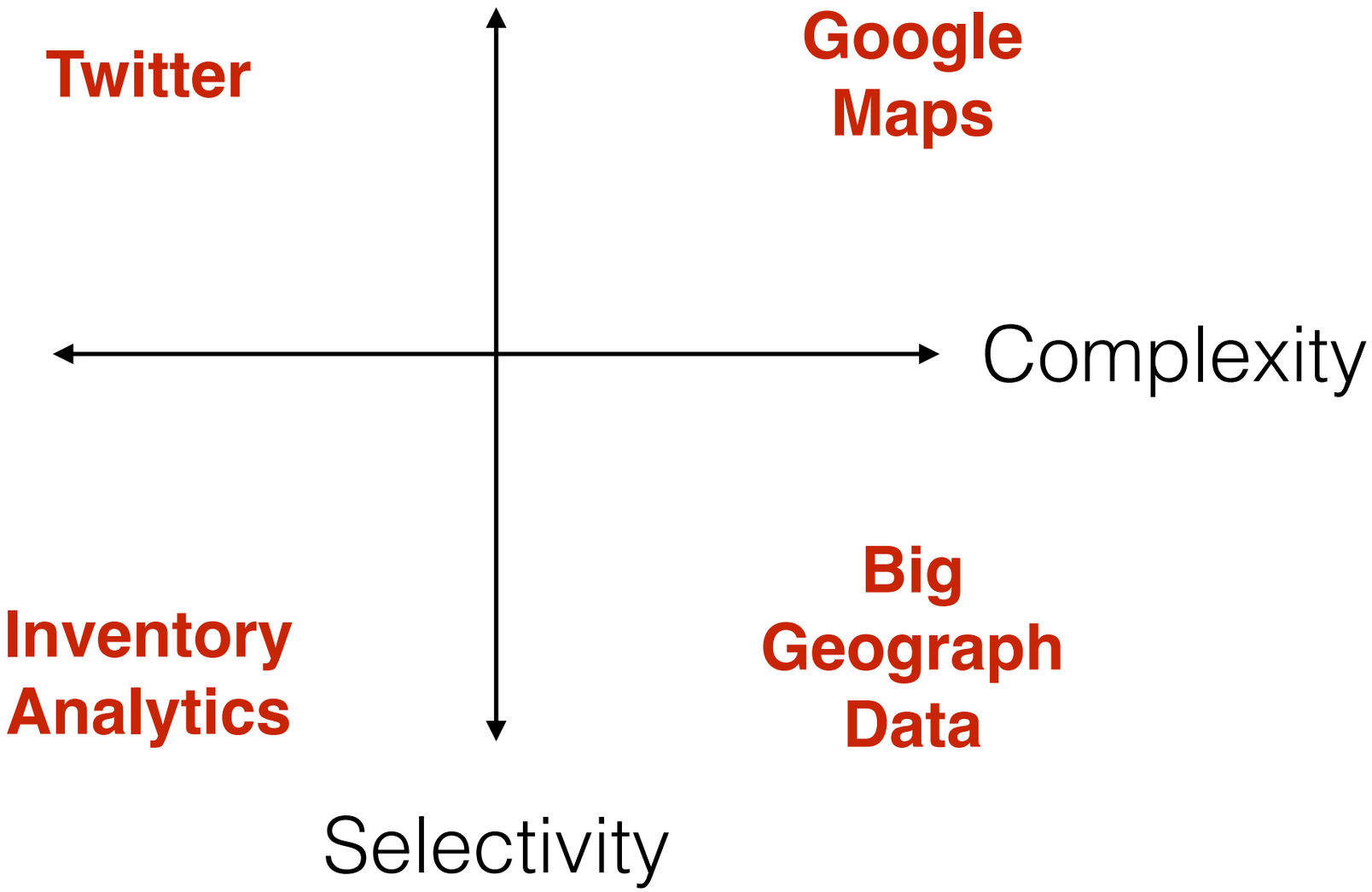
With data!

- Transportation Infrastructure
  - Road networks
  - Transit networks (subway, bus, etc.)
  - Sharing services (Hubway, ZipCar)
- Demographics
  - Census
  - Surveys

Now we need:

- A data pipeline to…
  - clear data 1TB+ of digital bread crumbs
  - transform and extract relevant features
  - merge multiple data sources (CDR + Census)
- Algorithms to…
  - find correlations
  - measure system behavior

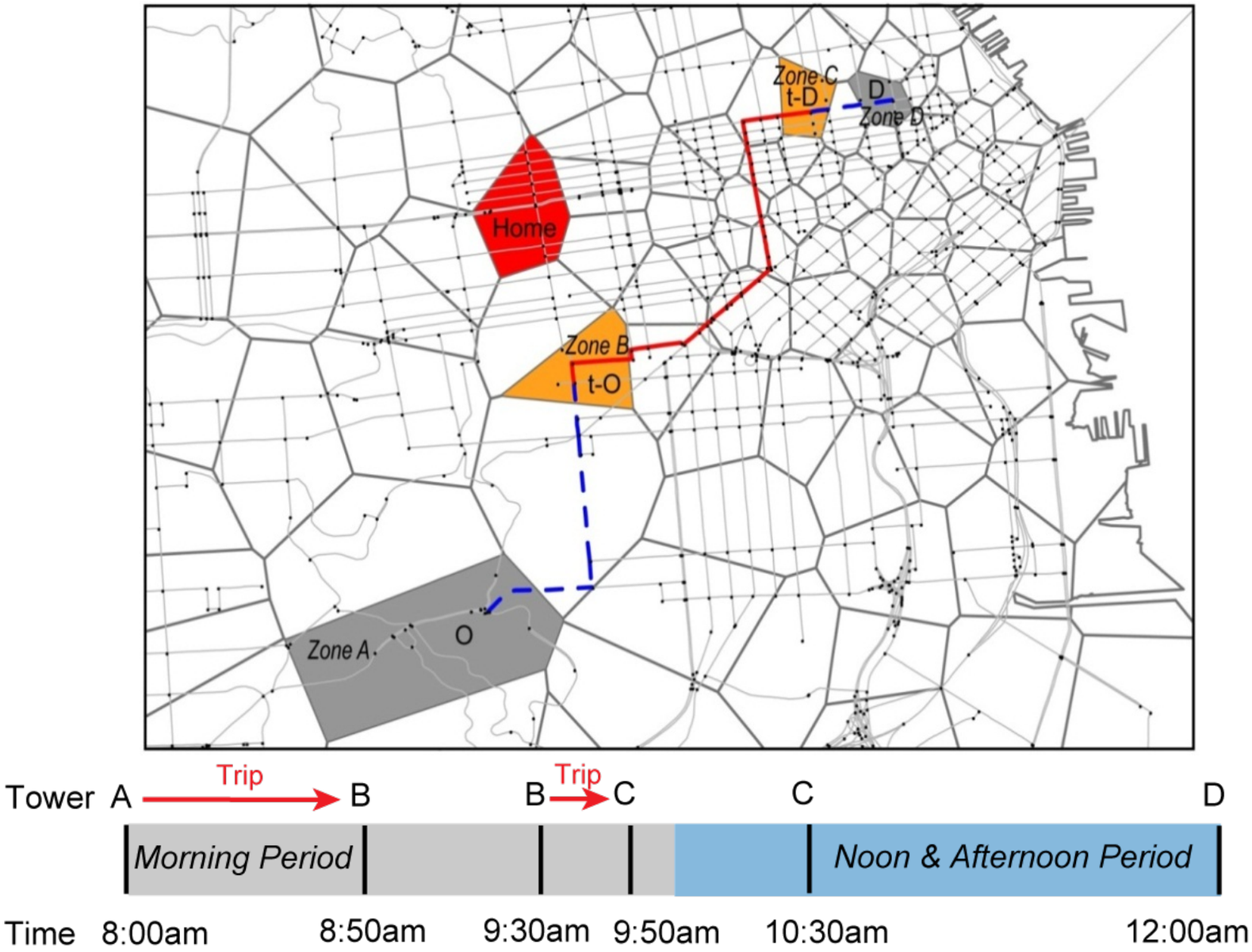
Use cases


Twitter
Google Maps
Complexity
Inventory Analytics
Big Geograph Data
Selectivity

What we want to do.

Massive, parallel routing.

Generate the OD Matrix



Understanding Road Usage Patterns in Urban Areas by *P. Wang et al.*, Scientific Reports, 2012

Generate the OD Matrix

OD Flows:

s, t, flow

0, 1, 20

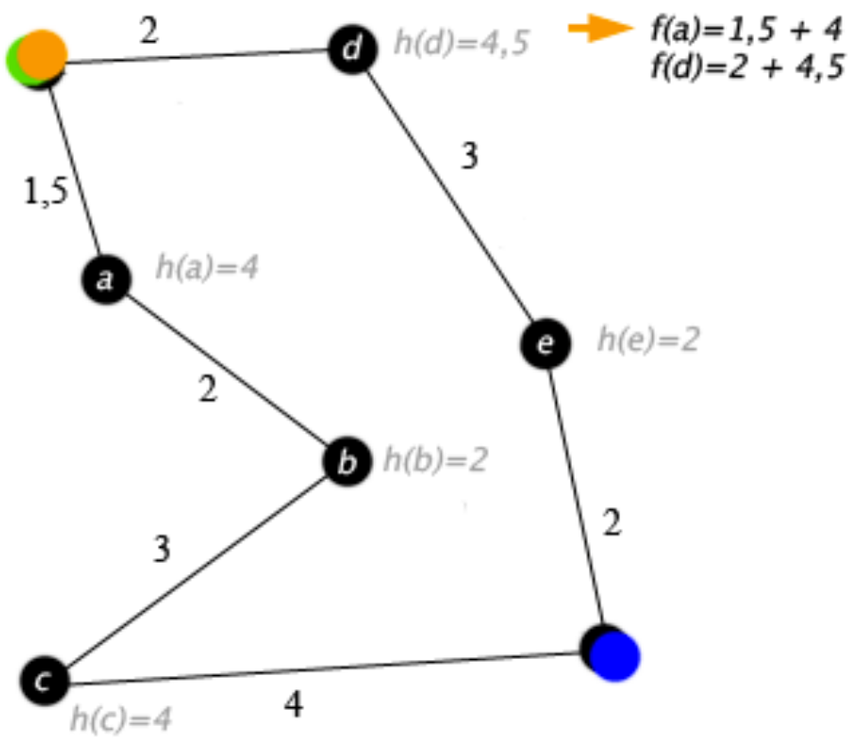
0, 2, 30

1, 2, 10

Route between source and target

A* algorithm

- Best-first search
- Heuristic cost function f(x) to guide search
- Known component, g(x)
- Estimated component h(x)
- Generalized Dijkstra's algorithm




```
function A*(start,goal)
    closedset := the empty set    // The set of nodes already evaluated.
    openset := {start}    // The set of tentative nodes to be evaluated,
    came_from := the empty map    // The map of navigated nodes.

    g_score[start] := 0    // Cost from start along best known path.
    // Estimated total cost from start to goal through y.
    f_score[start] := g_score[start] + heuristic_cost_estimate(start, goal)

    while openset is not empty
        current := the node in openset having the lowest f_score[] value
        if current = goal
            return reconstruct_path(came_from, goal)

        remove current from openset
        add current to closedset
        for each neighbor in neighbor_nodes(current)
            tentative_g_score := g_score[current] + dist_between(current,neighbor)
            tentative_f_score := tentative_g_score + heuristic_cost_estimate(neighbor, goal)
            if neighbor in closedset and tentative_f_score >= f_score[neighbor]
                continue

            if neighbor not in openset or tentative_f_score < f_score[neighbor]
                came_from[neighbor] := current
                g_score[neighbor] := tentative_g_score
                f_score[neighbor] := tentative_f_score
                if neighbor not in openset
                    add neighbor to openset

    return failure
```

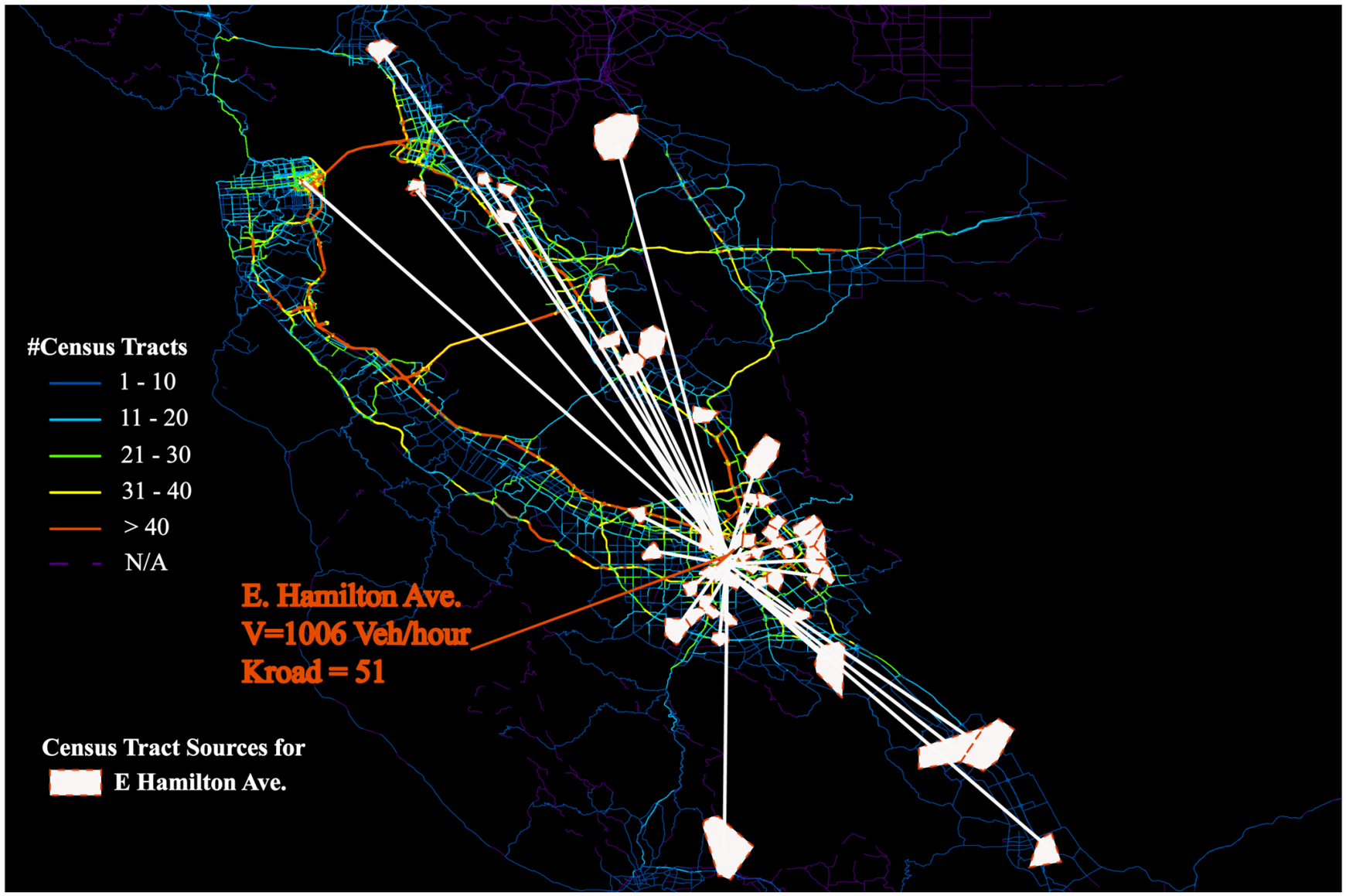
http://en.wikipedia.org/wiki/A*_search_algorithm

Incremental Traffic Assignment

ITA:

- Users aren't completely independent
- Externalities of travel mean that an individuals route choice affects the choices of others
- To account for this we divide our flows into increments:
  - Route the first 20% of users
  - Update costs on road segments factoring in how many users were assigned to a road
  - Route the next 20% of users with updated costs (paths may change)

What we want to do.



Understanding Road Usage Patterns in Urban Areas by *P. Wang et al.*, Scientific Reports, 2012

Which level to introduce parallelism?

- User centric
  - Because in many cases, the analysis for a single user is independent from the others, we can simply run the same algorithm for different users concurrently.
- Algorithmic
  - Write parallel algorithms that distribute the computation related to a single user or feature to multiple workers

Some options…

- A database management system:
  - Pros: Easy (standard) query language, transactions support concurrent use, easy to build an API or web application
  - Cons: Slow, hard to implement complex user defined functions
- A stand-alone software package
  - Pros: Fast, flexible
  - Cons: Difficult to share, opaque to system users

Database management system.

- Postgres + PostGIS + pgRouting
  - Open Source
  - Postgres is mature and reliable
  - PostGIS adds spatial features like indexing and complex joins
  - pgRouting has routing for spatial networks
- Parallelization Strategies
  - Partition data across many machines
  - Make many concurrent queries to the database and manage updates using transactions.

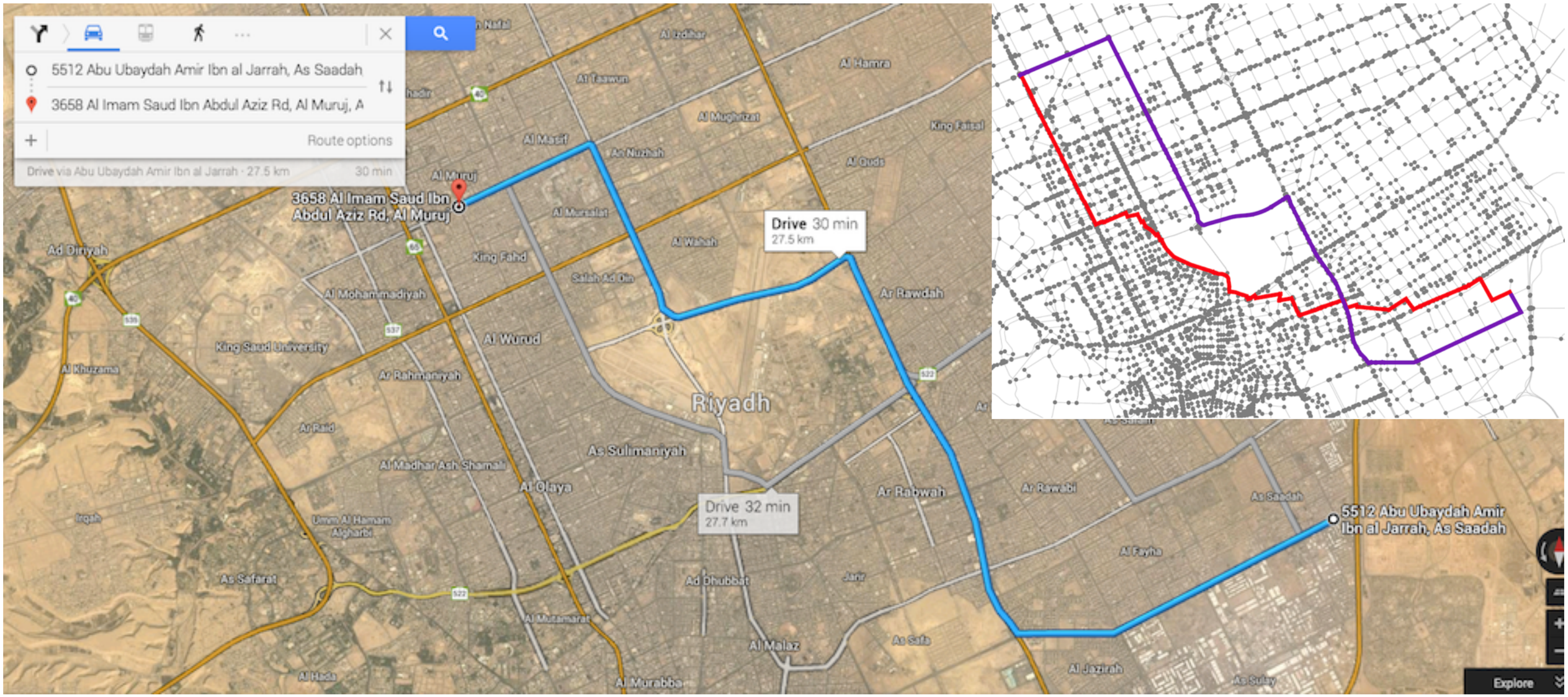
Database management system.

- Road Network
  - Schema:
    - edge_ID
    - source
    - target
    - cost
    - geometry
- System
  - Drivers connect python to the database.
  - Read a portion of the OD list and execute SQL Routing queries storing paths along the way.
  - Pause to aggregate counts per road segment and update costs

Database management system.

Database management system.

Google: **30min**
pgrouting (out of box): **22min**



Database management system.

The problem…it's SLOW!

.05 x 600,000 x 5 = 42 hours!

sec/route routes increments

Multithreaded C++ implementation

- System
  - Weighted-directed network class to store network
  - A class to store and split OD lists for parallel processing.
  - A multithreaded routine to execute multiple A* searchers concurrently.
  - Makes heavy use of the C++ Boost Graph Library
- **To the code!**

Multithreaded C++ implementation

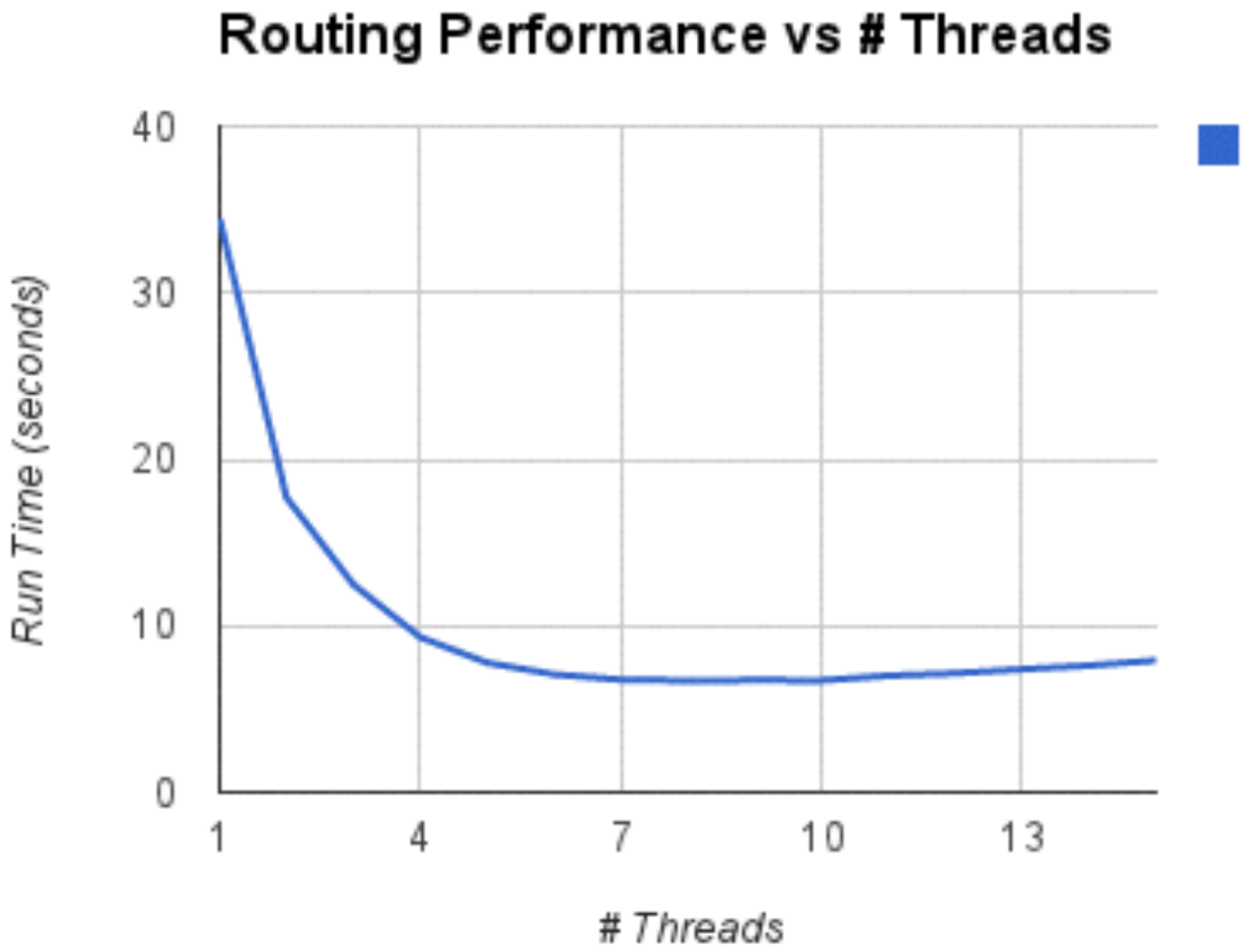
Now it’s over 6000x faster, but there is a lot more code!

.00005 x 600,000 x 5 = 150 seconds!

sec/route routes increments

Multithreaded C++ implementation

544,000 unique pairs. 22,000 edges. 10,000 nodes.



What do we do with all of these paths?

- Aggregate them on the roads and look for patterns!
- Visualize them! (must be from MIT IP)
- http://ec2-54-200-71-200.us-west-2.compute.amazonaws.com:7053/roads/