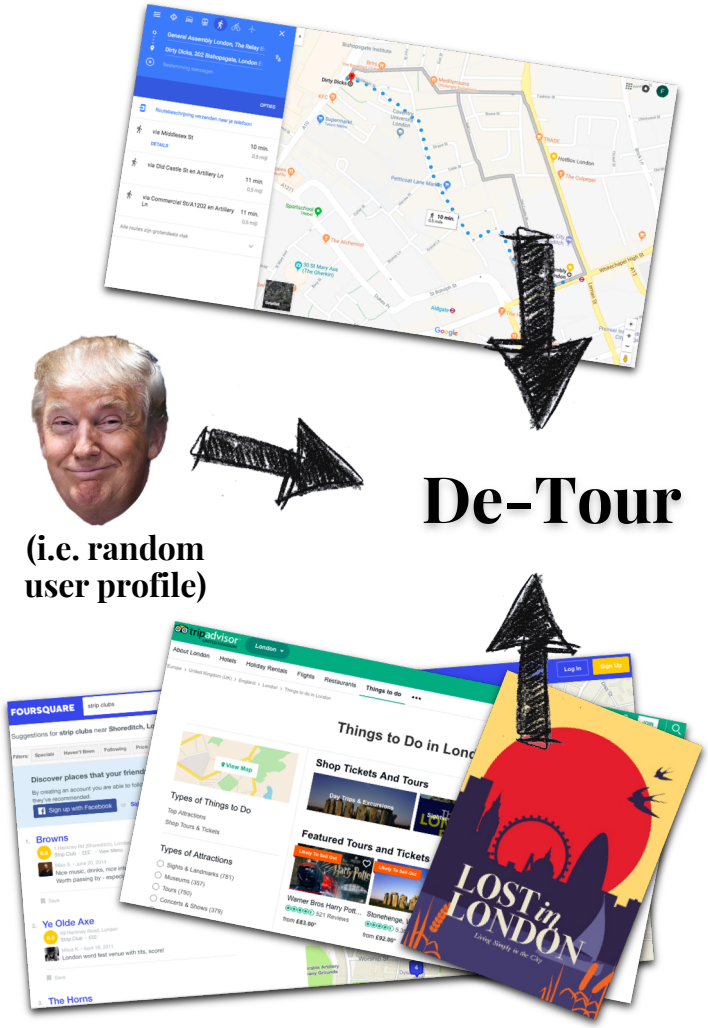
# De-Tour

## Your navigator through the streets and the web

Floris van Diest

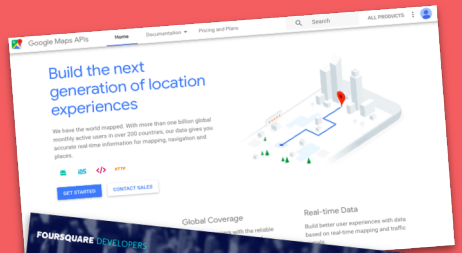


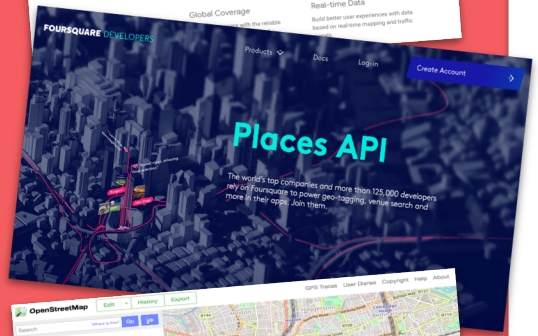

# De-Tour

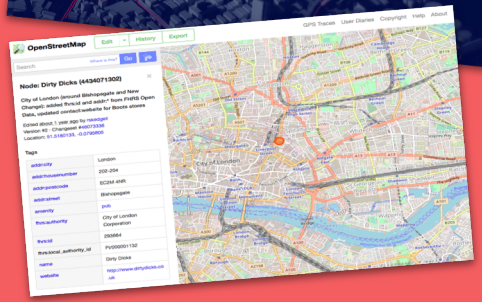
Feeling lost in the streets on your city trip and the piles of travel sites, blogs, reviews? Neurotically returning to the same places in your hometown?

De-Tour, a tailored walk planner tying your interests and preferences to the best the city has to offer!

**At least that was the idea...**

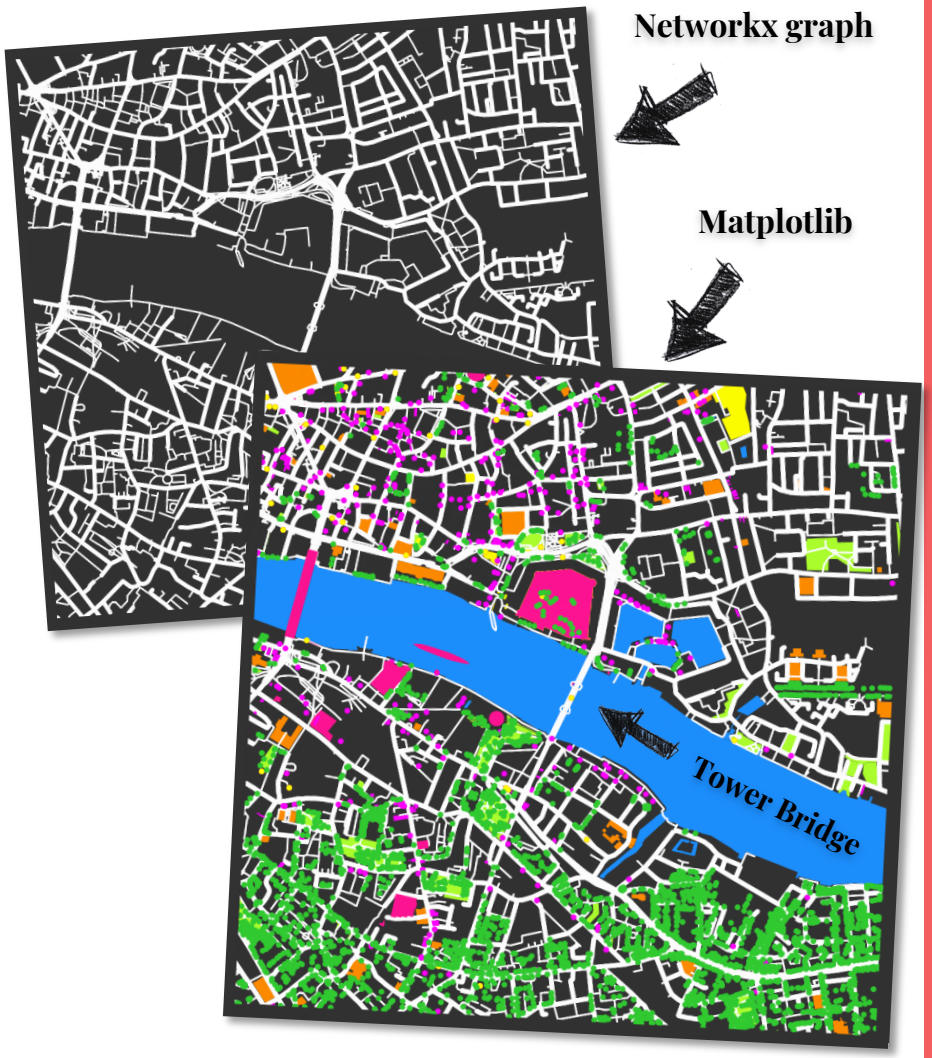



# The De-Tour process

1. Load map data
2. Make recommendations
3. Allocate map items to network
4. Define node and edge weighting models
5. Set up a ‘profit’ function
6. Optimize ‘profit’ function


# 1. Load map (data)

- **Open Street Map (Overpass API)**
  - Open data, community built free map database powering 1000+ websites and apps (incl. Tableau).
  - (Graph) Database of 64 GB (compressed size) XML files with global coverage of places, (road) networks, shapefiles, etc.
  - Unlimited requests a day.

1. **Define center of the map**
   - In this example Tower Bridge, London.
2. **Source the following map items**
   - Pedestrian street network (load as Networkx)
   - Parks, gardens and nature reserves ('Park')
   - Rivers, canals and lakes ('Water')
   - Historic attractions ('Historic')
   - Touristic attractions ('Touristic)
   - Bars & restaurants ('Gastronomy')
   - Trees ('Trees')

# 2. Make recommendations

### Foursquare Developers API

- Access to Foursquare database including 105+ million points-of-interest, globally.
- Venue (details) search, users, photos, etc.
- Max. 100,000 (free) requests a day.

### 1. Set up (Foursquare) user preferences for:

| | User preference (Floris) |
|---|---|
| **Coffee** | 'Coffee Shop' |
| **Lunch** | 'Bagel shop', 'Salad Place', 'Sandwich Place', 'Soup Place', 'Vegetarian / Vegan Restaurant' |
| **Bar** | 'Beer bar', 'Pub', 'Cocktail Bar', 'Speakeasy' |
| **Restaurant** | 'Vietnamese Restaurant', 'Indian Restaurant', 'Italian Restaurant', 'Middle Eastern Restaurant', 'Vegetarian / Vegan Restaurant |
| **Chains** | No |
| **Price tier** | 2 (out of 4; Cheap Dutch MF) |

### 2. Source venues in boundary box and make selection

- Filter venues on user preferences
- Find venues in different quadrants
- Per category, select venue with highest number of ratings in highest rating bucket (i.e. ratings rounded)


| | | Venue | Rating | No. Ratings |
|---|---|---|---|---|
| 1. | **Start** | Tower Bridge | | |
| 2. | **Coffee** | The Watch House | 8.9 | 74 |
| 3. | **Lunch** | EATalia | 8.0 | 46 |
| 4. | **Bar** | London Grind | 8.6 | 611 |
| 5. | **Restaurant** | Café Spice Namaste | 8.3 | 102 |

# 3. Allocate map items to network

To find density of interesting areas on the map, nodes underlying map items (e.g. historic, touristic, water, etc.) are allocated to closest nodes in the pedestrian graph network.

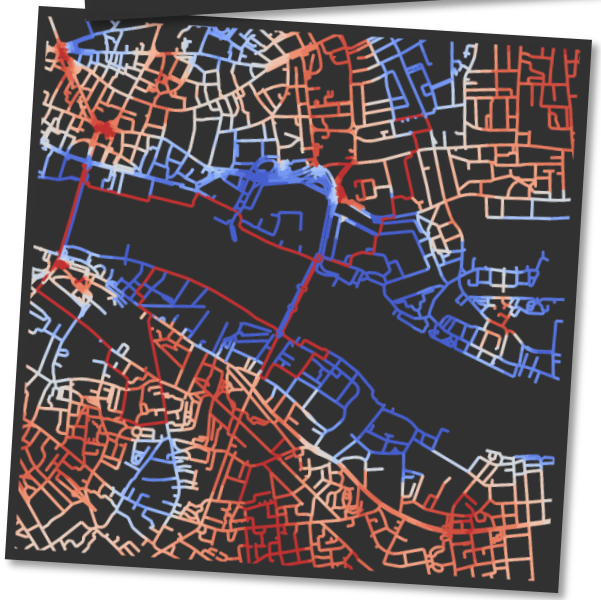
1. **Convert map items (patches and locations) to (unconnected) nodes**
2. **For every map item node, find the nearest node in the pedestrian graph network**
    - Using K-dimensional trees on the network nodes

1

2

# 4. Define node and edge weighting models

- Weights can be assigned to each **network node** as the weighted number of the number of map items allocated to this node per category.
- Subsequently, **network edges** can be weighted in accordance with the weights of the ($N$) neighboring nodes.
- Finally, these edge weights can be used to adjust the **edge lengths** and divert the route to nodes (and edges) with high weights.

1. **Set up node weighting model**
   - For each network node ($i$) assign a weight ($w_i$) as weighted (by **coefficient** $\beta_k$) number of map items allocated to node $i$ per category ($|M_{k,i}|$):

$$w_i = \sum_k [\beta_k \cdot |M_{k,i}|]$$

2. **Set up a edge weighting model ('Sink hole' model)**
   - For each network edge ($j$) assign a weight ($w_j^e$) as 'weighted' (again) sum of the weights of the $N$ neighboring nodes:

$$w_j^e = \sum_{n=1}^{N} [e^{-\alpha \cdot n} \cdot w_n]$$

3. **Adjust edge lengths with edge weights**
   - For each network edge ($j$) adjust the edge length ($l_j$) as follows:

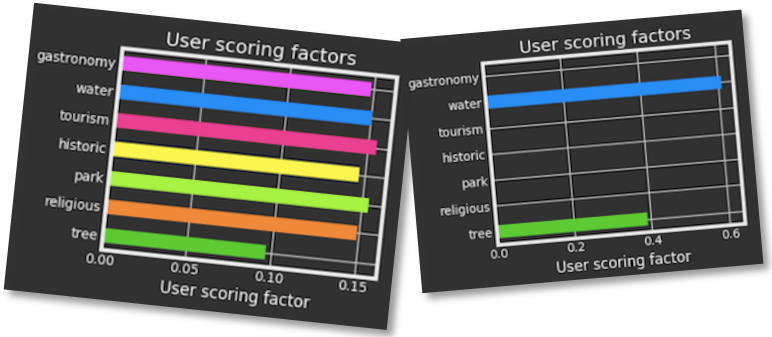
$$\widehat{l_j} = e^{-w_j^e} \cdot l_j$$

   - Note that if an edge is already included in the tour, I penalize the edge weight with $e^{+w_j^e}$ to avoid the navigator from taking the same paths

# 5. Set up a 'profit' function

The 'profit' (inverse of the loss) function is the optimizable function in search of the 'optimal' walking tour. The function should balance:

- Meeting the **target length** of the walking tour (e.g. 10 km)
- Passing by **interesting sites and items**




**1. Set up a function for the target length deviation**

- For each a given route $p$ with length $l_p$ and target length $L$, define the target deviation metric ($TD_p$):

$$TD_p = 1 - \left(\frac{L - l_p}{L}\right)^2$$

- $TD_p$ **varies between 0 and 1** and approaches 1 when deviation (+/-) from the target decreases.

**2. Set up a route score function**

- The **route 'score'** ($s_p$) is the sum of the map items it passes on the route ($M_{k,p}$) weighted by user scoring factors ($u_k$) per category:

$$s_p = \sum_k \left[u_k \cdot |M_{k,p}|\right]$$

- The **user scoring factors** express the user preference for different category and are normalized for the number of total occurrences of the category in the entire map.
- The **route score function** is defined as an improvement measure of the route score for a given route ($s_p$) and the route score of the shortest path ($s_b$)

$$\Delta s_p = 1 + e^{-\frac{s_b}{s_p}}$$

- $\Delta s_p$ **varies between 1 and 2** and approaches 2 when the route score improves over the baseline

**Profit function**

$$P_p = TD_p \cdot \Delta s_p$$

$P_p$ **varies between 0 and 1** and approaches 1 when the target deviation decreases and/or the route score improves

# 6. Optimize ‘profit’ function

## Iterative procedure

**In each iteration:**

1. **Shock each coefficient individually by increment ($\delta$)**
   Recalculate the route (networkx shortest path) for each (shocked) set of coefficients
2. **Combine shocks in the direction of the increase in the profit function**
   Recalculate the route for the new set of coefficients
3. **Compare new ‘profit’ with previous ‘profit’**
   - If lower, increase the size increment (larger shocks might be needed to ‘escape’ from local optimum)
   - If higher, reset size increment to base value
4. **Stop procedure if increment increases beyond threshold**
   None of the shock sizes improves the route further

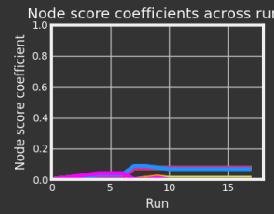

# An example

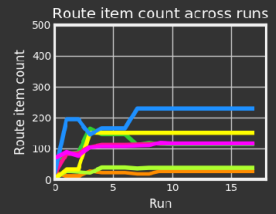
- No specific map item preference
- Target length 9,000 m
- Base $\delta = 0.01$
- Max $\delta = 10$

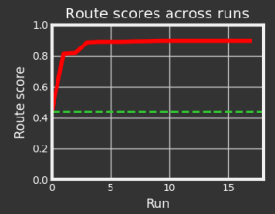
# Shorter route

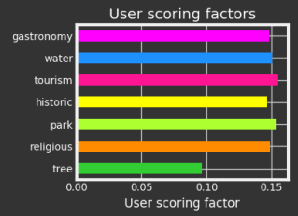
- No specific map item preference
- Target length **6,000 m**
- Base $\delta = 0.01$
- Max $\delta = 10$

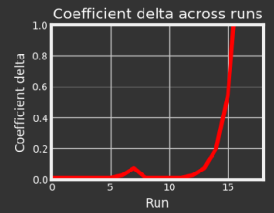


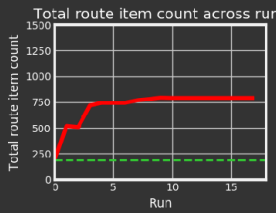



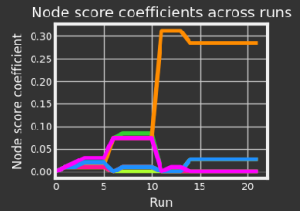



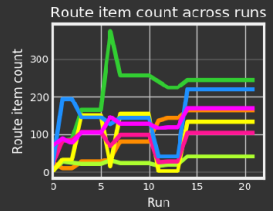



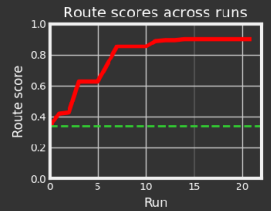



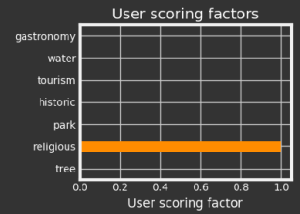


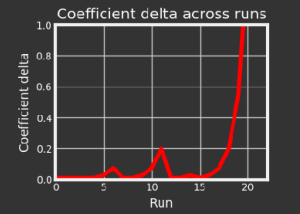


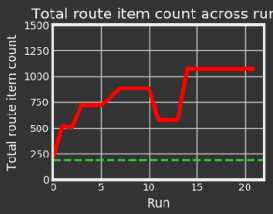


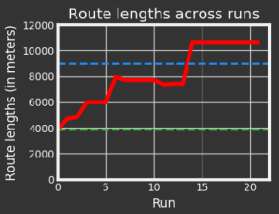



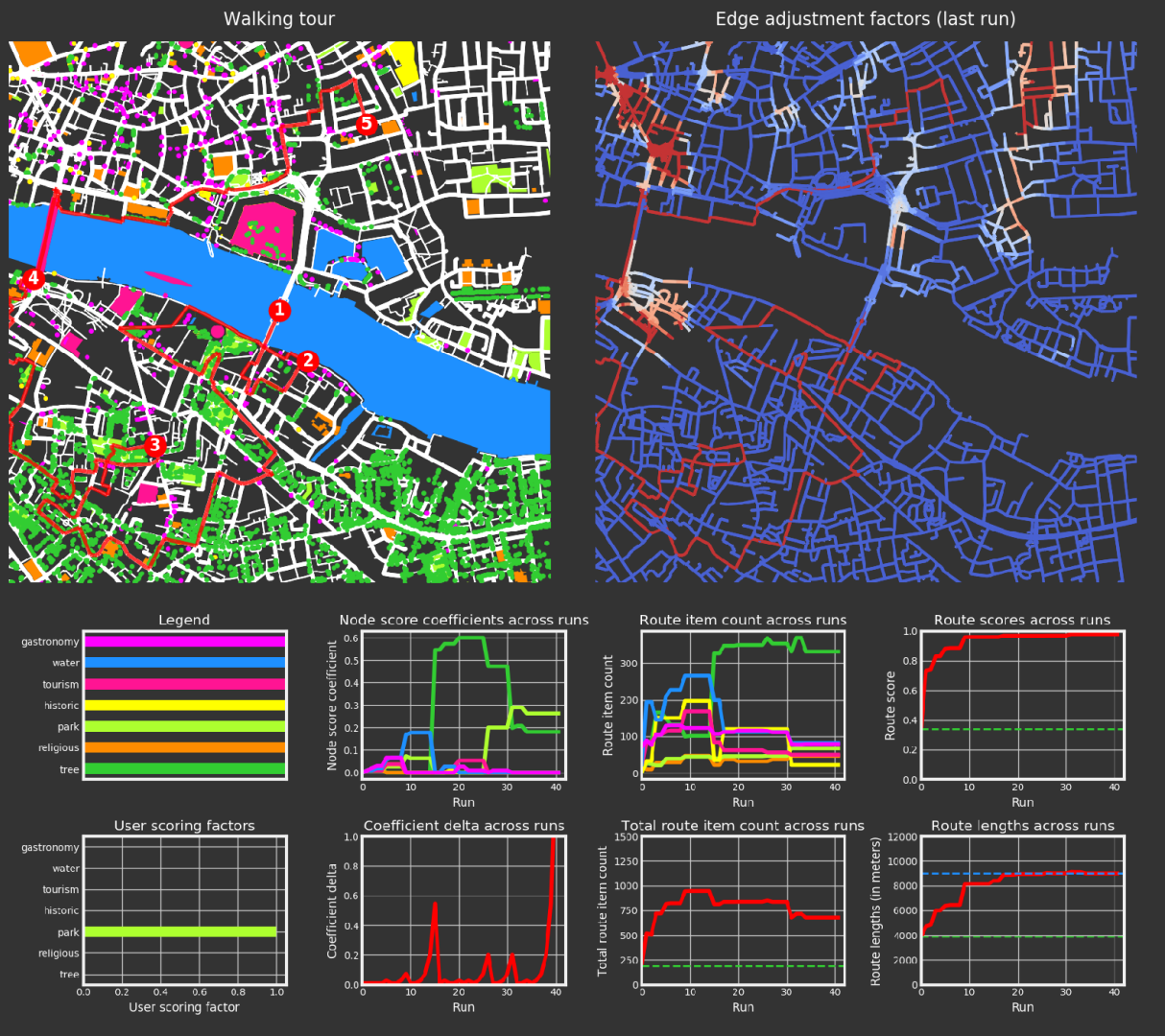

# Religous tour

- Only score on religious items
- Target length 9,000 m
- Base $\delta = 0.01$
- Max $\delta = 10$

.

# Park route

- Only score on parks
- Target length 9,000 m
- Base $\delta = 0.01$
- Max $\delta = 10$

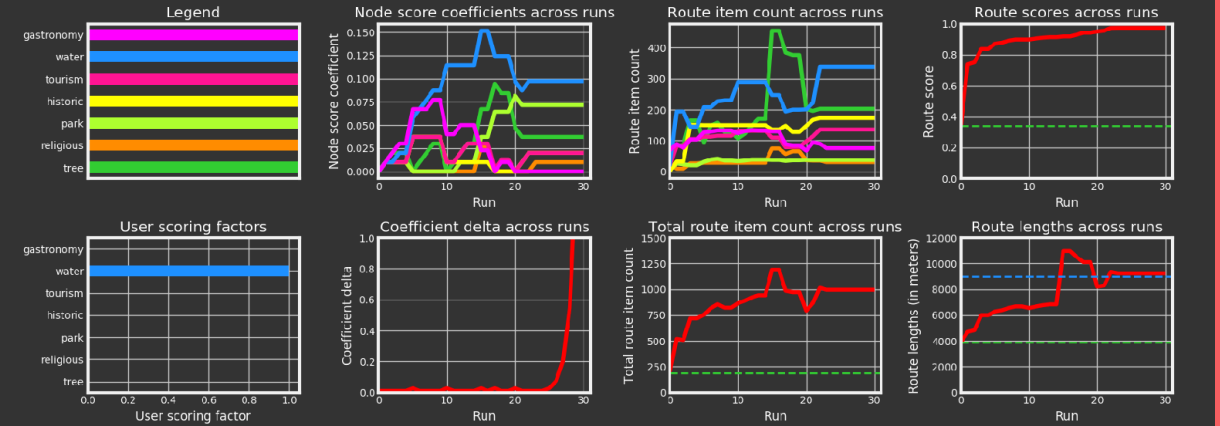


# Water tour

- Only score on water
- Target length 9,000 m
- Base $\delta = 0.01$
- Max $\delta = 10$

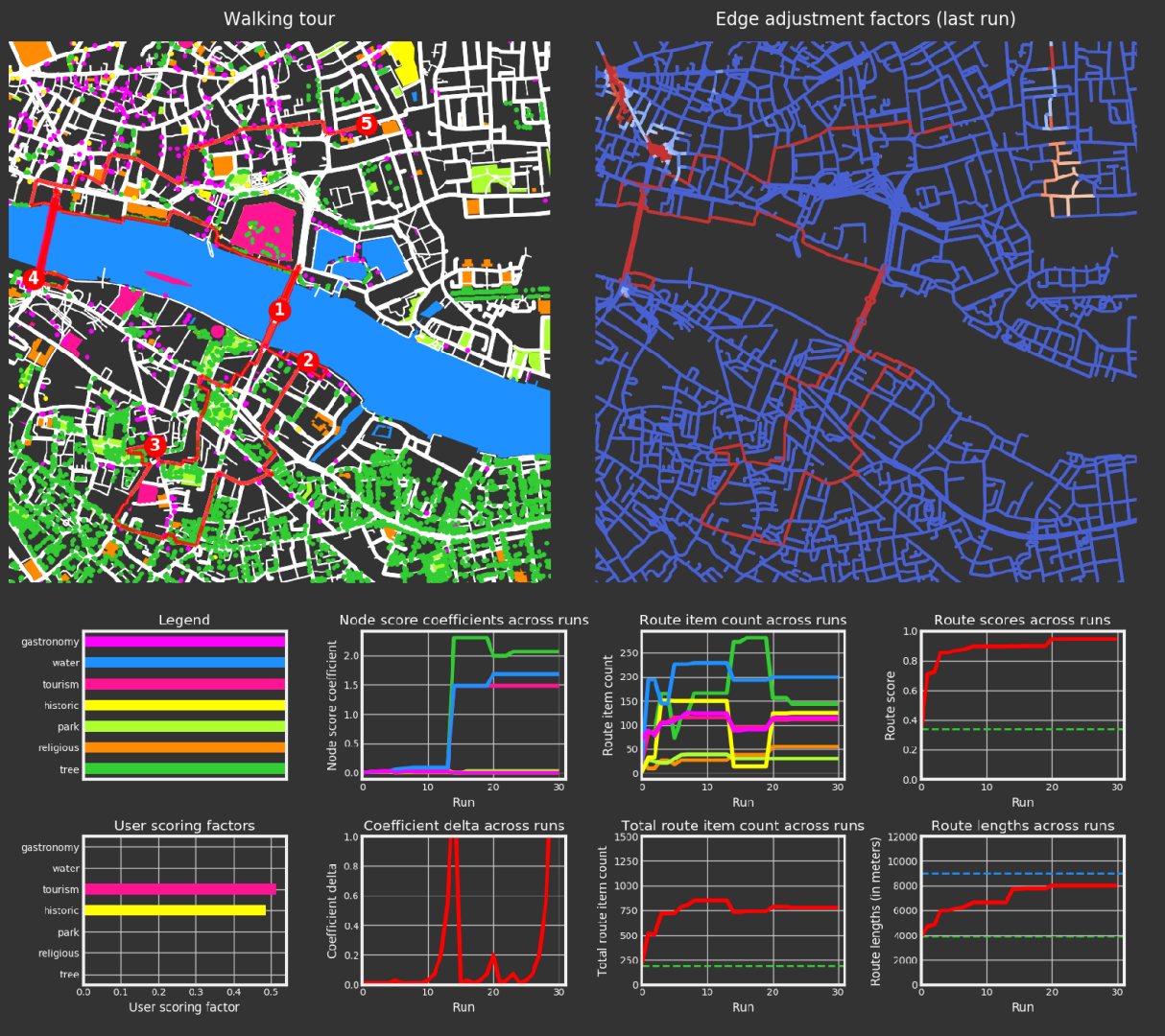
.

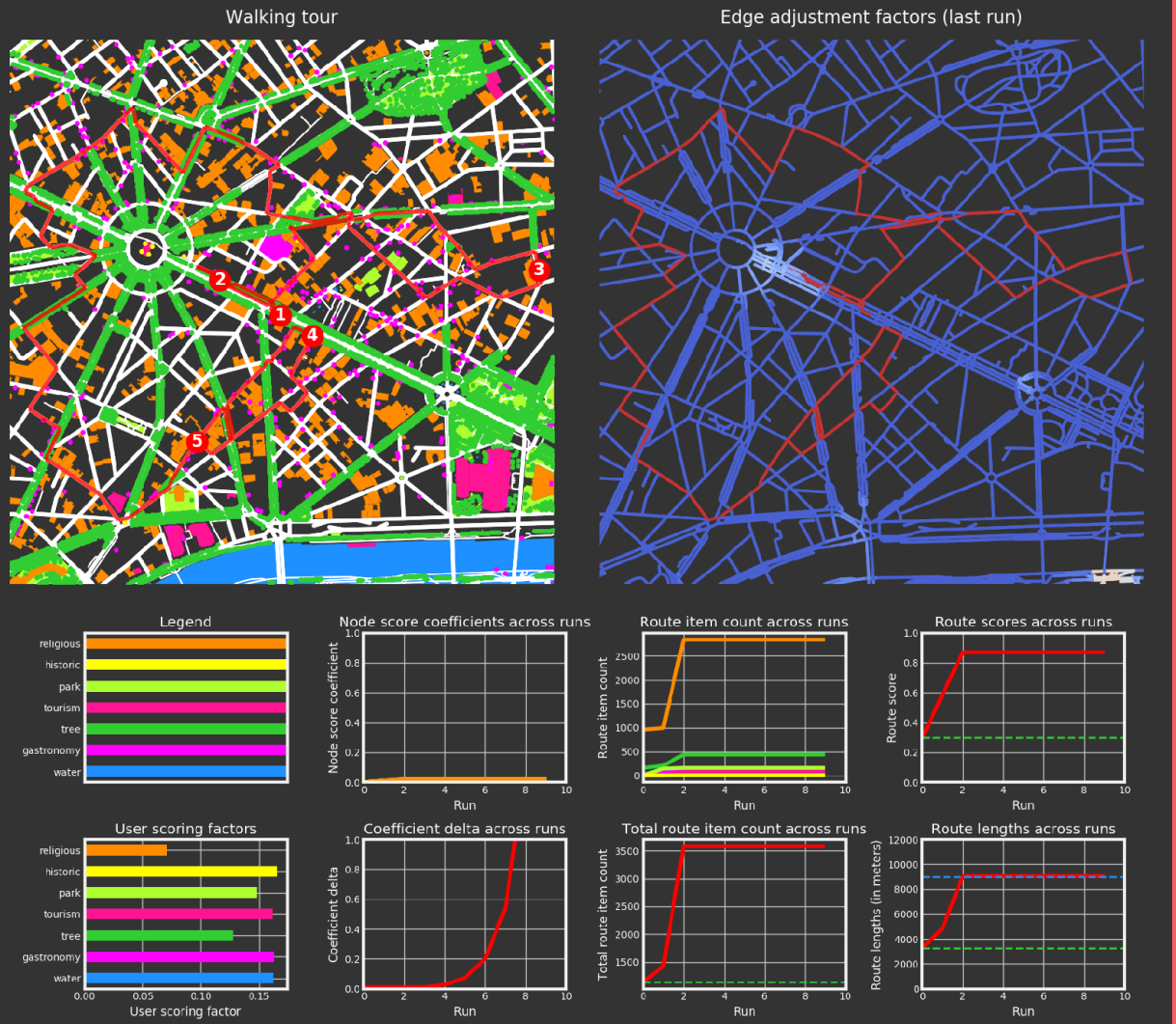




# Hist&Tour route

- Only score on historic and touristic items
- Target length 9,000 m
- Base $\delta = 0.01$
- Max $\delta = 10$





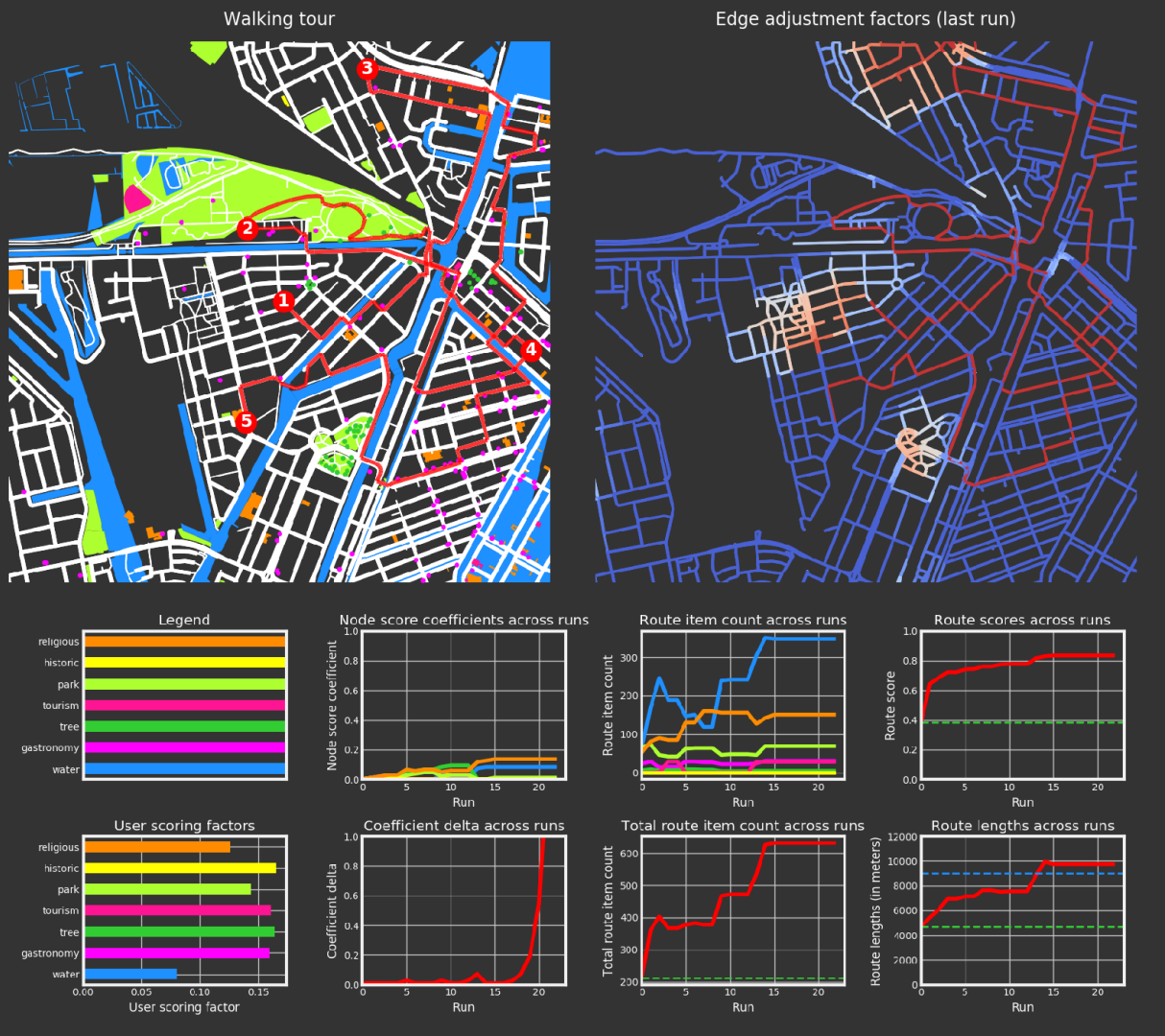

# Paris

- No specific map item preference
- Target length 9,000 m
- Base $\delta = 0.01$
- Max $\delta = 10$

.

# Amsterdam

- Only score on historic and touristic items
- Target length 9,000 m
- Base $\delta = 0.01$
- Max $\delta = 10$



# Future developments

- **Improve the optimization process**
  - Currently rather complex/not transparent methodology.
  - Lot of sequential steps (routing between checkpoints, shocking and combined route recalculation, increasing shocks to avoid local optima) hurts performance.
  - 'Profit' function depends on shortest path calculation (on changed edge weights) and is not an algebraic result from the edge weights. Preferred to have a more embedded solution (e.g. through Page Rank like algorithm).
- **Test performance in areas with little data**

  Methodology highly dependent on availability of OSM data.
- **Leverage more on ratings vs. counting**

  Optimization currently regards all items within a category as equal. It thereby might miss on not-to-be-missed sites.
- **Release it online through a web application**

  Develop a web application for users to try it out and provide feedback.
- **More features, more data, more ...**

  Endless opportunities to include more elements in the process.

# Do you have any questions?

Floris van Diest
f.van.diest@gmail.com
LinkedIn.com/in/floris-van-diest

**A big thanks to:**

- **Geoff Boeing** (Urban Planning post doc, Berkeley; for developing the OSMNX package)
- **Open Street Map community** (for tracking those trees)
- My mum