Assessment of an approximation method for TSP path length on road networks

Koen Stevens

May 17, 2025

Bachelor's Thesis Econometrics

Supervisor: dr. N.D. van Foreest

Second assessor: dr. W. Zhu

Assessment of an approximation method for TSP path length on road networks

Koen Stevens

Abstract

1 Introduction

The Traveling Salesman Problem (TSP) is an important problem in operations research. It is particularly relevant for last-mile carriers and other logistics companies where efficient routing directly impacts cost, time and service quality. Since the number of parcels worldwide has increased between 2013 and 2022 and is expected to keep increasing (Statista, 2025), the need for fast, scalable route planning methods becomes ever more pressing.

The TSP is an NP-hard problem, it is computationally intensive to find the exact solution for large instances. In many real-world scenarios, the exact optimal routes may not be needed, but instead a rough, reliable estimate of the optimal route length. For instance, consider a postal delivery company. This firm may need to assign a certain amount of deliveries or a certain area to each postman. Reliable estimates for the route length can provide valuable information for making such decisions.

Efficient approximation methods provide a solution for such practical applications where exact solutions are too computationally intensive to conduct or not feasible due to insufficient data. These methods aim at approximating the expected optimal total travel time or distance, while using minimal data and computational effort.

There is extensive research on such approximation methods and how they perform in the Euclidean plane. Consider n uniformly drawn locations from some area in $\mathbb{R}^2$ with area A. Beardwood, Halton, and Hammersley (1959) prove the relation:

$$L \to \beta\sqrt{nA}, \quad \text{as } n \to \infty \tag{1.1}$$

as an estimation for the length of the shortest TSP path measured by Euclidean distance through these random locations, where β is some proportionality constant. This formula is a very elegant result, and it requires very little data. However, its assumptions, uniform random locations and euclidean space differ from real-world applications, which are defined by complex geographic features, such as road networks.

This research investigates how well this approximation method performs when considering real road networks. Using OpenStreetMap (OpenStreetMap contributors, 2025) data, TSP instances are simulated in a wide variety of different urban areas in the Netherlands, then solve these for the actual shortest paths using the Lin–Kernighan heuristic (Lin and Kernighan, 1973). Then, the β from equation 1.1 is estimated and the performance of

this formula is analyzed. Additionally, the results for β and the performance across the selected areas is compared.

The core contributions of this research are:

1. An analysis of the BHH formula under more realistic conditions, specifically:
   (a) Relaxing the assumption of uniformly drawn locations. In this research, the locations are drawn from the set of postcodes in the area in question;
   (b) Applied to realistically sized real-world parts of cities and villages in the Netherlands;
2. A machine learning analysis to investigate how well the optimal TSP path length can be predicted, based on features of the area the path is in, including road network density, address density and other natural and man-made features of the area.

The analysis can easily be extended to any type of area in any part of the world, one would only have to download the OpenStreetMap (OpenStreetMap contributors, 2025) for another part of the world and add the names of the areas to apply it to. The source code of this project is available on GitHub.

In section 2 a deep dive in the context and previous research in this field is provided. In section 3 the experimental design is documented.

2 Literature Review

In this section the existing literature on the BHH formula and some applications, and on the Lin-Kernighan heuristic and its implementations is reviewed.

2.1 Applications of the BHH formula

This research concerns the performance of formula 1.1 for reasonable amounts of locations a delivery person can visit in a workday, say $10 \leq n \leq 90$. Lei, Laporte, Liu, and Zhang (2015) estimates the values of β for a selection of values for n. In their research, the points were generated uniformly and the L_2 distance metric was used. Table 1 lists the results.

Table 1: Empirical estimates of β as a function of n, $20 \leq n \leq 90$ (Lei et al., 2015)

n	$\beta(n)$
20	0.8584265
30	0.8269698
40	0.8129900
50	0.7994125
60	0.7908632
70	0.7817751
80	0.7775367
90	0.7773827

Figliozzi (2008) is the first research to apply approximation formulas to real-world instances of TSPs (and VRPs (Vehicle Routing Problems)). An extension of formula 1.1 that works for VRPs is assessed in a real-world setting. It is found that this model has an R^2 of 0.99 and MAPE (Mean Absolute Prediction Error) of 4.2%. This prediction error is slightly higher than when it is applied to a setting where Euclidean distances are considered (3.0%), but the formula still performs well (Figliozzi, 2008).

Merchán and Winkenbach (2019) use circuity factors to measure the relative detour incurred for traveling in a road network, compared to the Euclidean distance. This circuity factor is defined as, where p and q are locations:

$$c = \frac{d_c(p,q)}{d_{L_2}(p,q)} \tag{2.1}$$

By construction, c is greater or equal to 1, a value closer to 1 indicates a more efficient network. Then, β_c is estimated by $\beta_c = c\beta$. This value c, is estimated for three different areas in São Paulo, for which the results are listed in table 2. These values indicate real travel distances are on average 2.76 times longer in area 1 compared to the L_2 metric. These values were obtained by uniformly generating n locations (for n ranging from 3 to 250), computing near-optimal tour lengths under the Euclidean metric, and solving for β, then scaling by the empirical circuity factor. It is important to note,

Table 2: Estimates of the circuity factor c and its corresponding β_c (Merchán and Winkenbach, 2019)

	Area 1	Area 2	Area 3
c	2.76	2.34	1.82
β_c	2.48	2.10	1.64

however, that the assumptions in this study may limit the generality of the findings. In particular, the use of uniformly distributed locations does not accurately reflect the spatial distribution of delivery points in real urban environments, where locations tend to cluster in residential, commercial, or industrial zones. Additionally, within small urban areas, high-rise buildings and single-family homes may coexist in the same neighborhoods,

further challenging the assumption of uniformly distributed delivery points. Furthermore, the circuity factor c can vary significantly within a single city, depending on local street patterns, infrastructure, and topography. These variations suggest that a fixed circuity factor may oversimplify the complexity of real-world delivery contexts, especially when applied to smaller sub-regions or neighborhoods.

2.2 Lin-Kernighan Heuristic

To be able to efficiently solve many TSPs, to find a good estimate for β, a fast and reliable solution algorithm is needed. The Lin-Kernighan (Lin and Kernighan, 1973) heuristic provides outcome, it is generally considered to be one of the most effective methods of generating (near) optimal solutions for the TSP. In this research a modified implementation of the heuristic is used (Helsgaun, 2000). The run times of both heuristics increase by approximately $n^{2.2}$, but the modified heuristic is much more effective. It is able to find optimal solutions to large instances in reasonable times (Helsgaun, 2000).

PARAGRAPH ABOUT HOW THE HEURISTIC WORKS

3 Experimental design

In this section, a detailed explanation of the methodology is provided. This includes the characteristics of the data used, how this data is processed, as well as the approach taken to generate and solve TSP instances.

3.1 Data

In order to model the complex nature of real road networks, data from OpenStreetMap (OpenStreetMap contributors, 2025) is used. OpenStreetMap is an open-source project that provides geographic data, including accurate and detailed information about roads, buildings and natural features around the world. The data is continuously maintained and updated by a large community of users, making it a valuable resource for this research.

This data can be downloaded from Geofabrik, and then exported to a `PostgreSQL` database using `osm2pgsql` (Burgess and Contributors, 2025), in order to be able to efficiently use the data with `Python`. For this analysis, the database has three interesting tables: `planet_osm_polygon`, `planet_osm_nodes` and `planet_osm_ways`.

A large number of neighborhoods multiple towns and villages in the Netherlands have a polygon defined in the data. In OpenStreetMap a polygon is a closed shape formed by a set of geographic coordinates (`nodes`) that are connected by lines (`ways`). These objects can be used to define boundaries of geographic areas, such as lakes, parks, nature reserves and parts of cities and villages. In this research the polygons are used to filter the buildings and roads only in a certain area efficiently. These polygons are stored in

`planet_osm_polygon`.

In the database, the roads are defined as `ways`. These ways have three attributes: `id`, `nodes` and `tags`. The attribute `nodes` contains an ordered list of the nodes that this road contains of. In the `tags`, a large amount of information about the way is stored, for instance whether it is a one-way road, or the type of road that it is, i.e. primary, or trunk. The information about the roads that are needed for this analysis is the road id, the ids of the nodes the road consists of, the coordinates (Latitude, Longitude) of these nodes, and whether the road is a one-way road.

The buildings are stored as `nodes`. In this table (`planet_osm_nodes`), a large amount of other objects are stored as well. For this analysis, the potential delivery locations need to be extracted. Some of these nodes have a postcode defined, which can be used to extract all buildings that a potential delivery could take place. This way, for example a little shed in someones backyard is also filtered out, since this does not have its own postcode. For this research only the node id and the coordinates are needed.

3.2 Data processing

In listing 1 (Appendix), the query that is used to extract all roads in an area is listed. This query is used inside an `f-string` in `Python`, to be able to loop over the different areas and extract the roads from it. Note that a buffer of a few meters around the neighborhood is used for the filtering, since otherwise this results in edge cases, where a road is ever so slightly more to the outside of the area than the boundary, and it would get left out. `ST_Intersects` is used to extract all roads that are at least partly inside or on the boundary. A similar query is used to extract the nodes, but this is easier since for a building which is only defined by a single node, it is not needed to define a new geometry object.

One of the predictors of the TSP path length, is the area of the neighborhood the locations are drawn from. The OSM data provides the area of the polygons, but this value can not be used to predict TSP path length effectively. This value is a heavy overestimation of the correct value for A. In many cases, there are parts of the neighborhood that do not contain any buildings, for instance when there is a park in the neighborhood. To account for this, the value for A that is used, is the area of the convex hull around all buildings, which is calculated using the `shapely` module in `Python`. As an example visualization, figure 1 displays how the quarter for the inner city of Groningen is defined in OpenStreetMap. Using the area of this entire quarter would be an overestimation. A significant portion of this area is the canal and outer road around the inner city. There are many examples of quarters where this overestimation is even more significant than this.

Figure 1: The OpenStreetMap quarter for the inner city of Groningen (Binnenstad). (OpenStreetMap contributors, 2025)

Using the geographic information of the roads and buildings, a graph is constructed, using the `igraph` module in `Python`. This graph connects all buildings to each other over the road network. First, using the information about the roads and buildings the sets of nodes and edges need to be defined. An edge is a line that connects two edges to each other. Extracting the edges that connect the road network is straightforward, since all roads already have an ordered list of nodes defined. The subsequent nodes simply need to be stored as pairs, and all edges are defined.

When the road network is defined as a set of nodes and edges, the next step is to connect the buildings to the road network. This needs to be done manually, since no data is stored in OpenStreetMap about to which road the buildings belong. This algorithm needs to be very efficient, since many buildings are added in each area. An R-tree can be used to accomplish this efficiency. An R-tree is a dynamic index structure that is able to retrieve data items quickly according to their spatial locations (Guttman, 1984). Listing 2 displays the algorithm used to make the edges that connect the buildings to the road network. For each building node, the closest point on the closest road is found, using the shapely implementation of the R-tree, `STRtree`. Then, if this point is not an already

existing node, a new virtual node is added, which splits this existing edge in two parts. The road is reconnected with the new node in between. Finally, the building is connected to this new node.

In order to find the shortest path in terms of real distance, and to calculate the length of the TSP path, a 'weight' needs to be added. This weight represents the length of this edge in meters. The equirectangular approximation is used to calculate these weights. This approximation is very efficient, but only works when the points the distance is calculated between is small enough such that the rounding of the earth does not have a significant effect on the true distance. The nodes are all close enough together, so the effect of the rounding of earth's surface is negligible. Let A and B be two nodes, and (x_A, y_A), $(x_B), y_B)$ be their coordinates, in `radians`. Let $R = 6,371,000$ meters, Earth's radius. Then the distance between these nodes (the weight of the edge connecting them), using the equirectangular approximation is:

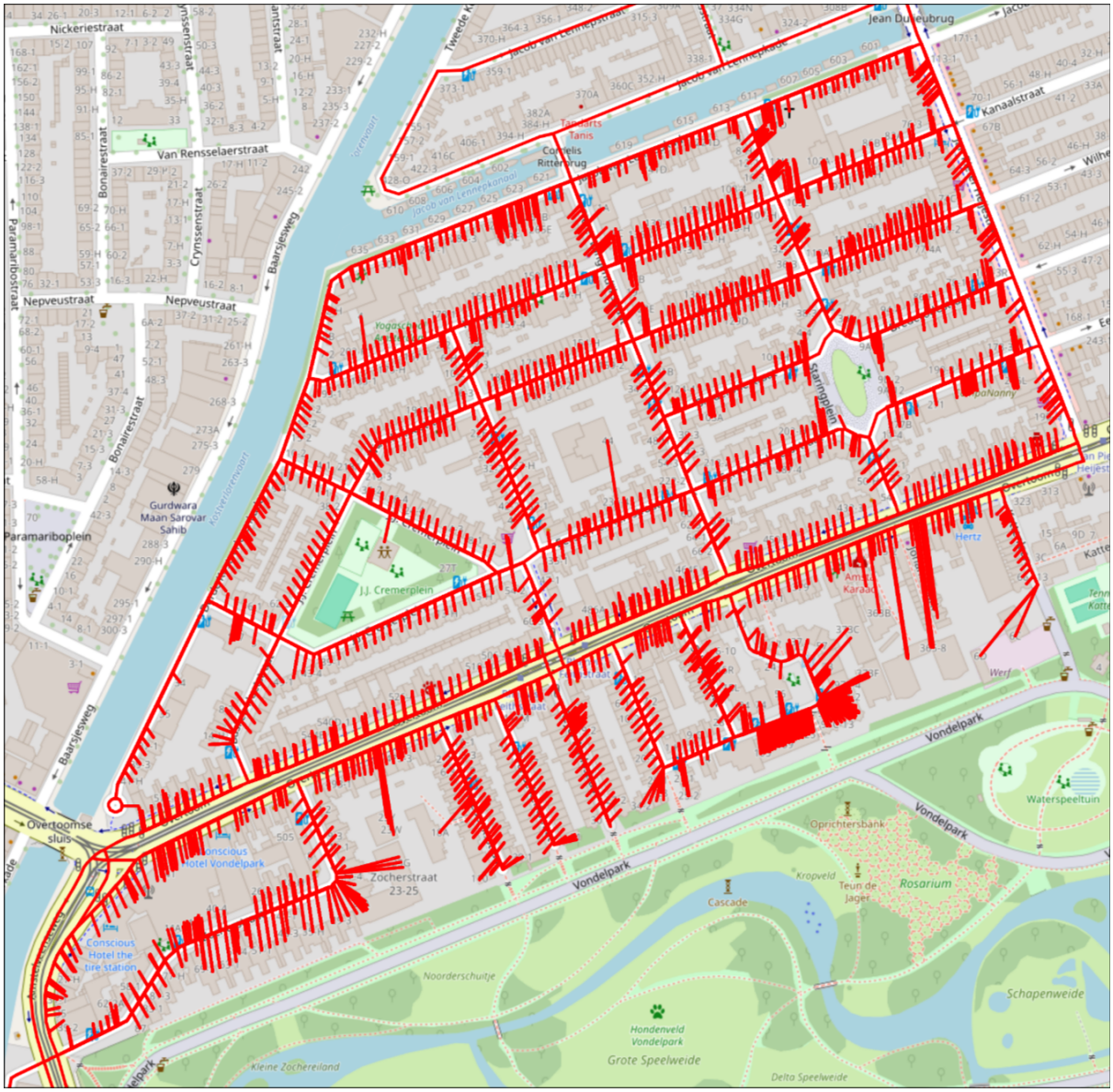
$$d(A, B) = R\sqrt{x^2 + y^2}, \tag{3.1}$$

$$\text{where } x = (x_B - x_A)\cos\left[\frac{y_A + y_B}{2}\right], \tag{3.2}$$

$$\text{and } y = y_B - y_A. \tag{3.3}$$

Using the `folium` module in `Python`, such a graph can be projected back onto the world map. One of such visualizations is provided in figure 2. All maps like this one, for the areas in this analysis can be viewed and interacted with via this `webpage`.

Figure 2: Visualization of the graph, for the Overtoomse Sluis quarter in Amsterdam.



Additionally, a number of features is required, to be able to perform the machine learning analysis. The OSM data is very complete, a large number of features can be extracted from there. For this analysis, natural features (i.e. water), leisure features (i.e. parks), landuse features (i.e. residential) and road network features (i.e. percentage of roads oneway) are used. In total, for each area information about 80 of these features is gathered.

3.3 Generating and solving TSPs

First, a uniform random sample is taken out of the list of buildings, of size $n \in \{20, 22, \cdots, 86, 88\}$. For each n, 100 samples are taken. Then, using the very neat builtin `igraph` function, `shortest_paths_dijkstra`, a distance matrix can be constructed over the graph in a very efficient manner. Using the sample of buildings and the

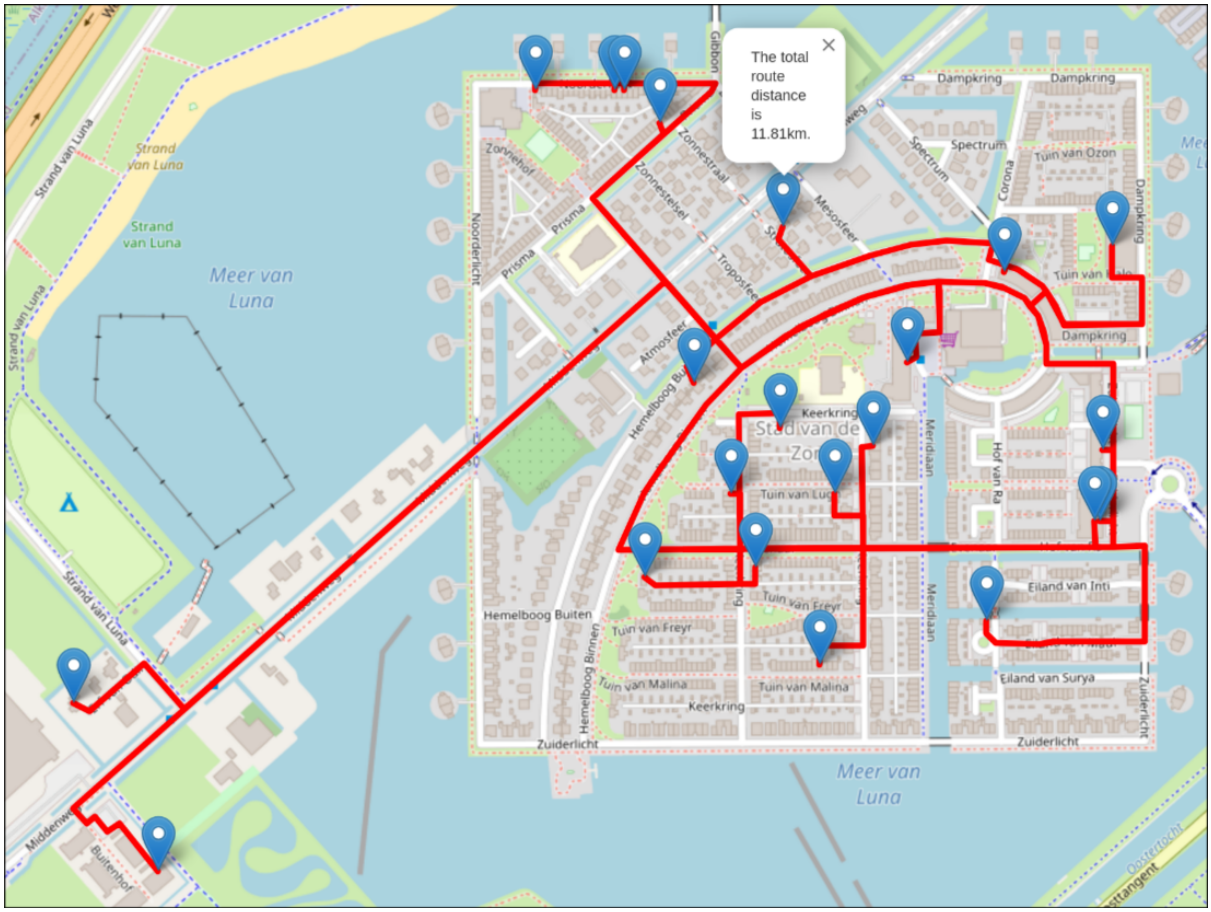
corresponding distance matrix, three files need to written per instance: a parameter file, a problem file and a `json` file.

The parameter file contains two lines, in this case: a path to the problem file and a path to the output tour file. This ensures that LKH solves the correct TSPs, and saves the solutions in the correct locations. The problem file contains the information about the problem, for instance n, and the distance matrix. Using these two inputs LKH can solve the TSPs. However with only these two files, the output tour can not be evaluated, since LKH starts indexing the visited locations from 1, and can not take different location ids as input. The `json` file saves a `Python` dictionary that maps these indexes back to the correct node indices, so the output can be read.

Using the `multiprocessing` module in `Python`, as many TSPs are solved at the same time as the number of threads in the processor that the project is ran on. LKH writes the solutions, with their respective path lengths to another file, that can then be read back into `Python`, to analyze the results. This process is repeated for a selection of 29 areas in the province of Groningen and for 52 areas in North Holland.

Again using the `folium` module, such a solution path can be visualized on the map. One of such paths is provided in figure 3. Only some of these paths are visualized, in order to check whether it looks like a reasonable solution, but it is not feasible, and it does not add value to visualize all TSP paths, since there is a total of 137,550 TSP instances in this analysis. Using the TSP solutions LKH provides and the estimated area of each neighborhood, formula 1.1 can be estimated.

Figure 3: Visualization of a solved TSP path with 22 locations, for the Stad van de Zon quarter in Heerhugowaard (North Holland).



3.4 Predicting TSP path length

Using the path length of the solved TSPs and the features of the area they are in, a dataset it constructed, of 137,550 TSPs with 80 features. A random forest model is used to predict TSP path length, using these features.

4 Results

5 Conclusion

6 References

Beardwood, Jillian, John H Halton, and John Michael Hammersley (1959). The shortest path through many points. In *Mathematical proceedings of the Cambridge philosophical society*, Volume 55, pp. 299–327. Cambridge University Press.

Burgess, Jon and Contributors (2025). osm2pgsql: OpenStreetMap data to PostgreSQL converter. Version 1.9.0, Accessed: 2025-04-25.

Figliozzi, Miguel Andres (2008). Planning approximations to the average length of vehicle routing problems with varying customer demands and routing constraints. *Transportation Research Record 2089*(1), 1–8.

Guttman, Antonin (1984). R-trees: A dynamic index structure for spatial searching. pp. 47–57.

Helsgaun, Keld (2000). An effective implementation of the lin–kernighan traveling salesman heuristic. *European journal of operational research 126*(1), 106–130.

Lei, H., G. Laporte, Y. Liu, and T. Zhang (2015). Dynamic design of sales territories. *Computers & Operations Research 56*, 84–92.

Lin, Shen and Brian W Kernighan (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations research 21*(2), 498–516.

Merchán, Daniel and Matthias Winkenbach (2019). An empirical validation and data-driven extension of continuum approximation approaches for urban route distances. *Networks 73*(4), 418–433.

OpenStreetMap contributors (2025). OpenStreetMap. Accessed: 2025-04-25.

Statista (2025). Global parcel shipping volume between 2013 and 2027 (in billion parcels)*.

7 Appendix

7.1 Listings

Listing 1: The query to extract roads inside a neighborhood.

```
-- First, get the neighborhood polygon, in the coordinate format we need.
WITH neighborhood AS (
    SELECT ST_Transform(way, 4326) AS geom
    FROM planet_osm_polygon
    WHERE place = 'quarter'
      AND name = '{neighborhood}'
),
-- Then, define the road geometries in a way that we can filter based
-- on whether they are inside the neighborhood.
road_geometries AS (
    SELECT
        w.id AS road_id,
        w.nodes AS node_ids,
        w.tags->>'oneway' AS oneway,
```

```
        ST_MakeLine(ARRAY(
            SELECT ST_SetSRID(
                ST_MakePoint(n.lon / 1e7, n.lat / 1e7), 4326
                )
            FROM unnest(w.nodes) WITH ORDINALITY AS u(node_id, ordinality)
            JOIN planet_osm_nodes n ON n.id = u.node_id
            ORDER BY u.ordinality
        )) AS road_geom
    FROM planet_osm_ways w
    -- Also filter based on the road type.
    WHERE w.tags->>'highway' IN (
        'trunk', 'rest_area', 'service', 'secondary_link',
        'services', 'tertiary', 'primary', 'secondary',
        'tertiary_link', 'road', 'motorway', 'motorway_link',
        'corridor', 'primary_link', 'residential', 'trunk_link',
        'living_street', 'unclassified', 'proposed'
    )
),
-- Filter on whether the roads are at least partly in the neighborhood.
filtered_roads AS (
    SELECT rg.*
    FROM road_geometries rg, neighborhood nb
    WHERE
        ST_Intersects(rg.road_geom, ST_Buffer(nb.geom, 0.0001))
)
-- Select the attributes that are needed.
SELECT
    fr.road_id,
    array_agg(n.id ORDER BY u.ordinality) AS node_ids,
    array_agg(n.lat / 1e7) AS node_lats,
    array_agg(n.lon / 1e7) AS node_lons,
    fr.oneway
FROM filtered_roads fr
JOIN planet_osm_ways w ON fr.road_id = w.id
JOIN LATERAL unnest(w.nodes)
    WITH ORDINALITY
    AS u(node_id, ordinality)
    ON true
JOIN planet_osm_nodes n ON n.id = u.node_id
GROUP BY fr.road_id, fr.oneway;
```

Listing 2: The algorithm to extract the edges to connect the buildings to the road network

```
tree = STRtree(road_segments) # make a tree of the road network

# Counter to add new nodes to connect buildings to road network correctly
new_node_idx = 0
for building_id, building_coords in buildings.items():
    building_point = Point(building_coords)
```

```
    # find the nearest edge
    nearest_segment_idx = tree.nearest(building_point)
    start_node, end_node, oneway = segment_info[nearest_segment_idx]

    # Project the building onto this nearest edge
    nearest_segment = road_segments[nearest_segment_idx]
    projected_point = nearest_segment.interpolate(
        nearest_segment.project(building_point)
    )

    # If the projected point is one of the segments end points, use this
    if projected_point.equals(Point(nodes[start_node])):
        connect_to = start_node
    elif projected_point.equals(Point(nodes[end_node])):
        connect_to = end_node
    # else, we need to create a virtual node
    else:
        virtual_node_id = f"virtual_{new_node_idx}"
        nodes[virtual_node_id] = (projected_point.x, projected_point.y)
        new_node_idx += 1

        # We split the road segment and add the virtual node
        edges.append((start_node, virtual_node_id))
        weights.append(
          distance(nodes[start_node], nodes[virtual_node_id])
          )
        edges.append((virtual_node_id, end_node))
        weights.append(
          distance(nodes[virtual_node_id], nodes[end_node])
          )

        if oneway != "yes": # if not one-way, add reverse
            edges.append((virtual_node_id, start_node))
            weights.append(
              distance(nodes[virtual_node_id], nodes[start_node])
              )
            edges.append((end_node, virtual_node_id))
            weights.append(
            distance(nodes[end_node], nodes[virtual_node_id])
            )

        # And we need to connect the building to the newly created node
        connect_to = virtual_node_id

    # Finally, we make the connections
    edges.append((str(building_id), connect_to))
    weights.append(distance(building_coords, nodes[connect_to]))
```

```
    edges.append((connect_to, str(building_id)))
    weights.append(distance(nodes[connect_to], building_coords))
```

7.2 Tables

Table 3: Empirical estimates for β, with prediction errors this beta gives for TSP path length in selected neighborhoods.

Province-Neighborhood	β	MAE (m)	MAPE (%)
groningen-Hortusbuurt	2.5284	727.0653	0.0769
groningen-Binnenstad	2.1407	945.7670	0.0668
groningen-Oosterpoort	2.2561	658.6700	0.0709
groningen-Rivierenbuurt	1.8219	565.7389	0.0702
groningen-De Wijert	1.6622	706.6520	0.0553
groningen-Oosterparkwijk	1.8281	959.3695	0.0618
groningen-De Hoogte	2.0862	879.5585	0.0851
groningen-Korrewegwijk	2.1646	1012.9386	0.0633
groningen-Schildersbuurt	2.3008	595.4837	0.0611
groningen-Paddepoel	1.6437	593.5363	0.0484
groningen-Oranjewijk	2.1890	858.1195	0.0739
groningen-Tuinwijk	3.8419	1161.7814	0.1459
groningen-Selwerd	1.6166	657.7040	0.0690
groningen-Vinkhuizen	1.4525	538.2826	0.0451
groningen-Hoogkerk-zuid	1.4815	759.6465	0.0689
groningen-Gravenburg	1.3771	1589.0864	0.1537
groningen-De Held	1.7796	399.1739	0.0428
groningen-Reitdiep	1.5787	444.5958	0.0439
groningen-Hoornse Meer	1.6191	635.5882	0.0674
groningen-Corpus den Hoorn	1.7618	865.1621	0.0753
groningen-Eemspoort	1.6039	535.7333	0.0497
groningen-Euvelgunne	1.8686	1052.7148	0.0976
groningen-Driebond	1.9411	1090.1420	0.1046
groningen-Winschoterdiep	2.2593	2261.2006	0.1532
groningen-Eemskanaal	1.7541	533.5686	0.0413
groningen-Helpman	2.1408	838.9278	0.0708
groningen-Lewenborg	2.0654	1237.3884	0.0631
groningen-Beijum	1.7453	741.9949	0.0443
groningen-Maarsveld	1.6674	391.2032	0.0499
drenthe-Assen Oost	1.4664	1200.6360	0.0585
drenthe-Assen West	1.1828	1877.6603	0.1094
drenthe-Centrum	1.5731	1242.5597	0.0698
drenthe-Kloosterveen	1.3101	2106.5607	0.0942
drenthe-Lariks	1.5640	984.1017	0.0732
drenthe-Marsdijk	1.4834	1813.0542	0.0771
drenthe-Noorderpark	1.6355	791.7199	0.0515
drenthe-Peelo	1.6629	929.8540	0.0685

Province-Neighborhood	β	MAE (m)	MAPE (%)
drenthe-Pittelo	1.7620	691.4879	0.0590
drenthe-Ter Borch	2.1360	785.8678	0.0551
friesland-Achter de Hoven	1.6660	497.5225	0.0769
friesland-Aldlân	1.9510	841.9472	0.0564
friesland-Bilgaard	1.8896	850.0155	0.0717
friesland-Binnenstad en Stationskwartier	1.9715	889.0453	0.0690
friesland-Blitsaerd	2.0280	594.8259	0.0531
friesland-Buitengebied Noordwest	1.4074	1716.9913	0.0768
friesland-Camminghaburen	1.6653	751.5120	0.0459
friesland-De Hemrik	1.4962	747.3613	0.0524
friesland-De Zuidlanden	1.1180	927.0308	0.0726
friesland-De Zwette	1.4970	1620.1902	0.0744
friesland-Heechterp	1.3685	790.5178	0.1224
friesland-Hempens, Teerns, en Zuiderburen	1.5409	1155.2527	0.0562
friesland-Huizum-Oost	1.4650	616.7353	0.0586
friesland-Huizum-West	1.6266	658.0781	0.0650
friesland-Middelsee	1.2457	833.2005	0.1017
friesland-Nijlân	1.4181	499.5333	0.0551
friesland-Oranjewijk	2.5773	468.7933	0.0622
friesland-Oud Oost	1.4975	716.6742	0.0546
friesland-Schepenbuurt	1.3704	1050.4308	0.1776
friesland-Schieringen en De Centrale	1.4875	822.4384	0.0982
friesland-Techum	1.7562	461.9544	0.0548
friesland-Vlietzone	1.7080	501.9054	0.0573
friesland-Vogelwijk en Componistenbuurt	1.8373	430.1407	0.0592
friesland-Vrijheidswijk	1.8264	469.3865	0.0521
friesland-Westeinde	1.7956	388.2131	0.0388
friesland-Wielenpôlle	1.5504	790.3753	0.1305
friesland-Zuiderburen	1.5318	1169.6306	0.0569
flevoland-Polderwijk	2.0397	1168.0344	0.0654
overijssel-Baalder	1.5088	633.7329	0.0631
overijssel-Baalderveld	1.5559	986.0537	0.0796
overijssel-Berflo Es	1.3169	1058.2400	0.0651
overijssel-Bergweide	1.6039	947.3912	0.0666
overijssel-De Graven Es	1.5472	1506.6178	0.0976
overijssel-De Meijbree	1.8145	334.3680	0.0514
overijssel-De Riet	1.5608	582.6573	0.0501
overijssel-De Thij	1.4553	865.6980	0.0656
overijssel-Groot Driene	1.6673	785.2460	0.0445
overijssel-Haardijk	2.1898	792.4527	0.0808
overijssel-Hanzeland	1.7341	749.5714	0.1052
overijssel-Hasseler Es	1.4518	964.2826	0.0549
overijssel-Heemsermars	1.0824	343.6100	0.0514
overijssel-Het Onderdijks	1.5631	310.5288	0.0378
overijssel-Hofkamp	1.4592	596.4896	0.0536
overijssel-Ittersum	1.5685	1219.5755	0.0635
overijssel-Keizerslanden	1.7004	1711.9010	0.0972

Province-Neighborhood	β	MAE (m)	MAPE (%)
overijssel-Kloosterlanden	1.3158	889.1588	0.0605
overijssel-Marslanden	2.3428	1676.9882	0.1027
overijssel-Nieuwe Haven	1.9547	731.3536	0.0546
overijssel-Nieuwstraatkwartier	1.6022	457.0575	0.0551
overijssel-Noorderkwartier	1.4499	535.0191	0.0481
overijssel-OosterDalfsen	2.1386	284.1659	0.0507
overijssel-Ossenkoppelerhoek	1.4826	685.3150	0.0548
overijssel-Rivierenwijk	1.6356	456.5307	0.0495
overijssel-Schelfhorst	1.5826	771.2058	0.0473
overijssel-Slangenbeek	1.3124	1047.3283	0.0533
overijssel-Sluitersveld	1.6373	851.9876	0.0597
overijssel-Twekkelerveld	1.4996	651.0840	0.0469
overijssel-Veerallee	1.8406	667.4959	0.0916
overijssel-Voorstad	1.1573	982.9321	0.0766
overijssel-Wierdensehoek	1.4463	797.5979	0.0593
overijssel-Windmolenbroek	1.4742	1362.2382	0.0587
overijssel-Zandweerd	1.4247	594.2441	0.0482
noord holland-Schrijverswijk	1.6992	439.2479	0.0477
noord holland-Stad van de Zon	1.2944	2374.6580	0.1849
noord holland-Stadshart	2.0086	501.7207	0.0516
noord holland-Jordaan	2.4697	1048.5404	0.0602
noord holland-Slotervaart	1.7308	687.6353	0.0490
noord holland-IJburg	1.3251	893.8233	0.0499
noord holland-Oostelijke Eilanden	1.6705	479.3620	0.0463
noord holland-Oostelijk Havengebied	1.7287	1115.7897	0.0534
noord holland-Frederik Hendrikbuurt	2.7322	908.9105	0.0823
noord holland-Van Lennepbuurt	3.3129	774.7296	0.0669
noord holland-Da Costabuurt	3.2245	1005.9659	0.0973
noord holland-Kinkerbuurt	3.1734	929.8317	0.0905
noord holland-Kersenboogerd	1.5912	1072.6652	0.0532
noord holland-Pax	2.0977	680.0580	0.0522
noord holland-Graan voor Visch	2.1891	520.1579	0.0567
noord holland-Vrijschot-Noord	2.6726	580.1329	0.0702
noord holland-Toolenburg	1.6414	1027.0372	0.0467
noord holland-Floriande	1.8543	1127.2496	0.0455
noord holland-Overbos	1.7422	678.0740	0.0462
noord holland-Bornholm	1.7596	548.3246	0.0406
noord holland-Beukenhorst-Oost	1.8570	930.4855	0.0773
noord holland-De Hoek	2.4540	679.2393	0.0530
noord holland-West	1.8634	301.6956	0.0490
noord holland-Zuid	2.2108	874.1950	0.0606
noord holland-Oost	1.8592	763.2687	0.0567
noord holland-Noord	1.6509	599.4562	0.0546
noord holland-De President	1.4466	979.7524	0.1218
noord holland-Graan voor Visch-Zuid	1.9357	590.9427	0.0655
noord holland-Zuidwijk	1.9350	432.3491	0.0432
noord holland-Buitenveldert-West	1.1466	1229.5719	0.0722

Province-Neighborhood	β	MAE (m)	MAPE (%)
noord holland-Buitenveldert	1.1064	1334.8021	0.0700
noord holland-Apollobuurt	1.8064	867.3723	0.0733
noord holland-Stadionbuurt	1.5822	820.9004	0.0735
noord holland-Prinses Irenebuurt e.o.	2.0185	462.0289	0.0644
noord holland-Hoofddorppleinbuurt	1.9547	1167.4108	0.0902
noord holland-Willemspark	2.3694	937.6364	0.0873
noord holland-Schinkelbuurt	3.2406	1231.6013	0.1177
noord holland-Vondelparkbuurt	3.1934	738.3031	0.0801
noord holland-Helmersbuurt	2.9906	878.2293	0.0780
noord holland-Overtoomse Sluis	2.9313	752.8070	0.0715
noord holland-Museumkwartier	1.7507	1082.9057	0.0770
noord holland-Rivierenbuurt	1.8522	1147.1064	0.0653
noord holland-IJselbuurt	3.6677	947.2657	0.0825
noord holland-Scheldebuurt	1.8645	970.3637	0.0693
noord holland-Rijnbuurt	1.9077	534.4458	0.0579
noord holland-De Baarsjes	2.2831	1292.5096	0.0662
noord holland-Landlust	1.9626	1006.6387	0.0730
noord holland-Staatsliedenbuurt	1.9201	705.2967	0.0727
noord holland-Spaarndammerbuurt	2.6024	793.8784	0.0725
noord holland-De Pijp	2.2733	1364.8035	0.0700
noord holland-Grachtengordel	2.1467	1385.3247	0.0704
noord holland-Oud-Zuid	1.4205	1421.3800	0.0585
utrecht-Bedrijventerrein Vathorst	1.7908	1021.0497	0.0663
utrecht-Binnenstad City- En Winkelgebied	1.1806	620.1733	0.0660
utrecht-Binnenstad Woongebied	2.0283	1119.9648	0.0598
utrecht-Bosgebied	0.9463	1334.9264	0.0667
utrecht-Buitengebied Oost	1.0068	1497.3125	0.0758
utrecht-Calveen	2.0332	886.3555	0.0910
utrecht-De Berg-Noord	1.6805	740.8549	0.0538
utrecht-De Berg-Zuid	1.5851	824.6435	0.0598
utrecht-De Hoef	1.6625	1123.0882	0.0950
utrecht-De Koppel	1.8108	404.2406	0.0511
utrecht-Dichterswijk, Rivierenwijk	1.8432	792.0172	0.0553
utrecht-Eemkwartier	2.3326	554.8346	0.0719
utrecht-Hoogland	1.6664	984.2913	0.0582
utrecht-Hoogland-West	1.1327	1771.1340	0.0658
utrecht-Hooglanderveen	1.9157	717.1319	0.0516
utrecht-Isselt	1.7041	710.9079	0.0485
utrecht-Kanaleneiland	1.7742	650.2125	0.0480
utrecht-Kattenbroek	1.7813	661.8285	0.0453
utrecht-Kruiskamp	1.7730	539.1563	0.0524
utrecht-Leusderkwartier	1.4610	1146.1960	0.0901
utrecht-Liendert	1.7739	596.0238	0.0489
utrecht-Lombok, Leidseweg	2.1839	925.2282	0.0665
utrecht-Nederberg	2.8111	709.0796	0.0705
utrecht-Nieuw Engeland, Schepenbuurt	1.6676	2140.2897	0.0735
utrecht-Nieuwland	1.7826	823.6962	0.0500

Province-Neighborhood	β	MAE (m)	MAPE (%)
utrecht-Oog in Al	1.8555	959.0929	0.0733
utrecht-Randenbroek	1.7924	716.2161	0.0539
utrecht-Rustenburg	1.5525	301.8927	0.0374
utrecht-Schothorst-Noord	1.4577	879.2264	0.0706
utrecht-Schothorst-Zuid	2.1886	673.7437	0.0523
utrecht-Schuilenburg	1.6973	500.4915	0.0581
utrecht-Soesterkwartier	1.5376	941.5048	0.0667
utrecht-Stadskern	2.5050	828.5688	0.0596
utrecht-Terwijde	1.9919	836.9398	0.0499
utrecht-Vathorst-Centrum	2.3005	712.6952	0.0583
utrecht-Vathorst-De Laak	1.5807	1236.5632	0.0659
utrecht-Vathorst-De Velden	1.8400	739.1278	0.0477
utrecht-Veldhuizen	1.4912	613.0585	0.0533
utrecht-Vermeerkwartier	1.5948	687.4064	0.0570
utrecht-Zielhorst	1.5006	737.5802	0.0619
gelderland-Aldenhof	2.0767	433.9962	0.0520
gelderland-Altrade	1.9249	698.2621	0.0603
gelderland-Benedenstad	2.2403	461.7761	0.0592
gelderland-Biezen	1.9073	752.0586	0.0545
gelderland-Bijsterhuizen	1.5322	1297.4129	0.1344
gelderland-Bottendaal	2.0446	543.1354	0.0666
gelderland-Brakkenstein	1.2203	581.6886	0.0615
gelderland-De Hoop	1.6966	372.7379	0.0498
gelderland-De Horsten	2.0480	514.9788	0.0732
gelderland-De Huet	1.6177	635.6865	0.0432
gelderland-De Kamp	2.2909	1062.2622	0.0714
gelderland-De Pas	1.8325	538.1064	0.0628
gelderland-Dichteren	1.7689	1078.1779	0.0788
gelderland-Druten-Zuid	1.4551	735.2720	0.0512
gelderland-Ede-Zuid	1.3842	794.8912	0.0626
gelderland-Enka	2.0286	407.3000	0.0456
gelderland-Galgenveld	1.8408	785.2952	0.0656
gelderland-Goffert	1.2473	1076.0512	0.0695
gelderland-Groenewoud	1.4469	658.1102	0.0544
gelderland-Grootstal	1.5571	665.0571	0.0593
gelderland-Hatert	1.6491	706.3607	0.0492
gelderland-Haven- en industrieterrein	1.5212	1392.6804	0.0824
gelderland-Hazenkamp	1.8706	641.5964	0.0530
gelderland-Hees	1.5558	545.6828	0.0495
gelderland-Heideslag	1.6071	975.2410	0.1388
gelderland-Heijendaal	1.6073	1200.7609	0.0947
gelderland-Hengstdal	1.6617	547.5640	0.0465
gelderland-Heseveld	1.8336	809.3034	0.0628
gelderland-Hunnerberg	1.6417	951.1800	0.0805
gelderland-Kerkenbos	1.7382	693.6390	0.0845
gelderland-Kernhem	2.0289	1850.1112	0.1242
gelderland-Kortenoord	1.8562	384.3648	0.0391

Province-Neighborhood	β	MAE (m)	MAPE (%)
gelderland-Kwakkenberg	1.4378	756.0676	0.0660
gelderland-Lankforst	2.0773	386.7511	0.0478
gelderland-Lent	1.5533	1618.6980	0.0697
gelderland-Maandereng	1.6478	440.8507	0.0445
gelderland-Malvert	2.1738	466.3427	0.0514
gelderland-Meijhorst	2.4381	572.9487	0.0503
gelderland-Methen	1.7507	425.6924	0.0434
gelderland-Neerbosch-Oost	1.8323	1063.6562	0.0763
gelderland-Nije Veld	1.5661	585.2526	0.0530
gelderland-Noordwest	1.4619	410.9665	0.0414
gelderland-Nude	1.5069	409.0421	0.0418
gelderland-Oosseld	1.3913	651.8807	0.0587
gelderland-Oosterhout	1.7417	920.4550	0.0583
gelderland-Ooyse Schependom	1.9134	452.2295	0.0961
gelderland-Overstegen	1.6601	619.1005	0.0463
gelderland-Ressen	1.2860	2477.5349	0.1652
gelderland-Rietkampen	1.8954	661.3551	0.0486
gelderland-Rijkerswoerd	1.6634	791.6755	0.0503
gelderland-Roodwilligen	2.0462	154.8905	0.0452
gelderland-Schöneveld	1.5860	348.5303	0.0439
gelderland-Staddijk	2.2683	605.8993	0.0389
gelderland-Stadscentrum	1.9931	970.3173	0.0702
gelderland-Stadsweiden	1.9297	564.7706	0.0428
gelderland-Tolhuis	2.2581	1032.3502	0.0769
gelderland-Veldhuizen	1.6321	632.9743	0.0430
gelderland-Weezenhof	2.2147	652.8908	0.0490
gelderland-Westkanaaldijk	1.8136	613.0853	0.0478
gelderland-Wijnbergen	1.9347	341.6040	0.0479
gelderland-Wolfskuil	1.8763	641.3040	0.0581
gelderland-Zwanenveld	2.0831	480.3584	0.0431
zuid holland-Afrikaanderwijk	2.4654	721.8422	0.0662
zuid holland-Berkel	1.7787	672.5325	0.0562
zuid holland-Beverwaard	1.4127	577.7932	0.0516
zuid holland-Binnenstad	0.4101	1071.5382	0.0605
zuid holland-Bloemendaal	1.4924	877.6564	0.0502
zuid holland-Bloemhof	2.4326	1100.6165	0.0759
zuid holland-Bospolder	2.6807	839.1196	0.0775
zuid holland-Buitenhof	1.6368	1221.2619	0.0782
zuid holland-Capelle-West	1.7204	685.4383	0.0761
zuid holland-Carnisse	2.2357	990.3210	0.0816
zuid holland-Cool	2.8742	1269.3949	0.0837
zuid holland-De Schenkel	1.8537	366.5598	0.0429
zuid holland-Delfshaven	2.3678	877.7223	0.0797
zuid holland-Fascinatio	1.4154	947.6871	0.1038
zuid holland-Feijenoord	1.7957	615.1246	0.0620
zuid holland-Goverwelle	1.5360	717.9833	0.0571
zuid holland-Groente- en Fruitmarkt	2.0900	581.8834	0.0612

Province-Neighborhood	β	MAE (m)	MAPE (%)
zuid holland-Groot-IJsselmonde	1.4505	1209.2257	0.0534
zuid holland-Heijplaat	1.7579	599.8172	0.0758
zuid holland-Hellevoet	1.8235	974.4529	0.0624
zuid holland-Het Lage Land	1.5567	927.9785	0.0668
zuid holland-Hillegersberg-Noord	1.2027	1402.8705	0.0946
zuid holland-Hillesluis	2.1622	992.7245	0.0696
zuid holland-Hof van Delft	1.4857	755.3555	0.0608
zuid holland-Hoog Dalem	1.8133	550.5851	0.0493
zuid holland-Katendrecht	2.3460	848.4079	0.0731
zuid holland-Kleinpolder	1.5306	534.3541	0.0478
zuid holland-Kleiwegkwartier	1.6862	1108.9772	0.0851
zuid holland-Kop van Zuid	4.0589	2635.9565	0.1803
zuid holland-Kop van Zuid-Entrepot	2.2073	1013.5264	0.0795
zuid holland-Kort Haarlem	1.6126	714.5429	0.0582
zuid holland-Korte Akkeren	1.4268	934.6143	0.0728
zuid holland-Leyenburg	1.7989	1105.6755	0.0704
zuid holland-Lombardijen	1.5453	877.8189	0.0575
zuid holland-Middelland	2.8351	1449.9492	0.0939
zuid holland-Middelwatering	1.5045	1027.9243	0.0573
zuid holland-Moerwijk	1.7662	911.6470	0.0532
zuid holland-Molenlaankwartier	1.4254	1057.7209	0.0766
zuid holland-Morgenstond	2.2568	1237.2026	0.0660
zuid holland-Nesselande	1.5881	1191.0324	0.0562
zuid holland-Nieuw-Helvoet	1.4789	727.4401	0.0570
zuid holland-Nieuw-Mathenesse	1.8853	1490.8770	0.0978
zuid holland-Nieuwe Westen	1.9554	1012.1868	0.0710
zuid holland-Noord	1.5213	962.6752	0.0619
zuid holland-Noordereiland	2.8096	546.1991	0.0657
zuid holland-Ommoord	1.6754	1006.0523	0.0453
zuid holland-Onyx	2.0266	245.1591	0.0600
zuid holland-Oosterflank	1.8375	921.5584	0.0638
zuid holland-Oostgaarde	1.6440	1002.5975	0.0609
zuid holland-Oud-Charlois	1.8822	998.3229	0.0654
zuid holland-Oud-IJsselmonde	1.4633	1276.5539	0.0702
zuid holland-Oud-Mathenesse	1.9691	684.9317	0.0655
zuid holland-Oude Westen	2.7588	1245.0457	0.0869
zuid holland-Overschie	1.3690	930.9699	0.0668
zuid holland-Pendrecht	1.5959	638.2521	0.0565
zuid holland-Plaswijck	1.6337	858.8201	0.0538
zuid holland-Portlandsehoek	1.9635	337.4392	0.0408
zuid holland-Prinsenland	1.5941	672.2612	0.0508
zuid holland-RijswijkBuiten	2.0129	852.4383	0.0602
zuid holland-Rivium	1.6956	492.7409	0.0637
zuid holland-Ruiven	0.9342	1085.9904	0.0947
zuid holland-Rustenburg en Oostbroek	2.3531	996.3351	0.0634
zuid holland-Schenkel	1.6121	728.4814	0.0603
zuid holland-Schiebroek	1.1736	1351.9866	0.0778

Province-Neighborhood	β	MAE (m)	MAPE (%)
zuid holland-Schieweg	1.5207	1199.9656	0.0941
zuid holland-Schollevaar	1.6165	2304.8204	0.1018
zuid holland-Spangen	2.3912	913.4136	0.0775
zuid holland-Stadsdriehoek	1.9833	1105.7357	0.0617
zuid holland-Stolersluis	1.1767	905.2548	0.1086
zuid holland-Tanthof-Oost	1.6836	533.9905	0.0518
zuid holland-Tanthof-West	1.6252	702.5259	0.0644
zuid holland-Tarwewijk	2.1290	1197.4825	0.0849
zuid holland-Transvaalkwartier	2.3932	1159.1211	0.0780
zuid holland-Tussendijken	2.5914	828.4997	0.0830
zuid holland-Vogelbuurt	1.8965	331.7062	0.0430
zuid holland-Voordijkshoorn	1.8969	1182.2247	0.0638
zuid holland-Voorhof	1.7787	687.5436	0.0547
zuid holland-Vreewijk	2.0796	1337.1264	0.0680
zuid holland-Vrijenban	1.5248	1177.3917	0.0863
zuid holland-Wateringse Veld	1.6851	889.2508	0.0418
zuid holland-Westergouwe	1.4591	1411.3738	0.0919
zuid holland-Westplaat	1.5769	327.1887	0.0404
zuid holland-Westpolder	1.9831	1201.3643	0.0788
zuid holland-Wielewaal	2.0599	514.1891	0.0736
zuid holland-Wippolder	1.3746	934.4171	0.0606
zuid holland-Zestienhoven	1.3352	1414.0629	0.0678
zuid holland-Zevenkamp	1.5541	893.7788	0.0567
zuid holland-Zuid	1.6404	425.8455	0.0548
zuid holland-Zuiderpark	1.7440	771.4564	0.0746
zuid holland-Zuidwijk	1.5034	657.6528	0.0549
noord brabant-Binnenstad	1.4873	988.4452	0.0570
noord brabant-Brandevoort	1.2806	1820.5737	0.1081
noord brabant-Brouwhuis	1.2729	1392.1591	0.0851
noord brabant-Dierdonk	1.0960	2001.3904	0.1358
noord brabant-Helmond-Noord	1.4375	823.4308	0.0520
noord brabant-Helmond-Oost	1.4731	714.1405	0.0596
noord brabant-Helmond-West	1.6143	824.8724	0.0717
noord brabant-Industriegebied Zuid	1.2590	1663.6083	0.0682
noord brabant-Mierlo-Hout	1.1230	831.9247	0.0480
noord brabant-Oosterheide	1.5064	1164.2297	0.0731
noord brabant-Rijpelberg	1.0201	2983.4971	0.1879
noord brabant-Stiphout	1.0598	1370.0206	0.0717
noord brabant-Warande	1.5735	946.7836	0.0576
noord brabant-West	1.2153	2095.1854	0.0854
noord brabant-Zuidoost	1.0094	2447.4908	0.1255
limburg-Blerick-Midden	1.5260	595.5677	0.0536
limburg-Blerick-Noord	1.7274	651.6155	0.0618
limburg-Blerick-Zuid	1.6030	415.9061	0.0540
limburg-Boekend	0.9041	1407.1170	0.1200
limburg-Centrum	1.5630	1571.1261	0.0698
limburg-Donderberg	1.5455	461.5097	0.0410

Province-Neighborhood	β	MAE (m)	MAPE (%)
limburg-Hoogvonderen	1.8311	437.1072	0.0486
limburg-Hout-Blerick	1.1087	1435.0019	0.1037
limburg-Klingerberg	1.3996	569.4924	0.0648
limburg-Lindenheuvel	1.3184	848.0495	0.0592
limburg-Maasniel	1.1796	1354.3674	0.0700
limburg-Maastricht-Centrum	1.9251	1312.6660	0.0587
limburg-Maastricht-Noordwest	1.3496	997.5150	0.0709
limburg-Maastricht-Oost	1.2976	1222.5390	0.0479
limburg-Maastricht-West	1.3181	1198.8582	0.0468
limburg-Maastricht-Zuidoost	1.1842	1151.9402	0.0530
limburg-Meuleveld	1.9084	341.2812	0.0497
limburg-Roermond-Oost	1.4770	663.6759	0.0581
limburg-Roermond-Zuid	1.0282	2050.1706	0.1189
limburg-Sanderbout	1.9377	765.6142	0.0907
limburg-Vossener	1.6321	478.9732	0.0562
zeeland-Binnenstad	1.5393	1009.2677	0.0750
zeeland-Burgh	1.5935	550.3192	0.0714
zeeland-Haamstede	1.6837	664.3283	0.0571
zeeland-Lammerenburg	1.4547	1275.8438	0.0617
zeeland-Middelburg-Zuid	1.6389	900.5887	0.0686
zeeland-Othene	1.6663	712.4926	0.0556
zeeland-Paauwenburg	1.3147	1126.0872	0.0835
zeeland-Terneuzen-Centrum	1.8762	638.5805	0.0613
zeeland-Terneuzen-Noord	1.5186	665.8760	0.0478
zeeland-Terneuzen-West	1.7567	1089.1792	0.0774
zeeland-Terneuzen-Zuid	1.5154	642.4476	0.0465
zeeland-West-Souburg	1.7832	464.8484	0.0501