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

Koen Stevens

May 5, 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 which spatial features (e.g., address density, road network characteristics) influence deviations from the BHH formula's predictions.

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 Beardwood 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:

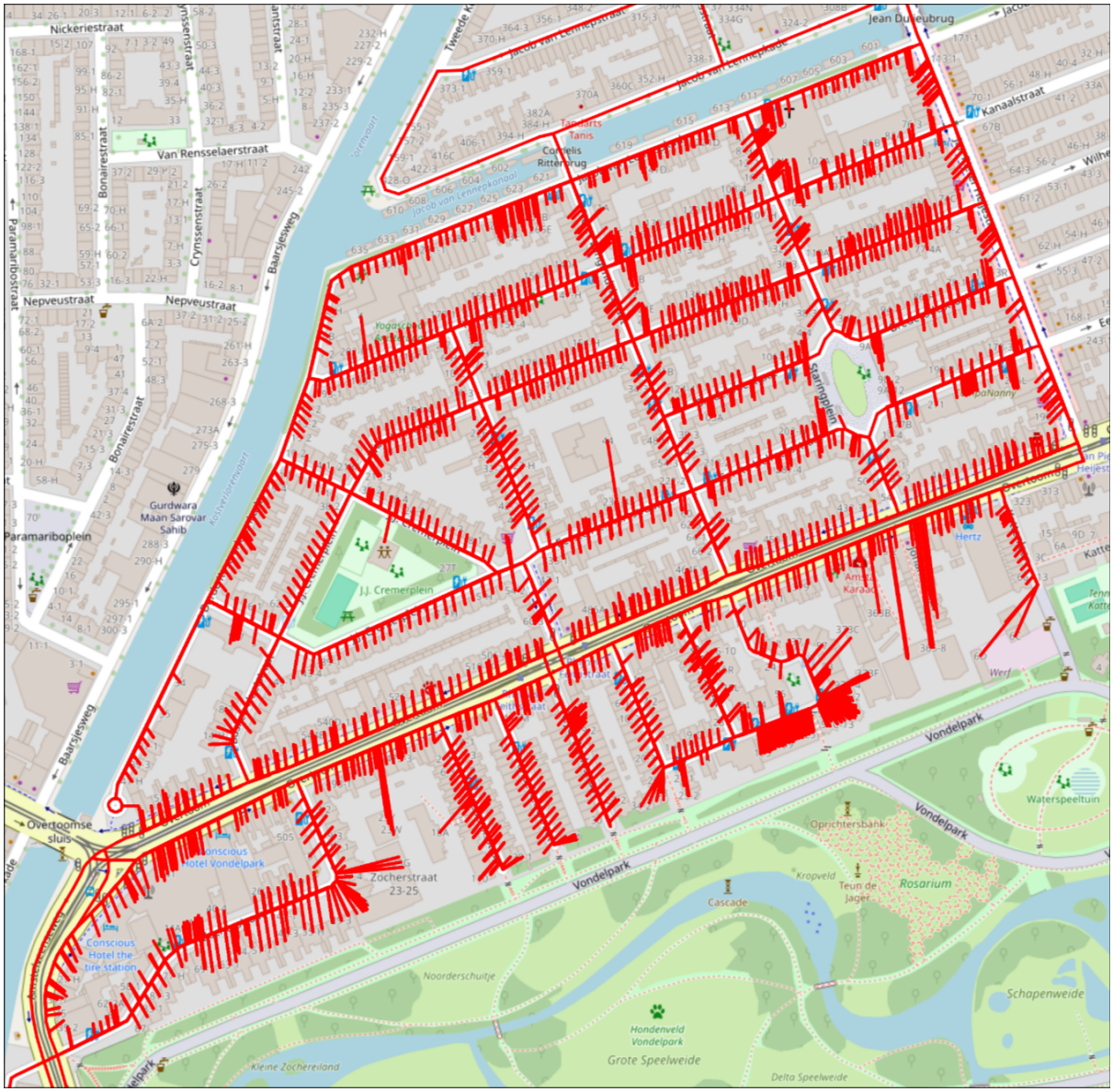
$$d(A, B) = R\sqrt{x^2 + y^2}, \tag{3.1}$$

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

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

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.



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.

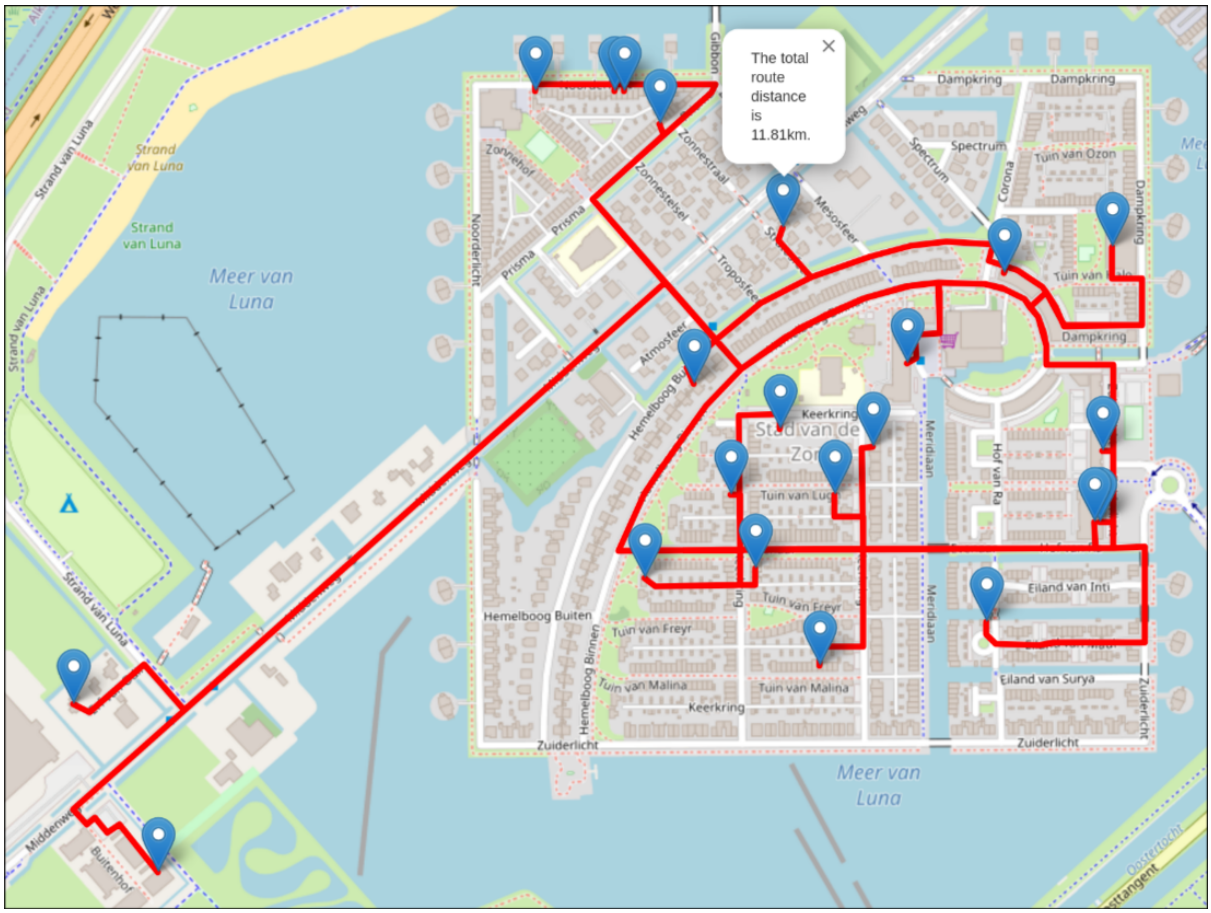
The parameter file contains two lines, in this case: a path to the problem file and a path to the output tour file. This ensures that LKH solves the correct TSPs, and saves the solutions in the correct locations. The problem file contains the information about the problem, for instance n, and the distance 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 $(29+52)\cdot 70\cdot 100 = 567,000$ TSP instances. 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).



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 in selected neighborhoods.

Province	Neighborhood	β	MAE
groningen	Hortusbuurt	2.5550	0.0762
groningen	Binnenstad	2.1603	0.0609
groningen	Oosterpoort	2.2497	0.0800
groningen	Rivierenbuurt	1.8399	0.0686
groningen	De Wijert	1.6627	0.0572
groningen	Oosterparkwijk	1.8157	0.0642
groningen	De Hoogte	2.0924	0.0867
groningen	Korrewegwijk	2.1584	0.0685
groningen	Schildersbuurt	2.3054	0.0635
groningen	Paddepoel	1.6441	0.0485
groningen	Oranjewijk	2.2013	0.0735
groningen	Tuinwijk	3.9071	0.1557
groningen	Selwerd	1.6129	0.0735
groningen	Vinkhuizen	1.4519	0.0479
groningen	Hoogkerk-zuid	1.4759	0.0681
groningen	Gravenburg	1.3842	0.1537
groningen	De Held	1.7873	0.0448
groningen	Reitdiep	1.5725	0.0470
groningen	Hoornse Meer	1.6270	0.0725
groningen	Corpus den Hoorn	1.7489	0.0729
groningen	Eemspoort	1.6025	0.0495
groningen	Euvelgunne	1.8694	0.0976
groningen	Driebond	1.9275	0.1117
groningen	Winschoterdiep	2.2539	0.1478
groningen	Eemskanaal	1.7523	0.0411
groningen	Helpman	2.1346	0.0669

Province	Neighborhood	β	MAE
groningen	Lewenborg	2.0532	0.0631
groningen	Beijum	1.7469	0.0418
groningen	Maarsveld	1.6708	0.0471
drenthe	Assen Oost	1.4572	0.0544
drenthe	Assen West	1.1837	0.1018
drenthe	Centrum	1.6032	0.0661
drenthe	Kloosterveen	1.3175	0.0867
drenthe	Lariks	1.5678	0.0851
drenthe	Marsdijk	1.4970	0.0709
drenthe	Noorderpark	1.6395	0.0484
drenthe	Peelo	1.6547	0.0620
drenthe	Pittelo	1.7842	0.0651
drenthe	Ter Borch	2.1396	0.0515
friesland	Achter de Hoven	1.6777	0.0685
friesland	Aldlân	1.9585	0.0543
friesland	Bilgaard	1.8856	0.0703
friesland	Binnenstad en Stationskwartier	1.9690	0.0666
friesland	Blitsaerd	2.0138	0.0504
friesland	Buitengebied Noordwest	1.4103	0.0698
friesland	Camminghaburen	1.6618	0.0480
friesland	De Hemrik	1.4894	0.0517
friesland	De Zuidlanden	1.1206	0.0783
friesland	De Zwette	1.4904	0.0676
friesland	Heechterp	1.3554	0.1213
friesland	Hempens, Teerns, en Zuiderburen	1.5408	0.0628
friesland	Huizum-Oost	1.4737	0.0624
friesland	Huizum-West	1.6214	0.0633
friesland	Middelsee	1.2532	0.0875
friesland	Nijlân	1.4162	0.0506
friesland	Oranjewijk	2.6047	0.0682
friesland	Oud Oost	1.4825	0.0561
friesland	Schepenbuurt	1.3838	0.1460
friesland	Schieringen en De Centrale	1.4722	0.1014
friesland	Techum	1.7453	0.0536
friesland	Vlietzone	1.7114	0.0608
friesland	Vogelwijk en Componistenbuurt	1.8395	0.0657
friesland	Vrijheidswijk	1.8332	0.0475
friesland	Westeinde	1.8069	0.0375
friesland	Wielenpôlle	1.5383	0.1205
friesland	Zuiderburen	1.5334	0.0595
flevoland	Polderwijk	2.0352	0.0661
overijssel	Baalder	1.4997	0.0629
overijssel	Baalderveld	1.5559	0.0758
overijssel	Berflo Es	1.3234	0.0698
overijssel	Bergweide	1.5792	0.0599
overijssel	De Graven Es	1.5550	0.0942
overijssel	De Meijbree	1.8157	0.0505

Province	Neighborhood	β	MAE
overijssel	De Riet	1.5612	0.0502
overijssel	De Thij	1.4448	0.0644
overijssel	Groot Driene	1.6459	0.0485
overijssel	Haardijk	2.1870	0.0838
overijssel	Hanzeland	1.7487	0.0882
overijssel	Hasseler Es	1.4540	0.0563
overijssel	Heemsermars	1.0859	0.0594
overijssel	Het Onderdijks	1.5658	0.0361
overijssel	Hofkamp	1.4501	0.0551
overijssel	Ittersum	1.5582	0.0632
overijssel	Keizerslanden	1.6999	0.0917
overijssel	Kloosterlanden	1.3116	0.0647
overijssel	Marslanden	2.3320	0.1007
overijssel	Nieuwe Haven	1.9628	0.0486
overijssel	Nieuwstraatkwartier	1.5939	0.0605
overijssel	Noorderkwartier	1.4451	0.0507
overijssel	OosterDalfsen	2.1215	0.0500
overijssel	Ossenkoppelerhoek	1.4821	0.0578
overijssel	Rivierenwijk	1.6418	0.0504
overijssel	Schelfhorst	1.5862	0.0488
overijssel	Slangenbeek	1.3195	0.0537
overijssel	Sluitersveld	1.6229	0.0598
overijssel	Twekkelerveld	1.5042	0.0427
overijssel	Veerallee	1.8578	0.0906
overijssel	Voorstad	1.1583	0.0674
overijssel	Wierdensehoek	1.4468	0.0600
overijssel	Windmolenbroek	1.4985	0.0569
overijssel	Zandweerd	1.4187	0.0490
noord holland	Schrijverswijk	1.7003	0.0533
noord holland	Stad van de Zon	1.3361	0.1540
noord holland	Stadshart	2.0102	0.0518
noord holland	Jordaan	2.4535	0.0663
noord holland	Slotervaart	1.7309	0.0447
noord holland	IJburg	1.3259	0.0477
noord holland	Oostelijke Eilanden	1.6890	0.0473
noord holland	Oostelijk Havengebied	1.7291	0.0539
noord holland	Frederik Hendrikbuurt	2.7253	0.0859
noord holland	Van Lennepbuurt	3.3330	0.0727
noord holland	Da Costabuurt	3.2210	0.0944
noord holland	Kinkerbuurt	3.1971	0.0833
noord holland	Kersenboogerd	1.6025	0.0509
noord holland	Pax	2.0927	0.0496
noord holland	Graan voor Visch	2.1769	0.0523
noord holland	Vrijschot-Noord	2.6790	0.0595
noord holland	Toolenburg	1.6331	0.0449
noord holland	Floriande	1.8360	0.0493
noord holland	Overbos	1.7443	0.0493

Province	Neighborhood	β	MAE
noord holland	Bornholm	1.7714	0.0416
noord holland	Beukenhorst-Oost	1.8658	0.0726
noord holland	De Hoek	2.4353	0.0504
noord holland	West	1.8581	0.0516
noord holland	Zuid	2.1917	0.0614
noord holland	Oost	1.8493	0.0583
noord holland	Noord	1.6465	0.0501
noord holland	De President	1.4345	0.1242
noord holland	Graan voor Visch-Zuid	1.9348	0.0728
noord holland	Zuidwijk	1.9307	0.0395
noord holland	Buitenveldert-West	1.1324	0.0669
noord holland	Buitenveldert	1.1079	0.0745
noord holland	Apollobuurt	1.8053	0.0738
noord holland	Stadionbuurt	1.5917	0.0747
noord holland	Prinses Irenebuurt e.o.	2.0043	0.0711
noord holland	Hoofddorppleinbuurt	1.9428	0.0855
noord holland	Willemspark	2.3903	0.0934
noord holland	Schinkelbuurt	3.2777	0.1018
noord holland	Vondelparkbuurt	3.2189	0.0870
noord holland	Helmersbuurt	3.0017	0.0717
noord holland	Overtoomse Sluis	2.8961	0.0736
noord holland	Museumkwartier	1.7495	0.0720
noord holland	Rivierenbuurt	1.8621	0.0685
noord holland	IJselbuurt	3.6492	0.0813
noord holland	Scheldebuurt	1.8598	0.0655
noord holland	Rijnbuurt	1.9132	0.0565
noord holland	De Baarsjes	2.2798	0.0673
noord holland	Landlust	1.9631	0.0723
noord holland	Staatsliedenbuurt	1.9353	0.0715
noord holland	Spaarndammerbuurt	2.6115	0.0688
noord holland	De Pijp	2.2832	0.0691
noord holland	Grachtengordel	2.1490	0.0724
noord holland	Oud-Zuid	1.4154	0.0653
utrecht	Bedrijventerrein Vathorst	1.8135	0.0628
utrecht	Binnenstad City- En Winkelgebied	1.1909	0.0578
utrecht	Binnenstad Woongebied	2.0563	0.0641
utrecht	Bosgebied	0.9471	0.0688
utrecht	Buitengebied Oost	1.0024	0.0762
utrecht	Calveen	2.0364	0.0830
utrecht	De Berg-Noord	1.6859	0.0505
utrecht	De Berg-Zuid	1.5662	0.0635
utrecht	De Hoef	1.6496	0.0950
utrecht	De Koppel	1.8068	0.0498
utrecht	Dichterswijk, Rivierenwijk	1.8415	0.0547
utrecht	Eemkwartier	2.3506	0.0707
utrecht	Hoogland	1.6517	0.0522
utrecht	Hoogland-West	1.1093	0.0806

Province	Neighborhood	β	MAE
utrecht	Hooglanderveen	1.9074	0.0524
utrecht	Isselt	1.7118	0.0515
utrecht	Kanaleneiland	1.7669	0.0486
utrecht	Kattenbroek	1.7866	0.0473
utrecht	Kruiskamp	1.7514	0.0576
utrecht	Leusderkwartier	1.4597	0.0806
utrecht	Liendert	1.7675	0.0486
utrecht	Lombok, Leidseweg	2.2158	0.0647
utrecht	Nederberg	2.8451	0.0731
utrecht	Nieuw Engeland, Schepenbuurt	1.6672	0.0784
utrecht	Nieuwland	1.7870	0.0442
utrecht	Oog in Al	1.8653	0.0718
utrecht	Randenbroek	1.8042	0.0549
utrecht	Rustenburg	1.5583	0.0370
utrecht	Schothorst-Noord	1.4630	0.0729
utrecht	Schothorst-Zuid	2.1753	0.0560
utrecht	Schuilenburg	1.7148	0.0592
utrecht	Soesterkwartier	1.5312	0.0676
utrecht	Stadskern	2.4799	0.0614
utrecht	Terwijde	1.9836	0.0479
utrecht	Vathorst-Centrum	2.2883	0.0628
utrecht	Vathorst-De Laak	1.5806	0.0649
utrecht	Vathorst-De Velden	1.8309	0.0506
utrecht	Veldhuizen	1.5030	0.0518
utrecht	Vermeerkwartier	1.6073	0.0597
utrecht	Zielhorst	1.5046	0.0639
gelderland	Aldenhof	2.0843	0.0529
gelderland	Altrade	1.9186	0.0642
gelderland	Benedenstad	2.2510	0.0553
gelderland	Biezen	1.9177	0.0528
gelderland	Bijsterhuizen	1.5520	0.1203
gelderland	Bottendaal	2.0470	0.0680
gelderland	Brakkenstein	1.2034	0.0629
gelderland	De Hoop	1.7000	0.0464
gelderland	De Horsten	2.0490	0.0726
gelderland	De Huet	1.6178	0.0399
gelderland	De Kamp	2.3013	0.0651
gelderland	De Pas	1.8275	0.0649
gelderland	Dichteren	1.7891	0.0852
gelderland	Druten-Zuid	1.4579	0.0517
gelderland	Ede-Zuid	1.3796	0.0626
gelderland	Enka	2.0268	0.0479
gelderland	Galgenveld	1.8428	0.0640
gelderland	Goffert	1.2505	0.0661
gelderland	Groenewoud	1.4433	0.0511
gelderland	Grootstal	1.5568	0.0564
gelderland	Hatert	1.6354	0.0512

Province	Neighborhood	β	MAE
gelderland	Haven- en industrieterrein	1.5206	0.0841
gelderland	Hazenkamp	1.8623	0.0522
gelderland	Hees	1.5594	0.0482
gelderland	Heideslag	1.5981	0.1374
gelderland	Heijendaal	1.5889	0.0983
gelderland	Hengstdal	1.6597	0.0488
gelderland	Heseveld	1.8658	0.0661
gelderland	Hunnerberg	1.6389	0.0723
gelderland	Kerkenbos	1.7347	0.0870
gelderland	Kernhem	2.0017	0.1226
gelderland	Kortenoord	1.8641	0.0373
gelderland	Kwakkenberg	1.4172	0.0695
gelderland	Lankforst	2.0773	0.0476
gelderland	Lent	1.5570	0.0736
gelderland	Maandereng	1.6384	0.0461
gelderland	Malvert	2.1687	0.0507
gelderland	Meijhorst	2.4404	0.0457
gelderland	Methen	1.7451	0.0455
gelderland	Neerbosch-Oost	1.8281	0.0755
gelderland	Nije Veld	1.5668	0.0532
gelderland	Noordwest	1.4599	0.0435
gelderland	Nude	1.5147	0.0444
gelderland	Oosseld	1.3948	0.0602
gelderland	Oosterhout	1.7406	0.0612
gelderland	Ooyse Schependom	1.9202	0.0991
gelderland	Overstegen	1.6702	0.0399
gelderland	Ressen	1.2925	0.1553
gelderland	Rietkampen	1.8914	0.0422
gelderland	Rijkerswoerd	1.6586	0.0478
gelderland	Roodwilligen	2.0537	0.0458
gelderland	Schöneveld	1.5825	0.0417
gelderland	Staddijk	2.2832	0.0358
gelderland	Stadscentrum	1.9755	0.0670
gelderland	Stadsweiden	1.9250	0.0449
gelderland	Tolhuis	2.2286	0.0805
gelderland	Veldhuizen	1.6364	0.0413
gelderland	Weezenhof	2.2079	0.0493
gelderland	Westkanaaldijk	1.8150	0.0483
gelderland	Wijnbergen	1.9383	0.0431
gelderland	Wolfskuil	1.8837	0.0586
gelderland	Zwanenveld	2.0751	0.0423
zuid holland	Afrikaanderwijk	2.4644	0.0685
zuid holland	Berkel	1.7768	0.0589
zuid holland	Beverwaard	1.4093	0.0481
zuid holland	Binnenstad	0.4099	0.0626
zuid holland	Bloemendaal	1.4969	0.0497
zuid holland	Bloemhof	2.4346	0.0693

Province	Neighborhood	β	MAE
zuid holland	Bospolder	2.6830	0.0771
zuid holland	Buitenhof	1.6136	0.0761
zuid holland	Capelle-West	1.7145	0.0744
zuid holland	Carnisse	2.2517	0.0768
zuid holland	Cool	2.9089	0.0862
zuid holland	De Schenkel	1.8481	0.0404
zuid holland	Delfshaven	2.3588	0.0809
zuid holland	Fascinatio	1.4151	0.0998
zuid holland	Feijenoord	1.7951	0.0582
zuid holland	Goverwelle	1.5495	0.0563
zuid holland	Groente- en Fruitmarkt	2.0848	0.0640
zuid holland	Groot-IJsselmonde	1.4497	0.0529
zuid holland	Heijplaat	1.7540	0.0755
zuid holland	Hellevoet	1.8278	0.0701
zuid holland	Het Lage Land	1.5732	0.0660
zuid holland	Hillegersberg-Noord	1.1909	0.0856
zuid holland	Hillesluis	2.1526	0.0712
zuid holland	Hof van Delft	1.4848	0.0576
zuid holland	Hoog Dalem	1.8086	0.0469
zuid holland	Katendrecht	2.3478	0.0711
zuid holland	Kleinpolder	1.5246	0.0505
zuid holland	Kleiwegkwartier	1.6855	0.0803
zuid holland	Kop van Zuid	4.1484	0.1773
zuid holland	Kop van Zuid-Entrepot	2.2034	0.0818
zuid holland	Kort Haarlem	1.6151	0.0566
zuid holland	Korte Akkeren	1.4413	0.0755
zuid holland	Leyenburg	1.7983	0.0700
zuid holland	Lombardijen	1.5576	0.0503
zuid holland	Middelland	2.8183	0.0910
zuid holland	Middelwatering	1.4998	0.0642
zuid holland	Moerwijk	1.7490	0.0534
zuid holland	Molenlaankwartier	1.4263	0.0756
zuid holland	Morgenstond	2.2670	0.0596
zuid holland	Nesselande	1.5834	0.0529
zuid holland	Nieuw-Helvoet	1.4777	0.0567
zuid holland	Nieuw-Mathenesse	1.8766	0.0868
zuid holland	Nieuwe Westen	1.9625	0.0695
zuid holland	Noord	1.5256	0.0591
zuid holland	Noordereiland	2.7875	0.0771
zuid holland	Ommoord	1.6861	0.0472
zuid holland	Onyx	2.0410	0.0586
zuid holland	Oosterflank	1.8234	0.0692
zuid holland	Oostgaarde	1.6407	0.0603
zuid holland	Oud-Charlois	1.8897	0.0644
zuid holland	Oud-IJsselmonde	1.4641	0.0680
zuid holland	Oud-Mathenesse	1.9834	0.0623
zuid holland	Oude Westen	2.7834	0.0809

Province	Neighborhood	β	MAE
zuid holland	Overschie	1.3661	0.0663
zuid holland	Pendrecht	1.5880	0.0599
zuid holland	Plaswijck	1.6459	0.0599
zuid holland	Portlandsehoek	1.9650	0.0394
zuid holland	Prinsenland	1.5855	0.0489
zuid holland	RijswijkBuiten	2.0161	0.0570
zuid holland	Rivium	1.7176	0.0618
zuid holland	Ruiven	0.9446	0.0988
zuid holland	Rustenburg en Oostbroek	2.3306	0.0677
zuid holland	Schenkel	1.6095	0.0593
zuid holland	Schiebroek	1.1726	0.0766
zuid holland	Schieweg	1.4981	0.0865
zuid holland	Schollevaar	1.6344	0.0891
zuid holland	Spangen	2.3666	0.0816
zuid holland	Stadsdriehoek	1.9862	0.0608
zuid holland	Stolersluis	1.1829	0.1014
zuid holland	Tanthof-Oost	1.6802	0.0534
zuid holland	Tanthof-West	1.6027	0.0693
zuid holland	Tarwewijk	2.1001	0.0795
zuid holland	Transvaalkwartier	2.4080	0.0741
zuid holland	Tussendijken	2.5959	0.0858
zuid holland	Vogelbuurt	1.8994	0.0459
zuid holland	Voordijkshoorn	1.8919	0.0614
zuid holland	Voorhof	1.7758	0.0558
zuid holland	Vreewijk	2.0695	0.0645
zuid holland	Vrijenban	1.5118	0.0897
zuid holland	Wateringse Veld	1.6811	0.0445
zuid holland	Westergouwe	1.4850	0.0879
zuid holland	Westplaat	1.5841	0.0437
zuid holland	Westpolder	1.9908	0.0875
zuid holland	Wielewaal	2.0689	0.0723
zuid holland	Wippolder	1.3848	0.0569
zuid holland	Zestienhoven	1.3374	0.0751
zuid holland	Zevenkamp	1.5427	0.0540
zuid holland	Zuid	1.6510	0.0611
zuid holland	Zuiderpark	1.7411	0.0744
zuid holland	Zuidwijk	1.5135	0.0532
noord brabant	Binnenstad	1.4886	0.0547
noord brabant	Brandevoort	1.2907	0.0960
noord brabant	Brouwhuis	1.2607	0.0877
noord brabant	Dierdonk	1.1284	0.1435
noord brabant	Helmond-Noord	1.4365	0.0503
noord brabant	Helmond-Oost	1.4666	0.0549
noord brabant	Helmond-West	1.6144	0.0725
noord brabant	Industriegebied Zuid	1.2525	0.0651
noord brabant	Mierlo-Hout	1.1170	0.0470
noord brabant	Oosterheide	1.5126	0.0732

Province	Neighborhood	β	MAE
noord brabant	Rijpelberg	1.0164	0.1972
noord brabant	Stiphout	1.0574	0.0755
noord brabant	Warande	1.5766	0.0570
noord brabant	West	1.2245	0.0824
noord brabant	Zuidoost	1.0307	0.1066
limburg	Blerick-Midden	1.5509	0.0549
limburg	Blerick-Noord	1.7352	0.0628
limburg	Blerick-Zuid	1.6055	0.0514
limburg	Boekend	0.9081	0.1153
limburg	Centrum	1.5609	0.0682
limburg	Donderberg	1.5480	0.0451
limburg	Hoogvonderen	1.8374	0.0448
limburg	Hout-Blerick	1.1105	0.1023
limburg	Klingerberg	1.3967	0.0698
limburg	Lindenheuvel	1.3089	0.0656
limburg	Maasniel	1.1827	0.0737
limburg	Maastricht-Centrum	1.9108	0.0602
limburg	Maastricht-Noordwest	1.3526	0.0688
limburg	Maastricht-Oost	1.3006	0.0499
limburg	Maastricht-West	1.3195	0.0469
limburg	Maastricht-Zuidoost	1.1844	0.0554
limburg	Meuleveld	1.9213	0.0417
limburg	Roermond-Oost	1.4822	0.0573
limburg	Roermond-Zuid	1.0218	0.1065
limburg	Sanderbout	1.9231	0.0974
limburg	Vossener	1.6196	0.0501
zeeland	Binnenstad	1.5437	0.0757
zeeland	Burgh	1.6014	0.0670
zeeland	Haamstede	1.6854	0.0554
zeeland	Lammerenburg	1.4350	0.0621
zeeland	Middelburg-Zuid	1.6464	0.0673
zeeland	Othene	1.6575	0.0515
zeeland	Paauwenburg	1.3287	0.0676
zeeland	Terneuzen-Centrum	1.8809	0.0613
zeeland	Terneuzen-Noord	1.5156	0.0480
zeeland	Terneuzen-West	1.7826	0.0711
zeeland	Terneuzen-Zuid	1.5153	0.0465
zeeland	West-Souburg	1.7884	0.0476