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

Koen Stevens

May 2, 2025

Bachelor's Thesis Econometrics

Supervisor: dr. N.D. van Foreest

Second assessor:

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 contribution of this research is the performance of the Beardwood formula is analyzed when:

1. relaxing the assumption of uniformly drawn locations. In this research, the locations are drawn from the set of postcodes in the area in question.
2. applied to realistically sized real-world parts of cities and villages in the Netherlands.

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 Beardwood 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:

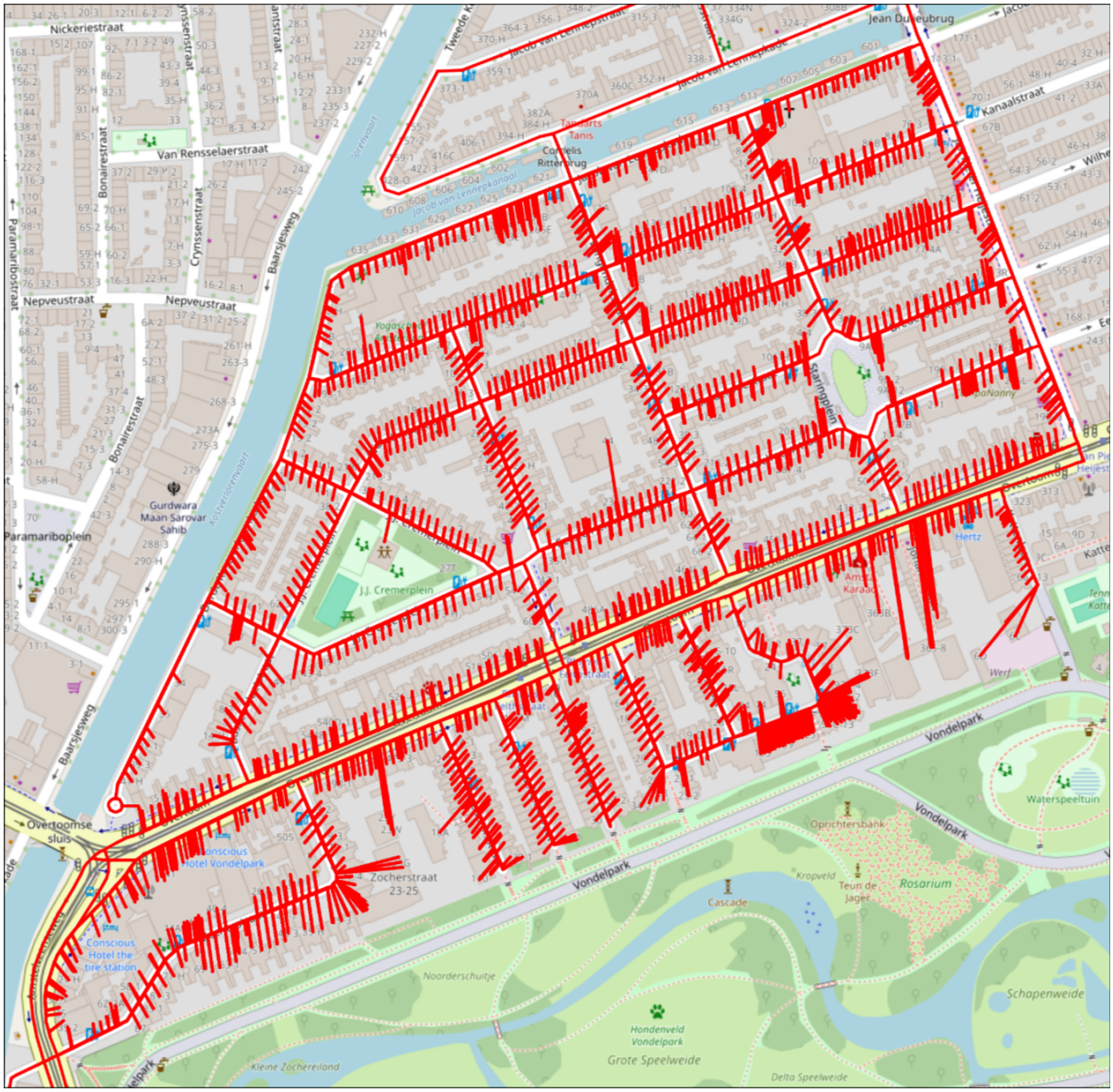
$$d(A, B) = R\sqrt{x^2 + y^2}, \tag{3.1}$$

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

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

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.

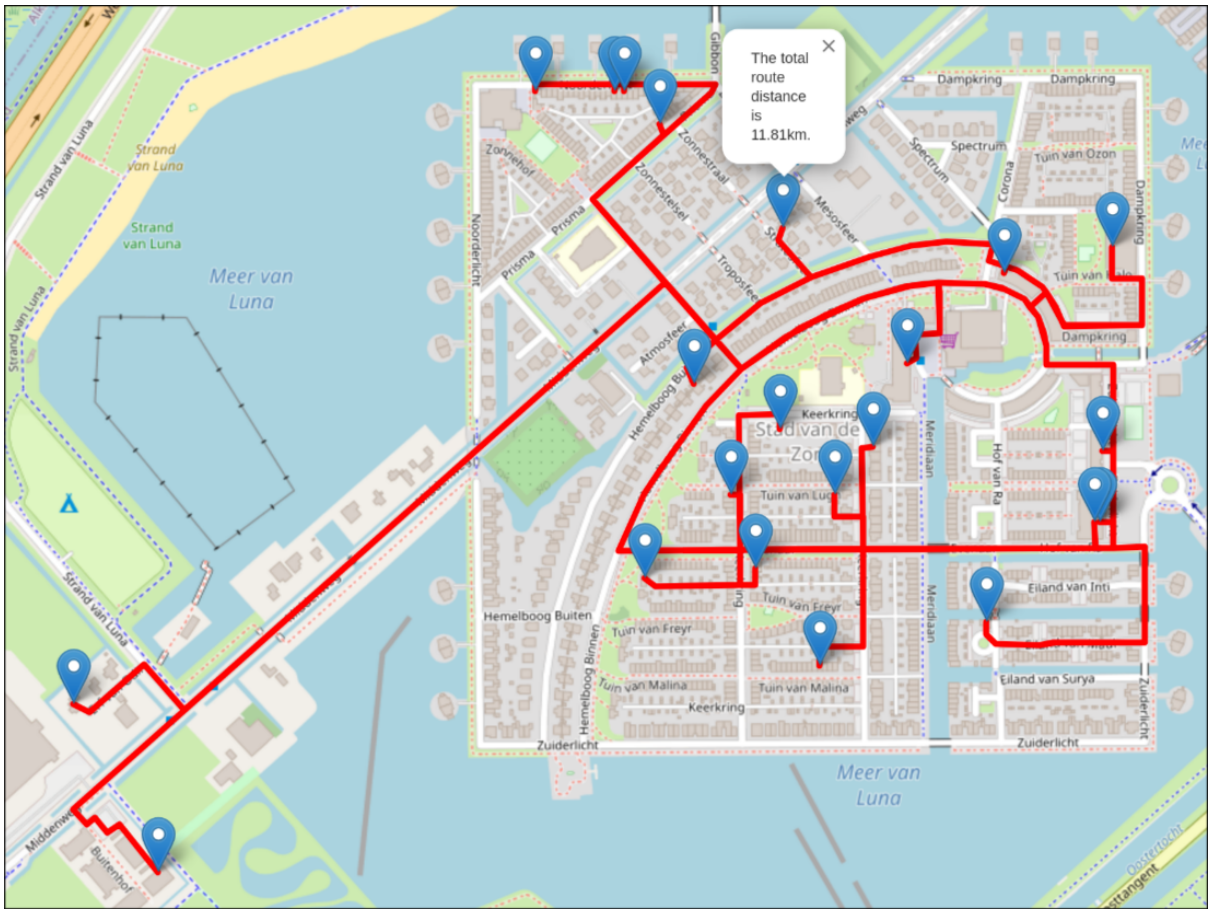
The parameter file contains two lines, in this case: a path to the problem file and a path to the output tour file. This ensures that LKH solves the correct TSPs, and saves the solutions in the correct locations. The problem file contains the information about the problem, for instance n, and the distance 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	2.5489	0.0820
groningen	Binnenstad	2.1402	0.0694
groningen	Oosterpoort	2.2544	0.0706
groningen	Rivierenbuurt	1.8407	0.0780
groningen	De Wijert	1.6620	0.0604
groningen	Oosterparkwijk	1.8317	0.0639
groningen	De Hoogte	2.0798	0.0825
groningen	Korrewegwijk	2.1485	0.0708
groningen	Schildersbuurt	2.3071	0.0651
groningen	Paddepoel	1.6446	0.0506
groningen	Oranjewijk	2.1883	0.0709
groningen	Tuinwijk	3.8908	0.1409
groningen	Selwerd	1.6236	0.0646
groningen	Vinkhuizen	1.4570	0.0452
groningen	Hoogkerk-zuid	1.4835	0.0715
groningen	Gravenburg	1.3761	0.1562
groningen	De Held	1.7886	0.0450
groningen	Reitdiep	1.5797	0.0413
groningen	Hoornse Meer	1.6244	0.0665
groningen	Corpus den Hoorn	1.7609	0.0778
groningen	Eemspoort	1.6033	0.0501
groningen	Euvelgunne	1.8820	0.0866
groningen	Driebond	1.9449	0.1074
groningen	Winschoterdiep	2.2893	0.1561
groningen	Eemskanaal	1.7571	0.0416
groningen	Helpman	2.1350	0.0676

Province	Neighborhood	β	MAE
groningen	Lewenborg	2.0640	0.0633
groningen	Beijum	1.7438	0.0436
groningen	Maarsveld	1.6662	0.0502
drenthe	Assen Oost	1.4594	0.0586
drenthe	Assen West	1.1837	0.1014
drenthe	Centrum	1.5878	0.0684
drenthe	Kloosterveen	1.3172	0.0819
drenthe	Lariks	1.5689	0.0777
drenthe	Marsdijk	1.4941	0.0730
drenthe	Noorderpark	1.6409	0.0482
drenthe	Peelo	1.6549	0.0638
drenthe	Pittelo	1.7726	0.0604
drenthe	Ter Borch	2.1463	0.0547
friesland	Achter de Hoven	1.6474	0.0719
friesland	Aldlân	1.9580	0.0584
friesland	Bilgaard	1.8985	0.0724
friesland	Binnenstad en Stationskwartier	1.9498	0.0676
friesland	Blitsaerd	2.0235	0.0502
friesland	Buitengebied Noordwest	1.3937	0.0722
friesland	Camminghaburen	1.6628	0.0462
friesland	De Hemrik	1.4981	0.0538
friesland	De Zuidlanden	1.1236	0.0793
friesland	De Zwette	1.4996	0.0782
friesland	Heechterp	1.3473	0.1178
friesland	Hempens, Teerns, en Zuiderburen	1.5257	0.0587
friesland	Huizum-Oost	1.4733	0.0606
friesland	Huizum-West	1.6289	0.0620
friesland	Middelsee	1.2396	0.0866
friesland	Nijlân	1.4201	0.0549
friesland	Oranjewijk	2.5764	0.0653
friesland	Oud Oost	1.4810	0.0523
friesland	Schepenbuurt	1.3801	0.1521
friesland	Schieringen en De Centrale	1.4851	0.1026
friesland	Techum	1.7456	0.0552
friesland	Vlietzone	1.7082	0.0581
friesland	Vogelwijk en Componistenbuurt	1.8545	0.0629
friesland	Vrijheidswijk	1.8219	0.0488
friesland	Westeinde	1.7959	0.0371
friesland	Wielenpôlle	1.5386	0.1199
friesland	Zuiderburen	1.5332	0.0578
flevoland	Polderwijk	2.0393	0.0667
overijssel	Baalder	1.5095	0.0630
overijssel	Baalderveld	1.5579	0.0771
overijssel	Berflo Es	1.3305	0.0586
overijssel	Bergweide	1.5838	0.0591
overijssel	Binnenstad	0.2522	0.0569
overijssel	De Graven Es	1.5537	0.0927

Province	Neighborhood	β	MAE
overijssel	De Meijbree	1.8144	0.0545
overijssel	De Riet	1.5580	0.0484
overijssel	De Thij	1.4554	0.0686
overijssel	Groot Driene	1.6519	0.0463
overijssel	Haardijk	2.1806	0.0818
overijssel	Hanzeland	1.7377	0.0894
overijssel	Hasseler Es	1.4665	0.0553
overijssel	Heemsermars	1.0834	0.0553
overijssel	Het Onderdijks	1.5657	0.0372
overijssel	Hofkamp	1.4459	0.0538
overijssel	Ittersum	1.5701	0.0632
overijssel	Keizerslanden	1.7098	0.0893
overijssel	Kloosterlanden	1.3210	0.0622
overijssel	Marslanden	2.3038	0.1012
overijssel	Nieuwe Haven	1.9520	0.0510
overijssel	Nieuwstraatkwartier	1.5863	0.0600
overijssel	Noorderkwartier	1.4366	0.0488
overijssel	OosterDalfsen	2.1333	0.0511
overijssel	Ossenkoppelerhoek	1.4911	0.0583
overijssel	Rivierenwijk	1.6222	0.0469
overijssel	Schelfhorst	1.5858	0.0488
overijssel	Slangenbeek	1.3192	0.0544
overijssel	Sluitersveld	1.6348	0.0615
overijssel	Twekkelerveld	1.4920	0.0457
overijssel	Veerallee	1.8506	0.0964
overijssel	Voorstad	1.1599	0.0664
overijssel	Wierdensehoek	1.4502	0.0601
overijssel	Windmolenbroek	1.4849	0.0592
overijssel	Zandweerd	1.4192	0.0493
noord holland	Schrijverswijk	1.7061	0.0490
noord holland	Stad van de Zon	1.3182	0.1501
noord holland	Stadshart	2.0048	0.0492
noord holland	Jordaan	2.4570	0.0658
noord holland	Slotervaart	1.7264	0.0452
noord holland	IJburg	1.3204	0.0516
noord holland	Oostelijke Eilanden	1.6889	0.0493
noord holland	Oostelijk Havengebied	1.7278	0.0529
noord holland	Frederik Hendrikbuurt	2.7373	0.0836
noord holland	Van Lennepbuurt	3.3163	0.0723
noord holland	Da Costabuurt	3.1936	0.0876
noord holland	Kinkerbuurt	3.2151	0.0867
noord holland	Kersenboogerd	1.5913	0.0546
noord holland	Pax	2.0991	0.0466
noord holland	Graan voor Visch	2.1820	0.0530
noord holland	Vrijschot-Noord	2.6600	0.0625
noord holland	Toolenburg	1.6488	0.0451
noord holland	Floriande	1.8443	0.0463

Province	Neighborhood	β	MAE
noord holland	Overbos	1.7498	0.0436
noord holland	Bornholm	1.7700	0.0436
noord holland	Beukenhorst-Oost	1.8520	0.0760
noord holland	De Hoek	2.4444	0.0511
noord holland	West	1.8568	0.0476
noord holland	Zuid	2.2117	0.0589
noord holland	Oost	1.8565	0.0584
noord holland	Noord	1.6421	0.0568
noord holland	De President	1.4477	0.1211
noord holland	Graan voor Visch-Zuid	1.9263	0.0635
noord holland	Zuidwijk	1.9293	0.0402
noord holland	Buitenveldert-West	1.1218	0.0633
noord holland	Buitenveldert	1.0955	0.0742
noord holland	Apollobuurt	1.8059	0.0723
noord holland	Stadionbuurt	1.5936	0.0727
noord holland	Prinses Irenebuurt e.o.	2.0234	0.0690
noord holland	Hoofddorppleinbuurt	1.9619	0.0848
noord holland	Willemspark	2.3634	0.0930
noord holland	Schinkelbuurt	3.2710	0.1127
noord holland	Vondelparkbuurt	3.2099	0.0837
noord holland	Helmersbuurt	3.0008	0.0774
noord holland	Overtoomse Sluis	2.9129	0.0704
noord holland	Museumkwartier	1.7544	0.0676
noord holland	Rivierenbuurt	1.8396	0.0769
noord holland	IJselbuurt	3.6422	0.0849
noord holland	Scheldebuurt	1.8628	0.0669
noord holland	Rijnbuurt	1.9140	0.0538
noord holland	De Baarsjes	2.2768	0.0695
noord holland	Landlust	1.9472	0.0681
noord holland	Staatsliedenbuurt	1.9180	0.0697
noord holland	Spaarndammerbuurt	2.6001	0.0776
noord holland	De Pijp	2.2747	0.0721
noord holland	Grachtengordel	2.1434	0.0720
noord holland	Oud-Zuid	1.4210	0.0607
utrecht	Bedrijventerrein Vathorst	1.7912	0.0627
utrecht	Binnenstad City- En Winkelgebied	1.1738	0.0683
utrecht	Binnenstad Woongebied	2.0391	0.0614
utrecht	Bosgebied	0.9469	0.0721
utrecht	Buitengebied Oost	1.0059	0.0779
utrecht	Calveen	2.0316	0.0836
utrecht	De Berg-Noord	1.6972	0.0533
utrecht	De Berg-Zuid	1.5694	0.0598
utrecht	De Hoef	1.6422	0.0869
utrecht	De Koppel	1.8135	0.0552
utrecht	Dichterswijk, Rivierenwijk	1.8312	0.0551
utrecht	Eemkwartier	2.3441	0.0700
utrecht	Hoogland	1.6486	0.0599

Province	Neighborhood	β	MAE
utrecht	Hoogland-West	1.1179	0.0748
utrecht	Hooglanderveen	1.9185	0.0517
utrecht	Isselt	1.7099	0.0491
utrecht	Kanaleneiland	1.7761	0.0494
utrecht	Kattenbroek	1.7824	0.0416
utrecht	Kruiskamp	1.7657	0.0536
utrecht	Leusderkwartier	1.4596	0.0758
utrecht	Liendert	1.7664	0.0478
utrecht	Lombok, Leidseweg	2.2143	0.0615
utrecht	Nederberg	2.8506	0.0745
utrecht	Nieuw Engeland, Schepenbuurt	1.6627	0.0771
utrecht	Nieuwland	1.7935	0.0485
utrecht	Oog in Al	1.8567	0.0712
utrecht	Randenbroek	1.8051	0.0548
utrecht	Rustenburg	1.5527	0.0391
utrecht	Schothorst-Noord	1.4759	0.0733
utrecht	Schothorst-Zuid	2.1820	0.0573
utrecht	Schuilenburg	1.7128	0.0545
utrecht	Soesterkwartier	1.5328	0.0713
utrecht	Stadskern	2.4995	0.0617
utrecht	Terwijde	1.9847	0.0495
utrecht	Vathorst-Centrum	2.2904	0.0640
utrecht	Vathorst-De Laak	1.5928	0.0694
utrecht	Vathorst-De Velden	1.8342	0.0512
utrecht	Veldhuizen	1.5004	0.0522
utrecht	Vermeerkwartier	1.5999	0.0561
utrecht	Zielhorst	1.5044	0.0603
gelderland	Aldenhof	2.0766	0.0541
gelderland	Altrade	1.9145	0.0631
gelderland	Benedenstad	2.2511	0.0570
gelderland	Biezen	1.9326	0.0533
gelderland	Bijsterhuizen	1.5348	0.1253
gelderland	Bottendaal	2.0493	0.0685
gelderland	Brakkenstein	1.2074	0.0575
gelderland	Centrum	0.3140	0.0716
gelderland	De Hoop	1.6989	0.0502
gelderland	De Horsten	2.0652	0.0679
gelderland	De Huet	1.6183	0.0415
gelderland	De Kamp	2.2876	0.0634
gelderland	De Pas	1.8306	0.0585
gelderland	Dichteren	1.7735	0.0806
gelderland	Druten-Zuid	1.4594	0.0520
gelderland	Ede-Zuid	1.3725	0.0577
gelderland	Enka	2.0257	0.0494
gelderland	Galgenveld	1.8439	0.0669
gelderland	Goffert	1.2599	0.0677
gelderland	Groenewoud	1.4407	0.0513

Province	Neighborhood	β	MAE
gelderland	Grootstal	1.5614	0.0538
gelderland	Hatert	1.6350	0.0476
gelderland	Haven- en industrieterrein	1.5214	0.0819
gelderland	Hazenkamp	1.8644	0.0538
gelderland	Hees	1.5451	0.0449
gelderland	Heideslag	1.6251	0.1398
gelderland	Heijendaal	1.5946	0.1006
gelderland	Hengstdal	1.6642	0.0423
gelderland	Heseveld	1.8577	0.0648
gelderland	Hunnerberg	1.6298	0.0748
gelderland	Kerkenbos	1.7384	0.0767
gelderland	Kernhem	2.0103	0.1282
gelderland	Kortenoord	1.8580	0.0412
gelderland	Kwakkenberg	1.4237	0.0661
gelderland	Lankforst	2.0800	0.0457
gelderland	Lent	1.5665	0.0743
gelderland	Maandereng	1.6415	0.0480
gelderland	Malvert	2.1752	0.0531
gelderland	Meijhorst	2.4224	0.0460
gelderland	Methen	1.7545	0.0481
gelderland	Neerbosch-Oost	1.8381	0.0735
gelderland	Nije Veld	1.5649	0.0516
gelderland	Noordwest	1.4672	0.0394
gelderland	Nude	1.5097	0.0492
gelderland	Oosseld	1.3981	0.0575
gelderland	Oosterhout	1.7439	0.0624
gelderland	Ooyse Schependom	1.9227	0.0974
gelderland	Overstegen	1.6659	0.0416
gelderland	Ressen	1.3153	0.1419
gelderland	Rietkampen	1.8911	0.0471
gelderland	Rijkerswoerd	1.6584	0.0530
gelderland	Roodwilligen	2.0570	0.0463
gelderland	Schöneveld	1.5785	0.0454
gelderland	Staddijk	2.2891	0.0400
gelderland	Stadscentrum	1.9654	0.0698
gelderland	Stadsweiden	1.9375	0.0477
gelderland	Tolhuis	2.1971	0.0858
gelderland	Veldhuizen	1.6340	0.0466
gelderland	Weezenhof	2.2244	0.0492
gelderland	Westkanaaldijk	1.8224	0.0485
gelderland	Wijnbergen	1.9374	0.0463
gelderland	Wolfskuil	1.8719	0.0678
gelderland	Zwanenveld	2.0639	0.0454
zuid holland	Afrikaanderwijk	2.4434	0.0684
zuid holland	Berkel	1.7835	0.0543
zuid holland	Beverwaard	1.4109	0.0508
zuid holland	Binnenstad	0.4081	0.0582

Province	Neighborhood	β	MAE
zuid holland	Bloemendaal	1.4879	0.0473
zuid holland	Bloemhof	2.4347	0.0742
zuid holland	Bospolder	2.6953	0.0804
zuid holland	Buitenhof	1.6193	0.0710
zuid holland	Capelle-West	1.7229	0.0683
zuid holland	Carnisse	2.2508	0.0716
zuid holland	Cool	2.9001	0.0905
zuid holland	De Schenkel	1.8490	0.0418
zuid holland	Delfshaven	2.3753	0.0767
zuid holland	Fascinatio	1.4135	0.1003
zuid holland	Feijenoord	1.7912	0.0600
zuid holland	Goverwelle	1.5327	0.0677
zuid holland	Groente- en Fruitmarkt	2.0943	0.0668
zuid holland	Groot-IJsselmonde	1.4445	0.0507
zuid holland	Heijplaat	1.7904	0.0814
zuid holland	Hellevoet	1.8013	0.0683
zuid holland	Het Lage Land	1.5726	0.0652
zuid holland	Hillegersberg-Noord	1.2025	0.0905
zuid holland	Hillesluis	2.1687	0.0675
zuid holland	Hof van Delft	1.4801	0.0623
zuid holland	Hoog Dalem	1.8175	0.0493
zuid holland	Katendrecht	2.3424	0.0793
zuid holland	Kleinpolder	1.5265	0.0486
zuid holland	Kleiwegkwartier	1.7037	0.0856
zuid holland	Kop van Zuid	4.0657	0.1805
zuid holland	Kop van Zuid-Entrepot	2.2315	0.0742
zuid holland	Kort Haarlem	1.6016	0.0577
zuid holland	Korte Akkeren	1.4279	0.0787
zuid holland	Leyenburg	1.8086	0.0688
zuid holland	Lombardijen	1.5642	0.0523
zuid holland	Middelland	2.8496	0.0876
zuid holland	Middelwatering	1.4986	0.0605
zuid holland	Moerwijk	1.7443	0.0577
zuid holland	Molenlaankwartier	1.4241	0.0711
zuid holland	Morgenstond	2.2738	0.0623
zuid holland	Nesselande	1.5891	0.0577
zuid holland	Nieuw-Helvoet	1.4695	0.0554
zuid holland	Nieuw-Mathenesse	1.8685	0.0877
zuid holland	Nieuwe Westen	1.9595	0.0686
zuid holland	Noord	1.5273	0.0598
zuid holland	Noordereiland	2.7948	0.0692
zuid holland	Ommoord	1.6854	0.0467
zuid holland	Onyx	2.0357	0.0560
zuid holland	Oosterflank	1.8200	0.0659
zuid holland	Oostgaarde	1.6401	0.0603
zuid holland	Oud-Charlois	1.8879	0.0651
zuid holland	Oud-IJsselmonde	1.4746	0.0742

Province	Neighborhood	β	MAE
zuid holland	Oud-Mathenesse	1.9725	0.0689
zuid holland	Oude Westen	2.7802	0.0779
zuid holland	Overschie	1.3662	0.0645
zuid holland	Pendrecht	1.5824	0.0541
zuid holland	Plaswijck	1.6438	0.0566
zuid holland	Portlandsehoek	1.9753	0.0412
zuid holland	Prinsenland	1.5927	0.0507
zuid holland	RijswijkBuiten	2.0306	0.0576
zuid holland	Rivium	1.7086	0.0609
zuid holland	Ruiven	0.9450	0.1013
zuid holland	Rustenburg en Oostbroek	2.3427	0.0688
zuid holland	Schenkel	1.6088	0.0636
zuid holland	Schiebroek	1.1906	0.0789
zuid holland	Schieweg	1.5169	0.0911
zuid holland	Schollevaar	1.6300	0.0874
zuid holland	Spangen	2.3839	0.0717
zuid holland	Stadsdriehoek	1.9881	0.0604
zuid holland	Stolersluis	1.2012	0.0961
zuid holland	Tanthof-Oost	1.6962	0.0560
zuid holland	Tanthof-West	1.6227	0.0706
zuid holland	Tarwewijk	2.1051	0.0819
zuid holland	Transvaalkwartier	2.3829	0.0733
zuid holland	Tussendijken	2.5872	0.0862
zuid holland	Vogelbuurt	1.8942	0.0458
zuid holland	Voordijkshoorn	1.9000	0.0613
zuid holland	Voorhof	1.7714	0.0518
zuid holland	Vreewijk	2.0712	0.0642
zuid holland	Vrijenban	1.5214	0.0960
zuid holland	Wateringse Veld	1.6834	0.0436
zuid holland	Westergouwe	1.4773	0.0904
zuid holland	Westplaat	1.5768	0.0468
zuid holland	Westpolder	1.9945	0.0918
zuid holland	Wielewaal	2.0511	0.0801
zuid holland	Wippolder	1.3909	0.0589
zuid holland	Zestienhoven	1.3321	0.0696
zuid holland	Zevenkamp	1.5333	0.0567
zuid holland	Zuid	1.6497	0.0625
zuid holland	Zuiderpark	1.7275	0.0658
zuid holland	Zuidwijk	1.5043	0.0535
noord brabant	Binnenstad	1.4904	0.0524
noord brabant	Brandevoort	1.2853	0.0983
noord brabant	Brouwhuis	1.2554	0.0870
noord brabant	Dierdonk	1.1021	0.1307
noord brabant	Helmond-Noord	1.4433	0.0457
noord brabant	Helmond-Oost	1.4658	0.0586
noord brabant	Helmond-West	1.6054	0.0669
noord brabant	Industriegebied Zuid	1.2557	0.0645

Province	Neighborhood	β	MAE
noord brabant	Mierlo-Hout	1.1247	0.0448
noord brabant	Oosterheide	1.5006	0.0734
noord brabant	Rijpelberg	1.0351	0.2101
noord brabant	Stiphout	1.0604	0.0757
noord brabant	Warande	1.5723	0.0609
noord brabant	West	1.2322	0.0813
noord brabant	Zuidoost	1.0077	0.1249
limburg	Blerick-Midden	1.5368	0.0536
limburg	Blerick-Noord	1.7316	0.0616
limburg	Blerick-Zuid	1.6044	0.0514
limburg	Boekend	0.9122	0.1203
limburg	Centrum	1.5512	0.0688
limburg	Donderberg	1.5472	0.0456
limburg	Hoogvonderen	1.8441	0.0443
limburg	Hout-Blerick	1.0943	0.1004
limburg	Klingerberg	1.3920	0.0681
limburg	Lindenheuvel	1.3033	0.0639
limburg	Maasniel	1.1914	0.0731
limburg	Maastricht-Centrum	1.9192	0.0603
limburg	Maastricht-Noordwest	1.3642	0.0744
limburg	Maastricht-Oost	1.2927	0.0515
limburg	Maastricht-West	1.3255	0.0477
limburg	Maastricht-Zuidoost	1.1853	0.0545
limburg	Meuleveld	1.9103	0.0494
limburg	Roermond-Oost	1.4925	0.0556
limburg	Roermond-Zuid	1.0161	0.1064
limburg	Sanderbout	1.9400	0.0843
limburg	Vossener	1.6278	0.0536
zeeland	Binnenstad	1.5282	0.0744
zeeland	Burgh	1.6157	0.0744
zeeland	Haamstede	1.6948	0.0539
zeeland	Lammerenburg	1.4488	0.0650
zeeland	Middelburg-Zuid	1.6237	0.0643
zeeland	Othene	1.6544	0.0521
zeeland	Paauwenburg	1.3216	0.0768
zeeland	Terneuzen-Centrum	1.8812	0.0663
zeeland	Terneuzen-Noord	1.5158	0.0466
zeeland	Terneuzen-West	1.7544	0.0663
zeeland	Terneuzen-Zuid	1.5198	0.0457
zeeland	West-Souburg	1.7924	0.0512