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

Koen Stevens

May 20, 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) find the relation:

$$\frac{L}{\sqrt{n}} \to \beta\sqrt{A}, \quad \text{as } n \to \infty \tag{1}$$

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 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 the following conditions:

(a) The locations are drawn from the set of postcodes in the area in question, a relaxation of the uniform distribution of visited locations. In reality, delivery locations for companies are not uniformly distributed across the delivery area, but they are clustered in certain parts of the area, for example in high rise buildings;

(b) Applied to realistically sized (in the sense that a delivery person could serve this area in a single workday) real-world parts of cities and villages in the Netherlands;

2. A supervised 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 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 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)), where distances between

locations are measured over the actual road network. An extension of formula 1 that works for VRPs is assessed in this setting. It is found that this model has an R^2 of 0.99 (which means 0.99 of the variation in TSP path length is explained by the formula) 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 for locations p and q is defined as:

$$c = \frac{d_c(p,q)}{d_{L_2}(p,q)} \tag{2}$$

By definition, $c \geq 1$, as the Euclidean distance is the shortest possible path between two points in a plane. 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.

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

It is important to note, 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 may vary significantly within a single city, depending on local street patterns, infrastructure, and topography. This seems plausible given observed variability in urban design, although further empirical investigation would be required to confirm this. 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).

3 Problem definition

This section provides a formal definition of the problem this research aims to solve. Define N to be a set of neighborhoods, and define X_n the set of all postal codes in a neighborhood $n \in N$. Let $TSP_{j,n} = \{x_i\}_{i=1}^{j}$, $x_i \in X_n$ be a subset of size j of the postal codes in neighborhood n. Then, $L_{j,n}$ can be defined to be the length of the shortest path through the set $TSP_{j,n}$. This is the length of the solution of the traveling salesman problem consisting of the points in $TSP_{j,n}$. This research consists of two parts.

1. Analyze the behavior of $L_{j,n}$ for various sample sizes j and neighborhoods n, in the context of evaluating the accuracy of predictions made by Equation 1.

2. Investigate how well TSP path length can be predicted using j and a number of features of the neighborhood the TSP is in, using supervised learning.

4 Data

In this section, a detailed explanation of the data is provided. This includes the characteristics of the data used and how this data is processed.

4.1 Source

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`, in order to be able to efficiently use the data with `Python`.

4.2 Description

For this analysis, the database has three interesting tables: `planet_osm_polygon`, `planet_osm_nodes` and `planet_osm_ways`. Many neighborhoods 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 consists 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 someone's 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.

In order to perform the supervised learning analysis, the second contribution of this research, features of the neighborhoods need to be extracted. In the `planet_osm_ways` table, not only roads are stored, there is also a large amount of other objects stored in here, such as parks, bodies of water, what a certain area is used for (i.e. residential, commercial) and other natural or man-made objects. Features like this might provide predictive value for TSP path length.

4.3 Graph construction

Using the geographic information of the roads and buildings, a graph is constructed for each neighborhood, 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 implemented 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 1 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 (φ_A, θ_A), (φ_B, θ_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:

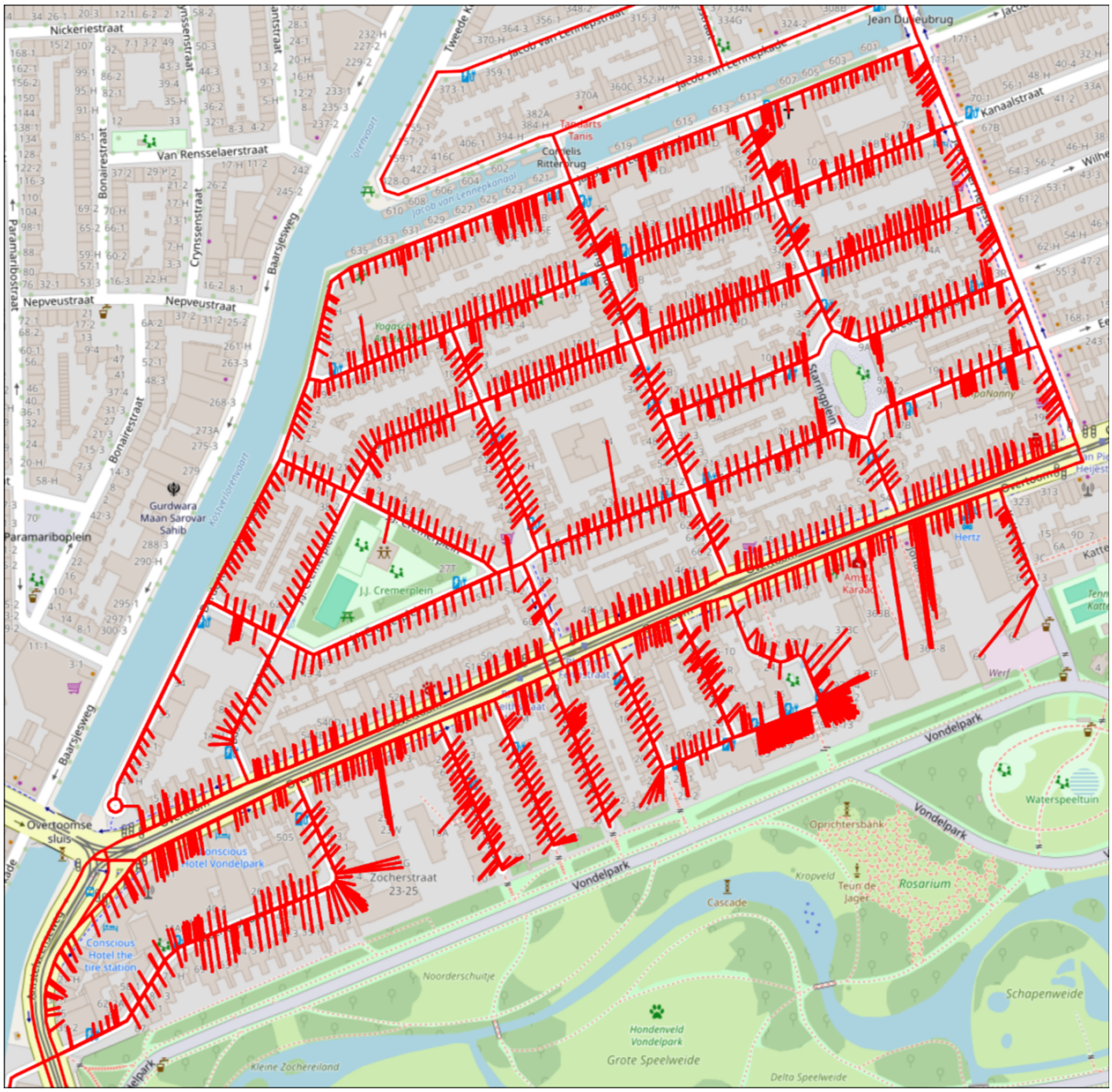
$$d(A,B) = R\sqrt{\varphi^2 + \theta^2}, \tag{3}$$

$$\text{where } \varphi = (\varphi_B - \varphi_A)\cos\left[\frac{\theta_A + \theta_B}{2}\right], \tag{4}$$

$$\text{and } \theta = \theta_B - \theta_A. \tag{5}$$

Using the `folium` module in `Python`, such a graph can be visualized on a geographic map. One of such visualizations is provided in figure 1. All maps like this one, for the areas in this analysis can be viewed and interacted with via this `webpage`.

Figure 1: Visualization of the graph, for the Overtoomse Sluis quarter in Amsterdam.



4.4 Extracting features

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 significantly overestimates the correct value for A. For many of the quarters, there is some area around where the buildings are, included in the quarter. 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 2 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. Note that this is still not a good estimate for A in some cases, for example when there is and area inside the neighborhood without addresses (for example a park or lake) in the neighborhood. When using the convex hull, this area (or at least a portion of it) is still included in A, which is still not the optimal way of calculating the area, although still better than using the pre-defined area in the data.

Figure 2: The OpenStreetMap quarter for the inner city of Groningen (Binnenstad). (OpenStreetMap contributors, 2025)

While the area of a neighborhood is an important predictor, it is only one of many features considered in this analysis. To train a supervised learning model that predicts the TSP path length based on neighborhood characteristics, a comprehensive set of features needs to be extracted.

In total, approximately 80 features are gathered for each neighborhood. These features describe various aspects of the built environment, infrastructure, and spatial distribution of delivery locations. They fall broadly into the following categories:

- **Geographic features:** Including the area of the convex hull around all postcodes, characteristics such as how much water or how many parks, and land use (i.e. residential or commercial).
- **Building characteristics:** number of buildings divided by the area of the convex hull.
- **Road network metrics:** Percentage of roads that are one-way, total length of road network divided by the area of the convex hull.

5 Experimental design

This section provides an overview of the methods and algorithms used to solve these problems.

5.1 Evaluation of BHH formula

The following algorithm is used for the first part of the research. Let $L = \{L_i\}$ and $\hat{L} = \{\hat{L}_i\}$ be the lists of observed and predicted TSP tour lengths, respectively, aggregated over all sample sizes j and repetitions k for neighborhood n. Let $|L|$ denote the total number of evaluated instances per neighborhood.

Algorithm 1 Procedure for evaluating the predictive accuracy of Equation 1

for each neighborhood $n \in N$ **do**
  Extract features of this neighborhood and save to a file for later use
  Construct a graph of the road network and visualize on a map
  Initialize empty lists for L and $\hat{L}$
  for each sample size $j \in \{20, 22, \ldots, 86, 88\}$ **do**
    for each repetition $k \in \{0, 1, \ldots, 10\}$ **do**
      Randomly sample a subset $TSP_{j,n} \subset X_n$ of size j uniformly
      Solve the TSP over $TSP_{j,n}$ to compute $L_{j,n}^{(k)}$
      Solve Equation 1 for β using these solutions $L_{j,n}^{(k)}$
      Compute the predicted length $\hat{L}_{j,n}^{(k)}$ using Equation 1 with the estimated β
      Append $L_{j,n}^{(k)}$ to list L, and $\hat{L}_{j,n}^{(k)}$ to list $\hat{L}$
  Compute the Mean Absolute Error (MAE) for neighborhood n:

$$\text{MAE}_n = \frac{1}{|L|} \sum_{i=1}^{|L|} \left| L_i - \hat{L}_i \right|$$

  Compute the Mean Absolute Percentage Error (MAPE) for neighborhood n:

$$\text{MAPE}_n = \frac{100\%}{|L|} \sum_{i=1}^{|L|} \left| \frac{L_i - \hat{L}_i}{L_i} \right|$$

  Visualize the results in graph form, a scatter plot of TSP length against number of locations, with a fitted equation 1 and a histogram of the errors $\epsilon_i = L_i - \hat{L}_i$.
  Save the results for L to a file for later use.

5.2 Supervised learning method

In the second part of this research, a supervised learning model is trained to predict the TSP path length from neighborhood-level features and the number of points j. Specifically, a Random Forest regression model is employed.

The data set is constructed using the results from Algorithm 1. Each entry of L (the list of TSP path lengths) is a separate observation in this data set. For every observation, the corresponding area features and the number of locations j is stored in the data set. Then, the following steps are performed:

1. Split the data into a training and test set, 80/20 split.
2. Normalize the features such that they all have mean 0 and variance 1.
3. Train a Random Forest regressor to predict TSP path length L using the number of points j and neighborhood-level features as input variables.
4. Evaluate model performance on the test set using the metrics (for the test data):
   - Mean Absolute Error (MAE)
   - Mean Squared Error (MSE)
   - Coefficient of determination R^2

The goal of this part is to determine how accurately TSP path length can be predicted based on readily available spatial and demographic characteristics of neighborhoods.

6 Results

In this section the results of the analysis that is described above is laid out. First the performance of Equation 1 is discussed, then the performance of the supervised learning approach is analyzed.

6.1 Evaluation of BHH formula

7 Conclusion

8 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.

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)*.

9 Appendix

9.1 Listings

Listing 1: 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))
```

9.2 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	3.5339	727.0653	7.6859
groningen-Binnenstad	2.8388	945.7670	6.6827
groningen-Oosterpoort	2.9193	658.6700	7.0928
groningen-Rivierenbuurt	3.6107	565.7389	7.0225
groningen-De Wijert	2.1588	706.6520	5.5321
groningen-Oosterparkwijk	2.6482	959.3695	6.1824
groningen-De Hoogte	2.9187	879.5585	8.5084
groningen-Korrewegwijk	2.6109	1012.9386	6.3294
groningen-Schildersbuurt	2.8402	595.4837	6.1082
groningen-Paddepoel	2.1387	593.5363	4.8357
groningen-Oranjewijk	2.8117	858.1195	7.3867
groningen-Tuinwijk	5.9951	1161.7814	14.5892
groningen-Selwerd	2.9315	657.7040	6.9024
groningen-Vinkhuizen	1.8359	538.2826	4.5054
groningen-Hoogkerk-zuid	2.0945	759.6465	6.8944
groningen-Gravenburg	2.0262	1589.0864	15.3712
groningen-De Held	3.3808	399.1739	4.2839
groningen-Reitdiep	2.2913	444.5958	4.3902
groningen-Hoornse Meer	2.2936	635.5882	6.7377
groningen-Corpus den Hoorn	2.5203	865.1621	7.5280
groningen-Eemspoort	2.9767	535.7333	4.9702
groningen-Euvelgunne	3.3373	1052.7148	9.7566
groningen-Driebond	3.0743	1090.1420	10.4602
groningen-Winschoterdiep	3.7654	2261.2006	15.3150
groningen-Eemskanaal	2.8624	533.5686	4.1314
groningen-Helpman	3.0333	838.9278	7.0836
groningen-Lewenborg	3.0920	1237.3884	6.3119
groningen-Beijum	2.3392	741.9949	4.4258
groningen-Maarsveld	2.6094	391.2032	4.9862
drenthe-Assen Oost	2.0025	1200.6360	5.8485
drenthe-Assen West	2.0257	1877.6603	10.9386
drenthe-Centrum	2.3252	1242.5597	6.9760
drenthe-Kloosterveen	2.0000	2106.5607	9.4208
drenthe-Lariks	2.1699	984.1017	7.3239

Province-Neighborhood	β	MAE (m)	MAPE (%)
drenthe-Marsdijk	1.9801	1813.0542	7.7146
drenthe-Noorderpark	2.1843	791.7199	5.1452
drenthe-Peelo	2.4662	929.8540	6.8506
drenthe-Pittelo	2.2824	691.4879	5.8984
drenthe-Ter Borch	2.5752	785.8678	5.5075
friesland-Achter de Hoven	2.5910	497.5225	7.6863
friesland-Aldlân	2.1495	841.9472	5.6392
friesland-Bilgaard	2.2841	850.0155	7.1729
friesland-Binnenstad en Stationskwartier	3.1541	889.0453	6.8955
friesland-Blitsaerd	2.5324	594.8259	5.3122
friesland-Buitengebied Noordwest	2.4701	1716.9913	7.6828
friesland-Camminghaburen	2.1439	751.5120	4.5905
friesland-De Hemrik	2.3984	747.3613	5.2427
friesland-De Zuidlanden	2.0929	927.0308	7.2644
friesland-De Zwette	2.7263	1620.1902	7.4431
friesland-Heechterp	1.9583	790.5178	12.2359
friesland-Hempens, Teerns, en Zuiderburen	2.2269	1155.2527	5.6225
friesland-Huizum-Oost	2.6184	616.7353	5.8553
friesland-Huizum-West	1.9839	658.0781	6.5048
friesland-Middelsee	2.3395	833.2005	10.1655
friesland-Nijlân	2.1580	499.5333	5.5088
friesland-Oranjewijk	3.2951	468.7933	6.2214
friesland-Oud Oost	1.9170	716.6742	5.4583
friesland-Schepenbuurt	2.1638	1050.4308	17.7603
friesland-Schieringen en De Centrale	2.5555	822.4384	9.8182
friesland-Techum	2.9018	461.9544	5.4775
friesland-Vlietzone	2.1078	501.9054	5.7278
friesland-Vogelwijk en Componistenbuurt	2.6567	430.1407	5.9155
friesland-Vrijheidswijk	2.8497	469.3865	5.2078
friesland-Westeinde	2.8245	388.2131	3.8764
friesland-Wielenpôlle	2.4770	790.3753	13.0510
friesland-Zuiderburen	2.4625	1169.6306	5.6943
flevoland-Polderwijk	3.2421	1168.0344	6.5423
overijssel-Baalder	2.1182	633.7329	6.3100
overijssel-Baalderveld	2.0892	986.0537	7.9558
overijssel-Berflo Es	2.0031	1058.2400	6.5051
overijssel-Bergweide	2.8875	947.3912	6.6569
overijssel-De Graven Es	2.3973	1506.6178	9.7635
overijssel-De Meijbree	2.7308	334.3680	5.1386
overijssel-De Riet	2.4211	582.6573	5.0100
overijssel-De Thij	1.9277	865.6980	6.5588
overijssel-Groot Driene	2.0992	785.2460	4.4491
overijssel-Haardijk	3.3307	792.4527	8.0839
overijssel-Hanzeland	3.0449	749.5714	10.5239
overijssel-Hasseler Es	1.9933	964.2826	5.4937
overijssel-Heemsermars	1.4597	343.6100	5.1421
overijssel-Het Onderdijks	2.3015	310.5288	3.7790

Province-Neighborhood	β	MAE (m)	MAPE (%)
overijssel-Hofkamp	2.1864	596.4896	5.3561
overijssel-Ittersum	2.0088	1219.5755	6.3532
overijssel-Keizerslanden	2.3742	1711.9010	9.7167
overijssel-Kloosterlanden	2.5233	889.1588	6.0477
overijssel-Marslanden	3.3647	1676.9882	10.2738
overijssel-Nieuwe Haven	3.2174	731.3536	5.4597
overijssel-Nieuwstraatkwartier	2.4444	457.0575	5.5113
overijssel-Noorderkwartier	1.9962	535.0191	4.8134
overijssel-OosterDalfsen	3.6083	284.1659	5.0704
overijssel-Ossenkoppelerhoek	2.1378	685.3150	5.4801
overijssel-Rivierenwijk	2.3779	456.5307	4.9535
overijssel-Schelfhorst	2.2123	771.2058	4.7327
overijssel-Slangenbeek	1.9166	1047.3283	5.3302
overijssel-Sluitersveld	2.0832	851.9876	5.9730
overijssel-Twekkelerveld	1.8288	651.0840	4.6910
overijssel-Veerallee	2.8231	667.4959	9.1617
overijssel-Voorstad	1.8231	982.9321	7.6580
overijssel-Wierdensehoek	2.4130	797.5979	5.9344
overijssel-Windmolenbroek	1.9876	1362.2382	5.8683
overijssel-Zandweerd	1.6568	594.2441	4.8177
noord holland-Schrijverswijk	2.4975	439.2479	4.7672
noord holland-Stad van de Zon	2.2334	2374.6580	18.4874
noord holland-Stadshart	3.9373	501.7207	5.1614
noord holland-Jordaan	2.7812	1048.5404	6.0221
noord holland-Slotervaart	1.9538	687.6353	4.8969
noord holland-IJburg	2.0558	893.8233	4.9919
noord holland-Oostelijke Eilanden	2.4006	479.3620	4.6300
noord holland-Oostelijk Havengebied	3.2151	1115.7897	5.3388
noord holland-Frederik Hendrikbuurt	3.9462	908.9105	8.2330
noord holland-Van Lennepbuurt	3.8147	774.7296	6.6922
noord holland-Da Costabuurt	7.6057	1005.9659	9.7311
noord holland-Kinkerbuurt	5.9117	929.8317	9.0461
noord holland-Kersenboogerd	2.0551	1072.6652	5.3154
noord holland-Pax	3.2297	680.0580	5.2167
noord holland-Graan voor Visch	2.8983	520.1579	5.6736
noord holland-Vrijschot-Noord	4.2031	580.1329	7.0226
noord holland-Toolenburg	2.3649	1027.0372	4.6654
noord holland-Floriande	2.5407	1127.2496	4.5550
noord holland-Overbos	2.1078	678.0740	4.6185
noord holland-Bornholm	2.4628	548.3246	4.0572
noord holland-Beukenhorst-Oost	3.7188	930.4855	7.7276
noord holland-De Hoek	4.3033	679.2393	5.2951
noord holland-West	2.5032	301.6956	4.9016
noord holland-Zuid	3.9029	874.1950	6.0621
noord holland-Oost	2.8828	763.2687	5.6653
noord holland-Noord	2.4511	599.4562	5.4559
noord holland-De President	2.2779	979.7524	12.1838

Province-Neighborhood	β	MAE (m)	MAPE (%)
noord holland-Graan voor Visch-Zuid	3.3932	590.9427	6.5459
noord holland-Zuidwijk	2.8922	432.3491	4.3163
noord holland-Buitenveldert-West	1.7087	1229.5719	7.2227
noord holland-Buitenveldert	1.5997	1334.8021	6.9965
noord holland-Apollobuurt	2.4099	867.3723	7.3299
noord holland-Stadionbuurt	2.4129	820.9004	7.3479
noord holland-Prinses Irenebuurt e.o.	3.4645	462.0289	6.4356
noord holland-Hoofddorppleinbuurt	2.7097	1167.4108	9.0199
noord holland-Willemspark	4.5669	937.6364	8.7341
noord holland-Schinkelbuurt	5.4453	1231.6013	11.7715
noord holland-Vondelparkbuurt	4.6890	738.3031	8.0135
noord holland-Helmersbuurt	3.5457	878.2293	7.8040
noord holland-Overtoomse Sluis	5.7106	752.8070	7.1547
noord holland-Museumkwartier	2.3359	1082.9057	7.6957
noord holland-Rivierenbuurt	2.2312	1147.1064	6.5338
noord holland-IJselbuurt	4.4026	947.2657	8.2520
noord holland-Scheldebuurt	3.0295	970.3637	6.9348
noord holland-Rijnbuurt	3.3822	534.4458	5.7860
noord holland-De Baarsjes	2.4769	1292.5096	6.6227
noord holland-Landlust	2.9761	1006.6387	7.3041
noord holland-Staatsliedenbuurt	2.3739	705.2967	7.2695
noord holland-Spaarndammerbuurt	3.9201	793.8784	7.2535
noord holland-De Pijp	2.5893	1364.8035	6.9982
noord holland-Grachtengordel	3.0265	1385.3247	7.0441
noord holland-Oud-Zuid	1.7976	1421.3800	5.8488
utrecht-Bedrijventerrein Vathorst	3.9651	1021.0497	6.6272
utrecht-Binnenstad City- En Winkelgebied	1.9418	620.1733	6.6015
utrecht-Binnenstad Woongebied	3.3696	1119.9648	5.9822
utrecht-Bosgebied	2.0821	1334.9264	6.6662
utrecht-Buitengebied Oost	1.9787	1497.3125	7.5809
utrecht-Calveen	3.5950	886.3555	9.1023
utrecht-De Berg-Noord	2.1228	740.8549	5.3759
utrecht-De Berg-Zuid	2.1515	824.6435	5.9790
utrecht-De Hoef	2.7785	1123.0882	9.4997
utrecht-De Koppel	2.7821	404.2406	5.1074
utrecht-Dichterswijk, Rivierenwijk	2.1396	792.0172	5.5268
utrecht-Eemkwartier	3.7914	554.8346	7.1873
utrecht-Hoogland	2.2642	984.2913	5.8228
utrecht-Hoogland-West	1.8327	1771.1340	6.5769
utrecht-Hooglanderveen	3.0780	717.1319	5.1563
utrecht-Isselt	2.6708	710.9079	4.8516
utrecht-Kanaleneiland	2.1108	650.2125	4.7998
utrecht-Kattenbroek	2.3699	661.8285	4.5277
utrecht-Kruiskamp	2.5919	539.1563	5.2446
utrecht-Leusderkwartier	1.9971	1146.1960	9.0074
utrecht-Liendert	2.4899	596.0238	4.8948
utrecht-Lombok, Leidseweg	2.4966	925.2282	6.6532

Province-Neighborhood	β	MAE (m)	MAPE (%)
utrecht-Nederberg	4.7971	709.0796	7.0463
utrecht-Nieuw Engeland, Schepenbuurt	2.4889	2140.2897	7.3533
utrecht-Nieuwland	2.3715	823.6962	5.0004
utrecht-Oog in Al	2.3990	959.0929	7.3277
utrecht-Randenbroek	2.4678	716.2161	5.3915
utrecht-Rustenburg	2.2353	301.8927	3.7402
utrecht-Schothorst-Noord	1.9845	879.2264	7.0617
utrecht-Schothorst-Zuid	3.1157	673.7437	5.2252
utrecht-Schuilenburg	2.4017	500.4915	5.8140
utrecht-Soesterkwartier	2.2073	941.5048	6.6668
utrecht-Stadskern	3.7056	828.5688	5.9648
utrecht-Terwijde	2.6678	836.9398	4.9889
utrecht-Vathorst-Centrum	3.7350	712.6952	5.8285
utrecht-Vathorst-De Laak	2.3985	1236.5632	6.5892
utrecht-Vathorst-De Velden	2.6537	739.1278	4.7713
utrecht-Veldhuizen	2.1381	613.0585	5.3329
utrecht-Vermeerkwartier	2.0948	687.4064	5.6960
utrecht-Zielhorst	2.4405	737.5802	6.1880
gelderland-Aldenhof	3.2478	433.9962	5.2018
gelderland-Altrade	2.6800	698.2621	6.0302
gelderland-Benedenstad	3.0954	461.7761	5.9228
gelderland-Biezen	2.6763	752.0586	5.4544
gelderland-Bijsterhuizen	3.6802	1297.4129	13.4444
gelderland-Bottendaal	3.5713	543.1354	6.6558
gelderland-Brakkenstein	1.7686	581.6886	6.1482
gelderland-De Hoop	2.6156	372.7379	4.9840
gelderland-De Horsten	3.0714	514.9788	7.3186
gelderland-De Huet	2.0399	635.6865	4.3236
gelderland-De Kamp	3.1382	1062.2622	7.1355
gelderland-De Pas	2.8751	538.1064	6.2808
gelderland-Dichteren	2.7341	1078.1779	7.8817
gelderland-Druten-Zuid	1.9570	735.2720	5.1219
gelderland-Ede-Zuid	1.8153	794.8912	6.2643
gelderland-Enka	2.8667	407.3000	4.5612
gelderland-Galgenveld	2.6778	785.2952	6.5649
gelderland-Goffert	2.1089	1076.0512	6.9497
gelderland-Groenewoud	2.5100	658.1102	5.4391
gelderland-Grootstal	2.3828	665.0571	5.9305
gelderland-Hatert	2.1649	706.3607	4.9164
gelderland-Haven- en industrieterrein	2.6808	1392.6804	8.2359
gelderland-Hazenkamp	2.2507	641.5964	5.2979
gelderland-Hees	2.4586	545.6828	4.9533
gelderland-Heideslag	2.6076	975.2410	13.8766
gelderland-Heijendaal	2.7466	1200.7609	9.4720
gelderland-Hengstdal	2.4657	547.5640	4.6460
gelderland-Heseveld	2.7133	809.3034	6.2825
gelderland-Hunnerberg	2.4426	951.1800	8.0527

Province-Neighborhood	β	MAE (m)	MAPE (%)
gelderland-Kerkenbos	2.7842	693.6390	8.4550
gelderland-Kernhem	2.2896	1850.1112	12.4176
gelderland-Kortenoord	3.1116	384.3648	3.9080
gelderland-Kwakkenberg	2.0866	756.0676	6.5955
gelderland-Lankforst	3.7527	386.7511	4.7766
gelderland-Lent	2.3546	1618.6980	6.9690
gelderland-Maandereng	2.1318	440.8507	4.4540
gelderland-Malvert	3.8077	466.3427	5.1391
gelderland-Meijhorst	4.3041	572.9487	5.0323
gelderland-Methen	2.4783	425.6924	4.3395
gelderland-Neerbosch-Oost	2.5813	1063.6562	7.6334
gelderland-Nije Veld	2.3657	585.2526	5.3024
gelderland-Noordwest	2.3877	410.9665	4.1394
gelderland-Nude	2.2204	409.0421	4.1757
gelderland-Oosseld	1.9905	651.8807	5.8749
gelderland-Oosterhout	2.5573	920.4550	5.8349
gelderland-Ooyse Schependom	3.7717	452.2295	9.6099
gelderland-Overstegen	2.7676	619.1005	4.6336
gelderland-Ressen	2.0614	2477.5349	16.5196
gelderland-Rietkampen	3.0399	661.3551	4.8568
gelderland-Rijkerswoerd	2.0819	791.6755	5.0261
gelderland-Roodwilligen	2.9153	154.8905	4.5237
gelderland-Schöneveld	2.5359	348.5303	4.3869
gelderland-Staddijk	3.3406	605.8993	3.8934
gelderland-Stadscentrum	2.7884	970.3173	7.0195
gelderland-Stadsweiden	2.2674	564.7706	4.2772
gelderland-Tolhuis	3.4271	1032.3502	7.6911
gelderland-Veldhuizen	2.0775	632.9743	4.3029
gelderland-Weezenhof	3.1640	652.8908	4.8969
gelderland-Westkanaaldijk	3.0967	613.0853	4.7775
gelderland-Wijnbergen	2.7985	341.6040	4.7918
gelderland-Wolfskuil	2.7179	641.3040	5.8110
gelderland-Zwanenveld	3.1812	480.3584	4.3102
zuid holland-Afrikaanderwijk	4.1298	721.8422	6.6210
zuid holland-Berkel	2.4437	672.5325	5.6182
zuid holland-Beverwaard	1.6731	577.7932	5.1595
zuid holland-Binnenstad	1.2545	1071.5382	6.0496
zuid holland-Bloemendaal	2.0469	877.6564	5.0235
zuid holland-Bloemhof	3.1819	1100.6165	7.5850
zuid holland-Bospolder	3.1527	839.1196	7.7513
zuid holland-Buitenhof	2.6534	1221.2619	7.8173
zuid holland-Capelle-West	2.5288	685.4383	7.6121
zuid holland-Carnisse	2.8566	990.3210	8.1649
zuid holland-Cool	4.3052	1269.3949	8.3745
zuid holland-De Schenkel	2.5914	366.5598	4.2901
zuid holland-Delfshaven	3.8619	877.7223	7.9651
zuid holland-Fascinatio	2.2324	947.6871	10.3775

Province-Neighborhood	β	MAE (m)	MAPE (%)
zuid holland-Feijenoord	2.8150	615.1246	6.2019
zuid holland-Goverwelle	2.2542	717.9833	5.7128
zuid holland-Groente- en Fruitmarkt	3.2655	581.8834	6.1209
zuid holland-Groot-IJsselmonde	1.9753	1209.2257	5.3353
zuid holland-Heijplaat	3.3354	599.8172	7.5773
zuid holland-Hellevoet	2.7637	974.4529	6.2371
zuid holland-Het Lage Land	2.7411	927.9785	6.6786
zuid holland-Hillegersberg-Noord	2.0510	1402.8705	9.4555
zuid holland-Hillesluis	3.1273	992.7245	6.9632
zuid holland-Hof van Delft	1.9176	755.3555	6.0762
zuid holland-Hoog Dalem	3.1116	550.5851	4.9342
zuid holland-Katendrecht	3.4492	848.4079	7.3088
zuid holland-Kleinpolder	2.4708	534.3541	4.7771
zuid holland-Kleiwegkwartier	2.5294	1108.9772	8.5147
zuid holland-Kop van Zuid	8.9279	2635.9565	18.0302
zuid holland-Kop van Zuid-Entrepot	3.6277	1013.5264	7.9457
zuid holland-Kort Haarlem	2.2017	714.5429	5.8154
zuid holland-Korte Akkeren	1.8921	934.6143	7.2767
zuid holland-Leyenburg	2.6492	1105.6755	7.0406
zuid holland-Lombardijen	2.5158	877.8189	5.7469
zuid holland-Middelland	4.0587	1449.9492	9.3874
zuid holland-Middelwatering	2.1607	1027.9243	5.7256
zuid holland-Moerwijk	2.4027	911.6470	5.3226
zuid holland-Molenlaankwartier	1.7451	1057.7209	7.6649
zuid holland-Morgenstond	2.6340	1237.2026	6.5960
zuid holland-Nesselande	2.4492	1191.0324	5.6239
zuid holland-Nieuw-Helvoet	2.4324	727.4401	5.6990
zuid holland-Nieuw-Mathenesse	3.3402	1490.8770	9.7788
zuid holland-Nieuwe Westen	2.3714	1012.1868	7.0980
zuid holland-Noord	2.0993	962.6752	6.1913
zuid holland-Noordereiland	4.2867	546.1991	6.5696
zuid holland-Ommoord	2.5510	1006.0523	4.5268
zuid holland-Onyx	3.1994	245.1591	5.9990
zuid holland-Oosterflank	2.4122	921.5584	6.3848
zuid holland-Oostgaarde	2.3032	1002.5975	6.0868
zuid holland-Oud-Charlois	2.5910	998.3229	6.5429
zuid holland-Oud-IJsselmonde	2.6498	1276.5539	7.0185
zuid holland-Oud-Mathenesse	2.9634	684.9317	6.5451
zuid holland-Oude Westen	4.1749	1245.0457	8.6854
zuid holland-Overschie	2.4124	930.9699	6.6810
zuid holland-Pendrecht	2.4870	638.2521	5.6508
zuid holland-Plaswijck	2.3022	858.8201	5.3770
zuid holland-Portlandsehoek	2.3840	337.4392	4.0836
zuid holland-Prinsenland	2.6127	672.2612	5.0840
zuid holland-RijswijkBuiten	2.9223	852.4383	6.0178
zuid holland-Rivium	3.4663	492.7409	6.3654
zuid holland-Ruiven	1.6341	1085.9904	9.4737

Province-Neighborhood	β	MAE (m)	MAPE (%)
zuid holland-Rustenburg en Oostbroek	2.5735	996.3351	6.3441
zuid holland-Schenkel	2.3929	728.4814	6.0319
zuid holland-Schiebroek	1.6411	1351.9866	7.7812
zuid holland-Schieweg	2.2404	1199.9656	9.4078
zuid holland-Schollevaar	2.4925	2304.8204	10.1820
zuid holland-Spangen	3.8169	913.4136	7.7468
zuid holland-Stadsdriehoek	2.9261	1105.7357	6.1710
zuid holland-Stolersluis	3.1483	905.2548	10.8613
zuid holland-Tanthof-Oost	2.4978	533.9905	5.1841
zuid holland-Tanthof-West	2.6141	702.5259	6.4351
zuid holland-Tarwewijk	2.6717	1197.4825	8.4852
zuid holland-Transvaalkwartier	2.8313	1159.1211	7.7995
zuid holland-Tussendijken	4.7390	828.4997	8.3005
zuid holland-Vogelbuurt	2.8233	331.7062	4.2976
zuid holland-Voordijkshoorn	2.8440	1182.2247	6.3758
zuid holland-Voorhof	3.0422	687.5436	5.4685
zuid holland-Vreewijk	2.4708	1337.1264	6.8031
zuid holland-Vrijenban	2.0001	1177.3917	8.6270
zuid holland-Wateringse Veld	2.0135	889.2508	4.1830
zuid holland-Westergouwe	2.6352	1411.3738	9.1938
zuid holland-Westplaat	2.1284	327.1887	4.0356
zuid holland-Westpolder	2.9003	1201.3643	7.8848
zuid holland-Wielewaal	3.5038	514.1891	7.3558
zuid holland-Wippolder	2.1530	934.4171	6.0648
zuid holland-Zestienhoven	2.4101	1414.0629	6.7802
zuid holland-Zevenkamp	2.0393	893.7788	5.6740
zuid holland-Zuid	2.6813	425.8455	5.4814
zuid holland-Zuiderpark	2.7503	771.4564	7.4627
zuid holland-Zuidwijk	1.9354	657.6528	5.4889
noord brabant-Binnenstad	1.8860	988.4452	5.7002
noord brabant-Brandevoort	1.9107	1820.5737	10.8091
noord brabant-Brouwhuis	1.8751	1392.1591	8.5124
noord brabant-Dierdonk	1.8306	2001.3904	13.5784
noord brabant-Helmond-Noord	2.0934	823.4308	5.1954
noord brabant-Helmond-Oost	1.9536	714.1405	5.9645
noord brabant-Helmond-West	2.4216	824.8724	7.1705
noord brabant-Industriegebied Zuid	2.0555	1663.6083	6.8181
noord brabant-Mierlo-Hout	1.6005	831.9247	4.8039
noord brabant-Oosterheide	1.9176	1164.2297	7.3069
noord brabant-Rijpelberg	1.8283	2983.4971	18.7872
noord brabant-Stiphout	1.6789	1370.0206	7.1730
noord brabant-Warande	2.3438	946.7836	5.7640
noord brabant-West	1.9129	2095.1854	8.5361
noord brabant-Zuidoost	1.6757	2447.4908	12.5495
limburg-Blerick-Midden	2.4307	595.5677	5.3623
limburg-Blerick-Noord	2.4127	651.6155	6.1779
limburg-Blerick-Zuid	2.6081	415.9061	5.3967

Province-Neighborhood	β	MAE (m)	MAPE (%)
limburg-Boekend	1.8872	1407.1170	11.9955
limburg-Centrum	2.1613	1571.1261	6.9788
limburg-Donderberg	2.1171	461.5097	4.1018
limburg-Hoogvonderen	2.6198	437.1072	4.8563
limburg-Hout-Blerick	1.7810	1435.0019	10.3675
limburg-Klingerberg	2.2053	569.4924	6.4843
limburg-Lindenheuvel	1.8446	848.0495	5.9188
limburg-Maasniel	1.9044	1354.3674	7.0017
limburg-Maastricht-Centrum	3.0078	1312.6660	5.8736
limburg-Maastricht-Noordwest	2.6891	997.5150	7.0901
limburg-Maastricht-Oost	1.7216	1222.5390	4.7935
limburg-Maastricht-West	1.7641	1198.8582	4.6823
limburg-Maastricht-Zuidoost	1.5854	1151.9402	5.3026
limburg-Meuleveld	2.8486	341.2812	4.9743
limburg-Roermond-Oost	1.7881	663.6759	5.8131
limburg-Roermond-Zuid	1.6711	2050.1706	11.8942
limburg-Sanderbout	3.1573	765.6142	9.0681
limburg-Vossener	2.6946	478.9732	5.6238
zeeland-Binnenstad	2.1766	1009.2677	7.4998
zeeland-Burgh	2.6285	550.3192	7.1392
zeeland-Haamstede	2.6743	664.3283	5.7081
zeeland-Lammerenburg	2.0721	1275.8438	6.1651
zeeland-Middelburg-Zuid	2.5989	900.5887	6.8581
zeeland-Othene	2.4960	712.4926	5.5557
zeeland-Paauwenburg	1.9163	1126.0872	8.3501
zeeland-Terneuzen-Centrum	3.1580	638.5805	6.1312
zeeland-Terneuzen-Noord	2.2055	665.8760	4.7845
zeeland-Terneuzen-West	2.1195	1089.1792	7.7356
zeeland-Terneuzen-Zuid	2.0249	642.4476	4.6504
zeeland-West-Souburg	2.4413	464.8484	5.0099