

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

Koen Stevens

April 26, 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:

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

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 |

Figliozi (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 (Figliozi, 2008).

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

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

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 circuitry factor. It is important to note,

Table 2: Estimates of the circuitry 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 circuitry factor c can vary significantly within a single city, depending on local street patterns, infrastructure, and topography. These variations suggest that a fixed circuitry 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, 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`. 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.

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`.

3.2 Generating and solving of TSPs

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.

4 Results

5 Discussion

6 Conclusion

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.
- Figliozi, 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.

- 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

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;

```

Table 3: Empirical estimates for β in selected neighborhoods.

| Province | Neighborhood | β | MAE |
|-----------|----------------|---------|--------|
| groningen | Hortusbuurt | 2.5585 | 0.0791 |
| groningen | Binnenstad | 2.1462 | 0.0761 |
| groningen | Oosterpoort | 2.2591 | 0.0623 |
| groningen | Rivierenbuurt | 1.8351 | 0.0658 |
| groningen | De Wijert | 1.6575 | 0.0550 |
| groningen | Oosterparkwijk | 1.8315 | 0.0559 |
| groningen | De Hoogte | 2.0912 | 0.0803 |
| groningen | Korrewegwijk | 2.1355 | 0.0591 |
| groningen | Schildersbuurt | 2.2993 | 0.0635 |
| groningen | Paddepoel | 1.6485 | 0.0461 |
| groningen | Oranjewijk | 2.2007 | 0.0697 |
| groningen | Tuinwijk | 3.8483 | 0.1129 |
| groningen | Selwerd | 1.6298 | 0.0675 |
| groningen | Vinkhuizen | 1.4551 | 0.0446 |
| groningen | Hoogkerk-zuid | 1.4752 | 0.0610 |
| groningen | Gravenburg | 1.3763 | 0.1492 |

| Province | Neighborhood | β | MAE |
|---------------|-------------------------|---------|--------|
| groningen | De Held | 1.7846 | 0.0413 |
| groningen | Reitdiep | 1.5713 | 0.0446 |
| groningen | Hoornse Meer | 1.6240 | 0.0690 |
| groningen | Corpus den Hoorn | 1.7593 | 0.0668 |
| groningen | Eemspoort | 1.6011 | 0.0411 |
| groningen | Euvelgunne | 1.8495 | 0.1000 |
| groningen | Driebond | 1.9508 | 0.0623 |
| groningen | Winschoterdiep | 2.0562 | 0.0592 |
| groningen | Eemskanaal | 1.7584 | 0.0389 |
| groningen | Helpman | 2.1376 | 0.0691 |
| groningen | Lewenborg | 1.9960 | 0.0604 |
| groningen | Beijum | 1.7477 | 0.0428 |
| groningen | Maarsveld | 1.6575 | 0.0428 |
| noord holland | Schrijverswijk | 1.6982 | 0.0445 |
| noord holland | Stad van de Zon | 1.4168 | 0.1631 |
| noord holland | Stadshart | 1.8336 | 0.0578 |
| noord holland | Jordaan | 2.0295 | 0.0605 |
| noord holland | Slotervaart | 1.7250 | 0.0444 |
| noord holland | IJburg | 1.3266 | 0.0503 |
| noord holland | Oostelijke Eilanden | 1.6790 | 0.0437 |
| noord holland | Oostelijk Havengebied | 1.7312 | 0.0475 |
| noord holland | Frederik Hendrikbuurt | 2.7022 | 0.0721 |
| noord holland | Van Lennepbuurt | 2.7463 | 0.0786 |
| noord holland | Da Costabuurt | 3.0877 | 0.0765 |
| noord holland | Kinkerbuurt | 2.5900 | 0.0906 |
| noord holland | Kersenboogerd | 1.5995 | 0.0516 |
| noord holland | Pax | 2.0994 | 0.0394 |
| noord holland | Graan voor Visch | 2.1737 | 0.0545 |
| noord holland | Vrijschot-Noord | 2.6586 | 0.0515 |
| noord holland | Toolenburg | 1.6453 | 0.0436 |
| noord holland | Floriande | 1.8705 | 0.0483 |
| noord holland | Overbos | 1.7493 | 0.0434 |
| noord holland | Bornholm | 1.7675 | 0.0417 |
| noord holland | Beukenhorst-Oost | 1.7780 | 0.0824 |
| noord holland | De Hoek | 2.4509 | 0.0506 |
| noord holland | West | 1.8557 | 0.0445 |
| noord holland | Zuid | 2.1954 | 0.0597 |
| noord holland | Oost | 1.8551 | 0.0565 |
| noord holland | Noord | 1.6516 | 0.0549 |
| noord holland | De President | 1.4329 | 0.1163 |
| noord holland | Graan voor Visch-Zuid | 1.9398 | 0.0675 |
| noord holland | Zuidwijk | 1.3511 | 0.0403 |
| noord holland | Buitenveldert-West | 1.1352 | 0.0651 |
| noord holland | Buitenveldert | 1.1033 | 0.0700 |
| noord holland | Apollobuurt | 1.7856 | 0.0625 |
| noord holland | Stadionbuurt | 1.5651 | 0.0657 |
| noord holland | Prinses Irenebuurt e.o. | 1.9873 | 0.0647 |

| Province | Neighborhood | β | MAE |
|---------------|---------------------|---------|--------|
| noord holland | Hoofddorppleinbuurt | 1.9575 | 0.0752 |
| noord holland | Willemspark | 1.9544 | 0.0662 |
| noord holland | Schinkelbuurt | 3.2523 | 0.0949 |
| noord holland | Vondelparkbuurt | 2.9876 | 0.0827 |
| noord holland | Helmersbuurt | 2.6397 | 0.0744 |
| noord holland | Overtoomse Sluis | 2.9447 | 0.0694 |
| noord holland | Museumkwartier | 1.7540 | 0.0613 |
| noord holland | Rivierenbuurt | 1.6830 | 0.0585 |
| noord holland | IJselbuurt | 2.4093 | 0.0778 |
| noord holland | Scheldebuilt | 1.7462 | 0.0638 |
| noord holland | Rijnbuurt | 1.8918 | 0.0535 |
| noord holland | De Baarsjes | 2.2849 | 0.0576 |
| noord holland | Landlust | 1.8583 | 0.0583 |
| noord holland | Staatsliedenbuurt | 1.9282 | 0.0629 |
| noord holland | Spaarndammerbuurt | 2.4826 | 0.0716 |
| noord holland | De Pijp | 2.2515 | 0.0563 |
| noord holland | Grachtengordel | 2.1485 | 0.0662 |
| noord holland | Oud-Zuid | 1.4295 | 0.0582 |