UNIVERSITEIT UTRECHT

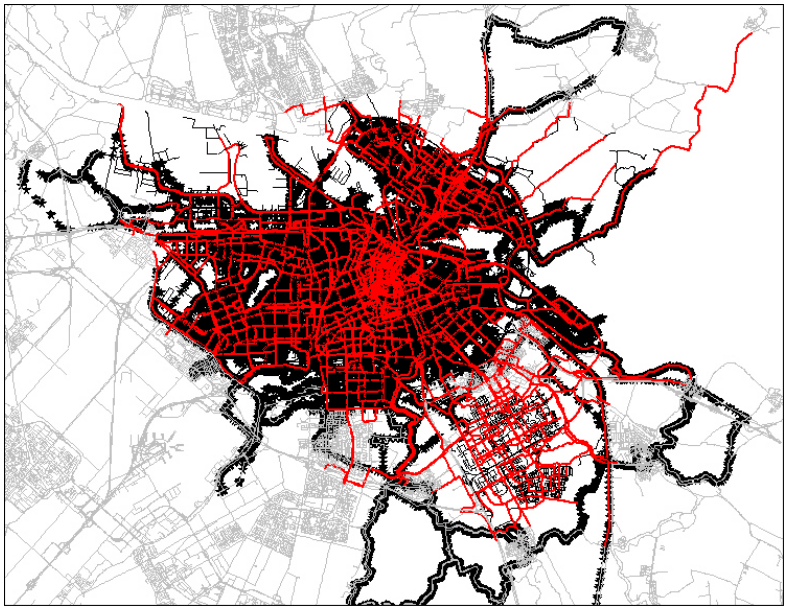
MASTER THESIS

On Bicycle Choice Set Generation

Author:
Niels WARDENIER

Supervisors:
Dr. E.R. Dugundji, CWI
Dr. Ir. L. Knapen, UHasselt
Dr. K. Dajani, UU
Second Corrector:
Prof. Dr. R.H. Bisseling, UU

17 October 2017



Abstract

In this thesis, we studied various methods of choice set generation. The Double Stochastic Generation Function (DSGF) method was looked after in more detail. We showed that the routes provided by this method are too complex in terms of the number of Basic Path Components (BPC). This problem is very clear when the predicted routes are compared with the observed routes provided by the FietsTelweek. In various ways we tried to cope with this problem. First we tried to scale back the cumulative distribution function (cdf) of the predicted data to the cdf of the observed data. Secondly, a maximum likelihood function was introduced to keep the most interesting predicted routes in terms of BPC's. The latter method proved to be more useful in constructing a choice set containing reasonable, not too complex routes.

Acknowledgements

First I would like to thank my supervisor Dr. Dugundji of the Centrum Wiskunde & Informatica for waking my interest to this topic, helping me with questions about my research and writing, making it possible to give a talk at the Science for Cycling colloquium during this years VeloCity and introducing me to various people in this expertise.

I would also like to thank my other supervisor Dr. Dajani of the Graduate School of Natural Sciences at University Utrecht for always being available to answer my questions about the process and monitoring my process working on this thesis.

I would also like to thank Dr. Ir. Knapen of School voor Mobiliteitswetenschappen at UHasselt for helping me out with coding problems, letting me use his previous work and suggesting ways to branch out my survey.

I would also like to acknowledge Prof. Dr. R.H. Bisseling of the Graduate School of Natural Sciences at University Utrecht as the second corrector of this thesis.

Finally, I want to thank my parents and friends for their support and interest during the time I was writing my thesis, with special thanks to Martin van Veelen for proofreading.

Niels Wardenier

Contents

List of Tables

List of Figures

Introduction

In 2016 over 40.000 people in the Netherlands participated in a week-long survey using a smartphone app to map travel movements by bicycle. The goal of this survey is to get more insight in cycling patterns of travelers. With this extensive data it is then possible to create stochastic models to understand behavioral choices, such as which routes people choose. This in turn can be used by municipalities to set priorities for improving the existing cycling network. The data generated by the FietsTelweek 2016 will be used in this thesis and discussed in chapter 1.

One typical part of the stochastic modeling effort involves generation of routes a person could have chosen to get from their origin to their destination. For cars there is a lot known on how to generate such route sets, but there is a lot less known on how to generate a good route choice set for bicycles. The generation of route choice sets for this mode of transportation is the focus of this thesis.

The multiple angles to look at the stated problem of choice set generation are described in more detail in chapter 2. We can use link elimination, branch and bound methods, route planning based on important landmarks or stochastic methods. There is no clear consensus on which angle is the most useful.

In this research we have chosen to use the method called Double Stochastic Generation Function (DSGF) . A significant benefit of this method is that in the cost function we can both vary the link cost and can account for preferences of people. For example, people may tend to use routes with nice scenery over a crowded city center. Furthermore, this method provides heterogeneous routes by how it is defined.

Our main goal is to combine the DSGF method with the idea of Basic Path Components (BPC), explained in chapter 3. The road network can be represented as a graph , where ways are edges and crossings are nodes . Knapen et al. [*1*] define the BPCs as either a least cost path or a non-least cost edge. This is interesting because in general people generate their route as a sequence of shortest paths. Knapen also came up with an algorithm to split a path in multiple BPCs. It is shown that we can find a minimal, non-unique splitting for every path. This can become useful in analyzing the data, for example provided by the FietsTelweek, to check if people indeed construct their route as a sequence of small number of shortest paths.

In chapters 4, 5 and 7 the codes of the DSGF method and BPC method and the combination of the two methods will be handled. Here we also give background information about the writers of the code, the purpose of the code and for which audience the code is written.

Chapter 6 shows how the number of basic path components is correlated with other attributes of the route. It all comes together in chapter 7, where an attempt is made to make the combination using the insights of earlier found results. It turns out that methods that have to work from a mathematical point of view, sometimes do not work in the real world. In that case, some engineering tricks are needed.

In conclusion, survey to the route complexity (i.e. the minimum number of embedded BPC) from GPS traces gives us information on how people tend to construct their routes. This leads to better route choice models and in turn allows evaluation of infrastructure design before investment.

1 Data: FietsTelweek

1.1 Information about the basic data

The basic data set is extracted from the Fietstelweek [2] 2016 . During, the FietsTelweek 42658 people across the Netherlands downloaded the mobile phone app, (App de Fiets). In total, these people recorded around 416.000 bicycle trips. The goal of the FietsTelweek is to get insight on where people are biking, how long are people biking and where people are having long waiting times. This survey is a joint initiative of the Fietsersbond, the Dutch Ministry of Infrastructure, Environment and local authorities. Below the infographic made about the second FietsTelweek, FietsTelweek 2016, FIGURE 1:

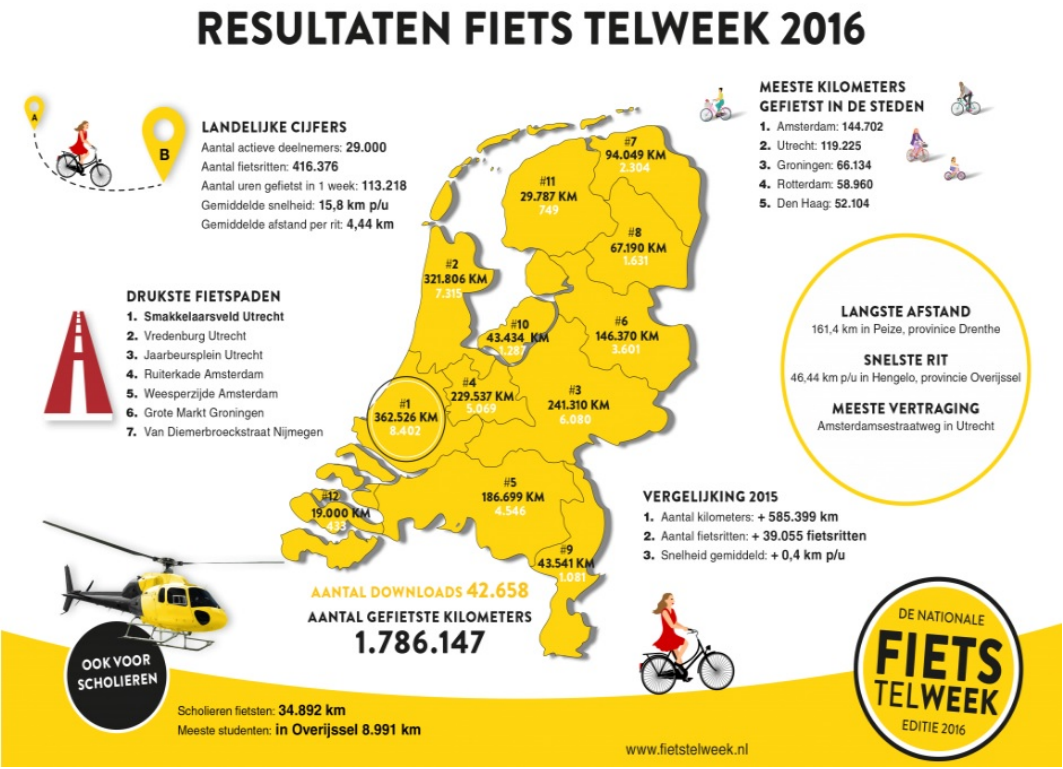


FIGURE 1 : Infographic FietsTelweek 2016 (in dutch)

The basic data may be freely used under certain conditions, namely the FietsTelweek has to be mentioned in the references and derived products have to be made publicly accessible. Also, FietsTelweek must be informed by email to what use the data is and the data may not be used for commercial purposes without permission by consortium Nationale Fiets Telweek.

1.2 Map matching the data

The data is map matched onto QGIS (Quantum Geographic Information System) . Map matching is the process of relating the recorded GPS track to edges in an existing graph. Recorded trips of the FietsTelweek are projected on the map as a sequence of brown dots. The green lines form the road network of Amsterdam. FIGURE 2 shows the nodes of 22 trips for the sake of visibility. The image on the title page shows all the data, summarized in fat (many routes) and small (few routes) red lines.

FIGURE 2 : Map matching of 22 routes in QGIS

2 Choice Set Generation Methods

The models described in this section are made to generate a discrete choice set . Hence, the name, the number of choices must be a finite number. Another name for choices is alternatives . There must be exactly one chosen alternative, and choosing one alternative means not choosing all other alternatives. So if there are 4 possible routes for one person to choose, say $\alpha, \beta, \gamma, \delta$, if this person chooses path β, that implies that he/she did not choose paths α, γ and δ.

In choice set generation, there is at first a universal set U containing all possible routes from an origin to a destination . After that, for each individual n there is created a choice set C_n. Going back to the example, maybe only routes α and γ are placed in the choice set for person n. So person n has only two routes to choose from instead of four. This choice can be made in a deterministic or probabilistic way.

There are various choice set generation models to approach the construction of a choice set. Thereby we will use FIGURE 3, where the traveler wants to travel from A to C. In this case all the edges have a positive cost. The links are named after the two edges it connects. For example, the edge connecting C and F is called cf.

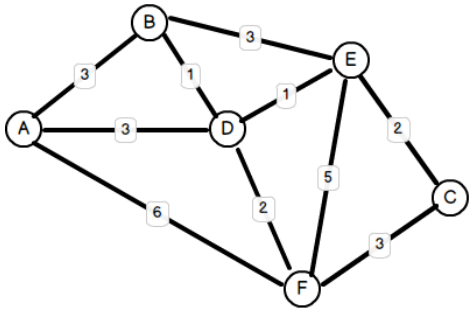


FIGURE 3 : Positive Weighted Graph

2.1 Breadth First Search-Link Elimination

Rieser-Schüssler et al. [*3*] came up with a shortest path method, called Breadth First Search-Link Elimination (BFS-LE) . First the shortest path is calculated, then the link elimination is done in various ways, like random or controlled by criteria. Looking at our example graph, the least path cost is 6 along the path *ADEC*. Then if link *de* is eliminated, the new found least cost path is *ADFC* or *ABEC*, both costing 8. Resulting in FIGURE 4

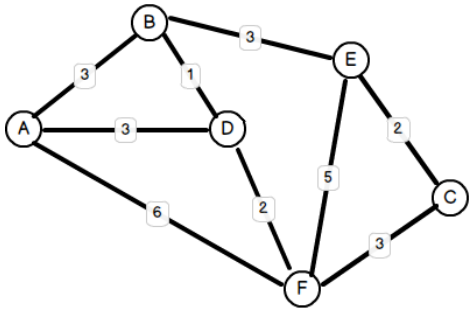


FIGURE 4 : Positive Weighted Graph with link de eliminated

Furthermore, Rieser-Schüssler described two variations. The first variation shuffles the order for which vertex the shortest path is calculated at depth d, this makes the run time of the algorithm shorter by preventing unneeded calculations. Here depth d means that already d edges have been removed from the original graph. The second variation cleans the network by removing junctions, dead ends and intersections. This second variation is tested against a method called Stochastic Choice Set Generation (SCSG), which uses stochastic repeated shortest path based algorithms. It turns out that the BFS-LE outperforms the SCSG method for trips under 10 km, relevant for bicycle trips, in terms of computational efficiency. Moreover, routes provided by BFS-LE are more diverse.

2.2 Branch and Bound

Prato and Bekhor [*4*] provide another method on the route choice, called Branch and Bound. This technique looks for paths in the branches of the graph which satisfy the boundary conditions. Supposed bounds are directional, temporal, similarity, loop and movement (avoiding left turns) constraints. For example the temporal constraint, a route will only be placed in the choice set if the travel time for this route does not exceed the least found travel time by a certain factor large than 1. All the other constraints are defined in a similar way. Considering our example and suppose links ad and cf have too many left turns, giving us the network in FIGURE 5 and resulting in the least cost path $ABDEC$

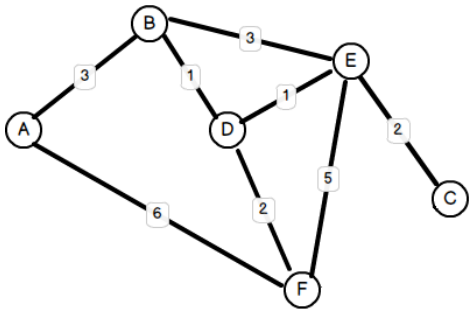


FIGURE 5 : Positive Weighted Graph where link ad and cf are not considered

In the article just mentioned the Branch and Bound technique is compared to other deterministic approaches; Label Approach (which calculates the shortest path considering attributes such as time, distance and delay), Link Elimination (described in 2.1), Link Penalty (a method where the links of the shortest path are made more expensive). The Branch and Bound technique outperforms all the others in reconstruction of paths. The Branch and Bound technique has equal computational times as methods that produce less results and performs well in terms of coverage. In this context, coverage means to what extent the set of found routes reproduces the actual behavior according to a certain overlap threshold. That is why Prato and Bekhor advise to use the Branch and Bound technique to obtain heterogeneous route sets, one big disadvantage of this method is to choose what are the right bounds.

2.3 Pure Statistical Methods

A method working around this problem is a pure statistical method based on biased random walks by Frejinger [*5*]. In this method the choice sets are defined as the sets of all paths connecting each origin-destination pair (OD-pair) . A probability is associated with a subpath based on its distance to the shortest path, according to the double bounded Kumaraswamy distribution. The cumulative distribution function F of this distribution is:

$$F(x;a,b) = 1-(1-x^a)^b \text{ where } x \in [0,1], a,b > 0, \tag{1}$$

here a and b are shaping parameters. The sampling protocol for path generation is as follows. A choice set $\tilde{C}_n$ is generated by taking R draws with replacement from the universal set adding the chosen path. Bovy [*6*] described the method of Frejinger in the following way: *"By definition, all links on the true shortest path have a link probability of one, all other links between zero and one. The shortest paths are not saved for use in a master set. Instead, in a second step a sample of routes are generated by a repeated random walk procedure starting from the origin from which links are successively selected and added from node to node where the link selection process at each node is governed by the probabilities of the associated next links. At the end a route probability results as the product of the associated link probabilities.* A possible problem with this method is that it constructs routes consisting of many small subpaths, which is not realistic.

2.4 Mental Representation Items

A fourth method by Kazagli and Bierlaire [*7*] to work around the challenges of route choice generation is the introduction of Mental Representation Items (MRI). To construct a data set, they made use of a layer system. The first layer is used to determine a MRI choice set for example as $C_l = \{avoidCC, aroundCC, throughCC\}$, where CC stands for city center. Layer $l+1$ provides additional detail in the representation of the choice in comparison with the one in layer l. In our example the set vertices $\{B,E\}$ and $\{D,F\}$ can represent two neighborhoods.

For the operationalization of the choice set, an attribute is assigned to each MRI by calculating the expected maximum utility. This can be done by taking the sum of the logarithms of all utilities of paths using this particular MRI. This can bring computational problems along, which were intended to avoid using this method. One clear advantage of this method stated by Kazagli is that several possible paths connecting OD-Pairs are bundled under MRIs while going from a lower layer to a higher layer. Hence, the choice set size decreases, and therefore the composition and correlation are simplified.

2.5 Double Stochastic Generation Function

The last choice set generation method that will be discussed is the Double Stochastic Generation Function method (DSGF) . This method, described by Nielsen [*8*] for public transportation and Bovy and Fiorenzo-Catalano [*9*], produces heterogeneous routes because the costs and parameters, used in the cost function for the links, are drawn from a probability distribution. A possible difficulty with this method is that the link cost randomization takes lots of computational time. However, Hood et al. [*10*] shows that the method is faster than the already mentioned method by Rieser-Schüssler. In the article of Halldorsdottir et al. [*11*], it is stated that the DSGF method has a high coverage level of replicating routes and performs well up to 10 km. Furthermore, Bovy and Fiorenzo-Catalano [*9*] state that the method guarantees, with high probability, that attractive routes are in the choice set and unattractive routes will not be put in the choice set. Also the size and composition of the choice set are called stable, where stable means that for a relative low number of randomizations the size and composition of the choice set are satisfactory. Ultimately, we have chosen this method for the reasons mentioned above. In the DSGF method the link costs and preference parameters are both drawn from a probability distribution, hence the name double stochastic. In the article of Halldorsdottir et al. [*11*] this method is described short and clear:

"In the DSGF method, a shortest path search is carried out iteratively using an implementation of the Dijkstra's algorithm (Dijkstra, 1959) on a realization of the network. At each iteration, the realization of the network is obtained by randomly drawing the cost of each link from a probability distribution and extracting attribute preferences for each traveler from another probability distribution. After each iteration, only unique routes not generated in previous ones are added to the route choice set as the same route may be found several times during the process, even though the realizations of the network are obtained from different costs and preference parameters. The shortest path search is repeated iteratively until the preselected maximum choice set size is achieved or the predefined time abort threshold is reached."

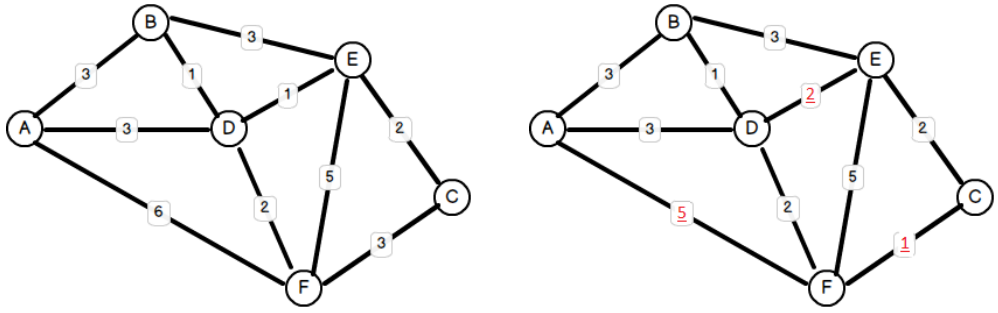
Dijkstra's algorithm finds the shortest path between two vertices in a directed graph with positive edge weights. Then name the starting vertex s for start. Set the distance from s to s to 0. All other vertices are placed in queue Q, are called unvisited and the distance from s to all to vertices in Q is set to ∞. We denote $d(x)$ for the distance from s to x. While Q is a non-empty set, we choose the vertex with the minimal distance to s. Remove this vertex from Q and call it v. (In the first iteration s is the vertex with the minimal distance.) For each vertex u that is a neighbor of the just removed vertex, if the distance from start vertex to v plus the length of the edge uv is smaller than the previous found distance, replace it. Otherwise, leave it as it was. Continue till the set Q is empty. Look again at FIGURE 3. Dijkstra's algorithm is performed for that graph.

- In the first iteration B, D and F are unvisited neighbors of A, the distances $d(B)$ and $d(D)$ are set to 3 and $d(F)$ is set to 6.
- In the second iteration B and D both have the shortest distance to A, we choose to pick B. D and E are unvisited neighbors of B, the distance $d(E)$ is set to 6 and $d(D)$ remains 3. We now remove B from Q.
- In the third iteration D has the shortest distance to A of the unvisited vertices. E and F are unvisited neighbors of D, the distance $d(E)$ is set to 4 and the distance $d(F)$ is set to

5. We now remove D from Q.

- In the fourth iteration E has the shortest distance to A of the vertices that are unvisited. C and F are unvisited neighbors of E, the distance $d(C)$ is set to 6 and the distance $d(F)$ remains 5. We now remove E from Q.
- In the fifth iteration F has the shortest distance to A of the unvisited vertices. C is the only unvisited neighbor of F, the distance DC remains 6. We now remove F from Q.
- In the sixth iteration C is the only unvisited vertex. It has no unvisited neighbors. We now remove C from Q. Q is empty, so the algorithm terminates.

Going back to the DSGF method, in our example this means that first the least cost path $ADEC$ is found and after that the linkcosts are adjusted (in a way discussed in the next paragraph). Assume now that the cost of de becomes 2 , cf becomes 1 and af becomes 5. Resulting in FIGURE 6b. In this case least cost path AFC is found.



(a) Original Graph (b) Graph with adjusted linkcosts (in red)

FIGURE 6 : Example graphs for the DSGF method

In the earlier mentioned article by Halldorsdottir, a survey has been held to show which of the four cost functions performs best in terms of coverage. The following cost attributes where tested: Road Type (large roads, small roads, other roads), Cycle Lanes (segregated lanes or not), Land Use (scenic roads, non-scenic roads, forest roads, non-forest roads) and a combination of the three attributes. It shows that the combination of the three attributes gives the highest coverage, namely from 63.5% at a 100% overlap threshold. This means that in 63.5% of the routes are replicated completely. Furthermore, up to 79.2% of the routes are replicated for at least 70%. The most successful cost function is the combination of the three cost attributes. This cost function looks as follows:

$$\begin{aligned} C_a = &\sum_k ((\beta_{Roadtype_k} + \xi_{Roadtype_{ak}}) \cdot RoadType_{ak} \cdot Length_a) \\ &+ \sum_k ((\beta_{Bikelanes_k} + \xi_{Bikelanes_{ak}}) \cdot Bikelanes_{ak} \cdot Length_a) \\ &+ \sum_k ((\beta_{LandUse_k} + \xi_{LandUse_{ak}}) \cdot LandUse_{ak} \cdot Length_a) + \varepsilon_a, \end{aligned} \tag{2}$$

where C_a is the random cost function of link a.

The components $Roadtype_{ak}, Bikelanes_{ak}$ and $LandUse_{ak}$ are deterministic for k, here k stands for RoadType k, BikeLanes k Land Use k respectively. From the first half of FIGURE 7 by Halldorsdottir et al. [*11*], we see that the initial cost of small roads is twice as high as the initial cost of large roads. Furthermore, the initial cost of other roads is three times the price as the initial cost of large roads. Likewise, for the other two attributes. Remark that the parameters for these attributes are scaled to 1. The next parameter in the costfunction, $Length_a$, stands for the length of link a, in meters. The length of a segment has a linear influence on the cost. The last deterministic parameters are β_{ik}, where $i \in \{RoadType, BikeLanes, LandUse\}$ are coefficients to be determined beforehand. The $\xi_{a,k}$ and ε_a are the two probabilistic parameters. The $\xi'_{ak}s$ are proportional to their resp. $\beta's$ multiplied by a variation factor and a standard normal distribution, so

$$\xi_{ai} = \beta_{a,k} \cdot N(0,1) \cdot vf$$

,where vf stands for variation factor. The variation factors can be found in the second half of FIGURE 7. The $\xi's$ accounts for the heterogeneous preferences of cyclists and can add both a positive or a negative value to the cost function of link a. The ε_a is proportional to $Length_a$ multiplied by a variation factor and a standard normal distribution, so

$$\varepsilon_a = Length_a \cdot N(0,1) \cdot 2$$

as this variation is always 2. Then this ε_a is added to vary in the linkcosts as an error term.

	Cost function			
	Road type	Cycle lanes	Land use	Road type, cycle lanes, and land use
Parameters				
Large roads	0.167	–	–	0.167
Small roads	0.333	–	–	0.333
Other roads	0.500	–	–	0.500
Segregated cycle lanes	–	0.400	–	0.400
No segregated cycle lanes	–	0.600	–	0.600
Scenic roads	–	–	0.125	0.125
Non-scenic roads	–	–	0.250	0.250
Forest roads	–	–	0.500	0.500
Non-forest roads	–	–	0.125	0.125
Variation factors				
$\xi_{Large\ roads}$	2	–	–	2
$\xi_{Small\ roads}$	10	–	–	10
$\xi_{Other\ roads}$	10	–	–	10
$\xi_{Segregated\ cycle\ lanes}$	–	2	–	2
$\xi_{No\ segregated\ cycle\ lanes}$	–	10	–	10
$\xi_{Scenic\ roads}$	–	–	2	2
$\xi_{Non\text{-}scenic\ roads}$	–	–	3	3
$\xi_{Forest\ roads}$	–	–	7	7
$\xi_{Non\text{-}forest\ roads}$	–	–	2	2
ε_a	2	2	2	2

FIGURE 7 : Overview of the calibrated parameters and variation factors

3 Basic Path Components

The complexity of a given path in a graph is the minimum number of basic path components in the decomposition of the path. A basic path component (BPC) is defined as either a least cost path or a non-least cost edge. A non-least cost edge e is an edge whose cost is larger than the least cost path connecting both vertices connected by that edge e. In the doctorate thesis of Knapen et al. [*1*], he states and proves the following hypothesis:*"In utilitarian trips (trips having the purpose to perform an activity at a particular location), individuals tend to construct their route as a concatenation of a small number of minimal cost routes i.e. basic path components (BPC)."* Any given path can be decomposed in this manner. Namely, the trivial decomposition in basic path components is to follow the edges along the path. Due to the hypothesis over the utilitarian trips, we are interested in the decomposition of a a path into a minimum number of BPC. The goal is in finding a way to incorporate the statement of Knapen in the cost functions that the DSGF method uses.

3.1 Knapen's algorithm

Knapen et al. [*1*] wrote an algorithm to split a path into BPC's. FIGURE 8 displays the pseudo code of the algorithm.

Algorithm 8.5.1 Algorithm for finding a partition of a path into BPC's

Input Graph G, edge costs c, $P = (v_0, v_1, \ldots, v_l)$ containing no non-shortest edges
$start \leftarrow 0$;
$k \leftarrow 1$
while ($P(v_{start}, v_l)$ is not a least cost path) **do**
  Find the first vertex v_{j_k} in $P(v_{start}, v_l)$ such that $lc(v_{start}, v_{j_k}) < c(P(v_{start}, v_{j_k}))$
  $start \leftarrow j_k - 1$
  $k \leftarrow k + 1$
end while
return join vertices $v_{j_1}, v_{j_2}, \ldots, v_{j_{k-1}}$

FIGURE 8 : Algorithm to divide a path into a minimum number of BPC's

In this algorithm, the input is a graph G with positive edge costs c and a path $P = (v_0, v_1, \ldots, v_l)$ with no non-shortest edges. Furthermore, there is defined a starting point called *start*, set to 0 and a counting variable k, set to 1. The while loop looks for a non-least cost subpath as follows. The first vertex where a shorter path is found is called v_{j_k}, V_{start} is replaced by $v_{j_k} - 1$, the vertex preceding v_{j_k}, and k is replaced by $k+1$. The vertices $v_{j_1}, v_{j_2}, \ldots, v_{j_{k-1}}$ are returned and called join vertices. Now the path can be split at vertices preceding the join vertices. These vertices are called split vertices. Using this algorithm, a splitting is found at $k-1$ vertices, so our path P splits into k BPC's. Knapen proved that this decomposition is minimal but not unique. For example, we can flip the whole setup and start in the end vertex v_l and find another minimal decomposition.

This algorithm was used on a number of data sets in Knapen's thesis, namely: BelgiumNavteq (mixed), BelgiumOSM (mixed) and ItalyNavteq (car trips). For the Belgium Navteq data, the number of minutes is added, this is the amount of time of inactivity before a stop is detected, (red=1 minute, green=2.5 minutes, dark blue=5 minutes). FIGURE 9 shows the relative amount of BPC's in trips, with on the horizontal axis the number of BPC's and on the vertical axis the relative frequency distribution.

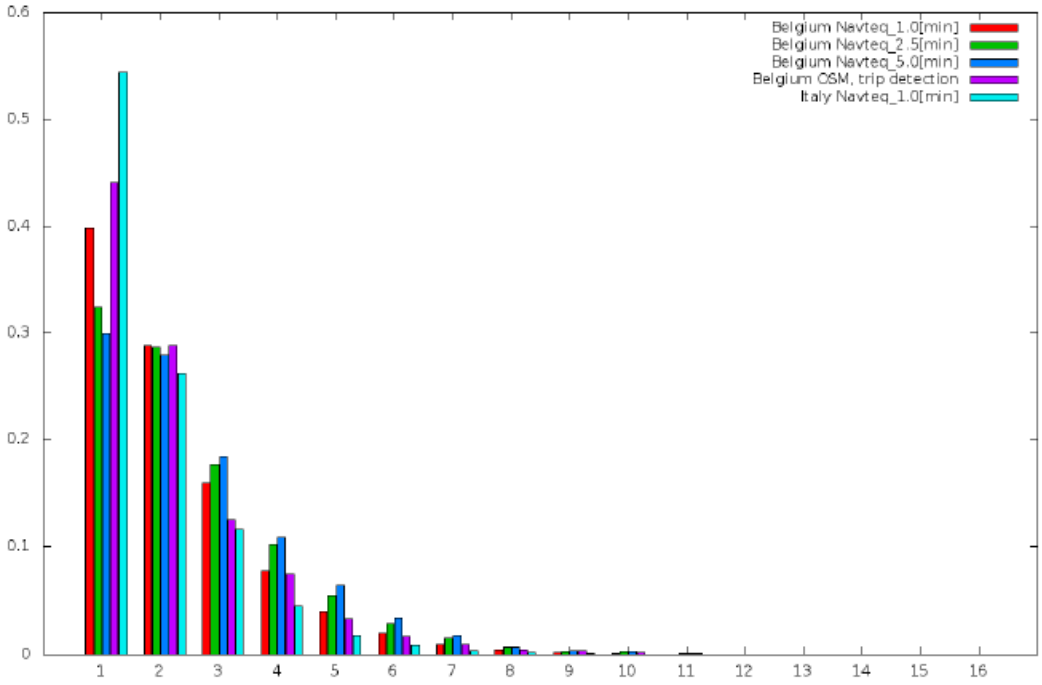


FIGURE 9 : Relative frequency distribution of BPC's

This graph supports the aforementioned hypothesis. Namely, in all cases the number of found routes consisting of 6 or more BPC's lies below 5%.

3.2 Basic Path Components in FietsTelweek Data

A table with symbols is included to avoid a stream of definitions in the next paragraph.

Symbol	Meaning
d_e	effective distance driven by the cyclist
d_{min}	shortest path between start- and endpoint
f_{NU}	fraction of non-utilitarian trips in the Netherlands
$\mathscr{P}_{A,O}$	set of observed paths assumed to be related to Amsterdam
$F_{A,O}(c)$	probability mass function for the complexity found in the Amsterdam set of observed routes
$F_{A,O}{}^{\mathrm{U}}(c)$	probability mass function for the complexity found in the Amsterdam set of observed routes that are qualified as utilitarian
c	the complexity (minimum number of basic path components) for a path

However, a difference shows up in the data of the FietsTelweek. For my thesis Knapen helped with splitting around 280.000 routes, collected by FietsTelweek 2016. FIGURE 10 shows that only 75% of the routes in the data consist of 5 or less BPC's. To reach the 95% benchmark, routes up to 11 BPC's have to be accepted.

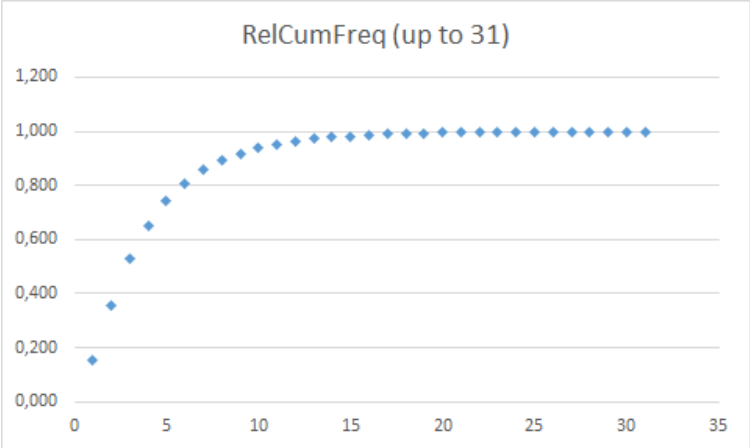


FIGURE 10 : Relative cumulative distribution of BPC's

Possible explanations for this difference are:

- Cyclists tend to use routes with more distinct shortest paths than mixed or car trips.
- The road network of the Netherlands is more dense and the infrastructure for cyclists is better. For example, in the center of Amsterdam there are lots of canals, having a road at both sides. This can lead to alternating shortest paths.

This means that we must accept routes with a greater number BPC's in our choice set, which is created by using Dutch data.

The basic FietsTelweek data does not provide all the information needed. Therefore, the advanced data is used in this survey. This data gives the direction in which the cyclist is moving and in which order the network nodes are visited. Such data is available for the city of Amsterdam, the capital of the Netherlands. Here an arbitrary rectangle of 22 km (north-south) $\times$ 27 km (west-east) is picked, to limit our very large data set. This rectangle is named R_0. After that, all routes with at least one link in R_0 are added back to our data and this new data set is named R_1. This data set consists of 31817 routes.

In FIGURE 11 all the routes of R_1 are made visible in QGIS:

FIGURE 11 : Routes of R_1

For this data set the purpose of the trip is unfortunately not known. In general, cyclists of this kind tend to pick routes close to the shortest path. This is the reason to look at the ratio $r_d = \frac{d_e}{d_{min}}$ for all the 31817 observed routes. The question is: "can we find suitable $\overline{r_d}$ such that: a trip is utilitarian if and only if $r_d < \overline{r_d}$?" If so, we can assume that every path p in our observed data set, for which the ratio $r_d(p)$ is smaller $\overline{r_d}$ is utilitarian. Define $F^{-1}(r_d)$, where $F^{-1}(r_d)$ is the inverse of the distribution function of the variable $r_d(p)$. Then $\overline{r_d}$ is defined as $F^{-1}(1-f_{NU})$, where f_{NU} is the fraction of non-utilitarian trips. According to Pucher and Buehler [*12*], the fraction of recreational trips is 0.27 for the Netherlands, hence $1-f_{NU} = 0.73$. Analyzing the data using, PgAdmin III (Pg Admin is an administration tool to search in a large data set), we found out that for the set $\mathscr{P}_{A,O}$, $\overline{r_d}$ equals 1.10. (For reference for whole Netherlands $\overline{r_d} = 1.08$.) $F_{A,O}(c)$ and $F^U_{A,O}(c)$ are compared in FIGURE 12 in terms of number of BPC's, one can see that there is a small shift of mass to the lower BPC's but there are no major differences. So we decide to use all 31817 routes in R_1.

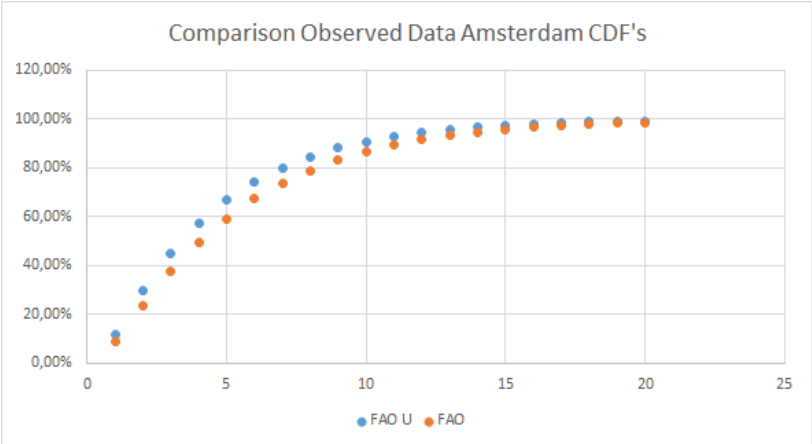


FIGURE 12 : Comparison between $F_{A,O}(c)$ and $F^U_{A,O}(c)$

4 ETH Zürich Code (POSDAP)

4.1 Background Information

For the DSGF method, the code is written in JAVA. It comes from the POSDAP (Position Data Processing) project(2012) of the ETH Zürich. The ETH (Eidgenössische Technische Hochschule) is a science, technology, engineering and mathematics university in Switzerland. The writers of the code are Lara Montini and Nadine Schüssler. The tool is developed for the automatic processing of GPS tracks to reconstruct travel diaries. Features of this project include detection of trips, mode detection and map matching. Intended audience of the code are developers and advanced end users, the people who are actually using the code. The full code can be found on `https://sourceforge.net/projects/POSDAP/`. This is a large web service where software developers can find, create and publish open source software for free.

The POSDAP project has the GNU General Public License Version 2.0. In short, it means that everyone can use, modify and even sell the code. Furthermore, when a code is published derived from the source code, the original author has to be mentioned.

4.2 Implementation: Examplechoice setGenerationStochasticHakatrin

The most important package of the POSDAP code for this thesis is:
hakatrin.choice setGeneration.hakatrinStochastic. This package consists of 17 classes of which *Examplechoice setGenerationStochasticHakatrin.java* is our main interest. This class has 245 lines of which 185 with code (line 49-236). Other classes of this package are described in Appendix A. Furthermore the class imports 20 other classes of which FIGURE 13 was made. Here the colors indicate the package from which the certain class is imported. Now a review of the code line by line is given(the classes that are imported are bold and when there is referred to lines that means lines in the code).

In line 52 it is started by defining a variable starting time, so the amount of time spent can be seen at the end. In the following lines 54-56 **configutils** and **config** are imported, a configuration file is created and the configuration settings are stored. Furthermore, an access to the settings at run time is created. In line 58 **HakatrinNetworkImpl** is imported in which links are added in the relevant network. The next line provides a helper class to store arbitrary attributes (identified by strings) for arbitrary objects (identified by String-Ids). The makers of the code indicates that this implementation takes lots of memory if too many attributes are stored for too many objects. This helper class is called **ObjectAttributes**. In the next five lines **MatsimNetworkReader** is imported. Matsim is a open source software development project founded by the ETH Zürich. This reader recognizes the format of the network-file and uses the correct reader for the specific network-version without manual settings. This class reads the network and the parameters. Then **ObjectAttributesXmlReader** reads object attributes from a file, in lines 67-75. Followed by **HakatrinNetworkLinkImpl**, which gets the attributes for each link in the network. Now **choice setStages** initialize the choice set generator and route characteristics calculator. In line 117-120, choice set generation is done with **choice setStageCSGStochastic**(discussed in detail in the next subsection), the network is adapted with **choice setStageCSGtochasticNetworkAdapter** and the cost function is adapted by **Costfunction**.

In line 122-155,there are instructions by the coder to:

- Implement your own network adapter and replace ExampleNetworkAdapterLengthCostNormal.
- Implement your own cost function here and replace ExampleStochasticCSGCostFunctionLengthCost.

This code can also be used for non-stochastic methods. To make clear that the DSGF method has to be used, **StochasticCSGSetDoublyStochasticParameters** is imported in lines 158-187. Then there is printed for which file the choice set generation is done and how many routes were found. In the last 37 lines **choice setStageAddChosenRoute** adds the found route if it was not already included in the choice set. Then the code writes out the choice set routes by **choice setStageWritechoice sets** and **choice setStageWritechoice setsForGIS**, where GIS stand for Geographic Information System. Lastly, the choice set routes are cleared and there is written down in seconds how long the choice set generation took by comparing the system time with the starting time.

4.3 Implementation: ChoiceSetStageCSGStochastic

Choice setStageCSGStochastic is part of the package choice setGeneration.algorithms. Here CSG stands for choice set generation. choice setStageCSGStochastic is an extension of choice setStageAlgorithm. It consists of 186 lines of which 157 with data. This class will be discussed globally. First the following member variables are defined: network, router (use Dijkstra to determine the "shortest path), choice set size (determine the desired choice set), time of day and time-out (by default set to 1 week). Furthermore, a network adapter and cost function are chosen. Then there is a Boolean which indicates if we want to use the DSGF method or similar method with $\xi_{Roadtype_{ak}} = \xi_{Bikelanes_{ak}} = \xi_{LandUse_{ak}} = \varepsilon_a = 0$. Second, two similar constructors are created, of which the second is applicable for the DSGF method. Then if an origin-destination pair (OD-pair) is found for which the origin is not the same as the destination (a trivial route of length zero) and if the desired choice set size is not yet reached, the doubly stochastic algorithm is executed. If it is the first iteration the default network and cost function is used. If this is not the case, the network and the costfunction are adapted. Now the routing happens. If the program found a route that is not already in our choice set, it adds the route to the choice set and add 1 to our counter. If a route is found that is already in our choice set, the program will discard it. The code prints: "No more paths found", when no satisfying route for the OD-pair is found. Now if at least one route is found, it is stored. If not, the whole iteration is discarded. The code keeps running this process until the desired number of routes is found or the time expires. For my thesis the stop criterion is adapted . The code now stops after an predefined number of trials. For more details, see chapter 7.

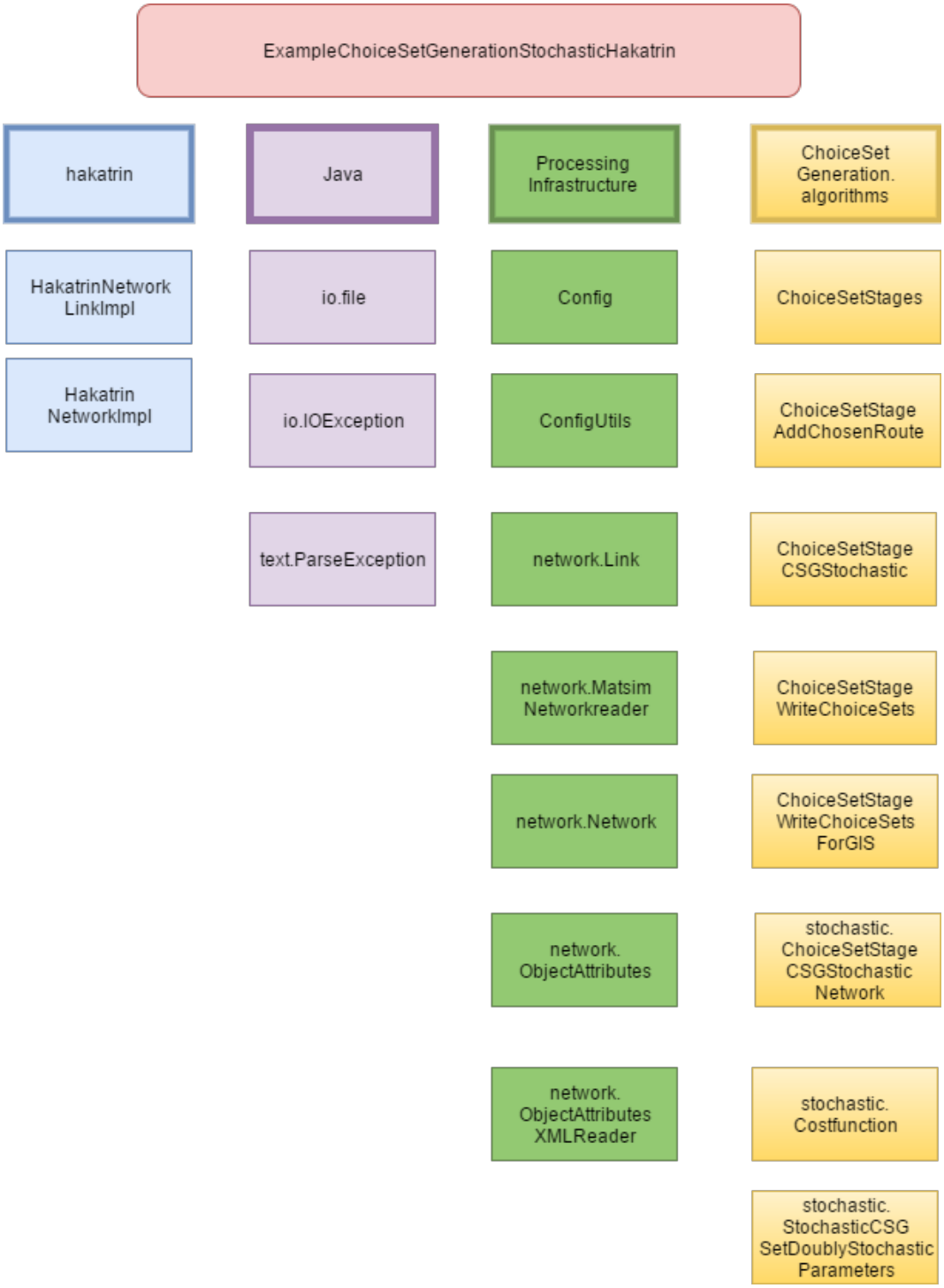


FIGURE 13 : imports of Examplechoice setGenerationStochasticHakatrin.java

5 UHasselt Code (Feathers)

5.1 Background Information

For the method of basic path components, we use the JAVA code made by IMOB , Instituut Mobiliteit connected to the University Hasselt. Knapen is author of the code. It is written for the FEATHERS (2010) project, where FEATHERS stands for: Forecasting Evolutionary Activity-Travel of Households and their Environmental RepercussionS. The main reason for this project is to facilitate the implementation of activity-based models for transport demand forecast. It is operational in Flanders, Belgium. South Korea, Slovenia and England are also interested. In these regions, FEATHERS tries to build models to predict the movements of cars, bicycles etc. However, the reality cannot be or is to describe analytically. Therefore, they split the problem in multiple aspects. For example, location choice, route choice, point of time choice and modus choice. This process is called micro simulation. In the end, all these aspects are coupled and create a compound model for the transportation behavior of an individual. To do this, they first have to predict the "demand" and have to know the "supply" of transportation facilities.

Part of the simulation model described before, is the route choice model. On basis of by GPS recorded trips, characteristics of the chosen routes are determined. Knapen handles in his doctorate thesis the complexity of the route, i.e. the number of basic path components, and his code is therefore part of the FEATHERS project.

5.2 Code

There is an application called "Route Service". This application consists of interfaces and implementation. The interfaces are available from:
code/java/src/be/uhasselt/imob/feathers2/services/routeService/
and the implementation in :
code/java/src/be/uhasselt/imob/feathers2/services/impl/routeService/.
Almost all names of the interfaces start with a capital I of interface. For the sake of completeness first the function of the interfaces are discussed, second the function of the implementation.

5.2.1 Interfaces BPC code

- **CmdArg Identifiers**: Identifiers are used iteratively for passing data. It keeps the number of argument names minimal.

- **IBasicRouteComponent**: Gives the definition of a basic path component. Determines what are the first and last node of a BPC and returns a list of links in the component.

- **ICanonicDecomposer:** This interface gives the route to be decomposed and checks if it is a simple path

- **ICanonicDecomposition:** This interface provides methods to extract data for statistics.

- **ICrdAnalyzer (Canonical Route Decomposition Analyzer)**: The CRD analyzer aims to find patterns in the canonical decomposition of large amounts of routes.
- **INetworkCanonical Route Decomposer**: Since a Canonical Route Decomposer needs to register some data in the network description, it needs to be opened to allocate data structures. After use, it shall be closed as soon as possible to free heap space.
- **RouteBasis**: Forward and backward route decomposition is done here. It will give a list of split vertex set pairs. The first node in each pair is generated by backward decomposition and the second node in each pair is generated by forward decomposition.
- **IRouteDecompStatsRepoter** This interface accepts results from multiple threads and melds them into one report.
- **RouteService**: Random Route Generation is done, to produce test data.
- **RouteSetLoader**:
  First loads the network, where the routes have to be embedded. Then delivers routes to the network.

5.2.2 Implementation BPC code

The implementation of the code consists of 17 classes. The names of the classes will be bold in the coming overview. **AESAS(Algorithm Execution Specific Attribute Set) Node Canonic Route Decomposition** keeps track off visited nodes, unvisited nodes and current distance to the origin. **Basic Route Component** defines the route Id and checks if the components constitutes a shortest path. To use the algorithm a simple path, a path with no repeating vertices, is needed, to check this **Canonic Decomposer** checks that the number of components of the forward algorithm is equal to the number of components of the backward algorithm. Furthermore, there is a method called HeadHunter. HeadHunter determines the head BPC of a route. Finds the shortest path using Dijkstra's algorithm and finds out whether or not the sub route in the shortest path covers a sequence of the route. After that, it compares the length of the shortest path equals the distance on given route from node to node. Then **Canonic Route Decomposer** provides methods, so that data can be extracted. One of the most important class **Canonical Route Decomposition Analyzer** embeds all routes, get all decompositions, can keep track of all decompositions (not advisable for large route sets) and keeps track of the run time. In **Route** all information about routes is kept track of: origin node, destination node, how the route is embedded in the network, the route Id, ownerId (who recorded the route), tripNr, walkNr, which links of the network are contained and the direction in which the link is traversed. It also detects cycles in the route, by the Boolean called Ispath. **Route Basis** checks the number of components in forward and backward direction again and prepares an array to keep track of SplitVertex defining pair offsets. **RouteCommandProvider** asks the application to: split all the routes in the data set, setup a test route generator, generate test data and to reload the configuration.

Route Decomp Stats Reporter creates a csv file, inserts the results into a sql database, generates a set of text files and generates a single xml document in a text file.The routes are loaded from sets of csv files specified by a directory and a regular expression of the file names. The class that does this is called **RouteSetLoader**.

6 Correlation with complexity

For the complexity, that is the number of basic path components, of a route, we are interested in how it is correlated with other characteristics of a path. These characteristics are length of the path **len(p)**, the number of links of the path **size(p)**, the ratio length of the path/length of the shortest path **len(p)/lenShortest(p)** and the ratio length of the path/Euclidean distance between endpoints of the path **len(p)/Euclid(p)**. The correlation coefficient (or population Pearson coefficient) for two random variables X, Y is given by

$$\rho_{X,Y} = \frac{Cov(X,Y)}{\sigma_X \sigma_Y} = \frac{\mathbb{E}[(X - \mathbb{E}X)(Y - \mathbb{E}Y)]}{\sigma_X \sigma_Y} \tag{3}$$

By the Cauchy-Schwarz inequality this coefficient is bounded between -1 and $+1$, where $+1$ means the two random variables are perfectly positive linearly related and -1 perfectly negative linearly related. When this coefficient goes to 0, the variables are more and more uncorrelated. A test in *R* is done to find out the correlation between the complexity and **len(p)**, **size(p)**, **len(p)/lenShortest(p)** and **len(p)/Euclid(p)**. The correlation is calculated by taking a full sample test, using all the 31817 routes in box R_1. The results stand in TABLE 1. Here the observed data for Amsterdam is used $(\mathscr{P}_{A,O})$, where again the ratio's are determined using Pg Admin III.

TABLE 1 : Correlation with complexity

Correlation with	Sample correlation coefficient	Confidence interval
$len(p)$	0.646	$(0.639; 0.651)$
$size(p)$	0.833	$(0.830; 0.837)$
$len(p)/lenShortest(p)$	0.239	$(0.229; 0.250)$
$len(p)/Euclid(p)$	0.149	$(0.138; 0.159)$

There is pretty much correlation between complexity and the size. Also there is a slight correlation between the complexity and length of the path. In the latter two cases, there is a lot less correlation. Plots were made for each of the four characteristics of interest (FIGURE 14). On the horizontal axis in all four cases the complexity and on the vertical axis the attribute we're interested in. All blue open circles are observations extracted from the data. The plots strengthen the statements about correlation just made.

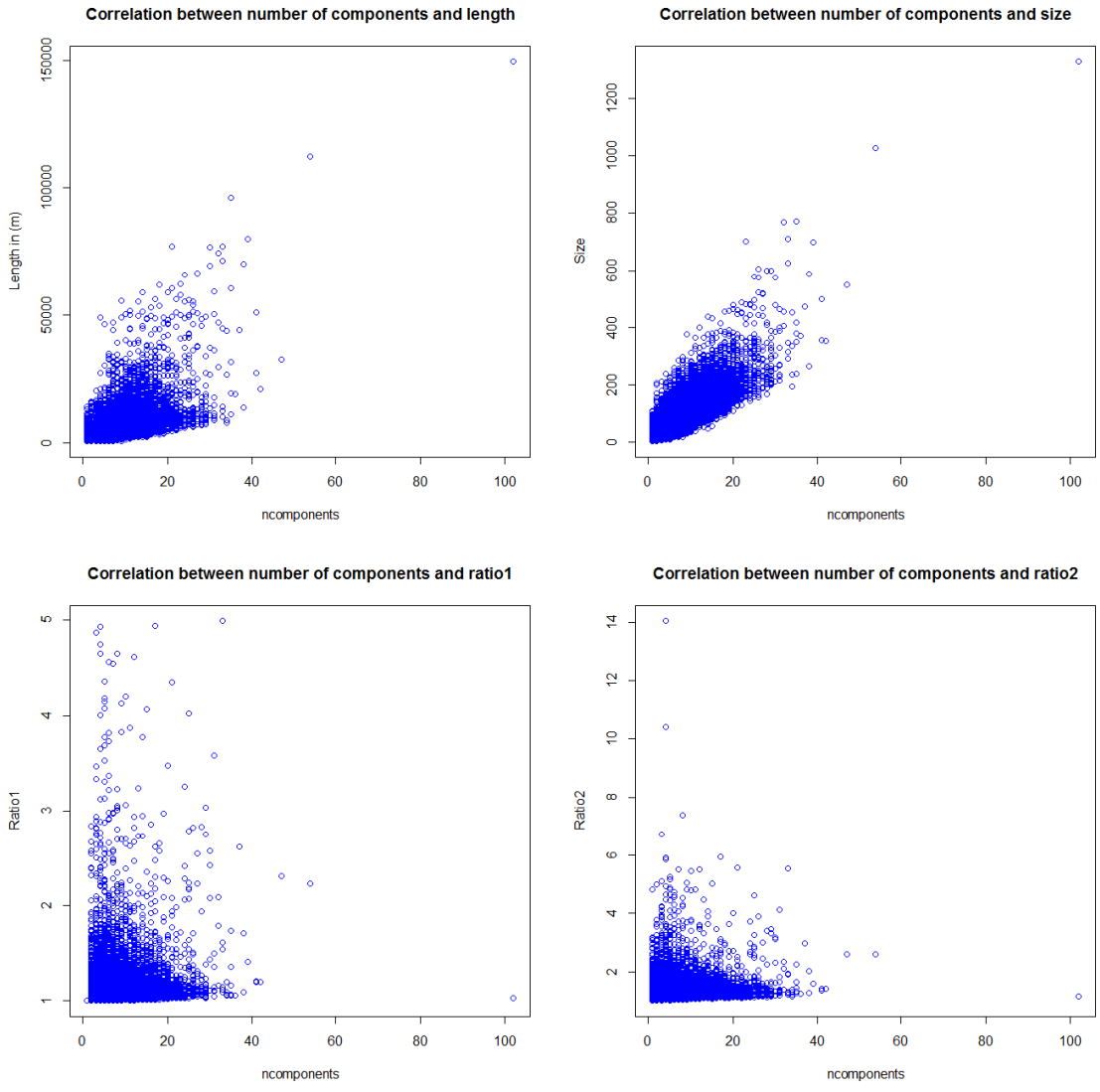


FIGURE 14 : Correlation plots

7 Combining the two methods

In this section we explain how a combination is made of the BPC method by Knapen and the DSGF method by Halldorsdottir.

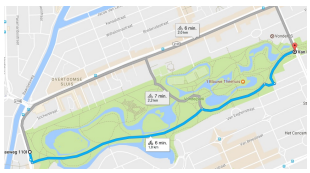
7.1 Making a combination

Two options to combine both methods come to mind. The first option is to add an extra attribute to the cost function. Then this attribute will add a positive amount to the cost function if the number of BPC's exceeds 5. In this manner routes consisting of many BPC's are more expensive and therefore less interesting. So it will be less likely that such routes will end up in our choice set. From a software design point of view this "on the fly" option seems to be complicated for this thesis as it will be very complicated to write this on the fly code.

The other option is to run the DSGF method, just like Halldorsdottir does. After that, we can make a route splitting of those routes with the method of Knapen. This splitting gives a distribution for the number of BPC's in the choice set made by the DSGF method. After that, the found choice set must be adapted, keeping in mind the idea of Knapen that routes with a high number of BPC's are highly unlikely. In the end, we chose this option. There is no agreement on how many routes a choice set generation method must produce. We arbitrary stated that the DSGF method should produce 16 routes for each OD pair and call this number N_0 for later use. The software is forced to stop at $M = 128$ iterations. For some OD-pairs the software is unable to find N_0 routes, in that case all the found routes are used. To calculate the number of BPC's for all this found POSDAP routes, the code of the ETHZ software was adapted to write all the routes to a csv file, so it can be read by the BPC code.

7.2 Choosing attributes

After that, we must choose which attributes are most interesting. Prato and Bekhor [*4*] use directional (a link is not taken in consideration if it brings the biker farther from the destination), temporal (a link is not taken in consideration if it takes significantly more time to travel this link in comparison with other links), similarity (a route is not included in the choice set if it is too similar to an already included route) and loop constraints (a segment is not included in the choice set if it causes a too large detour). They also use avoiding left turns, as turning left means interaction with other traffic and thus is considered dangerous. Fietsersbond Amsterdam (`https://fietsersbond.amsterdam/wat_vinden_wij`) pleads for bicycle routes with a low amount of cars like Weesperzijde instead of Wiboutstraat and Vondelpark instead of Overtoom (made visible in FIGURE 15).



(a) Multiple routes around Vondelpark

(b) Busy street Overtoom

(c) Quiet road in the Vondelpark

FIGURE 15 : Possible routes around the Vondelpark

Furthermore, if the maximum speed for cars exceeds 50 km/h, they want segregated bicycle lanes. This matches the survey of Halldorsdottir et al. [*11*] where she uses the attribute BikeLanes. Also they prefer red asphalt because it is very comfortable to use and remains of high quality. This in contrary to "*klinkers*", which are very bumpy and uncomfortable to ride on. This matches Halldorsdottir's attribute called RoadType, where she makes a distinction between large, small and others road. By considering the opinion of Fietsersbond Amsterdam and the will to compare with the results of Halldorsdottir, RoadType and BikeLanes are the attributes of interest and LandUse is left out. All parameters are kept as they were in the original POSDAP code.

7.3 Creating a sampler

A table with symbols is included to avoid a stream of definitions in the next paragraph.

Symbol	Meaning
$f_{A,O}(c)$	probability mass function for the complexity found in the Amsterdam set of observed routes
$F_{A,O}(c)$	cumulative distribution function for the complexity found in the Amsterdam set of observed routes
$f^P_{A,P}(c)$	probability mass function of the complexity found in the set of routes predicted for Amsterdam by the POSDAP software
$F^P_{A,P}(c)$	cumulative distribution function of the complexity found in the set of routes predicted for Amsterdam by the POSDAP software
c	the complexity (number of basic path components) for a path

Before adapting the choice set the probability distribution function (cumulative distribution function) of the observed data, $f_{A,O}(c)$ (resp. $F_{A,O}(c)$) (TABLE 3) and the routes provided by the POSDAP generation, $f^P_{A,P}(c)$ (resp. $F^P_{A,P}(c)$) (TABLE 4) have to be compared. To do this, a plot in R is made. (see FIGURE 16)

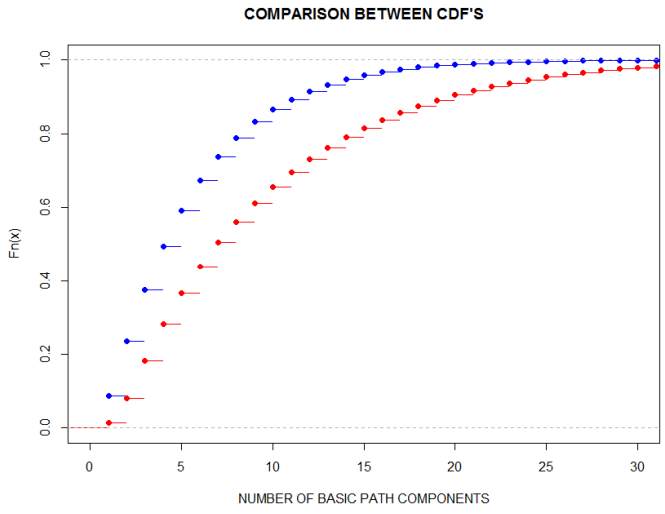


FIGURE 16 : Comparison between $F_{A,O}(c)$(blue) and $F^P_{A,P}(c)$(red)

The table shows that the cdf's are not very similar. This is also the result of various statistical tests, namely the Kolmogorov-Smirnov test (KS-test) and χ^2-test . Both test reject at an $\alpha = 0.05$ level the null hypothesis that the two data sets come from the same distribution. For the KS-test the ks.boot function in R is used to bootstrap the non-continuous distributions with 1000 repetitions. The null hypothesis is rejected if supremum of the absolute distance is higher than $c(\alpha)\sqrt{\frac{n+m}{nm}}$ where $c(\alpha) = 1.36$ and n,m are the number of data points in both samples. In our case the supremum is 0.1571, while $c(\alpha)\sqrt{\frac{n+m}{nm}} = 0.007$. So at $\alpha = 0.05$ level the null hypothesis is rejected with $p < 2.2e^{-16}$. The p-value is the probability for a given statistic model, that when the null hypothesis is true, the statistical summary (such as the sample mean difference between two compared groups) would be the same as or of greater magnitude than the actual observed results.

The result of the χ^2-test is very similar. The score of the χ^2-test statistic is 4077.39. This is much bigger than 146.57. So at $\alpha = 0.05$ level the null hypothesis is rejected with $p = 0.00050$.

We propose to alter $f^P_{A,P}(c)$ to a probability mass function very similar to $f_{A,O}(c)$, we want to do this to obtain routes that are more realistic. A discrete sampler function $f_s(c)$ is defined as follows:

$$\forall c \in \mathbb{N}^+ : f_s(c) = \beta \cdot \frac{f_{A,O}(c)}{f^P_{A,P}(c)} \tag{4}$$

$$\sum_{c\in\mathbb{N}^+} f_s(c) = \beta \cdot \sum_{c\in\mathbb{N}^+} \frac{f_{A,O}(c)}{f^P_{A,P}(c)} = 1 \tag{5}$$

$$\beta = (\sum_{c\in\mathbb{N}^+} \frac{f_{A,O}(c)}{f^P_{A,P}(c)})^{-1} \tag{6}$$

where the second expression is used to compute the constant β by normalization. After generating a candidate route r by the POSDAP code, the complexity $c(r)$ is determined. Then a random number $0 < k \leq 1$ is computed for this route r and the route is kept if $f_s(c(r)) \leq r(k)$. One difficulty in this approach is that for large values of c, i.e. many basic path components, there are very few observations. So, when there is such observation this method tends to assign a too high amount of mass to it. Two methods are tried to fix this problem. The first one is grouping boxes for larger values of c so that each box has a least 1% of the mass attached to it. In the second method all trips with $c \geq 41$ are discarded. The second method proved to work better since,

- It is easier to apply because we do not have to define the boxes containing less then 1% of the mass.
- It keeps more routes; $21k$ vs $17k$.
- The maximum error in comparison with $f_{A,O}(c)$ is 0.3% in the second method and 0.6% in the box method.

The found sampler distribution $f_s(c)$ (TABLE 5) was applied to the 494546 found routes of the POSDAP method and kept 21425 routes. The used $\beta = 0.043$. Initially the POSDAP software tried to find 16 alternatives for each OD pair and after applying the sampling function the following number of alternatives are kept (TABLE 2).

TABLE 2 : Distribution of the number of kept alternatives

Number of Alternatives	Frequency
0	16985
1	9831
2	3711
3	1043
4	200
5	40
6	6
7	1

If the sampler function is applied the maximal error between $f_{A,O}(c)$ and probability mass function of the sampled data is 0.3%, so in that sense the sampler is working well. However, in $84,2\%$ of the cases there is at most one route kept by the sampler. The goal was to make a choice set, so another method is proposed in the next paragraph.

7.4 Using a maximum likelihood function in the sampler

Symbol	Meaning
$\mathscr{P}_{A,P}$	set of routes predicted for Amsterdam
$\mathscr{P}_{A,O}$	set of observed paths assumed to be related to Amsterdam
N_0	desired number of predictions by the POSDAP software
N_1	the number of alternatives we want in our choice set for each OD-pair

For each OD pair we want at least N_1 alternatives in our choice model. To do this, define $N_0(i)$ as the initial number of found alternatives for OD-pair $\langle O_i, D_i \rangle$ and $N_1(i)$ as the number of alternatives we want to keep for OD-pair $\langle O_i, D_i \rangle$. This number is equal to the minimum of $N_0(i)$ and N_1. Now call the set of POSDAP predictions for OD-pair $\langle O_i, D_i \rangle$; $\mathscr{P}_{A,P}(O_i, D_i)$. We are interested in all subsets of $\mathscr{P}_{A,P}(O_i, D_i)$ with cardinality $N_1(i)$. Denote S_i^k as one of these subsets. The likelihood for S_i^k to have been drawn from a set for which is the complexity distribution is $f_{A,O}(c)$ is given by:

$$L(S_i^k) = \prod_{r \in S_i^k} f_{A,O}(c(r)) \tag{7}$$

The subset with the maximal value is kept as the choice set. Define this subset as S_i:

$$S_i = \underset{p \subset \mathscr{P}_{A,P}(O_i, D_i)}{\arg\max} \; L(p) \tag{8}$$

The required number of subset evaluations n_E is $\binom{N_0(i)}{N_1(i)}$. One possible problem with this approach is the case that $L(p) = 0, \forall p \subset P_{A,P}$, so that a random subset S_i has to be selected. To fix this, $f'_{A,O}(c(r))$ is defined as:

$$f'_{A,O}(c(r)) = \begin{cases} f_{A,O}(c(r)) & \text{if } f_{A,O}(c(r)) > 0 \\ \frac{1}{|\mathscr{P}_{A,O}| \cdot c(r)} & \text{otherwise} \end{cases} \tag{9}$$

And replace $L(S_i^k)$ by $\prod_{r \in S_i^k} f'_{A,O}(c(r))$. This is done in order to favor routes with low complexity.

This process is executed, by running the program sampleCsFromPosdapPreds, which performs the process described above, for $N_1 = \{3,4,5,6,7,8,9\}$ to see what the effect of choosing N_1 is. Something strange is occurring for routes using only one BPC for all values of N_1. Namely, the original Amsterdam observed data consists for 8.7% of only one BPC. Using this method, a percentage of 0.4% is found of routes consisting of only one BPC for all possible values of N_1. This effect probably occurs because $f_{A,O}(2) > f_{A,O}(1)$, so if there are routes consisting of a few BPC's, it is more likely to include only the routes consisting of 2 BPC's. In general we observe that the amount of mass given to routes consisting of more BPC's increases if N_1 increases. In TABLE 6 is this made visible. By determining which number N_1 performs best, look at the maximal error in relative frequency leaving out the frequencies belonging to a complexity of 1 and where errors occur. For $N_1 = \{6,7,8\}$ is the maximal error minimal around 2%. For all these values there is an overestimation for the tails and for $N_1 = 6$ there is also a large overestimation when the complexity is 2. Concluding for this method, $N_1 = 7$ or $N_1 = 8$ are not perfect, but the best we can do for this method. A large advantage for this method is that for almost all OD-pair a choice set consisting of 7 or 8 routes can be found. So a bit of quality is given up for quantity.

8 Making the choice: Logit Model

8.1 Multinomial Logit Model

Using previous section we can construct a choice set consisting of 7 or 8 routes for almost each OD-pair. To make a choice out of these routes, one often uses the logit method. The name logit is derived from the analogy with the probit model. The probit function is the quantile function of a standard normal distribution. In this logit method a utility function is needed. The utility U_{in} for choicemaker n and alternative i is given by V_{in} an observable part and ε_{in} a stochastic error part ($U_{in} = V_{in} + \varepsilon_{in}$). In general, a choicemaker chooses the alternative with the highest utility. This choice can be inconsistent because the stochastic error plays a role. We can define $P_i(n)$ as

$$P_i(n) = P(U_{in} \geq U_{jn}, \quad \forall j \in C_n, i \neq j) \tag{10}$$

the probability that choicemaker n chooses alternative i, where C_n is the choice set of choicemaker n as defined earlier.

In the most basic multinomial logit model (MNL) , if we assume that the errors are i.i.d $Gumbel(0, \mu)$, so with cumulative density function $F(\varepsilon) = e^{-e^{\mu\varepsilon}}$, it can be derived that

$$P_i(n) = \frac{e^{V_{in}}}{\sum_{j \in C_n} e^{V_{jn}}} \tag{11}$$

where one can check that the sum of probabilities for alternatives in C_n is equal to 1. From now on we omit choicemaker n, because we have built a choice set which is equal for each cyclist. The MNL model seems very simple but is used to study travel modes, choice of occupation, brand of automobile purchase, and decisions on marriage and number of children. Actually, McFadden received in 2000 a Nobel Prize in economics for his work.

In our case a choice has to be made out of 7 alternatives for each $OD-$pair. The most important attribute in this thesis is the number of basic path components. We do not know right now how to construct a utility function for this particular case, hence we are right now not able to choose between the 7 routes found by our program. Once we know how the utility function looks like, one can use the Biogeme software written by Bierlaire [*13*]. Biogeme is an open source freeware designed for the maximum likelihood estimation in discrete choice models. Also multiple logit models are part of this software. In chapter 9 more on these other logit models. The model that is needed for the MNL model is called *01logit*. It is written for the hypothetical choice for people to choose between train, car and Swiss metro (a vacuum tube train). Therefore, it should be adapted to be useful for choosing between routes.

9 Further Research

For further research it might be interesting to look at other areas than the city of Amsterdam, for example at smaller cities like Amersfoort or rural areas such as the north of the Netherlands (Groningen, Friesland, etc.). It is expected that the distributions of complexity $(f_{X,O})$, where X is a place to be determined, are different here and also might influence, for example, the sampler function we have to use to make a realistic choice set for these places.

In the POSDAP software there are other predictors described. One can maybe look at how the distributions differ from the distribution we found $(f^P_{A,P})$. Due to timing considerations, there was not unfortunately enough time to answer this question in this thesis because each run will take around 60 to 70 hours of computing time. This because over 500.000 routes are predicted using single threaded software which means on average that 2 routes are predicted per second. The algorithm can be improved by rewriting the POSDAP code as a multi threaded software.

With respect to split vertices, there could be research done on the occurrence frequency. What makes it that some vertices occur very frequently as split vertex and others do not and is it possible to see patterns in how these are distributed in a network.

The model in section 8 can be improved in various ways. A slight adaption can be made to this model by grouping very similar paths into a so-called nest, where each alternative is grouped into exactly one of the nests. This model is called the Nested Logit (NL) . However, these methods are not suitable for this type of route choice sets. Prato [*14*] states: *"MNL does not allow to account for similarity among alternatives, while NL assumes that each alternative belongs exclusively to one nest while in real-size networks routes share links with hundreds of other paths."* Therefore, he discusses multiple variations of the MNL and NL. I used his work to give an overview of some variations of the MNL.

Cascetta et al. [*15*] came up with C-logit, where a commonality factor CF_i is defined to give very similar routes less utility. The probability of choosing route i is

$$P_i = \frac{e^{V_i+\beta_{CF}\cdot CF_i}}{\sum_{j\in C} e^{V_j+\beta_{CF}\cdot CF_j}} \tag{12}$$

where β_{CF} is a parameter to be estimated. For the commonality factor CF_i various expressions are proposed and tested of which

$$CF_i = ln[1+\sum_{j\in C, i\neq j}(\frac{L_{ij}}{\sqrt{L_iL_j}})(\frac{L_i-L_{ij}}{L_j-L_{ij}})] \tag{13}$$

seemed to perform as expected and satisfactory. Here L_i, L_j are lengths of routes i and j and L_{ij} is the overlapping length between those two routes. Remark that CF_i is positive, so our estimator β_{CF} should be negative in order to penalize similar routes.

For Path Size Logit by Ben-Akiva and Bierlaire [*16*] the probability of choosing route i is of a very similar form, namely:

$$P_i = \frac{e^{V_i + \beta_{PS} \cdot \ln(PS_i)}}{\sum_{j \in C} e^{V_j + \beta_{PS} \cdot \ln(PS_j)}} \tag{14}$$

where β_{PS} is a parameter to be estimated. The original expression of PS_i is:

$$PS_i = \sum_{a \in \Gamma_i} \frac{L_a}{L_i} \frac{1}{\sum_{j \in C} \delta_{aj}} \tag{15}$$

here L_a is the length of link a, Γ_i is the set of links belonging to route i and

$$\delta_{aj} = \begin{cases} 1 & \text{if route } j \text{ uses link } a \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

(*Remark: Earlier the size of the path was defined as the number of links of this path. In this case it counts the number of new alternatives the routes add to the choice set.*)

Looking at this formula, the first that one notices is that one accounts for the fraction of the length a link makes part of the route. Second, if a link from a certain route is unique then $\frac{1}{\sum_{j \in C} \delta_{aj}}$ equals 1 and if a link appears in 4 routes $\frac{1}{\sum_{j \in C} \delta_{aj}}$ will be $\frac{1}{4}$.

Prato [*14*] states that Path Size Logit modifications outperform C-logit models and that is the reason more survey is done to improve the PSL method. One of this modifications is the Path Size Correction Logit (PSC) by Bovy et al. [*17*]. In this model $\ln(PS_i)$ is replaced by PSC_i which is of the following form:

$$PSC_i = -\sum_{a \in \Gamma_i} \left(\frac{L_a}{L_i} \ln \sum_{j \in C} \delta_{aj}\right) \tag{17}$$

This PSC_i is clearly negative resulting in subtracting utility in the case of very similar paths, making this method a more intuitive angle of view. It can be viewed as a combination of both the Multinomial and Nested Logit model. In predictions this PSC provides better performance that the original PSL model.

Also Dugundji, Knapen and myself are planning to write a paper about the conclusions of this thesis.

10 Discussion and Conclusion

Concluding, there are various methods to generate choice sets. In this thesis the Doubly Stochastic Generation Function is chosen. Mostly, because it generates heterogeneous routes, performs well up to trips of 10 km and puts the more attractive routes in the choice set. A common problem for these methods is that the routes they produce, are overcomplicated and unreal.

To overcome this problem, we looked at the data provided by the FietsTelweek in Amsterdam. We looked at the distribution of complexity for all routes in box R_1 and at the distribution of complexity for routes in R_1 with a low r_d value. We found out that there was only a slight mass shift towards lower complexities, so box R_1 is the data set we worked with along this thesis.

After that, routes were generated with a slightly adapted POSDAP code and compared in FIGURE 16 showing the cumulative density functions of the observed and the predicted data. From this it was observed that the DSGF method indeed produced way too many complex routes. This is in line with what we expected from earlier results. In an attempt to fix this a sampler was build to filter routes with a high complexity out of predicted data. By doing this the idea was to get a choice set with routes with a complexity distribution similar to the observed data in Amsterdam. This worked in terms of getting a complexity distribution very similar to the observed data but only in $16,2\%$ of the origin destination pairs, there was an actual choice set left, with at least 2 choices.

We decided that giving up a bit of quality for quantity was needed. In order to do this, we looked at the likelihood of the observed routes to value routes with a relative low complexity over routes with a relative high complexity. This method also has it limitations as it is underestimating the number of routes with only one basic path component and overestimating the number of routes with two basic path components. Using this method we observed that keeping 7 or 8 out of the 16 predicted routes, provide us a choice set that is most similar to the observed data. One of the limitations of this conclusion is that this only holds true for the city Amsterdam and maybe for other cities with a high dense network.

From our results it might be useful for other researchers in this field to filter out unlikely routes when constructing a choice set. This can be done by applying the method of basic path components but maybe there are other working methods or methods yet to be found.

Furthermore, it is shown that there is correlation between the number of components of a route and the length of the route and correlation between the number of components of the route and size of the route. So if one wants to construct a choice set for longer bicycle routes, we expect that routes with a higher number of basic path components must be kept.

Appendices

A Tables

complexity	absFreq	relFreq	relCumFreq
1	2782	0,087	0,087
2	4691	0,147	0,235
3	4424	0,139	0,374
4	3855	0,121	0,495
5	3065	0,096	0,592
6	2583	0,081	0,673
7	2039	0,064	0,737
8	1671	0,053	0,789
9	1370	0,043	0,832
10	1062	0,033	0,866
11	866	0,027	0,893
12	717	0,023	0,916
13	565	0,018	0,933
14	444	0,014	0,947
15	363	0,011	0,959
16	293	0,009	0,968
17	225	0,007	0,975
18	175	0,006	0,980
19	139	0,004	0,985
20	96	0,003	0,988
21	81	0,003	0,990
22	63	0,002	0,992
23	49	0,002	0,994
24	27	0,001	0,995
25	40	0,001	0,996
26	32	0,001	0,997
27	16	0,001	0,998
28	19	0,001	0,998
29	20	0,001	0,999
30	6	0,000	0,999
31	10	0,000	0,999
32	3	0,000	0,999
33	6	0,000	1,000
34	4	0,000	1,000
35	5	0,000	1,000
36	1	0,000	1,000
37	1	0,000	1,000
38	2	0,000	1,000
39	1	0,000	1,000
40	0	0,000	1,000
Total	**31811**		

TABLE 3 : Observed Data FietsTelweek Amsterdam

complexity	absFreq	relFreq	relCumFreq
1	6660	0,013	0,013
2	32737	0,066	0,080
3	50918	0,103	0,183
4	49167	0,099	0,282
5	41976	0,085	0,367
6	36155	0,073	0,440
7	31759	0,064	0,504
8	27983	0,057	0,561
9	24848	0,050	0,611
10	22345	0,045	0,656
11	19364	0,039	0,695
12	17555	0,035	0,731
13	15488	0,031	0,762
14	13972	0,028	0,790
15	12297	0,025	0,815
16	11044	0,022	0,838
17	9778	0,020	0,857
18	8715	0,018	0,875
19	7841	0,016	0,891
20	6819	0,014	0,905
21	6050	0,012	0,917
22	5428	0,011	0,928
23	4638	0,009	0,937
24	4180	0,008	0,946
25	3768	0,008	0,953
26	3273	0,007	0,960
27	2862	0,006	0,966
28	2538	0,005	0,971
29	2146	0,004	0,975
30	1887	0,004	0,979
31	1621	0,003	0,982
32	1372	0,003	0,985
33	1238	0,003	0,988
34	1020	0,002	0,990
35	847	0,002	0,991
36	699	0,001	0,993
37	623	0,001	0,994
38	504	0,001	0,995
39	444	0,001	0,996
40	325	0,001	0,997
Total	**492884**		

TABLE 4 : Predicted Data by POSDAP for Amsterdam

complexity	fAO/fAP	fS
1	6,494	0,278
2	2,228	0,095
3	1,351	0,058
4	1,219	0,052
5	1,135	0,049
6	1,111	0,048
7	0,998	0,043
8	0,928	0,040
9	0,857	0,037
10	0,739	0,032
11	0,695	0,030
12	0,635	0,027
13	0,567	0,024
14	0,494	0,021
15	0,459	0,020
16	0,412	0,018
17	0,358	0,015
18	0,312	0,013
19	0,276	0,012
20	0,219	0,009
21	0,208	0,009
22	0,180	0,008
23	0,164	0,007
24	0,100	0,004
25	0,165	0,007
26	0,152	0,007
27	0,087	0,004
28	0,116	0,005
29	0,145	0,006
30	0,049	0,002
31	0,096	0,004
32	0,034	0,001
33	0,075	0,003
34	0,061	0,003
35	0,092	0,004
36	0,022	0,001
37	0,025	0,001
38	0,062	0,003
39	0,035	0,001
40	0,000	0,000
sum	**23,357**	
beta	**0,043**	

TABLE 5 : Application of the sampler method

Complexity	Observed Data	Value of N_1						
		3	4	5	6	7	8	9
1	0,087	0,004	0,004	0,004	0,004	0,004	0,004	0,004
2	0,147	0,256	0,222	0,192	0,168	0,147	0,130	0,116
3	0,139	0,142	0,149	0,156	0,160	0,161	0,159	0,155
4	0,121	0,099	0,101	0,102	0,104	0,106	0,108	0,110
5	0,096	0,077	0,079	0,081	0,083	0,084	0,084	0,085
6	0,081	0,063	0,064	0,066	0,067	0,068	0,069	0,070
7	0,064	0,054	0,055	0,056	0,058	0,059	0,060	0,060
8	0,053	0,047	0,048	0,049	0,050	0,051	0,052	0,052
9	0,043	0,039	0,041	0,042	0,043	0,044	0,045	0,046
10	0,033	0,033	0,035	0,037	0,038	0,038	0,039	0,040
11	0,027	0,029	0,030	0,031	0,032	0,032	0,034	0,034
12	0,023	0,024	0,026	0,027	0,028	0,029	0,030	0,030
13	0,018	0,021	0,022	0,023	0,024	0,025	0,025	0,026
14	0,014	0,018	0,019	0,020	0,021	0,022	0,023	0,023
15	0,011	0,014	0,016	0,017	0,018	0,019	0,020	0,020
16	0,009	0,013	0,014	0,015	0,016	0,017	0,018	0,018
17	0,007	0,011	0,011	0,012	0,013	0,014	0,015	0,015
18	0,006	0,010	0,010	0,011	0,011	0,012	0,013	0,013
19	0,004	0,009	0,010	0,010	0,010	0,011	0,011	0,012
20	0,003	0,007	0,008	0,008	0,008	0,009	0,009	0,010
21	0,003	0,006	0,007	0,007	0,008	0,008	0,008	0,009
22	0,002	0,005	0,006	0,006	0,007	0,007	0,008	0,008
23	0,002	0,004	0,005	0,005	0,006	0,006	0,006	0,007
24	0,001	0,001	0,002	0,002	0,002	0,003	0,003	0,004
25	0,001	0,004	0,005	0,005	0,006	0,006	0,006	0,006
26	0,001	0,003	0,004	0,004	0,004	0,005	0,005	0,005
27	0,001	0,001	0,001	0,001	0,001	0,002	0,002	0,002
28	0,001	0,001	0,001	0,002	0,002	0,002	0,003	0,003
29	0,001	0,002	0,003	0,003	0,003	0,003	0,004	0,004
30	0,000	0,000	0,001	0,001	0,001	0,001	0,001	0,001
31	0,000	0,001	0,001	0,001	0,001	0,002	0,002	0,002
32	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
33	0,000	0,001	0,001	0,001	0,001	0,001	0,001	0,002
34	0,000	0,000	0,000	0,000	0,001	0,001	0,001	0,001
35	0,000	0,001	0,001	0,001	0,001	0,001	0,001	0,001
36	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
37	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
38	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,001
39	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
40	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000

TABLE 6 : Relative Frequencies for the observed data and different values of N_1

B Functions of classes in hakatrin.choice setGeneration.hakatrinStochastic

- Examplechoice setGenerationBalmerHakatrin.java
  Same function as Examplechoice setGenerationStochasticHakatrin.java
- Examplechoice setGenerationStochasticHakatrin.java
  Discussed in chapter 4
- Examplechoice setStagechoice setGenerationBalmer.java
  Same function as Examplechoice setGenerationStochasticHakatrin.java
- ExampleCostFunctionLengthLandUse.java
  Determine parameters Length and LandUse
- ExampleCostFunctionLengthPath.java
  Determine parameters Length and Path
- ExampleCostFunctionLengthType.java
  Determine parameters Length and RoadType
- ExampleCostFunctionLengthTypePathLandUse.java
  Determine Length, LandUse, Path and RoadType
- ExampleNetworkAdapterLengthLandUseNormal.java
  Computes $\xi_{landuse}$ and ε_a
- ExampleNetworkAdapterLengthPathNormal.java
  Computes ξ_{path} and ε_a
- ExampleNetworkAdapterLengthTypeNormal.java
  Computes $\xi_{lroadtype}$ and ε_a
- ExampleNetworkAdapterLengthTypePathLandUseNormal.java
  Computes $Length, \xi_{roadtype}, \xi_{landuse}, \xi_{lpath}$ and ε_a
- ExampleStochasticSetDoublyStochasticParametersLengthLandUse.java
  Sets $\xi_{landuse}$ and ε_a
- ExampleStochasticSetDoublyStochasticParametersLengthPath.java
  Sets ξ_{lpath} and ε_a
- ExampleStochasticSetDoublyStochasticParametersLengthType.java
  Sets $\xi_{lroadtype}$ and ε_a
- ExampleStochasticSetDoublyStochasticParametersLengthTypePathLandUse.java
  Sets $Length, \xi_{roadtype}, \xi_{landuse}, \xi_{lpath}$ and ε_a
- config.xml
- configpara.xml

Alphabetical Index

REFERENCES

[1] Knapen, L., I. B. Hartman, D. Schulz, T. Bellemans, D. Davy Janssens, and G. Wets, Determining structural route components from GPS traces. *Transportation Research Part B: Methodological (90)*, 2016, pp. 156–171.

[2] Fietstelweek, Data Fietstelweek, 2015.

[3] Rieser-Schüssler, N., M. Balmer, and K. W. Axhausen, Route choice sets for very high-resolution data. *Working paper Transport and Spatial Planning*, 2012.

[4] Prato, C. G. and S. Bekhor, Applying Branch-and-Bound Technique to Route Choice Set Generation. *TRANSPORTATION RESEARCH RECORD JOURNAL OF THE TRANSPORTATION RESEARCH BOARD*, 2006.

[5] Frejinger, E., Route sampling of alternatives in a route choice context, 2007.

[6] Bovy, P. H. L., On Modelling Route Choice Sets in Transportation Networks: A Synthesis. *Transport Reviews, Vol. 29, No. 1*, 2009, p. 58.

[7] Kazagli, E. and M. Bierlaire, A route choice model based on Mental Representations. *15th Swiss Transport Research Conference*, 2015.

[8] Nielsen, O. A., A stochastic transit assignment model considering differences in passengers utility functions. *Transportation Research Part B 34 (2000)*, 2000.

[9] Bovy, P. H. and S. Fiorenzo-Catalano, Stochastic route choice set generation: Behavioral and probabilistic foundations. *Transportation Research Part B 34 (2000)*, 2007.

[10] Hood, J., E. Sall, and B. Charlton, A GPS-based bicycle route choice model for San Francisco, California. *Transportation Letters: The International Journal of Transportation Research (2011) 3: (63-75)*, 2007.

[11] Halldorsdottir, K., N. Rieser-Schüssler, K. W. Axhausen, O. A. Nielsen, and C. G. Prato, Efficiency of choice set generation methods for bicycle routes. *EJTIR 14(4)*, 2014.

[12] Pucher, J. and J. Buehler, City Cycling, 2012, p. 13.

[13] Bierlaire, M., BIOGEME: A free package for the estimation of discrete choice models. *Proceedings of the 3rd Swiss Transportation Research Conference, Ascona, Switzerland*, 2003.

[14] Prato, C. G., Route choice modeling: past, present and future research directions. *Journal of Choice Modeling*, 2009, pp. 77–87.

[15] Cascetta, E., A. Nuzzolo, F. Russo, and A. Vitetta, A modified logit route choice model overcoming path overlapping problems: specification and some calibration results for interurban networks. *Proceedings of the Thirteenth International Symposium on Transportation and Traffic Theory*, 1996, pp. 697–711.

[16] Ben-Akiva, M. and M. Bierlaire, Discrete choice methods and their applications to short term travel decisions. *Handbook of Transportation Science*, 1996.

[17] Bovy, P., S. Bekhor, and C. Prato, The factor of revised path size: an alternative derivation. *Transportation Research Record, 2076*, 2008, pp. 132–140.