Specialization: Transport Engineering and Logistics

Report number: 2016.TEL.8048

Title: Routing algorithm for a time-dependent, stochastic network

Author: D.L. Overbeek

Title (in Dutch) Route bepalend algoritme voor een tijdsafhankelijk, stochastisch netwerk

Assignment: Research assignment

Confidential: no

Initiator (university): prof.dr.ir. G. Lodewijks

Supervisor: Dr.ir. F. Corman

Date: August 19, 2016

This report consists of 90 pages and 3 appendices. It may only be reproduced literally and as a whole. For commercial purposes only with written authorization of Delft University of Technology. Requests for consult are only taken into consideration under the condition that the applicant denies all legal rights on liabilities concerning the contents of the advice.
Subject: Shortest Paths in Time-Dependent, Stochastic networks

The availability of track-and-trace devices such as GPS has provided the possibility to record real-life trajectories of many vehicles. This unleashes possibilities for accurate and reliable routing information, critical for the logistics performance. This research assignment is to research a model and algorithm for defining shortest paths in a network, where the travel time is a stochastic variable described by continuously recorded data. This relates to extending a time-independent existing model (and related software) available, that is able to analyze transport networks and compute shortest paths in a stochastically described network. The challenge for this research assignment is to include a degree of time-dependency in the approach and the resolution algorithm, i.e. travel time distributions can be characterized by different stochastic distributions at different times of the day.

The research in this assignment should cover the following:

- Develop a model for describing shortest paths in transport networks, and an algorithm for computing them, in the setting described above;
- Briefly review time-dependent approaches for shortest paths problems; and shortest path problems in stochastic settings;
- Define theoretical characteristics of a shortest path in the setting described above, and a general procedure that is theoretically able to address the problem;
- Implement the procedure in a realistic, usable setting, by extending the existing code (in C#) or defining suitable add-ons to it;
- Test the resulting approach in a given real-life test case

Based on your research, it is expected that you conclude with a recommendation for future research opportunities and potential for more ideas and/or applications. The report must be written in English and must comply with the guidelines of the section. Details can be found on the website.

For more information, contact Dr.ir. F. Corman (B 3 290; f.corman@tudelft.nl).

The professor.
Prof. dr. ir. G. Lodewijks
Summary

Reliable routing is important in transportation, up to 75% of freight is transported by road in 2012 in the EU. Routing systems nowadays use deterministic link travel times that causes uncertainty in arrival time. The most reliable method would be an routing algorithm that takes network stochasticity and time-dependency into account. Per given starting time, the algorithm advises a set of non-dominated routes and its arrival time reliability. The development of such an algorithm is described in this report.

Data is generated that is used in the algorithm, this is done with a normal distribution. Several grid sizes are used, ranging from a 3x3 to a 25x25 grid. Different blocksizes are assumed (blocksize representation given in parentheses): 2 (30 min.), 4 (15 min.), 6 (10 min.), 10 (6 min.) and 12 (5 min.). Five correlation scenarios are taken into account: strong negative, negative, positive, strong positive and no correlation.

Experiments showed that positive correlated links shifted the arrival times to right, whereas negative correlated links shifted the arrival times to left. This effect is not seen when clustering is applied. Instead, a tiny decrease in arrival time is noticed when correlation links are clustered. Previous research disagrees with this effect; further research should be done to reveal the source of this effect, it might be due to the self-generated data.

A city example is made in which some links are correlated and the others are almost stationary. Positive correlated links increased arrival time, as congestion of one link continues on adjacent links. The city example also showed that detours become beneficial.

In larger grids, with size 10x10 and larger, many advised routes show large similarities. This might be related to the fact that alternative routes have higher link travel times, a wider variance range and suffer more from peak times.

The computational performance of the algorithm is also tested. A larger blocksize (i.e. more accurate data, more time steps in a day) increased the computational time. Blocksize 2 has an enormous increase in computational time for larger grids (≥20x20). The number of non-dominated routes also increases rapidly for this blocksize, hence the long computational time. It is advised to use larger blocksizes for data; data with blocksize 2 is less accurate in arrival time (each block is 30 minutes) and suffers long computational times.

Shorter routes are calculated faster due to the termination statement, the influence of blocksize on the results is comparable.

A basis is laid for a time-dependent routing algorithm. It would be interesting to test the algorithm with real-time data and networks, also to study the clustering effect. Further improvements can be made in the algorithm though, for example making data input more generic, speed improvements (also, more efficient route searching) or implementation of a generic clustering algorithm.
# Contents

Summary ........................................................................................................................................... v

1. Introduction.................................................................................................................................... 1

2. Background .................................................................................................................................. 3

3. Model fundamentals .................................................................................................................... 5

4. Data generation ............................................................................................................................ 9
   4.1 Link data generation ............................................................................................................... 9
   4.2 Correlated grids .................................................................................................................... 10
   4.3 File storage and importation ................................................................................................. 11

5. Algorithm .................................................................................................................................... 13
   5.1 General information ................................................................................................................ 13
   5.2 Scripts and functions ............................................................................................................. 17
   5.3 Example run of the algorithm ............................................................................................... 24

6. Experiments and results .............................................................................................................. 29
   6.1 Standard outputs ................................................................................................................... 29
   6.2 City effect .............................................................................................................................. 33
   6.3 Recurring route parts ............................................................................................................ 38
   6.4 Correlation effect & Clustering ............................................................................................ 42
      6.4.1 Correlation effect .......................................................................................................... 42
      6.4.2 Clustering ..................................................................................................................... 43
   6.5 Computational times .............................................................................................................. 46

7. Conclusion ................................................................................................................................... 51

References ....................................................................................................................................... 53

Appendix A – Effect of blocksize on arrival times ........................................................................ 55

Appendix B – Results of computational times .............................................................................. 57
   B.1 Correlation scenarios ........................................................................................................... 57
   B.2 Profiler ................................................................................................................................... 59

Appendix C – Matlab scripts ......................................................................................................... 65
   C.1 Routing algorithm scripts .................................................................................................... 65
   C.2 Data generation scripts ........................................................................................................ 78
1. Introduction
Routing is important in the world of transportation engineering and logistics, up to 75% of freight is transported by road in the EU in 2012 [2]. This freight must be transported as fast as possible, thus reliable routing is essential.

Many routing systems advise the shortest or fastest route based on a deterministic network, they do not consider network stochastics and thus congestions that increase the travel time are not uncommon on a route. Taking network stochastics into account, links that are sensitive to congestion can be avoided. Some algorithms are developed that take into account network stochastics, resulting in a more reliable route. These algorithms are time-independent.

Adding time-dependency to the routing algorithm, results in even more reliable routes to be advised. Depending on the desired departure time, the routing algorithm can advise one or more fastest routes (that are non-dominated). Or, given an arrival reliability, a set of routes with corresponding departure time can be advised.

Research question
The research question to be answered is:

“What routing algorithm can be developed that takes stochastic, time-dependent networks into account?”

Sub questions belonging to this research question are:
- What current algorithms are developed for routing using stochastic links (and time-dependency)?
- What are the fundamentals of the model?
- How to translate the fundamentals to an MATLAB algorithm?
- What types of link data should be generated to evaluate the algorithm?
- What is the influence of correlation and clustering on the results?

Structure of the report
The structure of this report is as follows. First a background into time-dependent stochastic network routing is given, Chapter 2. Chapter 3 explains some fundamentals of the model. Chapter 4 describes the data used in the model. Chapter 5 explains the routing algorithm. This is followed by experiments and results, described in Chapter 6. At last, in Chapter 7, a conclusion is drawn and recommendations are given.
2. Background

Corman et al. [3] studied how to find reliable routes in time-independent stochastic networks. This report is a continuation on their work, now assuming time-dependent stochastic networks. Routing in time-dependent, stochastic networks is studied by Miller-Hooks & Mahmassani [4] who created an algorithm to determine the least expected time paths in these networks. Nie & Wu [5] and Huang & Gao [6] have similar goals and used the study of Miller-Hooks & Mahmassani as the basis for both their studies. The algorithm described in this report is also based on their algorithm and shows similarities with the algorithms of Nie & Wu and Huang & Gao. This section explains the differences and similarities between the mentioned studies and the algorithm in this report.

Dependency

All studies, except the one from Huang & Gao, consider that link travel times are independent [4, 5]. Network dependency is not considered. This report generates independent link travel times and does not consider network dependency. Huang & Gao use data that is dependent in both time and space [6].

FIFO and cycles

First-in-first-out data is required to prevent the occurrence of cycles in the optimal path(s) [4, 5]. Huang & Gao do not mention cycles or FIFO data. The data used in this report is FIFO, this is checked with a script (explained later). Thus cycles will not be found in the optimal path(s). FIFO data is not a strict requirement in this report or the papers, the algorithm can run with non-FIFO data [4, 5]. Although cycles are always dominated by other paths, a script can be used to check for cycles (explained later). Nie & Wu use CYCLE-CHECK, but also mention the fact that it is not necessary [5]. The algorithm of CYCLE-CHECK is not described thus cannot be compared to the script in this report.

Peak period and data

The data generated in this report is random and time-dependent for each link at each time step. There are no static and deterministic data. Peaks are included in this data. Miller-Hooks & Mahmassani and Huang & Gao use so called peak periods [4, 6]. Data at some time steps are random and time-dependent, the peak period, whereas other data is static and deterministic. Nie & Wu do not explain their data.

Bellman’s principle of optimality and non-dominance

Bellman’s principle of optimality and non-dominance is valid in Miller-Hooks & Mahmassani [4] and Nie & Wu [5]. Huang & Gao [6] disagree with these principles and use pure paths to determine optimal paths. Pure paths are a subset of non-dominated paths. The algorithm in this report uses non-dominated paths.

Non-dominance check

There are three criteria methods to check for dominance [1]: deterministic dominance, first order stochastic dominance and expected value dominance. Miller-Hooks & Mahmassani [4] base their dominance on the last method; expected travel time $\lambda$. Huang & Gao [6] use expected disutility as an indicator, this is equal to the expected travel time if disutility is the travel time itself. The algorithm in this report uses first order stochastic dominance, like Nie & Wu [5]. Nie & Wu use the procedure LR-CHECK, Huang & Gao use an adapted version to determine dominated routes. This report uses a self-made procedure compare_functions. LR-CHECK first updates the Pareto frontier and then checks for each route if it exceeds this frontier (if so, the route is non-dominated). This report compares each unique pair of routes and determines which routes are non-dominated. If the cumulative probability function of two routes cross, these routes are non-dominated. A detailed explanation of compare_functions is given later.
Correlation
Corman et al [3] calculate correlation between links to determine whether or not these links can be clustered together. Miller-Hooks and Mahmassani [4], Nie and Wu [5] do not consider correlation and/or clustering. Huang and Gao [6] generate link data with a correlation coefficient between 0 and 1. In the tests, any pair of links has identical correlation coefficient. Clustering is not mentioned. This report describes some experiments with correlation and clustering, will be explained later.

Start nodes
The three papers ([4-6]) start their route building from the destination node and store all non-dominated routes that are found, a start node is not mentioned. Most likely, all routes starting from a specific node can be easily retrieved. The algorithm in this report starts the route building at the start node and stores all non-dominated routes that are found. The output requires a destination node, the route(s) from start to destination are given back.
3. Model fundamentals

This section explains the fundamentals of the model needed for time-dependent, stochastic routing. Only the basics are discussed here, implementation in the algorithm is discussed later.

Non-dominance
Every time a link is extended with an adjacent link and there is already a route to the end node of the adjacent link, these routes must be checked for non-dominance. If one of these routes is dominated, it may be removed.

As explained in the previous chapter, there are three methods to check for non-dominance. The method used in the algorithm of this report is first-order stochastic dominance. For a route to be non-dominated by first-order stochastics, the following condition must be fulfilled [1]:

\[ F_1^f(x) \geq F_2^f(x) \quad \forall x \quad \text{and} \quad F_1^f(x) > F_2^f(x) \quad \text{for at least one value of} \ x \]

The procedure `compare_functions` checks which routes fulfill this condition and removes the dominated routes (implementation discussed in Chapter 5).

Convolution
To extend a route with a link, the current route arrival times and probabilities must be convoluted with the travel times and probabilities of the link. Suppose:

\[ u_{ij}^{\text{Start}}(b) = \int_0^b u_{ij}^{\text{Start}}(b - w)p_{ij}(w)dw \] [5]. In which \( u_{ij}^{\text{Start}} \) represents the maximum probability of reaching node j from node Start in less than time budget b. This can be translated to a cumulative probability function if the previous probabilities are summed. w is the travel time of link i-j, \( p_{ij} \) is the probability density function that represents the travel times on link i-j, \( \int_0^b p_{ij}(w)dw = 1 \). Example:
The model uses the cumulative probability function that is easily deduced from previous example:

\[
\sum u_k^{\text{start-j}} = 1
\]

Table 1 - Convolution example.

<table>
<thead>
<tr>
<th>Time</th>
<th>(u_k^{\text{start-i}})</th>
<th>(p_{ij,t=1})</th>
<th>(p_{ij,t=2})</th>
<th>(p_{ij,t=3})</th>
<th>(p_{ij,t=4})</th>
<th>(p_{ij,t=5\text{...10}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0.5</td>
<td>0</td>
<td>0.8</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>0.6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The (cumulative) probability function is somewhat differently calculated in the algorithm. The algorithm does not work with the time budget \(b\), and travel times with a probability equal to zero are not stored. The travel times of route and link are added, and their probabilities multiplied. This leads to the same results. An example is given in Chapter 5.
FIFO
First-in-first-out links are required if one want to prevent:
- the occurrence of cycles in the advised route(s), or
- the advice to wait a time block so that one will arrive earlier at the destination.

The algorithm can run with non-FIFO data. In the case of cycles, these are filtered out by the algorithm (explained next). It does not have the function to check if waiting at a (intermediate) node is faster. However, only FIFO data is used. If the data is FIFO is checked with a script, which will be explained in Chapter 5.

Cycles
Cycles are not allowed in the output. The model has a function that discards routes containing a cycle (explained later). Also note that cycles are automatically filtered out when routes are checked for non-dominance since cycles are always dominated by its non-cycle route.

Time blocks
The algorithm calculates with so-called time blocks, thus in a discrete manner. Time blocks can differ in size (2, 4, 6, 10 or 12) that correspond to the number of minutes in a block (30, 15, 10, 6 or 5 respectively). Note that networks are generated per blocksize, thus only one blocksize is present per run. Output and corresponding graphs are also given with the unit [block]. Needless to say, a larger blocksize results in more accurate results.
Networks contain link data for an entire day, data is available for each time step. The amount of time steps in a day is thus: 24*blocksize.
4. Data generation
In this report, one type of network is used: the grid network. It can vary in size, but lengths are taken equal to widths. The 3x3, 4x4, 5x5, 6x6 and 10x10, 15x15, 20x20 and 25x25 grid are used to conduct the experiments in this report. The grids can be one-directional or bi-directional.

4.1 Link data generation
Data is generated randomly per link, given some boundaries. Generation is done in two steps.

Step 1: Determine general course throughout the day
To determine the general course throughout the day, five variables must be randomly chosen. These are: the peak means, minimal link travel time, intensity, peak intensity and deviation. Each day is subjected to three peaks: one with a mean between 7-9 o’clock, one with a mean between 9-13 o’clock and the last one with a mean between 15-19 o’clock. These three means are randomly chosen for each link. Next, the minimal link travel time is chosen randomly between 2 and 3 blocks. To determine the height of the peaks, intensity is randomly chosen. It is decided to insert a so-called peak time effect; smaller/faster links have higher intensity and vice versa. Intensity and minimal link travel time are added to form the final link minimal travel time. To give the peaks extra intensity, a value between 15 and 30 is chosen. Next, deviation is chosen; a higher deviation causes wider peaks and thus also less difference between the minimal link travel time and peak-time link travel time. Now that all the variables are randomly chosen, the general course throughout the day is generated using a normal probability density function (Matlab function is called normpdf).
Step 2: Generate travel time and probabilities per time step

The next step is to generate the travel time and probabilities for each time step following the general course throughout the day. This is also done with a normal probability density function. Per time step, the corresponding value of the general course throughout the day is inserted as the mean. Another deviation is decided randomly and also inserted in normpdf. The output are the travel times for the current link and its probabilities. It should be noted that probabilities < 0.005/blocksize are discarded (the generated probabilities are divided by themselves to obtain \( \sum p = 1 \)). An example:

![Figure 6 - Link travel time distributions during the day.](image)

Both a one directional and bi-directional grid can be generated. The data generation script generates a one directional grid and this data is recycled to create a bi-directional grid if wanted. With recycling is meant; data of link a -> b is copied to link b -> a.

4.2 Correlated grids

Some experiments have the aim to show the effect of link correlation in a network. There are five types of correlation considered:

- Strong positive correlation: ++  Range  0.8 and 1
- Positive correlation: +  Range  0.2 and 0.5
- No correlation: 0  Range  0 ± 0.000001
- Negative correlation: -  Range  -0.5 and -0.2
- Strong negative correlation: --  Range  -1 and -0.75

The previous data generation scripts are extended to generate grids that are as much correlated as wanted. The procedure is shown in Figure 7.
The correlation is determined for every adjacent link pair. If this correlation is not in the wanted range, the second link is regenerated and correlation is determined again. This regeneration is done max 20 times for each adjacent link pair. If no link could be generated that is within the correlation range, a temporary stored link (closest to the wanted correlation) is chosen as new link.

Some changes in course throughout the day must be made to let the correlation fall into the correlation range. The ++ correlation range does not require any changes. The + correlated grid has two instead of three peaks. The “no correlation” grid has the option to generate link data that is constant throughout the day, or with a small peak of drop. The -- correlated grid starts with a higher minimal travel time and has three inverted peaks. The -- correlated grid starts also with a higher minimal travel time and has three inverted peaks throughout the day. For the latter three correlation grids, attention is given to the fact that data must remain comparable to the positively correlated grids.

### 4.3 File storage and importation

Data is stored as “Links\textsuperscript{x}_\textsuperscript{y}_\textsuperscript{z}_t++.mat”. \textsuperscript{x} represents the network number, a 3x3 grid has network number 3, a 10x10 grid has network number 10 etc. \textsuperscript{y} represents the blocksize, either 2,4,6,10 or 12. \textsuperscript{z} represents the one- or bi-directionality of the grid, 1 or 2 respectively. \textsuperscript{c} represents the type of correlation, either ++, +, 0, - or --.

.mat is a Matlab file format. Data can be stored as other types, for example a .csv file, but this takes longer to import. If one wants to import other file types, the algorithm in RUN_network should be adapted. Currently:

```matlab
text = sprintf('Links%d_%d_%d_t++.mat',nr_network,blocksize,one_direction);
Links = load(text);
```

Change to:

```matlab
text = sprintf('Links%d_%d_%d_t++.csv',nr_network,blocksize,one_direction);
Links = csvread(text);
```
5. Algorithm

This section will discuss the implemented routing algorithm. The most important scripts and functions will be explained with a flow chart.

5.1 General information

Terms & information
Start The start node of the route
Finish The finish node of the route
Blocksize 60/blocksize is the number of minutes in a block.
t_start or start block The start block of the route.
Probability Mostly used as a meaning for the distribution of travel/arrival times and its probabilities.
Best route The route or routes that is/are non-dominated, thus the overall best and fastest route(s).
Subroute If there are multiple routes to the same (intermediate or finish) node.
Flag Variable that is either 1 or 0, used in an if-else-statement.
Cell Cells are used in Matlab to store each individual route.
Extending Adding a link to an existing route, i.e. route = 1-2, add link 2-3, new route: 1-2-3.
Steady state If a new iteration does not yield new results.
Nodes Table in which information of all calculated routes to every node is stored.
Links Table containing link data.
Tolerance, $\varepsilon$ Tolerance used when checking routes for dominance.

Notes:
- Functions are represented in italic, e.g. RUN_MAIN.
- The algorithm runs with time blocks, output is also given in blocks.
- Networks are numbered as follows: nr_network = 3 represents the 3x3 grid. nr_network = 15 the 15x15 grid etc.

RUN_network

User input
RUN_network is the script in which the user must specify several things regarding the calculation of the route. The script calls other functions that calculate the actual route (RUN_MAIN or Run_all_start_times). The user can specify:

```matlab
%% User input
nr_network = 3;  % Choose a network
Start = 1;        % Give the start node
Finish = 9;       % Give the finish node
blocksize = 4;    % Give blocksize. 1 = 1h, 2 = 30 min, 4 = 15 min.
                  % [1,2,4,6,10,12]
t_start = 1;      % Give the start block [min: 1, max: 24*blocksize]
one_direction = 1 % 1: Use the one directional grid. 2: Use the bidirectional grid.
correlation = 11; % ++ = 11, + = 1, 0 = 0, - = -1, -- = -11

% Run options
all_start_times = 0; % If 1: All possible start times are shown in the graph, best route is chosen from Start to Finish.
all_routes = 0;      % If 1: All possible routes from Start to Finish are shown in the graph, for a given start time.

% Output [only for visibility]
Reliability_checkpoint = 0.9; % Give wanted reliability
show_routes = 10;           % Amount of routes to plot in figures
```
Run options
There are three ways to run the algorithm:

1. Calculate the best route from Start to Finish node given a starting block. The output is an expected arrival block based on arrival times and its probabilities, the probabilities per arrival block and route. If there are multiple best routes, all are given back as output. A plot is made from the best route(s) showing the cumulative arrival probability function(s). Specify as: all_start_times = 0 & all_routes = 0. This is the option used for the experiments in Chapter 6.

2. Calculate all possible (non-cycle) routes from Start to Finish node given a starting block. The returning answer are the arrival blocks and routes. A 2D bar plot shows the travel time (given in blocks) for each route. Also a plot is made that show the cumulative arrival probability function for each route. Specify as: all_start_times = 0 & all_routes = 1.

3. Calculate the best route for each start block. A 2D bar plot shows the travel time (given in blocks) for each start block. A plot is made that shows the cumulative probability function of each start block (in the case that there are multiple best routes per start block, the one with minimum expected time is chosen to represent the start block). Specify as: all_start_times = 1 & all_routes = 0.

Output
The value of Reliability_checkpoint is plotted as a dotted horizontal line in the output graphs. The variable given at show_routes is the maximum number of graphs that is plotted.

RUN_MAIN
Links list
Links contains all information about the links. It is a cell with three columns. First column contains the begin node of a link, seconds column contains the end node of a link. The third column is divided into a number of rows, the amount of departure times. For example: for 5 minute-blocks during a day (24h): 24*12=288 blocks and thus rows. Each cell contains the travel times on that link for that start time and the probability of each travel time occurring.

Node list
At the beginning of a run a “Nodes” list is initialized. It consists of 7 columns and as much rows as there are nodes. In the columns the following data is written:

Column 1: expected arrival times. In row 1 the expected arrival time(s) to node 1 starting from the Start node is written. In row 2 the arrival time(s) from Start node to node 2 is given etc.
Column 2: expected arrival times of current iteration. Same as column 1, but now for the expected arrival times of the current iteration (see flow chart of RUN_MAIN).

Column 3: probability. The arrival times and probabilities from Start node to a link end node is saved. Also here, row number represents the end node of a link. The entire route arrival time and probability is saved, for example at row 3 via node 1 and 2, the probability of the route from 1 to 3 is saved.

Column 4: probability of current iteration. The probabilities found in the current iteration are saved here.

Column 5: route. The routes are saved per end node (thus per row).

Column 6: routes found in current iteration. The routes found in the current iteration are saved here.

Column 7: flag for changed nodes. If route(s) to a node is/are changed since last iteration, a 1 is put in this column for this node. If route(s) are equal compared to last iteration, a 0 is put in this column for this node. In the next iteration, if the begin node of the examined link has a 0 in column 7, this link is skipped because no new changes can happen. If the flag is 1, the link is examined.

Table 3 - Nodes list.

<table>
<thead>
<tr>
<th>To node 1</th>
<th>Expected arrival times</th>
<th>Expected arrival times of current iteration</th>
<th>Arrival/depature times and probabilities of current iteration</th>
<th>Routes</th>
<th>Routes found in current iteration</th>
<th>Flag for change of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>To node 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>To node 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>To node x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Initialization of Nodes

An empty cell with size nr_nodes x 7 is made. The first column (expected arrival time) at the start node row is initialized with 0. The third column (probabilities) at the start node row is initialized with \([t_{\text{start}} 1]\). The fifth column (route) is initialized as an empty 1x1 cell. The seventh column (indicator for node change in iteration) is initialized as 1.

Suppose there are 4 nodes, the start node is 2 and \(t_{\text{start}}\) (route departure block) is 30. Nodes is initialized as:

Table 4 - Initialization of Nodes.

<table>
<thead>
<tr>
<th>To node 1</th>
<th>Expected arrival time</th>
<th>Temp. exp. Arrival time</th>
<th>Times and probabilities</th>
<th>Temp. times and probabilities</th>
<th>Routes</th>
<th>Temp. Routes</th>
<th>Flag for change</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>To node 2</td>
<td>[0]</td>
<td></td>
<td>[30 1]</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>To node 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>To node 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Subroutes in Nodes
If there are multiple routes to an (intermediate) node and all routes are non-dominated, all options are saved and used for further calculations. This is done using cells. Each cell contains a subroute. Suppose a 2x2 grid, there are two routes to node 4: 1-2-4 and 1-3-4. Nodes looks like:

Table 5 - Example of Nodes showing storage of subroutes.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1x1 cell</td>
<td>[]</td>
<td>1x1 cell</td>
<td>[]</td>
<td>1x1 cell</td>
<td>1x1 cell</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1x1 cell</td>
<td>1x1 cell</td>
<td>1x1 cell</td>
<td>1x1 cell</td>
<td>1x1 cell</td>
<td>1x1 cell</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1x1 cell</td>
<td>1x1 cell</td>
<td>1x1 cell</td>
<td>1x1 cell</td>
<td>1x1 cell</td>
<td>1x1 cell</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1x2 cell</td>
<td>1x2 cell</td>
<td>1x2 cell</td>
<td>1x2 cell</td>
<td>1x2 cell</td>
<td>1x2 cell</td>
<td>0</td>
</tr>
</tbody>
</table>

If there are multiple cells, thus routes, for a node, a check for non-dominance is done. If a route is dominated, this route is removed. This means that the corresponding cell is removed in every column (expected arrival time, temp. expected arrival time etc.).

Determination of steady state
At the end of an iteration (all links are examined) a for loop checks for each node if the data in the temporary columns (2,4,6) differ from the data in the other columns (1,3,5). For each node, the flag that indicates differences (column 7) is updated (1 if row has changes, 0 if row has no changes). If one or more nodes are changed, another iteration will be performed.
After each completed iteration, all data from the temporary columns are copied to the other columns and the temporary columns are cleared. Thus: column 2 moves to column 1, 4 to 3 and 6 to 5. Iterations continue until a steady state (no changes) is reached.
5.2 Scripts and functions

This section will show the overall flow chart of the algorithm (Run_network and RUN_MAIN) and then explains shortly all functions and important if-statements within RUN_MAIN.

RUN_network and RUN_MAIN

RUN_network is the script where the algorithm starts running and the user specifies route properties. The actual route calculation is done in RUN_MAIN and is called within RUN_network.

Figure 8 - Flow chart of RUN_network and RUN_MAIN, part 1/2.
add_probability and add_link_to_node

add_probability searches travel times and probabilities of the to be extended link. add_link_to_node convolution of the link data and the current route data. If a departure time of the to be extended route exceeds the amount of time blocks in a day, modulo is used. Suppose there are 48 time blocks in a day, time block 49 corresponds with data of time block 1, 50 with time block 2 etc. Arrival time keeps on counting in blocks (49, 50 etc.). Flow chart:
Figure 10 - Flow chart of add_probability.

add_link_to_node takes the first departure time from node i (begin node of the to be added link) and adds all link travel times from link i-j corresponding to the correct departure time column. It then continues to the second departure time and so on. Example:

Table 6 - Example of convolution as implemented in the algorithm.

<table>
<thead>
<tr>
<th>t=0</th>
<th>t=2</th>
<th>t=3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Departure time of node i</td>
<td>Probability</td>
<td>Travel times on link i-j</td>
</tr>
<tr>
<td>2</td>
<td>0.5</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0.5</td>
<td>7</td>
</tr>
</tbody>
</table>

Result:

<table>
<thead>
<tr>
<th>Travel time to node j</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>2+3=5</td>
<td>0.5*0.8=0.4</td>
</tr>
<tr>
<td>2+7=9</td>
<td>0.5*0.2=0.1</td>
</tr>
<tr>
<td>3+6=9</td>
<td>0.5*0.4=0.2</td>
</tr>
<tr>
<td>3+7=10</td>
<td>0.5*0.6=0.3</td>
</tr>
</tbody>
</table>

Basically; travel times of a link are added to possible departure times, probabilities are multiplied. The results are equal to those using the convolution integral, but less redundant calculations are needed with this function. Multiple occurrences of arrival times, in the example arrival time = 9, are dealt with in the function clean_up_probability (see next).
**clean_up_probability**

In this function, the travel times and probabilities are merged. This is possible if there is a travel time with different probabilities. These probabilities are merged. For example:

<table>
<thead>
<tr>
<th>Arrival time</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.4</td>
</tr>
<tr>
<td>9</td>
<td>0.1</td>
</tr>
<tr>
<td>9</td>
<td>0.2</td>
</tr>
<tr>
<td>10</td>
<td>0.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arrival time</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.3 (=0.1+0.2)</td>
</tr>
<tr>
<td>9</td>
<td>0.3</td>
</tr>
</tbody>
</table>

**Table 7 - Example of clean_up_probability.**

**Termination if-statement**

“If min_max_t == 0 OR min_start_t < min_max_t” determines if the current link extension can be terminated. min_max_t is the minimal arrival time with probability 1 to the finish node. min_start_t is the least arrival time to the current link end node.

Suppose the following (unrealistic) network:

Both routes start at node 1 and finish at node 3. The short route has more distribution in travel time, but is much shorter than the long route. The long route, although deterministic links, cannot compete with the short route. The worst case scenario for the short route is an arrival time of 20, the long route has an arrival time of 100. There is no need to finish the calculation of the long route if the short route has reached the worst case scenario (arrival reliability of 1) and the long route exceeds this arrival time. The termination in **RUN_MAIN** goes as follows:

A flag variable called min_max_t is initialized at 0. At the end of an iteration, the arrival time with a cumulative probability of 1 is saved as min_max_t. If the finish node is not reached in an iteration, min_max_t remains 0. If there are multiple routes to the finish node, the minimal arrival time is saved.

During iteration, the new arrival time and probabilities are calculated for each link. After this, the minimal arrival time of the current route is calculated and saved as min_start_t (in case of multiple subroutes, the minimal arrival time is chosen). Next, an if-statement checks if either min_max_t = 0 or min_start_t < min_max_t. If true, the current loop continues. If false, terminate the current loop because this route cannot compete with the guaranteed arrival time to the finish node.
**expected_arrival_time**
Calculates the expected arrival time of the entire route from Start node to the current link end node. Uses the arrival time and probabilities calculated in `add_probability`. Note that the expected time is just an extra given as output, it is not used to decide for non-dominance etc.

![Figure 12 - Flow chart of expected_arrival_time.](image)

**add_route**
Updates the route in Nodes. The begin node of the current link is added to existing routes. If there is no route, a new one is created.

![Figure 13 - Flow chart of add_route.](image)

**loop_checker**
A function is made to filter loops (also called cycles) out. It should be noted that loops are always dominated by non-loop routes thus are automatically filtered out. But when it is chosen to show all routes in the output, loops may occur as alternatives. To prevent the occurrence of loops in the output for all cases, `loop_checker` is made.

Just after the `Add_probability` is calculated, `loop_checker` is called. Each route to the current link end node is tested on the presence of this end node in the route so far. If this node is already present, a loop is detected and this new route is discarded.
In case of a loop, the empty cell(s) is/are removed in a separate function right after \textit{loop\_checker} has finished.

\textbf{compare\_functions}

This function compares the probability functions from Start node to the current link end node. This function is called each time a route is added to Nodes and there already exists another route to the same end node. Flow chart:

\textbf{Step 1: Create time vector and cumulative probability functions.}

The overall minimum and maximum arrival times for all routes are sought. A time vector is made: [\textit{min\_arrival\_time}:1:\textit{max\_arrival\_time}]. For each function to be compared: this time vector is filled with the corresponding probabilities. The cumulative probability function is then determined.
Step 2: Compare each unique pair.
There are \( \frac{n^2-n}{2} \) unique pairs, in which \( n \) represents the amount of functions to be compared. For every time in time vector, the probability of both functions is compared. A table is made with size (number of times in time vector x number of unique pairs*2). Function 1 is compared with function 2 and vice versa. For each time element in time vector; flag the one with the highest probability with a 1. If the probabilities are the same, both functions in this table get a 1. A tolerance of \( \varepsilon=0.00001 \) is used.

Step 3: Determination of (non-)dominated routes.
All unique pairs are examined on the count of 1’s per column. If this count equals the number of elements in time vector, it can be concluded that this function dominates the other one. The other function may be removed. To illustrate:

![Figure 16 – Compare_functions example and result.](image1)

Function 2 is best for all arrival times, except the first one (\( t=1 \), shown with Inf). In this case, neither path is dominated, both are saved.

![Figure 17 - Compare_functions example and result.](image2)

Here, function 2 is best for all arrival times, thus function 1 may be removed.

Step 4: Remove the dominated route(s).
The routes to be removed are saved in “remove” and the function `remove_function` removes the corresponding cells in the three main variables: Add_time (the expected arrival time), Add_probability (arrival times and probabilities) and Add_route (the routes to current link end node).

`is_equal`
This function checks if the routes that are added are not already existing.
5.3 Example run of the algorithm

To provide more insight in the algorithm, the next step-by-step example is made. A print screen of Nodes is made after each iteration. Note that Matlab shows data in cells as 1x# cell. To clarify the example, the content of these cells is shown for the 1st and 5th column. The probabilities (3rd column) are only shown as #x2 double, due to the large amount of information.

Suppose a one directional 3x3 grid with strong positive correlation:

![Diagram of 3x3 grid](image)

**Figure 18 - Example: 3x3 one directional grid.**

Start node is 1 and Finish node is 9, blocksize is 6. Nodes is initialized as:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>2</td>
<td>[1]</td>
<td></td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>3</td>
<td>[1]</td>
<td></td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>4</td>
<td>[1]</td>
<td></td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>5</td>
<td>[1]</td>
<td></td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>6</td>
<td>[1]</td>
<td></td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>7</td>
<td>[1]</td>
<td></td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>8</td>
<td>[1]</td>
<td></td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>9</td>
<td>[1]</td>
<td></td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
</tbody>
</table>

Table 8 - Example: initialization of Nodes.

After first iteration:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.8080</td>
<td>1x1 cell</td>
<td>7x2 double</td>
<td>1x1 cell</td>
<td>1</td>
<td>1x1 cell</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>[1]</td>
<td></td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>3</td>
<td>[1]</td>
<td></td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>4</td>
<td>[1]</td>
<td></td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>5</td>
<td>[1]</td>
<td></td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>6</td>
<td>[1]</td>
<td></td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>7</td>
<td>[1]</td>
<td></td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>8</td>
<td>[1]</td>
<td></td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>9</td>
<td>[1]</td>
<td></td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
</tbody>
</table>

Table 9 - Example: Nodes after first iteration.

In the second iteration, two routes are found to intermediate node 5: 1-4-5 and 1-2-5. These are checked for non-dominance.
Both routes are non-dominated and thus saved in Nodes.

Nodes after second iteration:

### Table 10 - Example: Nodes after second iteration.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[]</td>
<td>[1]</td>
<td>[1]</td>
<td>[]</td>
<td>[]</td>
<td>[]</td>
<td>[]</td>
</tr>
<tr>
<td>2</td>
<td>5.8080</td>
<td>1x1 cell</td>
<td>7x2 double</td>
<td>1x1 cell</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>3</td>
<td>11.0017</td>
<td>1x1 cell</td>
<td>11x2 double</td>
<td>1x1 cell</td>
<td>[12]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>4</td>
<td>6.4904</td>
<td>1x1 cell</td>
<td>10x2 double</td>
<td>1x1 cell</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>5</td>
<td>12.2514</td>
<td>1x2 cell</td>
<td>17x2 double</td>
<td>10x2 double</td>
<td>[14]</td>
<td>[12]</td>
<td>[1]</td>
</tr>
<tr>
<td>6</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>7</td>
<td>12.5138</td>
<td>1x1 cell</td>
<td>17x2 double</td>
<td>1x1 cell</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>8</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>9</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
</tbody>
</table>

In the third iteration, there are multiple routes found to node 6 and node 8.

Three routes are found to node 6: 1-4-5, 1-2-5 and 1-2-3. The first route is dominated by the other two. Also note that due to tolerance in the first part of the graph, this result in yielded. Route 1 is discarded.

Also three routes are found to node 8: 1-4-7, 1-2-5 and 1-2-3. The second route is dominated and removed. See also Figure 20.

After third iteration:
In the fourth iteration, routes to node 9 must be checked for non-dominance. 

There are four routes to node 9: 1-4-7-8, 1-2-5-8, 1-2-5-6 and 1-2-3-6. Route 3 is dominated by the other three routes, see Figure 21. Although it seems competitive with route 1 after t = 12, due to tolerance it is decided that route 3 is dominated.

After fourth iteration:

Table 12 - Example: Nodes after fourth iteration.
Outputs after completion of the run:

![Cumulative probabilities vs arrival times from node 1 to node 9 starting at t = 1](image)

**Figure 22** - Example: cumulative probability distribution of output.

**Arrival time** =

- [23.4433] [22.4684] [22.5763]

**Route** =

- [1 4 7 8 9] [1 2 5 8 9] [1 2 3 6 9]

**Probability** =

- [22x2 double] [20x2 double] [19x2 double]

To illustrate the probabilities per route:

![Probability output](image)

**Figure 23** - Example: Probability output.
6. Experiments and results

The results of some experiments are presented here. Some effects are taken into account, for example the presence of a city, correlation and clustering between adjacent links and the combination of city and a combination of both.

6.1 Standard outputs

Some outputs will be presented for different grids in which a strong positive correlation is assumed. The Start node is the second node on the second row and the Finish node is the one but last node on the one but last row of the grid. The left figures are the outputs as given by Matlab, the right figures are zoomed in versions of the left figures.

5x5 bidirectional grid

Blocksize is 6, i.e. each block represents 10 minutes, and the route goes from node 7 to node 19:

Figure 24 – Advised routes from node 7 to node 19 at 6:40h, right: zoomed in.

The properties used to generate previous grid (intensity, deviation, minimum link travel time etc.) is reused to generate a grid with blocksize 12:

Figure 25 - Advised routes from node 7 to node 19 at 6:40h, right: zoomed in.
Two of the three non-dominated routes are advised for both block sizes. Using block size 6, the green and blue route have a large overlap, while a clear difference is found when using block size 12. The distribution of output probabilities of the similar routes (also block size 10 is presented):

![Figure 26 - Bar plot of travel time probabilities, blocksize 6.](image)

![Figure 27 - Bar plot of travel time probabilities, blocksize 10.](image)

![Figure 28 - Bar plot of travel time probabilities, blocksize 12.](image)

The left figure corresponds to route 7-12-13-14-19, the right figure represents route 7-12-13-18-19. In both cases, the travel time is shifted towards left when data is finer (block size is 10 or 12).

Variance for the bar plots:

<table>
<thead>
<tr>
<th>Route</th>
<th>Blocksize 6</th>
<th>Blocksize 10</th>
<th>Blocksize 12</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-12-13-14-19</td>
<td>42,17</td>
<td>35,0</td>
<td>38,50</td>
</tr>
<tr>
<td>7-12-13-18-19</td>
<td>50,0</td>
<td>-</td>
<td>46,0</td>
</tr>
</tbody>
</table>

No clear increase or decrease is noticed in the variance.
10x10 bidirectional grid
Blocksize is 6, i.e. each block represents 10 minutes, and the route goes from node 12 to node 89:

Figure 29- Advised routes from node 12 to node 89 at 3:20h, right: zoomed in.

Same grid properties, now with blocksize 12:

Figure 30- Advised routes from node 12 to node 89 at 3:20h, right: zoomed in.

One advised route comes up in both blocksizes (as well as blocksize 10), route 12-22-32-42-52-62-63-64-65-66-67-68-78-88-89. The probability bar plot of this route:
It can be seen that for a larger blocksize, thus more accurate travel times, the travel times become slightly larger. Also the spreading of the probabilities is smaller, variance is decreasing as blocksize increases: variance for blocksize 6 is 58.50, for blocksize 10 54.17 and for blocksize 12 46.0. This is the opposite effect of the 5x5 grid where travel times were shifted to the left when using a larger blocksize. No conclusion can be drawn from this effect. This effect is also researched in a 6x6 and a 15x15 grid but this did not lead to clearer conclusions, see also Appendix A.

It can be said that although the grids are bidirectional, this property is not used yet. All routes go in the most direct manner to the finish node, thus right and down. This makes sense, because a detour consists of at least 3 extra links. Given the fact that the links are generated to be competitive to each other and has peaks during the day, the chance of a detour is very low. The use of bidirectional links will be illustrated with a city, see Chapter 6.2.
6.2 City effect

The presence of a city influences the travel time distribution on links close to it. These links are crowded and thus travel time increases. Another effect is that these links have strong positive correlation. If an accident happens near the city or weather decreases the maximum travel speed, the consequences spill back to the other links. Two examples are made to illustrate this.

Link travel times near a city

A 5x5 bidirectional grid (++ correlation) is assumed and the most central node, node 13, is chosen as the city. Travel times on links towards and from node 13 are multiplied with a factor 4, i.e. very crowded links. Links adjacent to these links are multiplied with a factor 3 or 2, see Figure 32.

Figure 32 - City grid example.

Suppose one wants to travel from node 6 or node 7 to node 25 at 0:00h:

The advised routes avoid the links with multipliers. From node 7, the route is faster when a small detour is taken to avoid a crowded link.

The same route, now in peak time (7:30h):
The advised route from node 6 to node 25 remains the same. The advised route starting at node 7 differs in peak time. It follows the upper and right-side outer links of the grid instead of the left-side and bottom outer grids. The other option is a more direct route that travels on two somewhat crowded links (factor 2).

Travelling from the city, node 13, to node 25:

There are two equal advised routes in both non-peak time and peak time. One route goes directly down on a very crowded link, the other route advises a small detour. At both starting times, the route that goes directly down is best for arrival time reliabilities up to 50-60% (non-peak time and peak time respectively). The detour route is best when one would like a high arrival time reliability.

Travelling near the city center, from node 19 to node 7:
When one has to travel across the crowded links, it can be noticed that a detour along the non-multiplied links is not advised in both peak and non-peak times. This makes sense, because a detour takes at least 3 extra links. At both starting times, route 19-14-9-8-7 is the best option if one wants the fastest route with high arrival time reliability (>95%).

**Correlation near a city**

To show the effect of correlation near a city, two types of correlation are tested: strong positive correlation and strong negative correlation. Correlation is applied to the links close to the city. Both are compared to a non-correlated grid.

A 6x6 bidirectional grid is used, the city is between nodes 15, 16, 21 and 22.

The correlation effect is applied to the links around and near the city, represented with the red arrows in Figure 37. The other links have three travel times with probabilities, explained per correlation case, in order to be competitive with the correlated links.
No correlation
In this case, the entire grid has travel time distribution \([5 \ 0.25-0.45; 6 \ 0.25-0.45; 7 \ 0.1-0.5]\) for every link and every start time block.

Strong positive correlation
The correlated links have a correlation of \(0.8 \leq \rho \leq 1\) between each adjacent pair. The non-correlated links have travel time distribution \([5 \ 0.25-0.45; 6 \ 0.25-0.45; 7 \ 0.1-0.5]\) for every start block time and every link.

There are five non-dominated routes to node 36. Three of them (the red, blue and purple route) avoid the correlated links entirely. When we look at the zoomed in version, it can be seen that these three routes are the preferred ones if one wants a high arrival time reliability. For low arrival time reliabilities (<50%), the routes travelling on the correlated links are advised. This same effect is seen when the route goes to node 29 (not shown).
**Strong negative correlation**
The correlation between adjacent pairs should be $-1 \leq \rho \leq -0.75$. This is however impossible to obtain in a bidirectional grid, therefore the compromise is that in each adjacent link pair, one has upward peaks and the other downwards peaks. Travel time distributions on the non-correlated links are [4 0.3-0.6; 5 0.2-0.35; 6 0.05-0.5] for each time block.

![Figure 40 - Advised route from node 8 to node 36 at 10:00h with -- correlated links, right: zoomed in.](image)

The correlated links are somewhat avoided in this example. Especially in the upper arrival time reliability region (>50%), the route that avoids the correlated links dominates the others. This might be a coincidence, due to unfortunate link data generation. Another explanation can be the fact that a link with upwards peaks even out its adjacent link with downward peaks, resulting in a somewhat deterministic travel time that is higher than the non-correlated links. Thus the advantage of the link with downward peaks is vanished.
6.3 Recurring route parts

The same 10x10 grid as in Chapter 6.1 is used. The advised routes for four different starting blocks:

![Figure 41 - Advised route(s) for route from node 12 to node 89 for different starting blocks. Upper left: 3:30h. Upper right: 6:50h. Bottom left: 14:00h. Bottom right: 22:00h.](image)

As can be seen, there is not a route that is best for every start time, which makes sense. The advised routes at \( t = 41 \) and \( t = 132 \) are similar, one of these routes is also the advised route for \( t = 84 \). At start time \( t = 21 \), different routes are proposed, only one of the three is also seen at other start times (41 and 132).

It can also be noticed that there is a large part of each advised route is similar to the other advised routes. Every route goes via node 12-22-32-42-52-62-63-64-65-66-67. A quick investigation to this non-dominated route is given. The 10x10 grid:
The black route in Figure 42 are the advised routes. The red route is the part of the advised routes for which alternatives are investigated. There are three alternatives: the blue route (node 42-43-44-54-64), the orange route (node 42-43-53-54-64) and the green route (node 52-53-54-55-65).

The advised route (node 42-52-62-63-64) is compared with the blue and orange alternative (Figure 43, Figure 44 and Figure 45 respectively). A few things can be noticed: The mean link travel time during the day is comparable (link 132 and link 38, left figure) or lower (link 142, 56 or 57 compared to corresponding links, right three figures).

The peaks in the links of the advised route are relatively low, link travel time increases with 0.5 – 1.8 blocks at most, compared to an increase of up to 4 blocks in links of the advised routes.

The ranges of travel times showed by the variance intervals are quite wide for the advised route, especially the 90% interval. These ranges are more narrow for both alternative routes. A narrow range of travel times can be beneficial if mean travel time is low, but can be a disadvantage for longer mean travel times.

![Figure 42 - 10x10 grid with advised route and alternatives.](image)

![Figure 43 - Variance intervals for advised route 42-52-62-63-64.](image)
The red route (node 52-62-63-64-65) is compared with the green route, Figure 46 and Figure 47 respectively. Also here it can be seen that the mean during the day of the advised route is lower than for the alternative route. The mean of the advised route is between 4 and +6 blocks, whereas the mean in the alternative route is between 4.5 and +9 blocks. Alternative link 145 has clear peaks, where link travel time is almost doubled compared to non-peak times. Peaks are also clearly visible in the other three alternative links. The variance intervals of the advised route are narrower than the intervals for the advised route, thus the longer travel times cannot be avoided.

**Figure 44 - Variance intervals for alternative route 42-43-53-54-64.**

**Figure 45 - Variance intervals for alternative route 42-43-44-54-64.**

**Figure 46 - Variance intervals for advised route 52-62-63-64-65.**
Figure 47 - Variance intervals for alternative route 52-53-54-55-65.

All alternatives have a mean that is higher than the advised route. Peaks are more clearly present in the alternative routes, leading to even longer travel times for a significant part of the day. The variance intervals for the alternative routes are narrower, also contributing to longer travel times.

These examples were to gain more insight into the observation that a large part of the route is fixed, and is advised for routes with different start times. The observations show the disadvantageous effect of peak intensity and narrow variance intervals. However, no conclusions may be drawn on these examples and observations as they are far too incomplete. More research must be done for clear statements.
6.4 Correlation effect & Clustering

Correlation between adjacent links can have an influence on the output. The influence of correlation is tested in two experiments. In the first experiment, grids with one of the five correlation scenarios are compared to each other, assuming equal properties. The second experiment clusters adjacent links that exceed a certain correlation threshold.

6.4.1 Correlation effect

As mentioned in Chapter 4, grids are generated with five different correlation scenarios: ++, +, -, --, and 0 (no correlation). The no correlation grid is assumed as base case.

A grid with size 5x5 is assumed, blocksize is 10 and t_start is 6:06h. Route goes from node 7 to node 19. Correlation scenario 0 is assumed, data for each property (intensities, deviations, minimal link travel time etc.) are stored. Now, a grid is generated with these exact properties but different correlation scenario (++, +, -, --). Some boundaries are changed to ensure that the links remain comparable (thus a ++ grid is comparable with a -- grid for example). With comparable is meant; approximately the same link travel times for each time block. Outputs are shown in the left figure of Figure 48. Second, a grid with size 10x10 is assumed, blocksize is 10 and t_start is 11:42h. Route goes from node 12 to node 89. Again, a 10x10 grid with correlation scenario 0 is assumed and data is regenerated to generate a grid for the other four correlation scenarios. See Figure 48, right figure.

Figure 48 - Correlation effects. Left: 5x5 grid. Right: 10x10 grid.

Note: The line colors in Figure 48 are not of interest, only look at the average cumulative arrival time per graph and its shift over the different correlation scenarios.
Looking at the right figure, the 10x10 grid, a clear shifting of arrival time distribution is seen. Compared to the graph of no correlation, a negative correlation shifts the graph to the left, whereas a positive correlation shifts the graph to the right. This shifting is more clear in the right figure than in the left figure, where no correlation and positive correlation are quite similar. Probably because the route in the 5x5 grid is quite small and thus the effect can be easily influenced.

This shifting effect is mentioned in the paper by Charle et al. [7]. They showed that when correlations are ignored, the expected travel time and reliability are overestimated (thus a shift to the left of the cumulative arrival time distribution when a positively correlated grid is used and a shift to the right when a negatively correlated grid is used). Corman et al. [3] also mention and show this effect. Their experiment is different though; links are clustered together if a certain correlation threshold is crossed. This experiment is discussed next.

6.4.2 Clustering
Corman et al. [3] and Charle et al. [7] cluster their data. A correlation threshold \( \rho \) is used and all adjacent links that exceed this threshold are clustered, this process is repeated for the clustered links until a minimal correlation between adjacent (cluster)links is reached. Compared to the case in which no links are clustered, arrival times are shifted to right in case of \( \rho = 0.25 \).

Data is generated with a correlation of \( 0 \leq \rho \leq 1 \) between adjacent links. It is attempted to make \( 1/3 \) of the adjacent links between \( 0 \leq \rho \leq 0.25 \), \( 1/3 \) between \( 0.25 < \rho \leq 0.50 \) and \( 1/3 \) between \( 0.50 < \rho \leq 1 \). The procedure: Clustering is done per examination of Links, all links in Links are examined on adjacent link pairs and their correlation. If clustering is allowed, the new link and its data (sum of the two added links) is added to Links. When all links are examined, all links in Links are examined again and new clusterlinks (note: a new clusterlink can also exist of two clusterlinks) are added, etc. The procedure continues until no new clusterlinks are formed or the computational time exceeds > 1.5h. Clustering is shown in a one directional 4x4 and 10x10 grid.

4x4 grid
Route is taken from the upper left corner to the bottom right corner.

Corresponding results, as well as results for \( \rho = 0.5 \) and \( \rho = 0.12 \), are given in the following table:
Table 14 - Results of advised routes.

<table>
<thead>
<tr>
<th>Route</th>
<th>Expected arrival time [block]</th>
<th>Arrival time 50th percentile [block]</th>
<th>Arrival time 90th percentile [block]</th>
<th>Number of clusterlinks</th>
</tr>
</thead>
<tbody>
<tr>
<td>No clustering</td>
<td>1 5 9 13 14 15 16</td>
<td>165,20</td>
<td>165</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rho = 0.5</td>
<td>1 2 6 7 8 12 16</td>
<td>165,47</td>
<td>166</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1 6 7 8 12 16</td>
<td>164,48</td>
<td>165</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>1 10 11 12 16</td>
<td>164,57</td>
<td>165</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>1 10 14 16</td>
<td>164,65</td>
<td>165</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>1 10 14 15 16</td>
<td>164,82</td>
<td>165</td>
<td>10</td>
</tr>
<tr>
<td>Rho = 0.25</td>
<td>1 8 12 16</td>
<td>162,69</td>
<td>163</td>
<td>129</td>
</tr>
<tr>
<td></td>
<td>1 6 8 12 16</td>
<td>162,69</td>
<td>163</td>
<td>129</td>
</tr>
<tr>
<td>Rho = 0.12</td>
<td>1 8 12 16</td>
<td>162,69</td>
<td>163</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>1 6 8 12 16</td>
<td>162,69</td>
<td>163</td>
<td>256</td>
</tr>
</tbody>
</table>

The number of clusterlinks increases rapidly when correlation threshold is lower. The second route of the no clustering case is seen at ρ = 0.12, ρ = 0.25 and ρ = 0.5 but now partly clustered (1-6 via 2, 1-8 via 2-6 and 6-8 via 7). The first route does not appear using the correlation threshold. At ρ = 0.5 a new route is advised via node 10 using clusters 1-10 via nodes 2-6 and/or 14-16 via node 15. Although the amount of clusterlinks almost doubled using ρ = 0.12 compared to ρ = 0.25, none of these new clusters are used in a non-dominated route. The (expected) arrival times decrease a little bit when clusterlinks are used.

10x10 grid
Route is taken from the upper left corner to the bottom right corner.

![Figure 50 - Advised routes. Left: no clustering. Right: rho = 0.25.](image)

Corresponding results are given in the following table, as well as the results for ρ = 0.5 and ρ = 0.12:
Table 15: Results of advised routes.

<table>
<thead>
<tr>
<th>Route</th>
<th>Expected arrival time [block]</th>
<th>Arrival time 50&lt;sup&gt;th&lt;/sup&gt; percentile [block]</th>
<th>Arrival time 90&lt;sup&gt;th&lt;/sup&gt; percentile [block]</th>
<th>Number of clusterlinks</th>
</tr>
</thead>
<tbody>
<tr>
<td>No clustering</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Rho = 0.5</td>
<td>1 11 21 42 46 66 100</td>
<td>238.50</td>
<td>238</td>
<td>249</td>
</tr>
<tr>
<td></td>
<td>1 11 21 42 46 66 86 87 97 98</td>
<td>239.45</td>
<td>240</td>
<td>249</td>
</tr>
<tr>
<td></td>
<td>1 11 21 42 46 66 89 99 100</td>
<td>240.04</td>
<td>240</td>
<td>250</td>
</tr>
<tr>
<td>Rho = 0.25</td>
<td>1 11 21 42 43 53 73 75 99 100</td>
<td>237.56</td>
<td>238</td>
<td>247</td>
</tr>
<tr>
<td></td>
<td>1 11 21 42 43 53 73 87 99 100</td>
<td>237.62</td>
<td>238</td>
<td>247</td>
</tr>
<tr>
<td></td>
<td>1 11 21 42 46 66 99 100</td>
<td>237.70</td>
<td>238</td>
<td>248</td>
</tr>
<tr>
<td></td>
<td>1 11 21 42 43 53 73 87 97 99 100</td>
<td>237.79</td>
<td>238</td>
<td>247</td>
</tr>
<tr>
<td>Rho = 0.12</td>
<td>1 11 21 42 43 53 73 97 99 100</td>
<td>237.26</td>
<td>237</td>
<td>247</td>
</tr>
</tbody>
</table>

The route advised with non-clustered links is not advised when clusterlinks are used. At $\rho = 0.5$ three different routes are advised. One of these routes is also advised at $\rho = 0.25$, but with a clustered part: 66-99 via node 89. Three other routes are also advised at $\rho = 0.25$. Two of them are alike but one has an extra clusterlink: 87-99 via node 97. Lowering the correlation threshold to $\rho = 0.12$ leads to one advised route, one that came also up at $\rho = 0.25$ but now with clusterlink 73-97 via node 87. Expected arrival time decreases when clusterlinks are used. Decreasing the correlation threshold further to $\rho = 0.25$ and $\rho = 0.12$ leads to more decrease in (expected) arrival time. The number of clusterlinks increases rapidly when correlation threshold is lowered.

For both grid sizes, the amount of clusterlinks increase exponentially when the correlation threshold is lowered. Despite the extra clusterlinks, only a small decrease in (expected) arrival time can be noticed.

Charle et al. [7] and Corman et al. [3] noticed that a lower correlation threshold leads to an increase in travel time. This effect is not seen in these experiments. Further research is needed to show if this is due to the generated data (perhaps links are too much alike, correlation should be lower) or the cluster method etc. Repeating the experiment with data from the papers should help.
6.5 Computational times
To test the performance of the algorithm, the computational times of a run are compared. This is done for different grid sizes, block sizes and correlations. The difference between a one directional and bidirectional grid is also taken into account. This chapter only shows the results for the strongly positively correlated grids, results for other correlations are presented in Appendix B. This appendix also includes some profiler outputs.

Sensitivity of blocksize
Route is taken from node \((m+2)\) to node \((m^2 - m - 1)\), close to the upper left corner to close to the bottom right corner respectively. For grid 3x3 and 4x4, start node is 1. The average of three runs is taken, plots show each measurement with a circle.

![Computational times strong positive correlated one directional grid, right: zoomed in.](image1)

![Computational times strong positive correlated bidirectional grid, right: zoomed in.](image2)

The corresponding average computational times per measurement point:
Table 16 - Computational times long routes.

<table>
<thead>
<tr>
<th>Blocksize</th>
<th>One directional</th>
<th>Bidirectional</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.036</td>
<td>0.044</td>
</tr>
<tr>
<td>4</td>
<td>0.056</td>
<td>0.089</td>
</tr>
<tr>
<td>6</td>
<td>0.063</td>
<td>0.129</td>
</tr>
<tr>
<td>10</td>
<td>0.094</td>
<td>0.227</td>
</tr>
<tr>
<td>12</td>
<td>0.277</td>
<td>0.985</td>
</tr>
<tr>
<td>3x3</td>
<td>0.027</td>
<td>0.053</td>
</tr>
<tr>
<td>4x4</td>
<td>0.076</td>
<td>0.118</td>
</tr>
<tr>
<td>5x5</td>
<td>0.102</td>
<td>0.177</td>
</tr>
<tr>
<td>6x6</td>
<td>0.159</td>
<td>0.292</td>
</tr>
<tr>
<td>10x10</td>
<td>0.474</td>
<td>5.529</td>
</tr>
<tr>
<td>15x15</td>
<td>1.038</td>
<td>2.304</td>
</tr>
<tr>
<td>20x20</td>
<td>4.802</td>
<td>2.374</td>
</tr>
<tr>
<td>25x25</td>
<td>13.465</td>
<td>19.062</td>
</tr>
<tr>
<td>30x30</td>
<td>17.102</td>
<td>26.957</td>
</tr>
<tr>
<td>35x35</td>
<td>21.525</td>
<td>30.143</td>
</tr>
<tr>
<td>40x40</td>
<td>25.908</td>
<td>34.267</td>
</tr>
<tr>
<td>45x45</td>
<td>30.143</td>
<td>39.049</td>
</tr>
<tr>
<td>50x50</td>
<td>34.573</td>
<td>40.245</td>
</tr>
<tr>
<td>55x55</td>
<td>38.939</td>
<td>43.854</td>
</tr>
<tr>
<td>60x60</td>
<td>43.215</td>
<td>47.867</td>
</tr>
<tr>
<td>65x65</td>
<td>47.573</td>
<td>51.880</td>
</tr>
<tr>
<td>70x70</td>
<td>51.930</td>
<td>55.893</td>
</tr>
<tr>
<td>75x75</td>
<td>56.287</td>
<td>59.906</td>
</tr>
</tbody>
</table>

What directly stands out is the rapid increase of computational times for grids ≥ 25x25 with blocksize 2. Grid 50x50 took more than 7.5 hours to calculate the route, it is stopped after this time. Grid 75x75 is not tested with blocksize 2 as well as the one directional 50x50 grid.

A probable explanation could be that because there are little time steps (48 in a day), routes are more alike and thus more non-dominated routes. The 50x50 grid found already 4637 non-dominated routes to node 2285 compared to 21 non-dominated routes with blocksize 4.

Table 17 shows the amount of non-dominated routes that is given as output. For grids with sizes 10x10 and larger, blocksize 2 has (significant) more non-dominated routes. Blocksizes 10 and 12 find only one route. This is not seen for the smaller grid sizes.

The number of non-dominated routes is not related to the computational time. Grid 75x75 has 15 non-dominated routes for blocksize 4 and 12 routes for blocksize 6, but calculation time of blocksize 4 is more than three times the calculation time of blocksize 6. Blocksize 10 finds only one route, but computational time is close to blocksize 6. In general, it can be seen that a larger blocksize (i.e. more time steps) increases computational times.

Table 17 - Number of non-dominated routes.

<table>
<thead>
<tr>
<th>Blocksize</th>
<th>One directional</th>
<th>Bidirectional</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>20</td>
<td>17</td>
<td>9</td>
</tr>
<tr>
<td>25</td>
<td>23</td>
<td>13</td>
</tr>
<tr>
<td>30</td>
<td>29</td>
<td>17</td>
</tr>
<tr>
<td>35</td>
<td>35</td>
<td>21</td>
</tr>
<tr>
<td>40</td>
<td>41</td>
<td>25</td>
</tr>
<tr>
<td>45</td>
<td>45</td>
<td>29</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>33</td>
</tr>
<tr>
<td>55</td>
<td>55</td>
<td>37</td>
</tr>
<tr>
<td>60</td>
<td>60</td>
<td>41</td>
</tr>
<tr>
<td>65</td>
<td>65</td>
<td>45</td>
</tr>
<tr>
<td>70</td>
<td>70</td>
<td>49</td>
</tr>
<tr>
<td>75</td>
<td>75</td>
<td>53</td>
</tr>
</tbody>
</table>

Computational times double roughly when a bidirectional grid is used instead of a one directional grid. No difference can be noticed between the different correlations, see Appendix B.
Sensitivity of route length

A short route is calculated for different grid sizes, directionalities and block sizes. Only the strong positive correlated grids are assumed.

For the 3x3 grid, route goes from node 1 to node 5 (most direct route length is 2 links). For grid 4x4 and 5x5 from node 1 to node (2*m+3), most direct route length is 4 links. For the remaining grids the route goes from node 1 to node (3*m+4), most direct route length is 6 links.

Table with corresponding computational times:

| Blocksize | One directional | | | | | Bidirectional | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 2 | 4 | 6 | 10 | 12 | | 2 | 4 | 6 | 10 | 12 | |
| 3x3 | 0.108 | 0.066 | 0.071 | 0.072 | 0.093 | | 0.068 | 0.077 | 0.098 | 0.153 | 0.144 | |
| 4x4 | 0.059 | 0.083 | 0.100 | 0.139 | 0.151 | | 0.093 | 0.134 | 0.175 | 0.249 | 0.341 | |
| 5x5 | 0.080 | 0.112 | 0.144 | 0.245 | 0.261 | | 0.120 | 0.206 | 0.313 | 0.435 | 0.656 | |
| 6x6 | 0.167 | 0.157 | 0.214 | 0.347 | 0.390 | | 0.180 | 0.400 | 0.471 | 0.630 | 0.775 | |
| 10x10 | 0.252 | 0.350 | 0.495 | 0.768 | 0.930 | | 0.367 | 0.602 | 0.904 | 1.805 | 1.800 | |
| 15x15 | 0.314 | 0.585 | 0.818 | 1.412 | 1.647 | | 0.566 | 1.126 | 1.580 | 2.764 | 3.280 | |

Figure 54 shows the difference in computational times per blocksize:
The red line represents the longer routes, the blue line the shorter routes. Differences become clear for the larger grids (>10x10), in all cases is the computational time for the shorter route faster. This is a result of the termination statement. The differences are not linearly related to blocksize. Results for a one directional grid are similar and are shown in Appendix B.

In general, computational time increases as blocksize increases (blocksize 2 excluded and some exceptions here and there in the ≥40x40 grids). Computational times for a small blocksize increase rapidly for larger grids (> 20x20), in this case more non-dominated routes are found. However, the question arises if one would want a routing algorithm with this blocksize. With blocksize 2, no clear route is advised and thus accuracy of arrival time reliability is not guaranteed (since the output is given in blocks and has a range of 30 minutes). It is recommended to use data with larger block sizes (≥6, thus more time steps and accurate data).

Sensitivity of correlation threshold

The route is taken from node 1 to node 11 or from node 12 to node 89, network 4x4 and 10x10 respectively. A one directional grid is used. The average of three runs is shown in the following table:

<table>
<thead>
<tr>
<th>Blocksize = 6 (1 block = 10 min.)</th>
<th>Blocksize = 12 (1 block = 5 min.)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rho</strong></td>
<td><strong>Rho</strong></td>
</tr>
<tr>
<td>No clustering</td>
<td>No clustering</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>0.12</td>
<td>0.12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4x4</th>
<th>10x10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.17</td>
<td>0.49</td>
</tr>
<tr>
<td>0.17</td>
<td>0.49</td>
</tr>
<tr>
<td>0.12</td>
<td>4.80</td>
</tr>
<tr>
<td>0.38</td>
<td>22.49</td>
</tr>
<tr>
<td>0.49</td>
<td>35.26</td>
</tr>
<tr>
<td>0.13</td>
<td>0.89</td>
</tr>
<tr>
<td>0.17</td>
<td>2.54</td>
</tr>
<tr>
<td>0.32</td>
<td>27.09</td>
</tr>
<tr>
<td>1.07</td>
<td>136.92</td>
</tr>
<tr>
<td>1.95</td>
<td>251.48</td>
</tr>
</tbody>
</table>

As seen in the other two sensitivity cases, computational times increase when blocksize and/or network size increases. Using a correlation threshold increases computational times. Computational times are further increased when the correlation threshold is lowered, which makes sense because there are more (cluster)links to examine.
7. Conclusion

Reliable routing is important in the field of Transportation Engineering and Logistics. Reliable routing is obtained with an algorithm that takes network stochastics and time-dependency into account. A distribution function shows the cumulative distribution, so that one can see exactly which arrival time has which probability. The aim of this research was to develop such an algorithm in Matlab. Network data is also generated with a script.

The effect of correlation is studied and it appears that the expected arrival time and reliability is underestimated when a grid has a (strong) negative correlation, resulting in a shift of cumulative arrival time distribution to the left. Expected arrival time and reliability is overestimated in case of a (strong) positive correlated grid, thus a shift of cumulative arrival time distribution to the right. This is also seen when a city is simulated. The links near the city have strong positive correlation, the outer links are almost deterministic, but all links are competitive. If one wants to travel to the other side of the city, the algorithm avoids the correlated links and advises to go around the city. This experiment is repeated for strong negative correlated links, but no such clear conclusion can be drawn. This might be due to the fact that a (strong) negative correlated bidirectional grid is impossible (in each adjacent link pair, one has upward peaks and the other has downward peaks). The shift in arrival time is however not seen when clustering is applied. Instead, a small decrease in arrival time is noticed. If this is due to the generated data or the cluster algorithm should be further investigated.

The benefit of a bidirectional grid becomes clear when a city is simulated, and links close to the city have long travel times. The preferred direction is not only to travel in the most direct to the destination, but detours might become beneficial.

Larger grids show overlap in advised non-dominated routes. The variance intervals (10\(^{th}\), 50\(^{th}\) and 90\(^{th}\) %) of the advised and some alternative routes are studied. The studied alternatives are disadvantageous because their minimum link travel time is higher, they have more intense peaks and the variance intervals lay closer to the mean. Therefore, these alternatives are easy to dominate.

Computational times of different grid sizes and blocksizes are tested for one directional and bidirectional grids. Computational times for a bidirectional grid are roughly doubled compared to a one directional grid. The termination statement has an influence on computational times when short routes are calculated, a clear decrease in computational time is seen for the larger grids (≥20x20). Blocksize 2 increases computational times enormously in grids ≥20x20 and is correlated with the number of non-dominated routes, these increase rapidly. Blocksize 2 is not recommended, based on the negative influence on computational time and the fact that a less accurate advise can be given compared to larger blocksizes (these have more time steps, thus the arrival time range is less wide). Lowering the correlation threshold (applied when one uses clusterlinks) increases computational times.

Recommendations
- Generate more real-life like data and networks, or
- Use real-time data to test the algorithm and its computational speed.
- Fine-tune the tolerance in compare_functions to check for first order stochastic dominance.
- Implement the algorithm in another programming language, so it can handle larger grids in a faster way.
- Make speed improvements (for example: use a Pareto frontier in compare_functions).
- Implement an generic clustering technique.
- Make the algorithm more generic for different types of data input.
- Make the route searching algorithm more efficient. Do route searching from both start and finish node instead of only the start node.
References


Appendix A – Effect of blocksize on arrival times

Chapter 6.1 showed an opposite effect in blocksize 6 on the output probabilities. The probabilities of routes in the 5x5 grid shifted to the left, while the probabilities in the 10x10 grid shifted to the right when a larger blocksize is used. In this appendix, this test is repeated for a 6x6 and 15x15 grid.

6x6 bidirectional grid

Two similar routes are found with every blocksize. Route 8-14-20-21-22-23-29 (left figure) and route 8-14-20-26-27-28-29 (right figure).

Figure 55 - Bar plot of travel time probabilities, blocksize 6.

Figure 56 - Bar plot of travel time probabilities, blocksize 10.

Figure 57 - Bar plot of travel time probabilities, blocksize 12.

A shift to the left is seen when blocksize increases (i.e. there are more time blocks in a day).
Table 20 - Variance of the bar plots, 6x6 grid.

<table>
<thead>
<tr>
<th>Route 8-14-20-21-22-23-29</th>
<th>Blocksize 6</th>
<th>Blocksize 10</th>
<th>Blocksize 12</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>54,17</td>
<td>50,0</td>
<td>54,17</td>
</tr>
<tr>
<td>Route 8-14-20-26-27-28-29</td>
<td>58,50</td>
<td>58,50</td>
<td>54,17</td>
</tr>
</tbody>
</table>

The variance shows no clear consensus per route, the second decreases a bit.

15x15 bidirectional grid


![Figure 58: Bar plot of travel time probabilities, blocksize 6.](image)

![Figure 59: Bar plot of travel time probabilities, blocksize 10.](image)

No shift is noticed for the 15x15 grid. The probability distribution is different though. Blocksize 6 has a clearer mean compared to blocksize 10, this one is more smooth. This is also seen in the variance; variance of the upper figure is 54,17 and of the bottom figure 77,50.

Four grid sizes are compared on their effect of blocksize on arrival time probabilities. For the smallest two grids, 5x5 and 6x6, arrival times shift to left when a larger blocksize is chosen. Grid 10x10 shows the opposite effect; a shift to the right. Grid 15x15 shows no shift at all, only an increase in arrival time variance.

This change in variance is not clearly seen in the other grids, let alone an increase. Grid 10x10 shows a decrease in variance. Grids 5x5 and 6x6 show some decrease.

No conclusions can be drawn from these results, it appears that the effects are quite random.
Appendix B – Results of computational times

This appendix presents the computational times per correlation and directionality in graphs. Also some profiler outputs are given.

B.1 Correlation scenarios

Correlation scenarios +, no correlation, - and - -

Figure 60 - Computational times for different grid sizes, positive correlated grid.

Figure 61 - Computational times for different grid sizes, none correlated grid.

Figure 62 - Computational times for different grid sizes, negative correlated grid.
Figure 63 - Computational times for different grid sizes, strong negative correlated grid.

These graphs are comparable to the ++ correlated graphs presented in Chapter 6.5.

**Time differences short vs long routes**

Time differences are plotted for the results of a one directional grid and for every blocksize.

Figure 64 - Computational time differences for long and short routes, one directional grids.

The results are comparable to the results of a bidirectional grid presented in Chapter 6.5.
### B.2 Profiler

**Profiler 5x5 blocksize 2**

Bidirectional grid, start node is 7, finish node is 19, start time is 6:00h.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Calls</th>
<th>Total Time</th>
<th>Self Time</th>
<th>Total Time Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN_network</td>
<td>1</td>
<td>0.652 s</td>
<td>0.166 s</td>
<td></td>
</tr>
<tr>
<td>RUN_MAIN</td>
<td>1</td>
<td>0.463 s</td>
<td>0.115 s</td>
<td></td>
</tr>
<tr>
<td>add_probability</td>
<td>96</td>
<td>0.130 s</td>
<td>0.037 s</td>
<td></td>
</tr>
<tr>
<td>compare_functions</td>
<td>53</td>
<td>0.102 s</td>
<td>0.082 s</td>
<td></td>
</tr>
<tr>
<td>remove_empty_cells</td>
<td>219</td>
<td>0.065 s</td>
<td>0.042 s</td>
<td></td>
</tr>
<tr>
<td>loop_checker</td>
<td>80</td>
<td>0.059 s</td>
<td>0.014 s</td>
<td></td>
</tr>
<tr>
<td>ismember</td>
<td>715</td>
<td>0.043 s</td>
<td>0.010 s</td>
<td></td>
</tr>
<tr>
<td>is_equal</td>
<td>53</td>
<td>0.043 s</td>
<td>0.034 s</td>
<td></td>
</tr>
<tr>
<td>ismember&gt;ismemberR2012a</td>
<td>715</td>
<td>0.038 s</td>
<td>0.024 s</td>
<td></td>
</tr>
<tr>
<td>add_link_to_node</td>
<td>130</td>
<td>0.033 s</td>
<td>0.033 s</td>
<td></td>
</tr>
</tbody>
</table>

**Profiler 5x5 blocksize 12**

Bidirectional grid, start node is 7, finish node is 19, start time is 6:00h.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Calls</th>
<th>Total Time</th>
<th>Self Time</th>
<th>Total Time Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN_network</td>
<td>1</td>
<td>1.385 s</td>
<td>0.345 s</td>
<td></td>
</tr>
<tr>
<td>RUN_MAIN</td>
<td>1</td>
<td>0.658 s</td>
<td>0.117 s</td>
<td></td>
</tr>
<tr>
<td>close</td>
<td>1</td>
<td>0.381 s</td>
<td>0.002 s</td>
<td></td>
</tr>
<tr>
<td>close&gt;request_close</td>
<td>1</td>
<td>0.305 s</td>
<td>0.010 s</td>
<td></td>
</tr>
<tr>
<td>add_probability</td>
<td>88</td>
<td>0.304 s</td>
<td>0.040 s</td>
<td></td>
</tr>
<tr>
<td>closereq</td>
<td>1</td>
<td>0.288 s</td>
<td>0.149 s</td>
<td></td>
</tr>
<tr>
<td>add_link_to_node</td>
<td>141</td>
<td>0.192 s</td>
<td>0.192 s</td>
<td></td>
</tr>
<tr>
<td>Legend.doSetup&gt;=(h_c)Obj.doDelete</td>
<td>1</td>
<td>0.134 s</td>
<td>0.003 s</td>
<td></td>
</tr>
<tr>
<td>Legend.doDelete</td>
<td>1</td>
<td>0.131 s</td>
<td>0.122 s</td>
<td></td>
</tr>
<tr>
<td>compare_functions</td>
<td>59</td>
<td>0.113 s</td>
<td>0.090 s</td>
<td></td>
</tr>
</tbody>
</table>
25x25 blocksize 2
Bidirectional grid, start node is 27, finish node is 599, start time is 6:00h.

Profile Summary

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Calls</th>
<th>Total Time</th>
<th>Self Time</th>
<th>Total Time Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN_network</td>
<td>1</td>
<td>283.795 s</td>
<td>4.486 s</td>
<td></td>
</tr>
<tr>
<td>RUN_MAIN</td>
<td>1</td>
<td>279.293 s</td>
<td>0.867 s</td>
<td></td>
</tr>
<tr>
<td>compare_functions</td>
<td>1780</td>
<td>218.401 s</td>
<td>172.719 s</td>
<td></td>
</tr>
<tr>
<td>ismember</td>
<td>1667514</td>
<td>52.548 s</td>
<td>9.645 s</td>
<td></td>
</tr>
<tr>
<td>ismemberangible</td>
<td>1667514</td>
<td>42.994 s</td>
<td>25.397 s</td>
<td></td>
</tr>
<tr>
<td>add_probability</td>
<td>2408</td>
<td>34.054 s</td>
<td>3.525 s</td>
<td></td>
</tr>
<tr>
<td>add_link_to_node</td>
<td>58147</td>
<td>19.884 s</td>
<td>19.884 s</td>
<td></td>
</tr>
<tr>
<td>is_equal</td>
<td>1780</td>
<td>19.410 s</td>
<td>17.027 s</td>
<td></td>
</tr>
<tr>
<td>ismemberangibleBuiltInTypes</td>
<td>1667514</td>
<td>17.507 s</td>
<td>17.507 s</td>
<td></td>
</tr>
<tr>
<td>clean_up_probability</td>
<td>58147</td>
<td>7.911 s</td>
<td>5.779 s</td>
<td></td>
</tr>
</tbody>
</table>

Inside **RUN_MAIN**:

Children (called functions)

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Function Type</th>
<th>Calls</th>
<th>Total Time</th>
<th>% Time</th>
<th>Time Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>compare_functions</td>
<td>function</td>
<td>1780</td>
<td>218.401 s</td>
<td>78.2%</td>
<td></td>
</tr>
<tr>
<td>add_probability</td>
<td>function</td>
<td>2408</td>
<td>34.054 s</td>
<td>12.2%</td>
<td></td>
</tr>
<tr>
<td>is_equal</td>
<td>function</td>
<td>1780</td>
<td>19.410 s</td>
<td>6.9%</td>
<td></td>
</tr>
<tr>
<td>loop_check</td>
<td>function</td>
<td>2408</td>
<td>5.162 s</td>
<td>1.5%</td>
<td></td>
</tr>
<tr>
<td>expected_serial_time</td>
<td>function</td>
<td>2408</td>
<td>1.147 s</td>
<td>0.4%</td>
<td></td>
</tr>
<tr>
<td>add_route</td>
<td>function</td>
<td>2408</td>
<td>0.250 s</td>
<td>0.1%</td>
<td></td>
</tr>
<tr>
<td>add_route_and_nodes</td>
<td>function</td>
<td>1</td>
<td>0.002 s</td>
<td>0.0%</td>
<td></td>
</tr>
<tr>
<td>Self time (built-ins, overhead, etc.)</td>
<td></td>
<td>0.867 s</td>
<td>0.3%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td>279.293 s</td>
<td>100%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Inside **compare_functions**:

Children (called functions)

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Function Type</th>
<th>Calls</th>
<th>Total Time</th>
<th>% Time</th>
<th>Time Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>ismember</td>
<td>function</td>
<td>1369037</td>
<td>43.000 s</td>
<td>18.7%</td>
<td></td>
</tr>
<tr>
<td>remove_functions</td>
<td>function</td>
<td>1472</td>
<td>2.652 s</td>
<td>1.2%</td>
<td></td>
</tr>
<tr>
<td>Self time (built-ins, overhead, etc.)</td>
<td></td>
<td>172.719 s</td>
<td>79.1%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td>218.401 s</td>
<td>100%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This functions requires lots of computational time in **compare_functions**, where all unique pairs of routes are compared for non-dominance.
Bidirectional grid, start node is 27, finish node is 599, start time is 6:00h.

### Profile Summary

**Profiler 25x25 blocksize 12**


<table>
<thead>
<tr>
<th>Function Name</th>
<th>Calls</th>
<th>Total Time</th>
<th>Self Time</th>
<th>Total Time Plot (dark band = self time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN_network</td>
<td>1</td>
<td>33.888 s</td>
<td>24.343 s</td>
<td></td>
</tr>
<tr>
<td>RUN_MAIN</td>
<td>1</td>
<td>9.530 s</td>
<td>0.643 s</td>
<td></td>
</tr>
<tr>
<td>add_probability</td>
<td>2408</td>
<td>7.078 s</td>
<td>0.381 s</td>
<td></td>
</tr>
<tr>
<td>add_link_to_node</td>
<td>3304</td>
<td>5.633 s</td>
<td>5.633 s</td>
<td></td>
</tr>
<tr>
<td>compare_functions</td>
<td>1780</td>
<td>0.846 s</td>
<td>0.518 s</td>
<td></td>
</tr>
<tr>
<td>clean_up_probability</td>
<td>3304</td>
<td>0.826 s</td>
<td>0.621 s</td>
<td></td>
</tr>
<tr>
<td>ismember</td>
<td>22754</td>
<td>0.734 s</td>
<td>0.148 s</td>
<td></td>
</tr>
<tr>
<td>remove_empty_cells</td>
<td>6596</td>
<td>0.508 s</td>
<td>0.324 s</td>
<td></td>
</tr>
<tr>
<td>ismember_ismemberR2012a</td>
<td>22754</td>
<td>0.586 s</td>
<td>0.303 s</td>
<td></td>
</tr>
<tr>
<td>loop_checker</td>
<td>2408</td>
<td>0.464 s</td>
<td>0.090 s</td>
<td></td>
</tr>
</tbody>
</table>

**Inside RUN_MAIN:**

**Children (called functions)**

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Function Type</th>
<th>Calls</th>
<th>Total Time</th>
<th>% Time</th>
<th>Time Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_probability</td>
<td>function</td>
<td>2408</td>
<td>7.078 s</td>
<td>74.3%</td>
<td></td>
</tr>
<tr>
<td>compare_functions</td>
<td>function</td>
<td>1780</td>
<td>0.846 s</td>
<td>8.9%</td>
<td></td>
</tr>
<tr>
<td>loop_checker</td>
<td>function</td>
<td>2408</td>
<td>0.464 s</td>
<td>4.9%</td>
<td></td>
</tr>
<tr>
<td>is_equal</td>
<td>function</td>
<td>1780</td>
<td>0.313 s</td>
<td>3.3%</td>
<td></td>
</tr>
<tr>
<td>expected_arrival_time</td>
<td>function</td>
<td>2408</td>
<td>0.136 s</td>
<td>1.4%</td>
<td></td>
</tr>
<tr>
<td>add_node</td>
<td>function</td>
<td>2408</td>
<td>0.043 s</td>
<td>0.6%</td>
<td></td>
</tr>
<tr>
<td>add_node_and_nodes</td>
<td>function</td>
<td>1</td>
<td>0.000 s</td>
<td>0.6%</td>
<td></td>
</tr>
<tr>
<td>Self time (built-ins, overhead, etc.)</td>
<td></td>
<td></td>
<td>0.643 s</td>
<td>6.7%</td>
<td></td>
</tr>
</tbody>
</table>

**Totals**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>9.530 s</td>
<td>100%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Inside add_probability:**

**Children (called functions)**

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Function Type</th>
<th>Calls</th>
<th>Total Time</th>
<th>% Time</th>
<th>Time Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_link_to_node</td>
<td>function</td>
<td>3304</td>
<td>6.633 s</td>
<td>79.6%</td>
<td></td>
</tr>
<tr>
<td>clean_up_probability</td>
<td>function</td>
<td>3304</td>
<td>0.826 s</td>
<td>11.7%</td>
<td></td>
</tr>
<tr>
<td>remove_empty_cells</td>
<td>function</td>
<td>2408</td>
<td>0.239 s</td>
<td>3.4%</td>
<td></td>
</tr>
<tr>
<td>Self time (built-ins, overhead, etc.)</td>
<td></td>
<td></td>
<td>0.361 s</td>
<td>5.4%</td>
<td></td>
</tr>
</tbody>
</table>

**Totals**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7.078 s</td>
<td>100%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Inside `add_link_to_node`:

Lines where the most time was spent

<table>
<thead>
<tr>
<th>Line Number</th>
<th>Code</th>
<th>Calls</th>
<th>Total Time</th>
<th>% Time</th>
<th>Time Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td><code>new_p[index(i)] = []; ...</code></td>
<td>53876</td>
<td>2.098 s</td>
<td>37.2%</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td><code>new_c[index(i)] = []; ...</code></td>
<td>53876</td>
<td>2.007 s</td>
<td>36.9%</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td><code>new_c = [new_c; t(i)+t(i)]; ...</code></td>
<td>112231</td>
<td>0.480 s</td>
<td>6.5%</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td><code>new_p = [new_p; p(i)+p(i)]; ...</code></td>
<td>112231</td>
<td>0.453 s</td>
<td>6.0%</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td><code>t(i,3) = probability_link(i,3) ...</code></td>
<td>112231</td>
<td>0.177 s</td>
<td>3.1%</td>
<td></td>
</tr>
<tr>
<td>All other lines</td>
<td></td>
<td></td>
<td>0.349 s</td>
<td>6.2%</td>
<td></td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td></td>
<td>5.633 s</td>
<td>100%</td>
<td></td>
</tr>
</tbody>
</table>

In line 32 and 33, the probabilities that are smaller than 0.001 are removed.

Profiler 40x40 blocksize 2

Bidirectional grid, start node is 42, finish node is 1599, start time is 6:00h.

Profile Summary


<table>
<thead>
<tr>
<th>Function Name</th>
<th>Calls</th>
<th>Total Time</th>
<th>Self Time*</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN_network</td>
<td>1</td>
<td>13975.173 s</td>
<td>4.382 s</td>
</tr>
<tr>
<td>RUN_MAIN</td>
<td>1</td>
<td>13970.778 s</td>
<td>3.876 s</td>
</tr>
<tr>
<td>compare_functions</td>
<td>4673</td>
<td>12551.005 s</td>
<td>10540.041 s</td>
</tr>
<tr>
<td>ismember</td>
<td>67214220</td>
<td>2093.172 s</td>
<td>386.183 s</td>
</tr>
<tr>
<td>ismemberR2012a</td>
<td>67214220</td>
<td>1706.980 s</td>
<td>682.602 s</td>
</tr>
<tr>
<td>is_equal</td>
<td>4673</td>
<td>891.990 s</td>
<td>884.245 s</td>
</tr>
<tr>
<td>ismemberR2012aBuiltinTypes</td>
<td>67214220</td>
<td>724.387 s</td>
<td>724.387 s</td>
</tr>
<tr>
<td>add_probability</td>
<td>6276</td>
<td>444.875 s</td>
<td>44.540 s</td>
</tr>
<tr>
<td>add_link_to_node</td>
<td>729609</td>
<td>268.497 s</td>
<td>268.497 s</td>
</tr>
<tr>
<td>clean_up_probability</td>
<td>729609</td>
<td>100.524 s</td>
<td>76.091 s</td>
</tr>
<tr>
<td>remove_empty_cells</td>
<td>17225</td>
<td>81.892 s</td>
<td>25.447 s</td>
</tr>
<tr>
<td>loop_checker</td>
<td>6276</td>
<td>59.951 s</td>
<td>4.817 s</td>
</tr>
<tr>
<td>remove_functions</td>
<td>4236</td>
<td>32.710 s</td>
<td>6.536 s</td>
</tr>
<tr>
<td>sortrows</td>
<td>729609</td>
<td>24.434 s</td>
<td>8.934 s</td>
</tr>
<tr>
<td>expected_arrival_time</td>
<td>6276</td>
<td>16.585 s</td>
<td>16.585 s</td>
</tr>
</tbody>
</table>
Inside *RUN_MAIN*:

**Children (called functions)**

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Function Type</th>
<th>Calls</th>
<th>Total Time</th>
<th>% Time</th>
<th>Time Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>compare_functions</td>
<td>function</td>
<td>4673</td>
<td>12551.005 s</td>
<td>89.8%</td>
<td></td>
</tr>
<tr>
<td>is_equal</td>
<td>function</td>
<td>4673</td>
<td>891.950 s</td>
<td>6.4%</td>
<td></td>
</tr>
<tr>
<td>add_probability</td>
<td>function</td>
<td>6276</td>
<td>444.875 s</td>
<td>3.2%</td>
<td></td>
</tr>
<tr>
<td>loop_checker</td>
<td>function</td>
<td>6276</td>
<td>59.951 s</td>
<td>0.4%</td>
<td></td>
</tr>
<tr>
<td>expected_annual_time</td>
<td>function</td>
<td>6276</td>
<td>16.665 s</td>
<td>0.1%</td>
<td></td>
</tr>
<tr>
<td>add_route</td>
<td>function</td>
<td>6276</td>
<td>2.493 s</td>
<td>0.0%</td>
<td></td>
</tr>
<tr>
<td>add_route_end_nodes</td>
<td>function</td>
<td>1</td>
<td>0.004 s</td>
<td>0.0%</td>
<td></td>
</tr>
<tr>
<td>Self time (built-ins, overhead, etc.)</td>
<td></td>
<td></td>
<td>3.876 s</td>
<td>0.0%</td>
<td></td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td></td>
<td>13970.778 s</td>
<td>100%</td>
<td></td>
</tr>
</tbody>
</table>

Inside *compare_functions*:

**Children (called functions)**

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Function Type</th>
<th>Calls</th>
<th>Total Time</th>
<th>% Time</th>
<th>Time Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>ismember</td>
<td>function</td>
<td>63431766</td>
<td>19782.249 s</td>
<td>15.8%</td>
<td></td>
</tr>
<tr>
<td>remove_functions</td>
<td>function</td>
<td>4236</td>
<td>32.716 s</td>
<td>0.3%</td>
<td></td>
</tr>
<tr>
<td>Self time (built-ins, overhead, etc.)</td>
<td></td>
<td></td>
<td>10540.041 s</td>
<td>84.0%</td>
<td></td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td></td>
<td>12551.005 s</td>
<td>100%</td>
<td></td>
</tr>
</tbody>
</table>

Profiler 75x75 blocksize 12
Bidirectional grid, start node is 77, finish node is 5549, start time is 6:00h.

**Profile Summary**


<table>
<thead>
<tr>
<th>Function Name</th>
<th>Calls</th>
<th>Total Time</th>
<th>Self Time</th>
<th>Total Time Plot (dark band = self time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN_network</td>
<td>1</td>
<td>312.266 s</td>
<td>225.952 s</td>
<td></td>
</tr>
<tr>
<td>RUN_MAIN</td>
<td>1</td>
<td>89.300 s</td>
<td>8.246 s</td>
<td></td>
</tr>
<tr>
<td>add_probability</td>
<td>22323</td>
<td>63.878 s</td>
<td>3.196 s</td>
<td></td>
</tr>
<tr>
<td>add_link_to_node</td>
<td>29166</td>
<td>51.765 s</td>
<td>51.765 s</td>
<td></td>
</tr>
<tr>
<td>clean_up_probability</td>
<td>29166</td>
<td>7.001 s</td>
<td>5.281 s</td>
<td></td>
</tr>
<tr>
<td>compare_functions</td>
<td>16695</td>
<td>6.501 s</td>
<td>3.763 s</td>
<td></td>
</tr>
<tr>
<td>ismember</td>
<td>202126</td>
<td>6.141 s</td>
<td>1.249 s</td>
<td></td>
</tr>
<tr>
<td>ismember=ismemberR2012a</td>
<td>202126</td>
<td>4.892 s</td>
<td>3.342 s</td>
<td></td>
</tr>
<tr>
<td>remove_empty_cells</td>
<td>81341</td>
<td>4.854 s</td>
<td>2.906 s</td>
<td></td>
</tr>
<tr>
<td>loop_checker</td>
<td>22323</td>
<td>3.864 s</td>
<td>0.888 s</td>
<td></td>
</tr>
</tbody>
</table>
Inside *RUN_MAIN*:

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Function Type</th>
<th>Calls</th>
<th>Total Time</th>
<th>% Time</th>
<th>Time Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_probability</td>
<td>function</td>
<td>22323</td>
<td>63.878 s</td>
<td>74.6%</td>
<td></td>
</tr>
<tr>
<td>compare_functions</td>
<td>function</td>
<td>16695</td>
<td>8.501 s</td>
<td>7.5%</td>
<td></td>
</tr>
<tr>
<td>loop_checker</td>
<td>function</td>
<td>22323</td>
<td>3.664 s</td>
<td>4.2%</td>
<td></td>
</tr>
<tr>
<td>is_equal</td>
<td>function</td>
<td>16695</td>
<td>2.511 s</td>
<td>2.9%</td>
<td></td>
</tr>
<tr>
<td>expected_arrival_time</td>
<td>function</td>
<td>22323</td>
<td>1.149 s</td>
<td>1.3%</td>
<td></td>
</tr>
<tr>
<td>add_route</td>
<td>function</td>
<td>22323</td>
<td>0.351 s</td>
<td>0.4%</td>
<td></td>
</tr>
<tr>
<td>add_route_and_node</td>
<td>function</td>
<td>1</td>
<td>0.000 s</td>
<td>0.0%</td>
<td></td>
</tr>
<tr>
<td>Self time (built-ins, overhead, etc.)</td>
<td></td>
<td></td>
<td>8.245 s</td>
<td>9.6%</td>
<td></td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td></td>
<td>66.300 s</td>
<td>100%</td>
<td></td>
</tr>
</tbody>
</table>

Inside *add_probability*:

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Function Type</th>
<th>Calls</th>
<th>Total Time</th>
<th>% Time</th>
<th>Time Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_link_to_node</td>
<td>function</td>
<td>29165</td>
<td>51.765 s</td>
<td>81.0%</td>
<td></td>
</tr>
<tr>
<td>clean_up_probability</td>
<td>function</td>
<td>29165</td>
<td>7.001 s</td>
<td>11.0%</td>
<td></td>
</tr>
<tr>
<td>remove_empty_cells</td>
<td>function</td>
<td>22323</td>
<td>1.917 s</td>
<td>3.0%</td>
<td></td>
</tr>
<tr>
<td>Self time (built-ins, overhead, etc.)</td>
<td></td>
<td></td>
<td>3.196 s</td>
<td>5.0%</td>
<td></td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td></td>
<td>63.878 s</td>
<td>100%</td>
<td></td>
</tr>
</tbody>
</table>

Within *add_link_to_node*, line 32 and 33 represent 37.6% and 37.4% of the computational time respectively. Same as seen in grid 5x5 and grid 25x25, both with blocksize 12.
Appendix C – Matlab scripts

C.1 Routing algorithm scripts

C.1.1 RUN_network.m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% RUN_network
% State properties of the route and let route calculate.
% Options:
- show arrival times for all start times, best route chosen
- show routes between 2 nodes, given a starting time
- give best route between 2 nodes, given a starting time

Daphne Overbeek, Research Assignment TEL, 2016
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear
close
clc

%% State route properties and run option
nr_network = 10;
Start = nr_network+2;
Finish = (nr_network*nr_network)-nr_network-1;
blocksize = 2;
t_start = 6*blocksize;
one_direction = 2;
correlation = 11;
version = 1;
all_start_times = 0;
all_routes = 0;
terminate = 1;
fifo = 0;
Reliability_checkpoint = 0.5;
show_routes = 7;

%% Import network
nr_times = 24*blocksize;
if correlation == 11
    text = printf('Links%d_%d_%d_t++_v%d.mat',nr_network,blocksize,one_direction,version);
elseif correlation == 1
    text = sprintf('Links%d_%d_%d_t+_v%d.mat',nr_network,blocksize,one_direction,version);
elseif correlation == 0
    text = sprintf('Links%d_%d_%d_t0_v%d.mat',nr_network,blocksize,one_direction,version);
elseif correlation == -1
    text = sprintf('Links%d_%d_%d_t-_v%d.mat',nr_network,blocksize,one_direction,version);
elseif correlation == -11
    text = sprintf('Links%d_%d_%d_t--_v%d.mat',nr_network,blocksize,one_direction,version);
end
Links = load(text);

%% FIFO check
if fifo == 1
    [loop] = FIFO(Links,nr_links,nr_times);
end

%% Calculate route
if all_start_times == 0 && all_routes == 0
    [Route, Arrival_time,ProbaBility] = RUN_MAIN(Start, t_start, Finish,Links,nr_links,nr_nodes,all_routes,terminate)
elseif all_start_times == 1 && all_routes == 0
    [Route, Arrival_time,Probability] = Run_all_start_times(Start,Finish,Links,nr_links,nr_times,nr_nodes,Reliability_checkpoint,terminate)
elseif all_start_times == 0 && all_routes == 1
    [Route, Arrival_time,Probability] = RUN_MAIN(Start, t_start, Finish,Links,nr_links,nr_nodes,all_routes,terminate)
elem = numel(Route); % Number of routes
if elem >= 1 % More than one route
    for i = 1:elem % For all routes
        Arrival_time_new(i,1) = Arrival_time{i}; % Save arrival time in array
        Travel_time_new(i,1) = Arrival_time_new(i,1)-t_start; % Calculate travel time
        Labels{i,1} = sprintf('%d ',Route{i}); % Update labels with route
    end
end
figure
bar(Travel_time_new, 'BarWidth', 0.6) % Make bar plot of travel times
set(gca, 'XTickLabel', Labels) % Set labels
xlabel('Routes') % Set xlabel
ylabel('Expected travel time') % Set ylabel
title('Possible routes and expected travel times') % Set title
end

%% Show output (not if all_start_times == 1)
if elem >= 1 && all_start_times == 0 % Check all_start_times is not 1
    Arrival_time = cell2mat(Arrival_time); % Make array of arrival_time
    [min_arr, min_arr_index] = sort(Arrival_time); % Sort arrival time
    show_routes = numel(Arrival_time); % Update show_routes if there are less routes than original value of show_routes
end
for i = 1:show_routes % For amount of routes to show
    prob_temp{1,i} = Probability{1,min_arr_index(i)}; % Find corresponding route data (arrival times and probabilities)
    route_temp{1,i} = Route{1,min_arr_index(i)}; % Find corresponding routes
end
Arrival_time_reliability = plot_RUN(prob_temp, Reliability_checkpoint, arrival, route_temp, min_arr_index(1:show_routes)); % Plot the routes
xlabel('Arrival time [block]'); % Set xlabel
ylabel('Cumulative probability [-]'); % Set ylabel
str = sprintf('Route(s) from node %d to node %d starting at t = d', Start, Finish, t_start); % Set title
title(str); % Set title
end
function [Routes, Arrival_times, Probability] = RUN_MAIN(Start, t_start, Finish, Links,nr_links,nr_nodes, all_routes,terminate)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% RUN_MAIN
% This function searches the non-dominated route(s) from Start to Finish node.
%% Daphne Overbeek, Research Assignment TEL, 2016
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% INITIALISATION
arrival_time = 1; % Nodes column index
new_arrival_time = 2; % Nodes column index
probability = 3; % Nodes column index
new_probability = 4; % Nodes column index
route = 5; % Nodes column index
new_route = 6; % Nodes column index
node_changed = 7; % Nodes column index

% MAKE NODE LIST
Nodes = cell(nr_nodes,7); % Make cell
Nodes{Start,arrival_time} = cell(1,1); % Make cell
Nodes{Start,arrival_time}{1} = 0; % Arrival time at Start node is initialized at 0
Nodes{Start,probability} = cell(1,1); % Make cell
Nodes{Start,probability}{1} = [t_start 1]; % Times and probability at Start node is initialized as [t_start 1] ([Time Probability])
Nodes{Start,route} = cell(1,1); % Make cell
Nodes{Start,route}{1} = []; % Route at Start node is initialized as empty
Nodes{Start,node_changed} = 1; % Initialize as 1 such that it is evaluated in iteration

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ITERATION
min_max_t = 0; % Initialise as 0 (termination condition)
Change = 1; % Flag for iteration (run while loop)
while Change == 1 % Iterate until a steady state is reached
    Change = 0; % Update iteration flag
    for i = 1:nr_links % For all links
        Begin = Links(i,1); % Begin node of link
        if Nodes{Begin,node_changed} == 1 % Continue if routes to begin node have changed in last iteration
            if isempty(Nodes{Begin,arrival_time}) == 0 % Route is extendable
                eind = Links(i,2); % End node of link
                probability_node = Nodes{Begin,probability}; % Find route data (arrival times and probabilities)
                [Add_probability, stop] = add_probability(probability_node, Links, i); % Sum route data with link data [function]
                if stop == 0 % Route is extendable
                    % Termination condition
                    min_start_t = Add_probability{1,1}(1,1); % Minimal arrival time of current route
                    if numel(Add_probability{1,1}) > 1 % More than one subroute
                        for n = 2:numel(Add_probability{1,1}) % Check if minimal arrival times of subroutes are smaller
                            if Add_probability{1,n}(1,1) < min_start_t % If min arrival time is smaller
                                min_start_t = Add_probability{1,n}(1,1); % Update min_start_t
                            end
                        end
                    end
                    if min_max_t == 0 || min_start_t < min_max_t % Termination condition:
                        % Current route has not exceeded the minimal finish arrival time
                        Add_time = expected_arrival_time(Add_probability); % Calculate expected arrival time [function]
                        Add_route = add_route(Nodes{Begin,route},Begin); % Update route
                        [Add_route, Add_time, Add_probability] = loop_checker(eind,Add_route,Add_probability,Add_time); % Loop checker
                        if isempty(Nodes{eind,new_arrival_time}) == 1 % There is no route to current link end node stored in Nodes
                            % Addition of route to Nodes
                            Nodes{eind,new_arrival_time} = cell(1,1); % Make cell
                            Nodes{eind,new_arrival_time}{1} = Add_time; % Arrival time at end node is initialized
                            Nodes{eind,new_probability} = cell(1,1); % Make cell
                            Nodes{eind,new_probability}{1} = Add_probability{1,1}; % Probability at end node is initialized
                            Nodes{eind,new_route} = cell(1,1); % Make cell
                            Nodes{eind,new_route}{1} = Add_route; % Route at end node is initialized
                            Nodes{eind,node_changed} = 1; % Initialize as 1 such that it is evaluated in iteration
                        end
                    end
                end
            end
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Loop checker
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Addition of route to Nodes
% There is no route to current link end node stored in Nodes
Nodes{eind,new_arrival_time} = Add_time; % Add expected arrival time
Nodes{eind,new_probability} = Add_probability; % Add corresponding probability
Nodes{eind,new_route} = Add_route; % Add corresponding route
else % There is/are route(s) to current link end node
[Add_route, Add_time, Add_probability] = is_equal(Nodes,
Add_route,Add_time, Add_probability,eind); % Check all routes for similarity, output: unique routes
Nodes{eind,new_arrival_time} = [Nodes{eind,new_arrival_time},
Add_time]; % Add expected arrival time in new cell
Nodes{eind,new_route} = [Nodes{eind,new_route}, Add_route]; % Add route in new cell
Nodes{eind,new_probability} = [Nodes{eind,new_probability},
Add_probability]; % Add route data in new cell
if all_route, % Remove routes that are dominated
[Arrival_time, Route, Probability] =
compare_functions(Nodes,eind); % Check which route(s) are non-dominated [function]
Nodes{eind,new_arrival_time} = Arrival_time; % Update expected arrival times
Nodes{eind,new_route} = Route; % Update route
Nodes{eind,new_probability} = Probability; % Update route data {{arrival times probabilities}
end
end
end
end
end
end
end
end
end
end
% Update min_max_t of termination condition
if terminate == 1
if isempty(Nodes{Finish,new_probability}) == 0 % Termination condition is active
num_fin = numel(Nodes{Finish,new_probability}(1,:)); % Number of subroutes to Finish node
min_max_t = Nodes{Finish,new_probability}(1,1:end,1); % Latest arrival time of 1st route to Finish
for n = 2:num_fin % For other subroutes
if Nodes{Finish,new_probability}(1,n,end,1) < min_max_t % If latest arrival time is faster
min_max_t = Nodes{Finish,new_probability}(1,n,end,1); % Update min_max_t
end
end
end
end % Check if something changed in Nodes
for i = 1:nr_nodes % For all nodes
changed = not(isequal(Nodes{i,route},Nodes{i,new_route})); % Check for equality of routes between routes found in current iteration and previous iteration
Nodes{i,node_changed} = changed; % Save outcome in 7th column of Nodes (1: change, 0: no change)
if changed == 1 % There is a change
Change = 1; % Do a new iteration
end
end
% Update Nodes and initialise Start row
for i = 1:nr_nodes % For all nodes
Nodes{i,arrival_time} = Nodes{i,new_arrival_time}; % Arrival times in 2nd column moves to 1st column
Nodes{i,probability} = Nodes{i,new_probability}; % Route data from 4th column move to 3rd column
Nodes{i,route} = Nodes{i,new_route}; % Routes from 6th column move to 5th column
end
Nodes{Start,arrival_time} = cell(1,1); % Make cell
Nodes{Start,arrival_time}{1} = 0; % Initialise at 0
Nodes{Start,probability} = cell(1,1); % Make cell
Nodes{Start,probability}{1} = [t_start 1]; % Data at Start node is initialized as [t_start 1] ([Time Probability])
Nodes{Start,route} = cell(1,1); % Make cell
Nodes{Start,route}{1} = []; % Initialise as empty
Nodes{Start,new_route} = Nodes{Start,route}; % Initialise as empty
Arrival_times = Nodes(Finish,arrival_time); % Find expected arrival times for Finish node
Probability = Nodes(Finish,probability); % Find route data ([arrival_time
\ probability]) for Finish node
Routes = Nodes(Finish,route); % Find routes for Finish node
Routes = add_route_end_nodes(Routes,Finish); % Add Finish node to all routes
end

C.1.3 add_probability.m

function [Add_probability, stop] = add_probability(probability_node,Links,i)

% This function calculates the new data for the route from start node to %
% current end node. Route extensions are sought, if none available for all% %
% departure times, route will not be saved (stop flag). If some departure% %
% times are non-extendable, these options are removed with remove_empty. %
% Daphne Overbeek, Research Assignment TEL, 2016

elem = numel(probability_node); % Number of subroutes
nr_times = numel(Links{i,3}(:,1)); % Number of time blocks
Add_probability = cell(1,elem); % Make cell
for a = 1:elem % For each subroute, search for route extensions
    time = probability_node{1,a}(:,1); % Departure times of current link begin node
    Probability = cell(numel(time),1); % Make cell
    empty = 0; % Flag to check if program should continue
    % Find link data
    for k = 1:numel(time) % Search link data for each starting time
        Probability{k,1} = Links{i,3}{mod(time(k)-1,nr_times)+1,1}; % Module such that
        if isempty(Probability{k,1}) == 1 % Check for an empty cell
            empty = 1; % Flag for empty cell
        end
    end % Add link data to route data
    if empty == 0 % If empty == 1, program should stop
        probability_link = Probability; % Probability of the to be added link
        new_probability = add_link_to_node(probability_node{1,a},probability_link); % Make new
        start-to-link-end-node route: arrival times and probability
        if isempty(new_probability) == 0 % Check for empty output
            new_probability = clean_up_probability(new_probability); % Clean up the probability
            >> merge probabilities with same time
        end
        Add_probability{1,a} = [Add_probability{1,a}, new_probability]; % Store new subroute
        data in Add_probability
    end % Remove empty cells
    add_prob = 1; % Flag for function remove_empty_cells (handle only Add_probability)
    Add_time = [] ; % Input for function remove_empty_cells
    Add_route = [] ; % Input for function remove_empty_cells
    [Add_probability,Add_time,Add_route] = remove_empty_cells(Add_probability,Add_time,Add_route,add_prob); % Remove empty cells from
data in Add_probability
    stop = 0; % Flag to terminate loop if stop == 1
    if isempty(Add_probability) == 1 || Add_probability{1}(1,2) == 0 % Conditions for loop
        stop = 1;
    end
end
C.1.4 add_link_to_node.m

```matlab
function new_probability = add_link_to_node(probability_node,probability_link)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% add_link_to_node
% This function combines the data of the current route from start node %
% with the data of the to-be-added link. %
% Daphne Overbeek, Research Assignment TEL, 2016
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Data of route (Start to current link begin node)
elem = numel(probability_node(:,1)); % Number of elements in node probability
t1 = probability_node(:,1); % Departure times of link begin node
p1 = probability_node(:,2); % Probabilities of link begin node

%% Data of link
t = cell(1,elem); % Make cell
p = cell(1,elem); % Make cell
for i = 1:elem
    t{i,1} = probability_link{i,1}(:,1); % Save travel times of link
    p{i,1} = probability_link{i,1}(:,2); % Save probabilities of link
end

%% Calculate new route data (Start to current link end node)
new_t = []; % Make array
new_p = []; % Make array
for i = 1:elem % Calculate the route travel times and probabilities
    new_t = [new_t; t1(i)+t{i,1}]; % Times are added
    new_p = [new_p; p1(i)*p{i,1}]; % Probabilities are multiplied
end
index = find(new_p < 0.001); % Arrival times with probabilities <0.001 should be removed
for i = numel(index):-1:1 % Remove rows with probability <0.001
    new_p(index(i)) = []; % Clear row
    new_t(index(i)) = []; % Clear row
end
new_probability = [new_t new_p]; % Save in new_probability and give back
end
```

C.1.5 clean_up_probability.m

```matlab
function new_probability = clean_up_probability(Probability)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% clean_up_probability
% Simplify Probabilities if possible. If same times occur multiple times, %
% add their probabilities. %
% Daphne Overbeek, Research Assignment TEL, 2016
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

probability = sortrows(Probability,1); % Sort on travel times
new_p = 0;
new_probability = [];
if numel(probability(:,1)) < 2 % If only one element in Probability
    new_probability = Probability;
else % >1 elements in Probability
    for i = 1:numel(probability(:,1))-1 % If there are 2 same times
        if probability(i,1) == probability(i+1,1)
            new_p = new_p + probability(i,2); % Add probabilities
        else
            new_probability = [new_probability; probability(i,1) new_p+probability(i,2)]; % Update new_probability
            new_p = 0; % Set probability to zero
        end
    end
    new_probability = [new_probability; probability(numel(probability(:,1)),1) new_p+probability(numel(probability(:,1)),2)]; % Add last time and probability to new_probability
end
```
C.1.6 remove_empty_cells.m

function [Add_probability, Add_time, Add_route] = remove_empty_cells(Add_probability, Add_time, Add_route, add_prob)

% remove_empty_cells
% This function removes empty cells from Add_probability, Add_time and Add_route.
% Daphne Overbeek, Research Assignment TEL, 2016

% Find empty cells
elem = numel(Add_probability(1,:));
index = [];
for i = 1:elem
    if isempty(Add_probability{i}) == 0
        index = [index; i];
    end
end

% Initialise output
temp_probability = cell(1,numel(index));
if add_prob ~= 1
    temp_route = cell(1,numel(index));
    temp_time = cell(1,numel(index));
end

% Update output
teller = 1;
for i = 1:elem
    if ismember(i,index) == 1
        temp_probability{1,teller} = [temp_probability{1,teller} Add_probability{i}];
        if add_prob ~= 1
            temp_route{1,teller} = [temp_route{1,teller} Add_route{i}];
            temp_time{1,teller} = [temp_time{1,teller} Add_time{i}];
        end
        teller = teller + 1;
    end
end

Add_probability = temp_probability;
if add_prob ~= 1
    Add_time = temp_time;
else if add_prob == 1
    Add_time = [];
    Add_route = [];
end

C.1.7 expected_arrival_time.m

function [ Arrival_time_exp ] = expected_arrival_time(Add_probability)

% Calculate the expected arrival time based on the times and its probabilities.
% Daphne Overbeek, Research Assignment TEL, 2016

elem = numel(Add_probability);
Avival_time_exp = cell(1,elem);
for a = 1:elem
    subelem = numel(Add_probability{1,a}(:,1));
    for i = 1:subelem
        Arrival_time_exp{1,a} = Arrival_time_exp{1,a} + Add_probability{1,a}(i,1)*Add_probability{1,a}(i,2); % Calculate expected time: multiply time with probability
    end
end
C.1.8 add_route.m
function Add_route = add_route(old_route,Begin)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% add_route                                                              %%
% This function updates the Route column                                 %
%% Daphne Overbeek, Research Assignment TEL, 2016                        %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if isempty(old_route) == 1
    Add_route = cell(1,1);
    Add_route{1} = Begin;
else
    elem = numel(old_route);
    for a = 1:elem
        old_route{1,a} = [old_route{1,a}, Begin];
    end
    Add_route = old_route;
end

C.1.9 loop_checker.m
function [Add_route, Add_time, Add_probability] = loop_checker(eind,Add_route,Add_probability,Add_time)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% loop_checker                                                           %%
% This function checks if a loop is present in the current route. If so, %%
% the corresponding cell is made empty such that the route is discarded %
% later (in remove_empty_cells).                                        %%
%% Daphne Overbeek, Research Assignment TEL, 2016                        %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialisation
num_add = numel(Add_route); % Number of subroutes
new_route = cell(1,num_add); % Make cell
new_probability = cell(1,num_add); % Make cell
new_time = cell(1,num_add); % Make cell

% Check for loops
for i = 1:num_add
    if ismember(eind,Add_route{1,i}) == 1
        new_route{1,i} = [];
        new_probability{1,i} = [];
        new_time{1,i} = [];
    else
        new_route{1,i} = Add_route{1,i};
        new_probability{1,i} = Add_probability{1,i};
        new_time{1,i} = Add_time{1,i};
    end
end

% Update output
Add_route = new_route;
Add_probability = new_probability;
Add_time = new_time;
add_prob = 0; % Flag for remove_empty_cells
[Add_probability,Add_time,Add_route] = remove_empty_cells(Add_probability,Add_time,Add_route,add_prob); % Remove empty subroutes
end
C.1.10 is_equal.m

function [Add_route, Add_time, Add_probability] = is_equal(Nodes, new_route, new_time, new_probability, eind)

% This function determines which to-be-added subroutes are already in Nodes and should thus not be added. Only the not existing subroutes are added.
% Daphne Overbeek, Research Assignment TEL, 2016

subroutes = Nodes{eind,6};
% Existing subroutes in Nodes

elem = numel(subroutes{1,:});
% Number of subroutes in Nodes

elem_new = numel(new_route{1,:});
% Number of subroutes to be added

% Determine equal subroutes

do_not_add = [];
% Make array

for i = 1:elem
% For all subroutes in Nodes
    for j = 1:elem_new
% For all subroutes in Add_time
        if isequal(subroutes{1,i},new_route{1,j}) == 1
% If subroute is equal
            do_not_add = [do_not_add j];
% Save index in do_not_add
        end
    end
end

% Save unique subroutes

tel = 1;
% Teller

for i = 1:elem_new
% For all subroutes in Add_time
    if ismember(i,do_not_add) == 0
% If subroute is not in do_not_add
        temp_route{1,tel} = new_route{1,i};
% Save probability of subroute in temporary cell
        temp_prob{1,tel} = new_probability{1,i};
% Save expected arrival time in temporary cell
        temp_time{1,tel} = new_time{1,i};
% Save empty cell
        tel = tel + 1;
% Update teller
    elseif ismember(i,do_not_add) == 1
        temp_route{1,tel} = [];
% Save empty cell
        temp_prob{1,tel} = [];
% Save empty cell
        temp_time{1,tel} = [];
% Save empty cell
        tel = tel + 1;
% Update teller
    end
end
if isempty(do_not_add) == 1
% Add new subroutes
    Add_route = new_route;
    Add_time = new_time;
    Add_probability = new_probability;
else
% Add new subroutes
    Add_route = temp_route;
    Add_time = temp_time;
    Add_probability = temp_prob;
end

% Remove empty cells

add_prob = 0;
% Flag for function remove_empty_cells (handle all three "Add_" variables)

[Add_probability,Add_time,Add_route] = remove_empty_cells(Add_probability,Add_time,Add_route,add_prob);
end
C.1.11 compare_functions.m

function [Arrival_time, Route, Probability] = compare_functions(Nodes,eind)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% compare_functions                                                      
% This function plots all the probability functions and checks which one%%
% is/are non-dominated, others can be discarded.                         
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
arrival_time = 1; % Index in Nodes
new_arrival_time = 2; % Index in Nodes
probability = 3; % Index in Nodes
new_probability = 4; % Index in Nodes
route = 5; % Index in Nodes
new_route = 6; % Index in Nodes
draw = 0; % Plot routes: draw = 1.

%% Initialisation
temp_probability = Nodes{eind,new_probability}; % Save probabilities
num_probs = numel(temp_probability(1,:)); % Number of subroutes
if draw == 1;
    Legend_label = cell(1,num_probs); % Make cell
    colors = hsv(num_probs); % Make colors, same amount as there are subroutes
    colors_new = cell(num_probs,1); % Make cell
    for i = 1:num_probs
        if i < 10
            temp = sprintf('Function = 00%d', i);
        elseif i < 100
            temp = sprintf('Function = 0%d', i);
        else
            temp = sprintf('Function = %d', i);
        end
        Legend_label{i} = temp; % Fill the legend labels
        colors_new{i} = [colors(i,1) colors(i,2) colors(i,3)]; % Fill the color cell
    end
    legenda = [];
end
cumsum_prob = cell(1,num_probs); % Make cell
Probability = cell(1,num_probs); % Make cell

%% Find time range and plot
min_time = Inf; % Set minimal time to Inf
max_time = 0; % Set maximal time to 0
for x = 1:num_probs
    Time = temp_probability{1,x}{:,1}; % Find arrival times of current subroute
    min_t = min(Time); % Find minimal time of this subroute
    if min_t < min_time % Update overall minimal time to current
        min_time = min_t; % Save current minimal time
    end
    max_t = max(Time); % Find maximal time of this subroute
    if max_t > max_time % Change overall maximal time to current
        max_time = max_t; % Save current maximal time
    end
Probability{1,x} = temp_probability{1,x}{:,2}; % Save probabilities
if draw == 1
    cumpsum_prob = cumsum(temp_probability{1,x}{:,2}); % Make cumulative sum of
    % probabilities
    elem = numel(Time); % Number of elements in time
    Marker = 'o'; % Set marker
    Color = colors_new{x}; % Set color
    if elem == 1 % If [x 1], draw a vertical line at the arrival/travel time
        line([Time(1);Time(1)], [0;cumpsum_prob(1)], 'Marker', Marker, 'Color', Color);
    else
        plot(Time,cumpsum_prob,'Marker', Marker, 'Color', Color)
    end
hold on
    legends = [legenda; Legend_label{x}]; % Update legend
end
if draw == 1
    legend(legenda); % Set legend
end
%% Interpolate and extrapolate for time range found in all subroutes

\begin{verbatim}
% time_vector = [min_time:max_time];  
% Make time vector from overall minimal time to overall maximum time  
Data = cell(1,2);  
Data{1,1} = time_vector';  
% Save time vector in 1st cell of Data  
for i = 1:num_probs  
    x = temp_probability{1,i}(:,1);  
    y = Probability{1,i};  
    y_new = zeros(1,numel(time_vector));  
    index = 1;  
    for t = 1:numel(time_vector)  
        if time_vector(t) == x(index)  
            y_new(t) = y(index);  
            if index < numel(x)  
                index = index + 1;  
            end  
        end  
    end  
    Data{1,2}(:,i) = cumsum(y_new);  
    end  
end
\end{verbatim}

%% Compare all function pairs

\begin{verbatim}
% Number of unique function pairs (1-2, 1-3, 2-3 etc): (n^2-n)/2  
% Initialise as Inf matrix  
check_table = Inf(numel(time_vector),2*nr_pairs);  
% Make cell  
compare = cell(1,2*nr_pairs);  
% Make cell  
comp = 1;  
% Number of unique function pairs (1-2, 1-3, 2-3 etc): (n^2-n)/2  
% Tolerance for which data is assumed similar  
tolerance = 0.001;  
% Make cell  
if num_probs > 1  
    opt2 = 2;  
    for opt1 = 1:num_probs  
        for i = 1:numel(time_vector)  
            if abs(Data{1,2}(i,opt1)-Data{1,2}(i,opt2)) <= tolerance  
                check_table(i,comp) = 1;  
                check_table(i,comp+1) = 1;  
            elseif Data{1,2}(i,opt1) > Data{1,2}(i,opt2)  
                check_table(i,comp) = 1;  
            else  
                check_table(i,comp+1) = 1;  
            end  
        end  
    end  
    compare{1,comp} = [opt1 opt2];  
    compare{1,comp+1} = [opt2 opt1];  
    comp = comp+2;  
end
\end{verbatim}

%% Determine best and to be removed routes

\begin{verbatim}
% Make array  
% For every unique pair, twice per pair  
best = [];  
remove = [];  
for i = 1:2:nr_pairs*2  
    counts_1 = nnz(check_table(:,i) == 1);  
    counts_2 = nnz(check_table(:,i+1) == 1);  
    if counts_1 == numel(time_vector) & counts_2 == numel(time_vector)  
        best = [best compare(1,i)(1,1)];  
        if ismember(compare(1,i)(1,2),remove) == 0  
            remove = [remove compare(1,i)(1,2)];  
        end  
    elseif counts_2 == numel(time_vector) & counts_1 == numel(time_vector)  
        best = [best compare(1,i)(1,2)];  
        if ismember(compare(1,i)(1,1),remove) == 0  
            remove = [remove compare(1,i)(1,1)];  
        end  
end
\end{verbatim}
else
    best = [best 0];
end

best_routes = [];
for i = 1:num_probs
    if ismember(i,remove) == 0
        best_routes = [best_routes i];
    end
end

remove = sort(remove);
best_routes = sort(best_routes);

%% Remove dominated routes
if numel(remove) >= 1
    [Arrival_time, Route, Probability] = remove_functions(remove, Nodes{eind,new_arrival_time}, Nodes{eind,new_route}, Nodes{eind,new_probability});
else
    Arrival_time = Nodes{eind,new_arrival_time};
    Route = Nodes{eind,new_route};
    Probability = Nodes{eind,new_probability};
end

if draw == 1
    close Figure 1
end
end

C.1.12 remove_functions.m

function [Arrival_time, Route, Probability] = remove_functions(remove, old_arrival_time, old_route, old_probability)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% remove_functions
% This function removes routes that are marked as "remove" in %
% \[compare_functions\] %
% Daphne Overbeek, Research Assignment TEL, 2016 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Initialization
elem = numel(old_arrival_time);
for i = 1:elem
    if ismember(i,remove) == 1
        old_arrival_time{1,i} = []; % Make expected arrival time of route an empty cell
        old_route{1,i} = []; % Make route an empty cell
        old_probability{1,i} = []; % Make link data of route an empty cell
    end
end
Arrival_time = cell(1,1); % Make cell
Route = cell(1,1); % Make cell
Probability = cell(1,1); % Make cell

%% Update routes
teller = 1;
for i = 1:elem
    if isempty(old_arrival_time{1,i}) == 0
        Arrival_time{1,teller} = old_arrival_time{1,i}; % Add expected arrival time of route
        Route{1,teller} = old_route{1,i}; % Add route
        Probability{1,teller} = old_probability{1,i}; % Add link data of route
        teller = teller + 1; % Update route index
    end
end
function Routes = add_route_end_nodes(Routes, Finish)

% This function adds the finish node to the found route(s) to show the complete route from start to finish.
% Daphne Overbeek, Research Assignment TEL, 2016

if isempty(Routes) == 0
    elem = numel(Routes(1,:));
    for i = 1:elem
        Routes(i) = [Routes(i) Finish];
    end
end
C.2 Data generation scripts

C.2.1 RUN_make_grid.m

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% RUN_make_grid                                                        %%%
%%% Generates grids and link data for all kinds of properties. A one     %%%
%%% directional grid is copied to a bidirectional grid.                 %%%
%%% Each type of network is saved in a .mat file.                       %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear
all
close
clc

for network = [3,4,5,6,10,15,20,25]
    length = network;
    width = length;
    for blocksize = [2,4,6,10,12]
        for c_rho = [-11,-1,0,1,11,5]
            for one_direction = 1:2
                if one_direction == 1
                    rho = [0.8 1];
                elseif c_rho == 1
                    rho = [0.2 0.5];
                elseif c_rho == 0
                    rho = [-0.00000001 0.00000001];
                elseif c_rho == -1
                    rho = [-0.5 -0.2];
                elseif c_rho == -11
                    rho = [-1 -0.75];
                elseif c_rho == 5
                    rho = [ ];
                end
                % Determine correlation ranges
                % Generate links in the network
                [Links,nr_links,nr_nodes] = make_grid(length,width);
                % Generate link data
                [Links,nr_times,p,definitive] =
                generate_link_data(Links,nr_links,blocksize,rho,c_rho);
                % Copy links of one directional grid to make bidirectional grid
                elseif one_direction == 2
                    begin_node = Links(:,1);
                    begin_node = cell2mat(begin_node);
                    end_node = Links(:,2);
                    end_node = cell2mat(end_node);
                    for i = 1:nr_links
                        Links(nr_links+i,1) = end_node(i);
                    end
                    Links(nr_links+i,2) = begin_node(i);
                    Links(i+nr_links,3) = Links(i,3);
                    p(i+nr_links,:) = p(i,:);
                    end
                    nr_links = 2*nr_links;
                end
                Links{1,4} = p;
            end
        end
    end
end
```

This script generates grids and link data for various types of properties. It copies one-directional grids to bidirectional grids and saves each type of network in a .mat file. The script iterates through different network sizes, block sizes, and correlation values, generating links and link data accordingly.
elseif c_rho == -1
    text = sprintf('Links%d_%d_%d_t-
    ',length,blocksize,one_direction);
delseif c_rho == -11
    text = sprintf('Links%d_%d_%d_t--
    ',length,blocksize,one_direction);
elseif c_rho == 5
    text = sprintf('Links%d_%d_%d_t
    ',length,blocksize,one_direction);
end
save(text);
end
end
end

C.2.2 make_grid.m

function [Links,nr_links,nr_nodes] = make_grid(m, n )
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% make_grid                                                              
% This function created a grid with size mxn (length x width).         
% % Daphne Overbeek, Research Assignment TEL, 2016                    
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

nr_nodes = m*n; % Number of nodes
nr_links_ver = (m-1)*n; % Number of vertical links
nr_links_hor = (n-1)*m; % Number of horizontal links
nr_links = nr_links_hor+nr_links_ver; % Total number of links
Links = cell(1,3); % Make cell

minus = 0; % Index
for l = 1:nr_nodes % For all nodes
    x = l/n;
    if ceil(x) > floor(x) % Links are on same row of grid
        Links{1,minus,1} = l; % Add link begin node
        Links{1,minus,2} = l+1; % Add link end node
    else
        minus = minus+1; % New row of grid
    end
end

num = numel(Links(:,1)); % Number of links so far
num = num + 1; % Add 1 for index
for l = 1:nr_nodes % For rest of nodes
    Links{num,1} = l; % Add link begin node
    Links{num,2} = l+n; % Add link end node
    num = num+1; % Update index
end

C.2.3 generate_link_data.m

function [Links,nr_times,p,backup] = generate_link_data(Links,nr_links,blocksize,rho,c_rho)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% generate_link_data                                                     
% This function generates link data for all links in the network.        
% Optional: taking a certain correlation range into account.             
% % Daphne Overbeek, Research Assignment TEL, 2016                     
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Properties
min_intensity = 0.1*blocksize; % Specify lower boundary
max_intensity = 0.3*blocksize; % Specify upper boundary
intensity_all = (max_intensity-min_intensity)*rand(1,nr_links)+min_intensity; % Determine link intensities

min_deviation = 1*blocksize; % Specify lower boundary
max_deviation = 2.5*blocksize; % Specify upper boundary
deviation = (max_deviation-min_deviation)*rand(nr_links,1)+min_deviation; % Determine link deviations

deviation_all(:,1) = deviation(:,1); % Deviation of first peak
deviation_all(:,2) = deviation(:,1); % Deviation of second peak
deviation_all(:,3) = deviation(:,1)-0.5; % Deviation of third peak

79
%% Generate link data
[nr_times,probabilities,p,backup] = link_generator(nr_links,intensity_all,deviation_all,blocksize); % Make link probabilities

%% Check for correlation
if c_rho ~= 5
  TT = p';
  nr_links = numel(Links(:,1));
  % Link travel times
  % Number of links
  ext = [];
  % Find adjacent link pairs
  for i = 1:nr_links
    for j = 1:nr_links
      if Links{i,2} == Links{j,1} && Links{i,1} ~= Links{j,2} % Adjacent link pair condition
        ext = [ext; i j];
        % Save link indices
      end
    end
  end

  change = 1;
  % Flag for while loop
  tel = 0;
  % Flag
  while change == 1
    % Iterate until all adjacent link pairs are in correlation range
    change = 0;
    % Update flag
    for i = 1:numel(ext(:,1))
      % For all adjacent link pairs
      link_a = ext(i,1);
      % First link
      link_b = ext(i,2);
      % Second link
      cor(i) = corr(TT(:,link_a),TT(:,link_b)); % Correlation between link a and link b
      count = 0;
      % Flag
      cor_temp = Inf;
      % Initialise temporary correlation
      check = 0;
      % Flag
      while (cor(i) < rho(1,1) || cor(i) > rho(1,2)) && check == 0
        % Iterate until correlation is within range or termination
        if c_rho == 11
          [nr_times,probabilities,p,backup] = link_generator_cor_p2(intensity_all,deviation_all,blocksize,link_b,probabilities,p,backup);
        elseif c_rho == 1
          [nr_times,probabilities,p,backup] = link_generator_cor_p1(intensity_all,deviation_all,blocksize,link_b,probabilities,p,backup);
        elseif c_rho == 0
          [nr_times,probabilities,p,backup] = link_generator_cor_0(blocksize,link_b,probabilities,p,backup);
        elseif c_rho == -1
          [nr_times,probabilities,p,backup] = link_generator_cor_m1(intensity_all,deviation_all,blocksize,link_b,probabilities,p,backup);
        elseif c_rho == -11
          [nr_times,probabilities,p,backup] = link_generator_cor_m2(intensity_all,deviation_all,blocksize,link_b,probabilities,p,backup);
        end
        change = 1;
        % Update flag
        change = 1;
        % Update flag
        TT = p';
        % Update link travel times
        cor(i) = corr(TT(:,link_a),TT(:,link_b)); % Calculate correlation
        if abs(cor(i)-rho(1,1)) < abs(cor_temp-rho(1,2)) || abs(cor(i)-rho(1,1)) < abs(cor_temp-rho(1,1)) % If correlation not within range, save the one that is closest to range
          cor_temp = cor(i);
          p_temp = p;
          prob_temp = probabilities;
          def_temp = backup;
        end
        if isnan(cor(i))
          cor(i) = 0;
        end
      end
      if count > 15
        % Terminate generation of new link_b
        check = 1;
      end
      if abs(cor(i)-rho(1,1)) < abs(cor_temp-rho(1,2)) || abs(cor(i)-rho(1,1)) < abs(cor_temp-rho(1,1)) % If correlation not within range, save the one that is closest to range
        cor_temp = cor(i);
        p_temp = p;
        prob_temp = probabilities;
        def_temp = backup;
      end
      if tel > 8
        % Terminate 1st while loop
C.2.4 link_generator.m

Link_generator per correlation scenario are not presented.

function [nr_times,probabilities,p,backup] =
link_generator(nr_links,intensity_all,deviation_all,blocksize)
% This function generates link data (travel times and probabilities) %
% using a normal distribution, using traffic peaks, intensities and %
% deviations. %
% Daphne Overbeek, Research Assignment TEL, 2016 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

draw = 0; % Plot function is activated (1) or not activated (0)
_nr_times = 24*blocksize; % Number of blocks in a day
limit = 0.005/blocksize; % Min probability
min_mean_1 = 7; % Lower bound of first peak
max_mean_1 = 9; % Upper bound of first peak
min_mean_2 = 9; % Lower bound of second peak
max_mean_2 = 13; % Upper bound of second peak
min_mean_3 = 15; % Lower bound of third peak
max_mean_3 = 19; % Upper bound of third peak

for m = 1:nr_links % For all links
    mean_1 = (max_mean_1-
    min_mean_1)*rand+min_mean_1; % Determine mean1, mean of peak
    mean_2 = (max_mean_2-
    min_mean_2)*rand+min_mean_2; % Determine mean2
    mean_3 = (max_mean_3-
    min_mean_3)*rand+min_mean_3; % Determine mean3
    mean = [mean_1*blocksize,mean_2*blocksize,mean_3*blocksize]; % Save mean
    min_dev = 0.2*blocksize; % Lower bound of deviation used to create probabilities
    max_dev = 1*blocksize; % Upper bound
    dev = (max_dev-min_dev)*rand+min_dev; % Determine deviation
    for t = 1:nr_times % For all time blocks
        probabilities{m,t} = make_probs(minimum,p(m,t),dev,limit); % Make travel times and probabilities [function]
        max_tt(t) = max(probabilities{m,t}(:,1)); % Max travel time on a link and time
    end
end

end
%% Plot data
if draw == 1
    max_ttt = max(max_tt);
    bar_data = zeros(max_ttt,nr_times);
    for t = 1:nr_times
        for y = 1:numel(probabilities{m,t}{(:,1)})
            bar_data(probabilities{m,t}(y,1),t) = probabilities{m,t}(y,2);
        end
    end
    figure
    bar3(bar_data)
    xlabel('Starting block (block = 30 min)');
    ylabel('Link travel time [blocks]');
    zlabel('Probability [%/100]');
end

%% Backup properties
backup(m,1) = mean(1,1);
backup(m,2) = mean(1,2);
backup(m,3) = mean(1,3);
backup(m,4) = minimum_2;
backup(m,5) = intensity;
backup(m,6) = deviation(1,1);
backup(m,7) = deviation(1,2);
backup(m,8) = deviation(1,3);
backup(m,9) = spits_intensity;
backup(m,10) = minimum;
backup(m,11) = dev;
end

C.2.5 make_probs.m

function probabilities = make_probs(minimum,mean,deviation,limit)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% make_probs
% This function generates probabilities of the links using a normal distribution. No probabilities < 0.05 are accepted. To ensure a sum probability of 1, the elements are normalized.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
probabilities = [];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generate left side of normal distribution
for i = round(minimum):round(mean)
    x = normpdf(i,mean,deviation)^2;
    if x > limit
        chance = x;
        tijd = i;
        probabilities = [probabilities; tijd chance];
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generate right side of normal distribution
i = round(mean)+1;
while x > limit
    x = normpdf(i,mean,deviation)^2;
    if x > limit
        chance = x;
        tijd = i;
        probabilities = [probabilities; tijd chance];
    end
    i = i+1;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Rearrange probabilities
sum_link = sum(probabilities(:,2));
probabilities(:,2) = probabilities(:,2)/sum_link;
end