LinTim

An integrated environment for mathematical public transport optimization

Documentation

Version 2024.12

Currently developed at

Department of Mathematics and Systems Analysis
Aalto University, Espoo

and

Optimization Research Group
RPTU Kaiserslautern-Landau

and

Fraunhofer ITWM
Kaiserslautern

Originally developed at

Institute for Numerical and Applied Mathematics
University of Göttingen

Contributors

Head

- Philine Schiewe (current head)
- Anita Schöbel (former head and founder)

Technical Lead

- Sven Jäger

Researchers

- Sebastian Albert
- Christine Biedinger
- Thorsten Dahlheimer
- Vera Grafe
- Olli Herrala
- Klara Hoffmann
- Sarah Roth
- Alexander Schiewe
- Moritz Stinzendörfer
- Reena Urban

Student Assistants

- Ricardo Reicherz

Former Researchers

- Urs Baumgart
- Rasmus Fuhse
- Konstantinos Gkoumas
- Marc Goerigk
- Jonas Harbering
- Jonas Ide
- Julius Pätzold
- Michael Schachtebeck
- Jochen Schulz
- Michael Siebert
- Felix Spühler
- Anke Uffmann

Former Student Assistants

- Florentin Hildebrandt
- Jonas Hürter
- Jarkko Jalovaara
- David Kaiser
- Eero Ketola
- Benjamin Lieser
- Kim Reece
- Michael Rihlmann
- Leevi Rönty
- Mridul Roy
- Lisa Sandig
- Christopher Scholl
- Linda Sieber
- Vitali Telezki

Contents

Chapter 1

Introduction

1.1 What is LinTim?

LinTim is an academic algorithm and dataset library for mathematical public transport optimization. Problems in public transport optimization range from finding suitable locations for stations over calculating passenger-friendly timetables to handling unexpected delays. As it would be too complicated (though best in theory) to handle all these problems at the same time, they are split up and solved sequentially.
However, what seems to be best for one particular problem may have devastating influence on a different problem: For example a good timetable might not be well suited for delay management. **LinTim** (standing for **Line**planning and **Tim**etabling) addresses this issue by integrating the various public transport optimization problems and algorithms into one single environment. It hence gives the possibility to go back and forth in the sequence of public transport optimization problems in order to find solutions that work well on a greater scope and not only for the respective problem.
The data files are based on simple plain text formats that allow the implementation of algorithms in whatever programming language the developer likes to use. Thus, it is made easy to extend the current LinTim-library and keep up to date with new developments and ideas.
LinTim is designed for the use in UNIX, and will not work flawlessly in a native Windows environment.

Throughout the documentation, we will use some markers to indicate what certain `teletyped` texts mean:

- [Fo] foldername (relative paths w.r.t. the current dataset),
- [Fi] filename (relative paths w.r.t. the current dataset),
- [R] command that can run in some shell,
- [C] config entry with key and value,
- [CK] config key,
- [CV] config value,
- [S] statistic entry with key and value,
- [SK] statistic key,
- [SV] statistic value.
- [CK]([Fi]) a config key for a filename, followed by the default value

1.2 Installation and Requirements

LinTim uses many different programming languages. For the most parts, it is enough to have Java (≥ 11 with ant ≥ 1.9.8 and maven ≥ 4), C, C++ and Python3 (≥ 3.5) installed on your system. There may be some special algorithms requiring additional programming languages, but if this is the case this is noted in the respective section of the documentation.

Using Windows 10 The easiest way to run LinTim under Windows 10 is using a WSL installation. For installation instructions, see `https://docs.microsoft.com/en-us/windows/wsl/install-win10`. Using the WSL you can follow the installation notes listed for Linux below.

Using macOS Although macOS is a Unix-based operating systems, some of the below mentioned installation commands need to be adjusted when using macOS. The most important difference is the unavailability of apt-get for package management. Please check the different installed packages for the best way to install on macOS but for most of them, there are easy installation procedures using Homebrew, see `https://brew.sh/`. With that, see the installation notes for Linux below for more information.

Using a Linux distribution In this section, we list the commands to install all dependencies available in the Debian GNU/Linux Package index using apt-get. If you use another package manager, you need to adapt the corresponding commands.
To install all package manager dependencies of LinTim, run

R `sudo apt-get install build-essential openjdk-11-jdk ant graphviz python3-pip`

To install the python package dependencies using pip, run

R `sudo pip3 install numpy networkx pulp holoviews weightedstats pandas matplotlib seaborn`

Also for using all of LinTim, you will have to fulfill other third-party dependencies. For more information, have a look at Fi `/libs/README.md`. For a list of supported integer programming solvers and how to connect them with LinTim, see the next section.

1.2.1 Connecting LinTim with a solver

Some programs make use of integer programming solvers like Xpress, CPLEX and Gurobi. However, for each of the planning stages line planning, timetabling and vehicle scheduling there are also algorithms working without a solver installed. See the instructions of the respective algorithms for configuring LinTim to use your chosen solver and Chapter 6 for a general overview which methods support which solver. If you want to use an integer programming solver, make sure to install it using the corresponding documentation and to set the environment variables correctly. In the following, we give a short overview which environment variables need to be set for LinTim to find the corresponding solver. We suggest adding the below code snippets to your /.bashrc-file (or your equivalent, depending on your used environment), for automatic environment variable setting.

Gurobi For Gurobi, the CLASSPATH and LD_LIBRARY_PATH variables need to be set. On your machine, this might mean to run

R `export GUROBI_HOME=/opt/gurobi/linux64`

R `export CLASSPATH=${GUROBI_HOME}/lib/gurobi.jar:${CLASSPATH}`

R `export LD_LIBRARY_PATH=${GUROBI_HOME}/lib/:${LD_LIBRARY_PATH}`

For more information, check the Gurobi documentation.

Xpress For Xpress, source the `xpvars.sh` script provided with the installation. On your machine, this might mean to run

```
R source /opt/xpressmp/bin/xpvars.sh
```

This will take care of setting the appropriate environment variables for Xpress. For more information, check the Xpress documentation.

CPLEX For CPLEX, the PATH, CLASSPATH and LD_LIBRARY_PATH variables need to be set. On your machine, this might mean to run

```
R export CPLEX_HOME=/opt/ibm/ILOG/CPLEX_Studio201/cplex
R export CLASSPATH=${CPLEX_HOME}/lib/cplex.jar:${CLASSPATH}
R export LD_LIBRARY_PATH=${CPLEX_HOME}/bin/x86-64_linux/:${LD_LIBRARY_PATH}
R export PATH=${CPLEX_HOME}/bin/x86-64_linux/:${PATH}
```

Additionally, make sure to run the python installation script provided with the CPLEX installation to install the CPLEX python package. On your machine, this might mean to run

```
R sudo python3 /opt/ibm/ILOG/CPLEX_Studio201/python/setup.py install
```

For more information, check the CPLEX documentation.

SCIP For SCIP, the PATH and LD_LIBRARY_PATH variables need to be set. On your machine, this might mean to run

```
R export SCIPOPTDIR=/opt/scipoptsuite-7.0.2
R export LD_LIBRARY_PATH=${SCIPOPTDIR}/build/lib/:${LD_LIBRARY_PATH}
R export PATH=${SCIPOPTDIR}/build/bin/:${PATH}
```

If you want to use SCIP from a Java program, make sure to install JSCIPOpt as well, see https://github.com/scipopt/JSCIPOpt. After installing, extend the above environment variables with

```
R export JSCIPOPTDIR=/opt/scipoptsuite-7.0.2
R export LD_LIBRARY_PATH=${JSCIPOPTDIR}/build/Release:${LD_LIBRARY_PATH}
R export CLASSPATH=${JSCIPOPTDIR}/build/Release/scip.jar:${CLASSPATH}
```

For more information, check the SCIP and JSCIPOpt documentation.

GLPK To use GLPK as a solver in LinTim, only the binary `glpsol` needs to be in the PATH. You can install GLPK e.g. with

```
R sudo apt-get install glpk-utils
```

COIN and CBC The coin and cbc solver are both bundled with the PuLP python package. Therefore you don't need to install anything additionally here.

1.3 Installation Script

The installation script is a Python script which leads you through the most parts of the installation of LinTim. By following the instructions of the script you install the required system dependencies, LinTim, the LinTim dependencies, Gurobi and the Python dependencies. If you want to use the installation script you have to start it from the shell by running

R `python3 install.py`

after downloading Fi `install.py` and Fi `util.py`. If you already downloaded LinTim you can find the installation file in Fo `src/installation`. The installation script attempts to install the required python packages via pip. It should therefore be executed within the desired (virtual) environment. Note that certain installations require sudo access where you will be prompted for your password.

1.4 Typical Usage: A Hands-On-Example

In the following we describe the typical usage of LinTim and give an overview over the structure of the repository.
Its root directory consists of the following:

- Fo `/ci`
  Folder for continuous integration tests.
- Fo `/datasets`
  The LinTim instances and their customized configuration files.
- Fo `/doc`
  All documents regarding the LinTim project (e.g. this documentation).
- Fo `/libs`
  A folder to place dependencies. If necessary, the dependency will be described in the corresponding algorithm section.
- Fo `/src`
  The source code of the LinTim algorithms.

In Fo `/datasets` you can see all the datasets which are implemented in LinTim for the time being. For further information on these datasets see Chapter 9, including information on how to add your own datasets to LinTim.

Our goal in this example will be to calculate a disposition timetable for the "toy"-dataset and describe several of the in- and output files that you can find during the process. Note that in general, LinTim provides the capability to configure all file paths. For simplicity, we will only provide the default values for this config keys in this chapter. For more information, see the following chapters.

Change into the folder
Fo `/datasets/toy`
in order to run algorithms on the "toy"-dataset. You find an exemplary folder-structure of a dataset folder:

- Fo `basis`
  Contains all the data describing the instance like OD matrix, edges, loads, line pool, headways, etc.
- Fo `delay-management`
  Will contains all the data related to delay-management and aperiodic planning.
- Fo `graphics`
  Will contain all graphical output of the algorithms you might use.

- Fo `line-planning`
  Will contains all the data related to line planning.
- Fo `statistic`
  Will contain all output of evaluations you might run (may not exist yet, will be created automatically on evaluation).
- Fo `timetabling`
  Will contain all data related to periodic timetabling.
- Fo `vehicle-scheduling`
  Will contain all data related to vehicle scheduling.

As you can see, the folder names (and thus the contents) are related to the different steps of mathematical public transport optimization.

Every output you produce will by default be written into the respective folders.

This means, if you somehow produce an output regarding e.g. the delay-management, it will be written to Fo `delay-management`.

Also, each dataset folder contains a Makefile.

LinTim algorithms are used by calling make.

For instance typing

R `make lc-line-concept`

while being located in the "toy"-folder will compile all necessary files, calculate a line concept for the "toy"-instance and write it into Fi `line-planning/Line-concept.lin`.
Note that by default, this will use Xpress as an integer optimization solver. Therefore to successfully run this step, Xpress needs to be installed. See Chapter 1.2 for more information.

For calculating a line concept, LinTim uses the data given in Fo `basis`.
Having a look into the makefile the line

```
line-concept:
${SRC_DIR}/line-planning/line-planning.sh ${FILENAME_CONFIG}
```

tells us, that the line concept is calculated using the algorithms from
Fo `/src/line-planning` with the configurations given in
Fi `${FILENAME_CONFIG}`, which is Fi `basis/Config.cnf` by default.
For detailed instructions on configuration files and how to change them see Section 8.1.
If you want wo use different algorithms see Chapter 2 to know which are already implemented, Chapter 3 for detailed information on the implemented algorithms and Chapter 11 for instructions on how to implement your own into LinTim.

So let's have a look at what we got from our call

R `make lc-line-concept`

The file Fi `line-planning/Line-concept.lin` should contain something like this:

```
# line-id; edge-order; edge-id; frequency
1;1;1;0
1;2;6;0
1;3;7;0
2;1;2;3
2;2;6;3
...
```

LinTim usually works with text files structured similarly (# comments a line). The advantage of this concept is that they are very independent of the programming language used.
In the most text files, like in this example, an explanation will be given on how to read them.

So now we got ourselves a first line concept for our “toy”-example. Next thing to do would be calculating a feasible timetable. For this we first have to provide an Event-Activity-Network (EAN). We can make LinTim calculate this by calling

R `make ean`

Note that in order to calculate this EAN LinTim of course needs a public transportation network (PTN), given by the network itself and a line concept on this network.
Of course it would be possible to design the algorithms in a way that a call of

R `make ean`

automatically generates a line concept if none is existent so far but for different reasons we refrained from this.
Therefore before calling

R `make ean`

you will always have to provide a line concept. Calling it before calculating a line concept will result in an error.
By calling

R `make ean`

we calculated the events and activities of our EAN. These are written to
Fi `timetabling/Activities-periodic.giv` and Fi `timetabling/Events-periodic.giv`.
For instance Fi `timetabling/Events-periodic.giv` should look something like this:

```
# event_id; type; stop-id; line-id; passengers; line-direction;
    line-freq-repetition
1; "departure"; 1; 1; 20; >; 1
2; "arrival"; 3; 1; 20; >; 1
3; "departure"; 3; 1; 20; >; 1
...
```

The first line again tells us how to read the file, i.e. e.g. event 2 is an arrival of line 1 at stop 3 carrying 20 passengers.

In order to calculate a timetable from this data we just call

R `make tim-timetable`

and LinTim will write a timetable to Fo `timetabling/timetable-periodic.tim` in which you can look up the event given by its index and the time it is scheduled to take place.
Given this timetable we can now concentrate on the delay-management or the vehicle-scheduling.

We will try out the DM step first. This is a little bit more complex because there are some prerequisites we have to provide.
First of all we need an aperiodic timetable since the DM-algorithms only work for these.
But we do not really need a new aperiodic timetable. We just need our periodic timetable expanded that is we have to adhere the periods.
For LinTim we call this "Rollout" and calculate it by calling

R `make ro-rollout`

The needed "aperiodic" timetable will be written to
Fi `delay-management/Timetable-expanded.tim` and will also be included in
Fi `delay-management/Events-expanded.tim`.
After calculating this timetable we can create some delays by calling

R `make dm-delays`

This will call the delay-generator which generates source delays for our given network. More on how the delay-generator works and how to control it can be found in Section 4.10.
After creating some delays we finally want to calculate a disposition timetable and do that by calling

R `make dm-disposition-timetable`

The timetable will be calculated and written to
Fi `delay-management/Timetable-disposition.tim`.

For concluding our first LinTim-cycle we now want to calculate a vehicle scheduling.
For this we first have to consider, that all the trips that have to be completed by some vehicle have to be known. In a periodic timetable this might not be the case. Because of this we have to rollout the whole trips and we can do so by setting C `rollout_whole_trips` to true. Changing a config-parameter is done in the following way:
Change to

Fo `basis`

and write

```
rollout_whole_trips; true
```

into Fi `basis/Private-Config.cnf`.
Now for calculating the vehicle-schedules we first have to repeat the steps from and including

R `make ro-rollout`

We then have to calculate the trips, the vehicles have to do. We can do so by typing

R `make ro-trips`

and the trips will be written to Fi `delay-management/Trips.giv`.
Now calling

R `make vs-vehicle-schedules`

calculates the vehicle schedule and it is written to
Fi `vehicle-scheduling/Vehicle_Schedules.vs`.

In the end, we want to evaluate the created vehicle schedule. By running

R `make vs-vehicle-schedules-evaluate`

we evaluate the current vehicle schedule and the computed properties will be written to Fi `statistic/statistic.sta`, e.g. SK `vs_cost`, the cost of the vehicle schedule and SK `vs_feasible`, whether the computed schedule is feasible.

Beside this few make-targets we introduced there are a lot more in LinTim. Have a look into the makefiles to see which possible targets exist. Which algorithm will be called exactly is defined by the configuration file. For a description of which parameter setting will call which algorithm, see Chapter 2.

Chapter 2

Overview on the Planning Steps

The different public transport optimization problems can be summarized in the following figure:

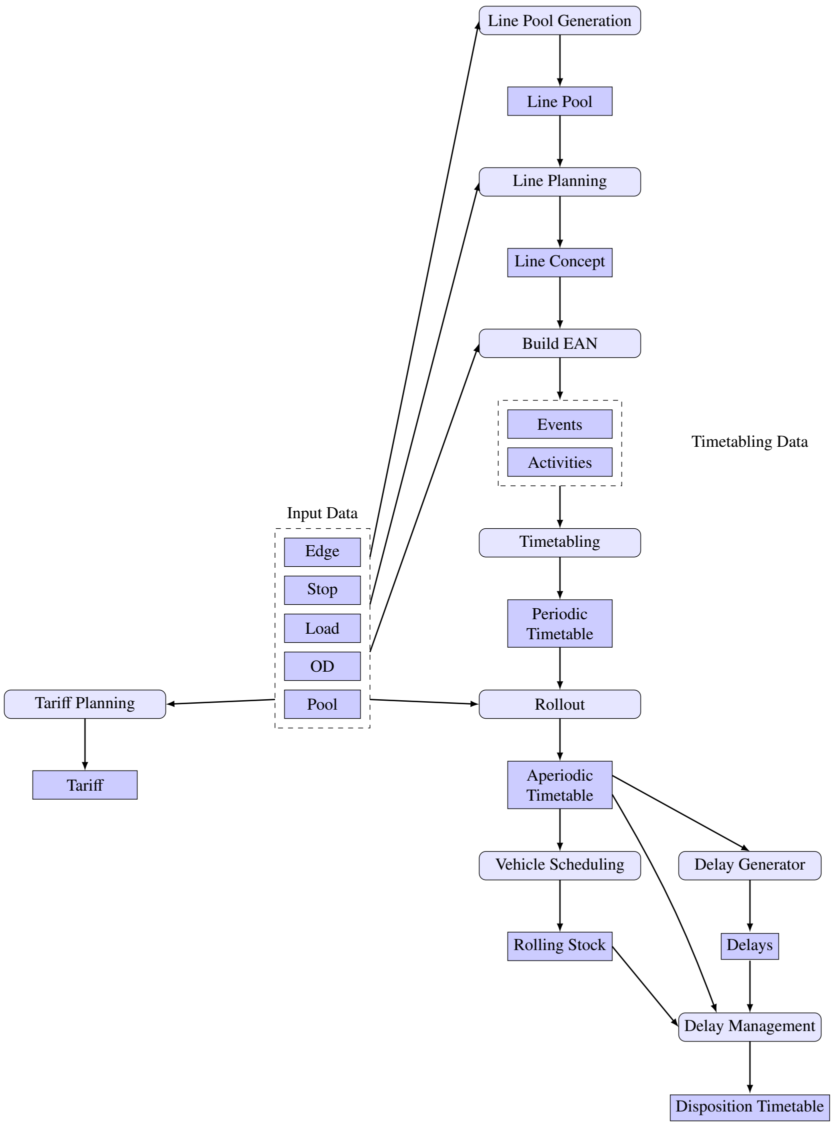


Figure 2.1: Different planning steps considered in LinTim

2.1 Stop Location

In the the stop location step a new PTN is computed according to a given demand and a given infrastructure of stations and tracks.

2.1.1 Input

The following files are needed as input for the classical stop location problems:

- CK `default_existing_stop_file` (Fi `basis/Existing-Stop.giv`) stops of the existing infrastructure network
- CK `default_existing_edge_file` (Fi `basis/Existing-Edge.giv`) edges of the existing infrastructure network
- CK `default_demand_file` (Fi `basis/Demand.giv`) demand at geographical positions

Additionally, there are models for a given infrastructure network. For this, the following files are needed as input:

- CK `filename_node_file` (Fi `basis/Node.giv`) the nodes of the network, including possible stops
- CK `filename_infrastructure_edge_file` (Fi `basis/Edge-Infrastructure.giv`) direct connections between the nodes suitable for public transport
- CK `filename_walking_edge_file` (Fi `basis/Edge-Walking.giv`) possible walking edges between infrastructure nodes
- CK `filename_od_nodes_file` (Fi `basis/OD-Node.giv`) od data based on infrastructure nodes

2.1.2 Output

The following files are produced as output.

- CK `default_stops_file` (Fi `basis/Stop.giv`) stops of the new PTN
- CK `default_edges_file` (Fi `basis/Edge.giv`) edges of the new PTN

2.1.3 Algorithms

Running

R `make sl-stop-location`

will create a new PTN with respect to the given demand points. The following algorithms are available:

- CK `sl_model` CV `dsl` finds an optimal solution for the stop location problem with fixed travel times on PTN edges.
- CK `sl_model` CV `greedy` finds a feasible solution for the stop location problem with fixed travel times on PTN edges with a greedy approach.
- CK `sl_model` CV `dsl-tt` solves CV `dsl` while considering the travel time, including acceleration and deceleration.
- CK `sl_model` CV `dsl-tt-2` solves CV `dsl` while considering the travel time, including acceleration and deceleration.
- CK `sl_model` CV `tt` finds a travel time optimal solution for a given infrastructure network with walking times for the passengers
- CK `sl_model` CV `all` adds every possible stop in a given infrastructure network to the new PTN.

2.2 Line Pool Generation

In the line pool generation step a possible set of lines is computed to use during the line planning step.

2.2.1 Preparation

Run

R make ptn-regenerate-load

to compute a new load.

2.2.2 Input

The following files are needed as input:

- CK default_stops_file (Fi basis/Stop.giv) stops of the PTN
- CK default_edges_file (Fi basis/Edge.giv) edges of the PTN
- CK default_loads_file (Fi basis/Load.giv) expected distribution of passengers to PTN edges (depending on CK lpool_model)
- CK default_od_file (Fi basis/OD.giv) OD matrix (depending on CK lpool_model)

2.2.3 Output

The following files are produced as output.

- CK default_pool_file (Fi basis/Pool.giv) line pool, set of possible lines
- CK default_pool_cost_file (Fi basis/Pool-Cost.giv) costs of lines in line pool

2.2.4 Algorithms

To compute a line pool run

R make lpool-line-pool

The following algorithms are available:

- CK lpool_model CV tree_based a heuristic based on MST which computes a line pool that at least allows for a feasible line concept for a given load (see 3.2.1)
- CK lpool_model CV restricted_line_duration same as CV tree_based but with additional constraints on the duration of a line (see 3.2.2)
- CK lpool_model CV k_shortest_paths a heuristic which computes the *k* shortest path for all OD pairs as line pool (see 3.2.3)
- CK lpool_model CV terminal-to-terminal enumerates the complete line pool, starting and ending each line at a terminal (see 3.2.4).
- CK lpool_model CV center-periphery identifies some nodes as centers and some nodes as periphery and then constructs lines between these different pairs of nodes (see 3.2.5).

2.3 Line Planning

In the line planning step a feasible line concept is determined by assigning frequencies to all lines in the line pool.

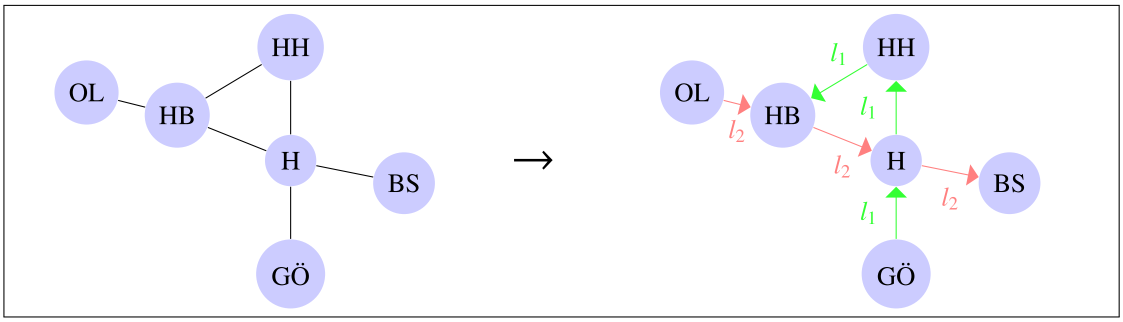


Figure 2.2: Line Planning Step

2.3.1 Preparation

Run

R `make ptn-regenerate-load`

to compute a new load.

2.3.2 Input

The following files are needed as input:

- CK `default_stops_file` (Fi `basis/Stop.giv`) stops of the PTN
- CK `default_edges_file` (Fi `basis/Edge.giv`) edges of the PTN
- CK `default_pool_file` (Fi `basis/Pool.giv`) line pool
- CK `default_pool_cost_file` (Fi `basis/Pool-Cost.giv`) costs of line pool
- CK `default_loads_file` (Fi `basis/Load.giv`) expected distribution of passengers to PTN edges (depending on CK `lc_model`)
- CK `default_od_file` (Fi `basis/OD.giv`) OD matrix (depending on CK `lc_model`)

2.3.3 Output

The following file is produced as output.

- CK `default_lines_file` (Fi `line-planning/Line-Concept.lin`) line pool, set of possible lines

2.3.4 Algorithms

To compute a line concept run

R `make lc-line-concept`

The following algorithms are available:

- CK `lc_model` CV `cost` optimization model minimizing the total costs of a line concept (see 3.3.1)
- CK `lc_model` CV `cost_restricting_frequencies` the CV `cost`-model, but with a restriction on the number of frequencies (see 3.3.1)

- CK lc_model CV direct optimization model maximizing the number of passengers who can travel on a shortest path from their origin to their destination without having to transfer (see 3.3.2)
- CK lc_model CV direct_restricting_frequencies the CV direct-model, but with a restriction on the number of frequencies (see 3.3.2)
- CK lc_model CV direct_relaxation relaxation of CK lc_model CV direct
- CK lc_model CV cost_greedy_1 greedy heuristic trying to minimize the costs
- CK lc_model CV cost_greedy_2 another greedy heuristic trying to minimize the costs
- CK lc_model CV mult-cost-direct an IP minimizing the weighted sum of costs and direct travelers
- CK lc_model CV mult-cost-direct-relax an IP minimizing the weighted sum of costs and direct travelers. Capacity restrictions are aggregated for each edge.
- CK lc_model CV traveling-time-cg a column generation procedure minimizing the estimated travel time of passengers. (see 3.3.4)
- CK lc_model CV traveling-time-mip (M)IP model for choosing line frequencies and passenger routes minimizing the estimated traveling times. (see 3.3.5)
- CK lc_model CV minchanges_ip integer program trying to minimize the weighted number of transfers (see 3.3.7)
- CK lc_model CV minchanges_cg column generation procedure trying to minimize the weighted number of transfers (see 3.3.7)
- CK lc_model CV game a game-theoretic approach which distributes lines equally among the edges in order to avoid congestion and delays

2.4 Ridepool Generation

In addition to the generation of a line pool and a line concept, it is also possible to generate a ride pool and a ride concept. In ridepooling, we generate several areas of the PTN, i.e. strongly connected subgraphs represented as edge lists. For each area a number of vehicles can be assigned that are allowed to travel freely in the area and operate as an on-demand ridepooling service.
By running

R make rpool-ride-pool

we generate a ride pool, to select some of the areas in the next step.

2.4.1 Input

The following files are needed as input:

- CK default_stops_file (Fi basis/Stop.giv) stops of the PTN
- CK default_edges_file (Fi basis/Edge.giv) edges of the PTN
- CK default_pool_file (Fi basis/Pool.giv) line pool
- CK default_loads_file (Fi basis/Load.giv) expected distribution of passengers to PTN edges (depending on CK lc_model)

2.4.2 Output

The following file is produced as output.

- CK filename_rpool_file (Fi basis/Ridepool.giv) ride pool, set of possible areas

2.4.3 Algorithms

The following models are available:

- CK rpool_model CV all Only one ridepooling area is constructed consisting of all edges of the PTN.
- CK rpool_model CV demand_heuristic Ridepooling areas are generated from edges with low demand.
- CK rpool_model CV tree_based Ridepooling areas are generated using a maximal spanning tree with respect to the edge loads.

2.5 Ride Concept

By running

R make rc-ride-concept

a line concept and a ride concept are generated simultaneously.

2.5.1 Input

The following files are needed as input:

- CK default_stops_file (Fi basis/Stop.giv) stops of the PTN
- CK default_edges_file (Fi basis/Edge.giv) edges of the PTN
- CK default_pool_file (Fi basis/Pool.giv) line pool
- CK filename_rpool_file (Fi basis/Ridepool.giv) ride pool, set of possible areas
- CK default_loads_file (Fi basis/Load.giv) expected distribution of passengers to PTN edges (depending on CK lc_model)

2.5.2 Output

The following file is produced as output.

- CK filename_rc_file (Fi basis/Ride-Concept.lin) ride concept, set of areas with assigned number of vehicles
- CK default_lines_file (Fi line-planning/Line-Concept.lin) line concept, set of lines with assigned frequencies

2.5.3 Algorithms

The following models are available:

- CK rc_model CV LPRP_ALPHA An IP is solved, where the edge-specific demand factors $\alpha_{r,e}$ are part of the input.
- CK rc_model CV LPRP_BETA An IP is solved, where the edge-specific demand factors $\alpha_{r,e}$ are are determined as continuous variables.

2.6 Periodic Timetabling

In periodic timetabling for each Event of a previously created Event-Activity-Network is assigned a time, resulting in a timetable.

2.6.1 Preparation

Run

R `make ean`

to create an Event-Activity-Network from an existing line concept.

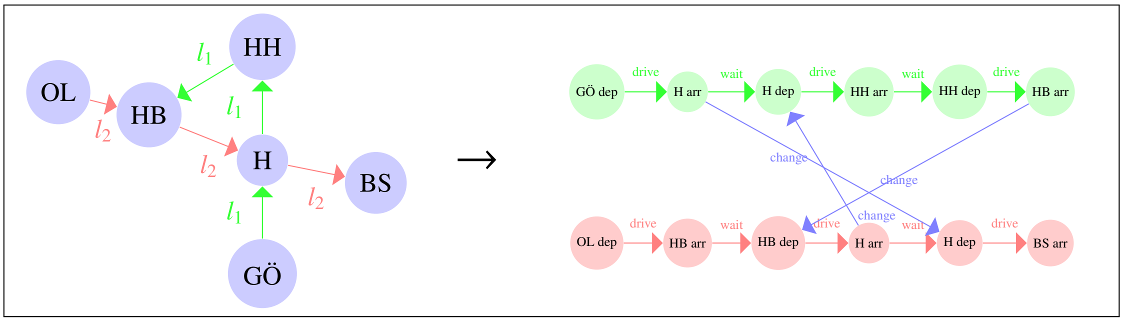


Figure 2.3: Creation of an EAN

2.6.2 Input

The following files are needed as input:

- CK `default_activities_periodic_file` (Fi `timetabling/Activities-periodic.giv`) Activities generated by the line concept.
- CK `default_events_periodic_file` (Fi `timetabling/Events-periodic.giv`) Events generated by the line concept.

For some timetabling procedures also the following files are necessary:

- CK `default_stops_file` (Fi `basis/Stop.giv`) stops of the PTN
- CK `default_edges_file` (Fi `basis/Edge.giv`) edges of the PTN
- CK `default_lines_file` (Fi `line-planning/Line-Concept.lin`) line concept calculated in the previous planning step
- CK `filename_tim_fixed_times` (Fi `timetabling/Fixed-timetable-periodic.tim`) fixed time intervals for some events

2.6.3 Output

The following files are produced as output.

- CK `default_timetable_periodic_file` (Fi `timetabling/Timetable-periodic.tim`)

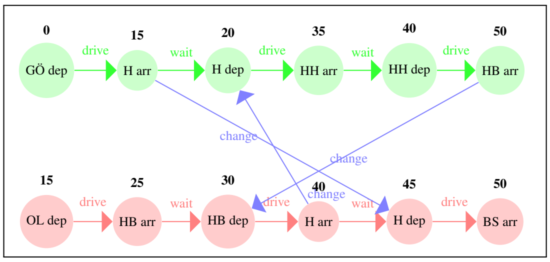


Figure 2.4: Periodic Timetabling Step

2.6.4 Algorithms

To compute a line concept run

R `make tim-timetable`

The following algorithms are available by setting the config parameter CK `tim_model` to one of the following:

- CV `MATCH` (default value) Heuristic that sets the times of driving and waiting activities to their lower bounds and then tries to minimize change durations.
- CV `con_prop` Heuristic that fixes events and propagates the implied constraints to the whole network.
- CV `csp` Heuristic that transforms the problem to a Constraint Satisfaction Problem and finds a feasible solution for it. *Currently not included in the release version of LinTim.*
- CV `ns_improve` Improvement procedure (known as Network-Simplex or Modulo-Simplex) that requires a feasible timetable.
- CV `csp_ns` Runs csp and ns_improve afterwards. *Currently not included in the release version of LinTim.*
- CV `con_ns` Runs con_prop and ns_improve afterwards.
- CK `ip` Models the Periodic Timetabling Problem as an IP and solves it.
- CK `cb_ip` Models the Periodic Timetabling Problem as a cycle based IP and solves it.
- CK `ns_cb` First improve a given feasible solution using the network simplex and afterwards optimize it using a cycle based IP
- CK `phase-one` Uses a phase 1 simplex method for finding a feasible timetable

2.7 Tariff Planning

Tariff planning computes a new tariff minimizing the deviation from given reference prices.

2.7.1 Input

The following files are needed as input:

- CK `default_stops_file` (Fi `basis/Stop.giv`), stops of the PTN
- CK `default_edges_file` (Fi `basis/Edge.giv`), edges of the PTN
- CK `default_od_file` (Fi `basis/OD.giv`), OD matrix

- CK filename_tariff_reference_price_matrix_file (Fi basis/Reference-Price-Matrix.giv), matrix of reference prices

Depending on the parameter values a routing in the PTN is also needed:

- CK filename_routing_ptn_input (Fi basis/Routing-ptn.giv), routing in the ptn

2.7.2 Output

The following files are produced as output independent of CK taf_model:

- CK filename_tariff_price_matrix_file (Fi tariff/Price-Matrix.taf), prices for each OD pair
- CK filename_tariff_properties_file (Fi statistic/tariff-properties.sta), statistic file containing information whether the no-elongation property and the no-stopover property (see Section 3.7.4) are fulfilled for the computed tariff. For a zone tariff it is also checked whether the zones are connected. Additionally, the tariff model and in case of a zone tariff the counting type are specified.

If CK taf_model is CV zone, then the two following files are produced as output as well:

- CK filename_tariff_zone_file (Fi tariff/Zones.taf), assignment of stops to zones
- CK filename_tariff_zone_price_file (Fi tariff/Zone-Prices.taf), prices per number of traversed zones

If CK taf_model is CV network_distance or CV zone, the routing that is computed is also written as output file:

- CK filename_routing_ptn_output (Fi basis/Routing-ptn.giv), routing in the ptn

2.7.3 Algorithms

To compute a tariff, run

R make taf-tariff

The following algorithms are available:

- CK taf_model CV flat, optimization model determining a flat tariff
- CK taf_model CV beeline_distance, optimization model determining an affine beeline distance tariff,
- CK taf_model CV network_distance, optimization model determining an affine network distance tariff,
- CK taf_model CV zone, optimization model determining a zone tariff.

2.8 Vehicle Scheduling

In the vehicle scheduling problem a set of routes for service vehicles is calculated to serve the given public transportation system. There are two base models, one based on an aperiodic timetable, the other only on a line concept. The following information is based on the classic formulations, based on the aperiodic timetable.

2.8.1 Preparation

Run

```
R make ro-rollout
```

and

```
R make ro-trips
```

with CK `rollout_whole_trips` set to CV `true` to create all input files needed for the vehicle scheduling problem.

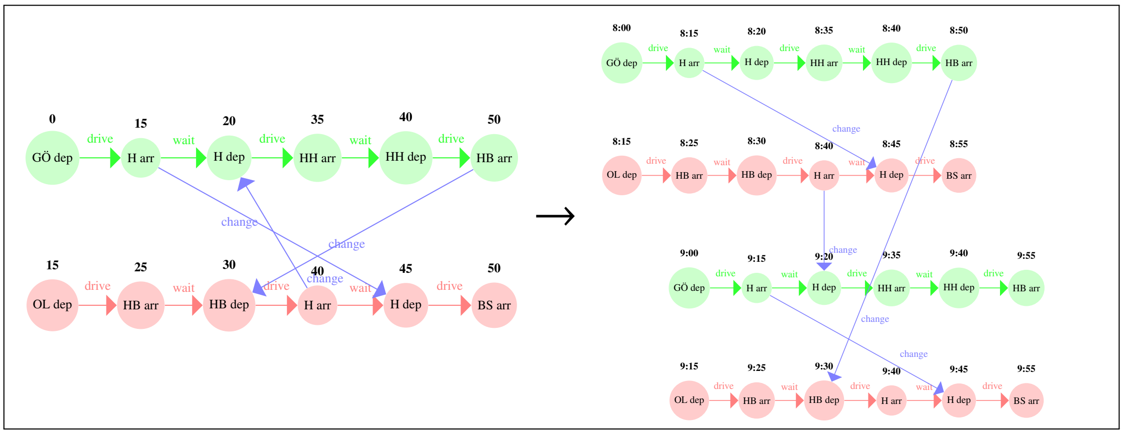


Figure 2.5: Rollout Step

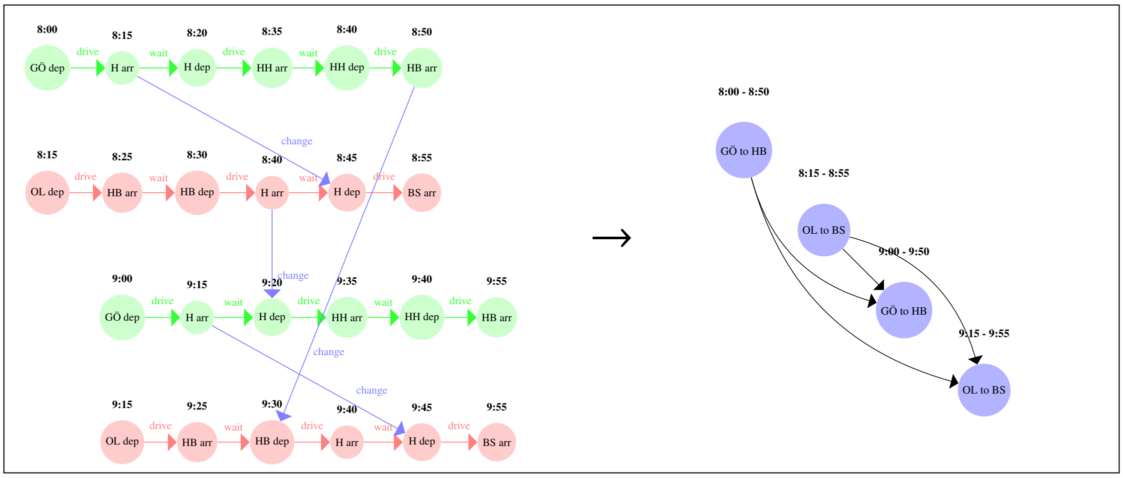


Figure 2.6: Rollout to Trips Step

2.8.2 Input

For the rollout

The following files are needed as an input for the rollout-step:

- CK `default_edges_file` (Fi `basis/Edge.giv`) edges of the PTN
- CK `default_headways_file` (Fi `basis/Headway.giv`) headways of the PTN

- CK default_lines_file (Fi line-planning/Line-Concept.lin) frequencies of the lines
- CK default_events_periodic_file (Fi timetabling/Events-periodic.giv) periodic events
- CK default_activities_periodic_file (Fi timetabling/Activities-periodic.giv) periodic activities
- CK default_timetable_periodic_file (Fi timetabling/Timetable-periodic.tim) periodic timetable

Only for the model

The following files are needed as an input for the vehicle scheduling step:

- CK default_stops_file (Fi basis/Stop.giv) stops of the PTN
- CK default_edges_file (Fi basis/Edge.giv) edges of the PTN
- CK default_trips_file (Fi delay-management/Trips.giv) trips for the vehicle schedule
- CK default_events_expanded_file (Fi delay-management/Events-expanded.giv) aperiodic events

2.8.3 Output

The following files will be produced:

- CK default_vehicle_schedule_file (Fi vehicle-scheduling/Vehicle_Schedules.vs) the vehicle schedule

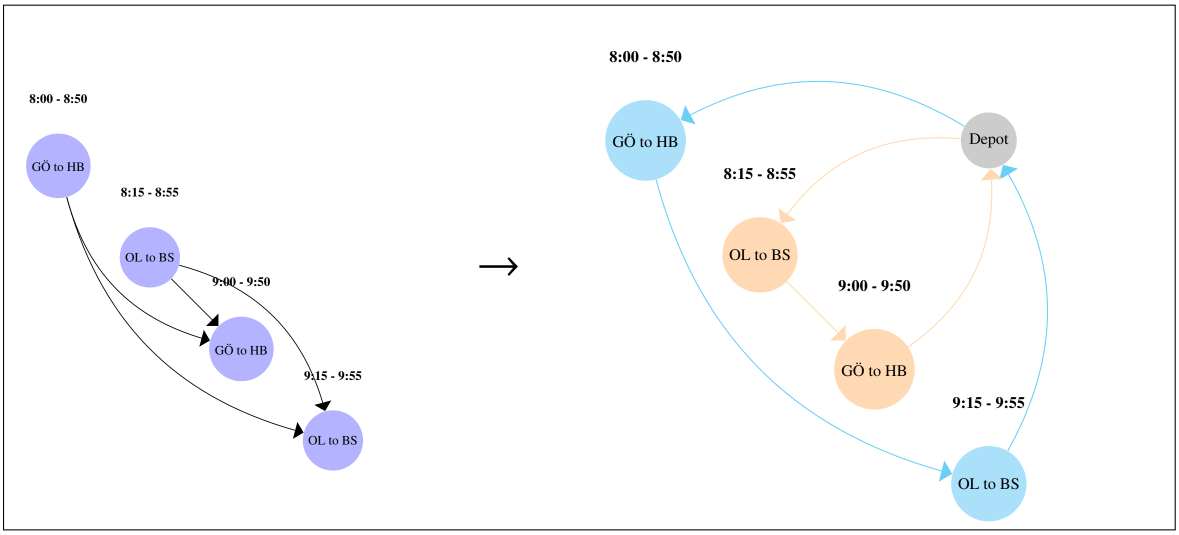


Figure 2.7: Vehicle Scheduling

2.8.4 Algorithms

To compute a vehicle schedule run

R make vs-vehicle-schedules

. The following models are available

- CK vs_model CV MDM1 Minimizing the number of vehicles (see 3.8.1)
- CK vs_model CV MDM2 Minimizing the number of vehicles (see 3.8.2)
- CK vs_model CV ASSIGNMENT_MODEL Minimizing the overall costs (see 3.8.3)
- CK vs_model CV TRANSPORTATION_MODEL Minimizing the overall costs (see 3.8.4)
- CK vs_model CV NETWORK_FLOW_MODEL Minimizing the overall costs (see 3.8.5) (see 3.8.1)
- CK vs_model CV CANAL_MODEL More detailed version of CV ASSIGNMENT_MODEL (see 3.8.6)
- CK vs_model CV LINE_BASED vehicle scheduling only based on line planning (see 3.8.7)
- CK vs_model CV SIMPLE will create a vehicle schedule driving the lines back and forth (see 3.8.8)
- CK vs_model CV IP solve a simple ip model (see 3.8.9)

2.9 Delay Management

Delay management computes a new (disposition) timetable based on an existing timetable and unforeseen delays that make the original timetable infeasible.

2.9.1 Preparation

If you have not already done so for the vehicle scheduling part, run

R make ro-rollout

to expand a previously computed periodic timetable on a periodic Event-Activity Network into an aperiodic timetable on an aperiodic Event-Activity Network.

2.9.2 Input

For the rollout

The following files are needed as an input for the rollout-step:

- CK default_edges_file (Fi basis/Edge.giv) edges of the PTN
- CK default_headways_file (Fi basis/Headway.giv) headways of the PTN
- CK default_lines_file (Fi line-planning/Line-Concept.lin) frequencies of the lines
- CK default_events_periodic_file (Fi timetabling/Events-periodic.giv) periodic events
- CK default_activities_periodic_file (Fi timetabling/Activities-periodic.giv) periodic activities
- CK default_timetable_periodic_file (Fi timetabling/Timetable-periodic.tim) periodic timetable

Aperiodic Event-Activity Network

These files, generated by the rollout step, are actually used for delay management:

- CK `default_events_expanded_file` (Fi `delay-management/Events-expanded.giv`) for the events
- CK `default_activities_expanded_file` (Fi `delay-management/Activities-expanded.giv`) for the activities
- CK `default_timetable_expanded_file` (Fi `delay-management/Timetable-expanded.tim`) for the initial timetable

Delays

There are two types of delays, which are both optional, and which go into separate files:

- CK `default_event_delays_file` (Fi `delay-management/Delays-Events.giv`)
- CK `default_activity_delays_file` (Fi `delay-management/Delays-Activities.giv`)

You can either manually enter delays on events and/or activities through these files, or use an automatic (random) delay generator by running

R `make dm-delays`

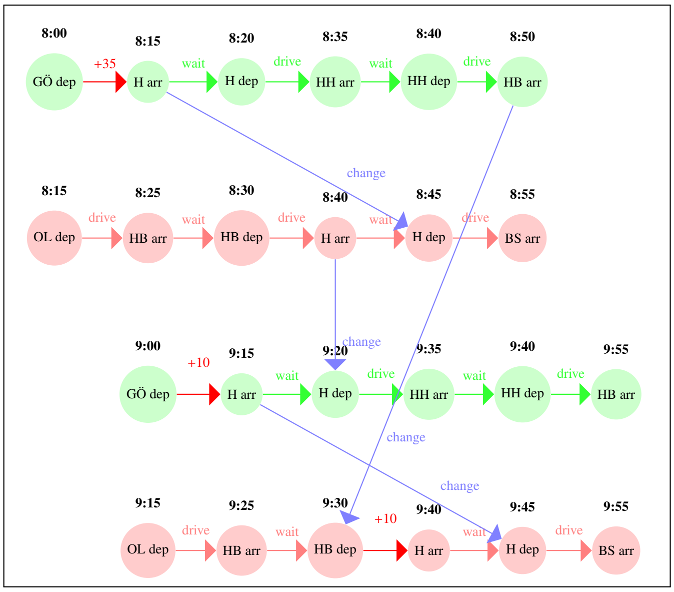


Figure 2.8: Generation of Delays

2.9.3 Output

The result of the delay management step is a new disposition timetable with no departure earlier than in the original timetable, and with all the delays respected: CK `default_disposition_timetable_file` (Fi `delay-management/Timetable-disposition.tim`)

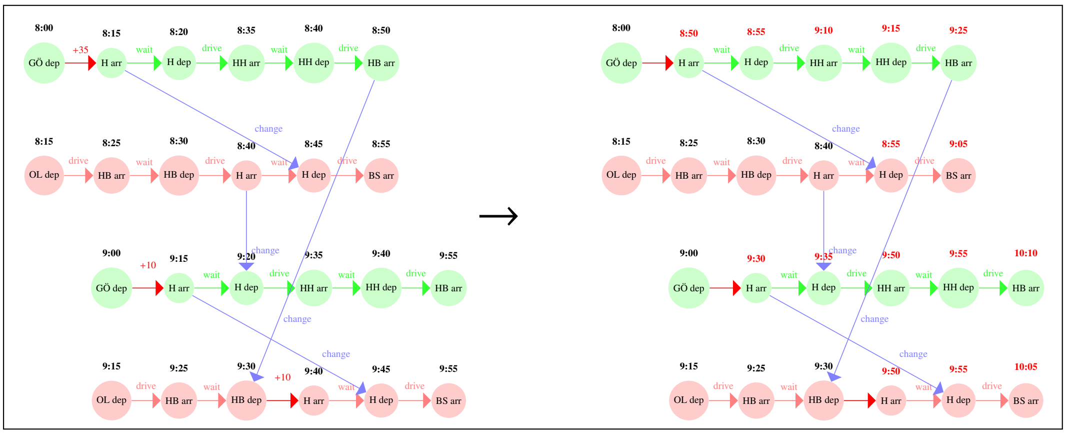


Figure 2.9: Delay Management Step

2.9.4 Algorithms

The delay management step is invoked via

R make dm-disposition-timetable

The main algorithms implemented in LinTim are the IP-based algorithms

- CK DM_method CV DM1
- CK DM_method CV FSFS
- CK DM_method CV FRFS
- CK DM_method CV EARLYFIX
- CK DM_method CV PRIORITY
- CK DM_method CV PRIOREPAIR
- CK DM_method CV best-of-all which computes all of the above and then chooses the best solution
- CK DM_method CV DM2
- CK DM_method CV DM2-pre

These need a solver configured via CK DM_solver (like CV Xpress or CV Gurobi, see Section 1.2 for details). In contrast, the most basic method without any optimization is just delaying all the events according to the delays, CK DM_method CV propagate, where a maximum waiting time for change activities can be configured in seconds by CK DM_propagate_maxwait, and headway activities can be turned around automatically whenever this would not result in additional delay for the train that was originally scheduled to go first, by setting CK DM_propagate_swapHeadways to CV true (the default).

2.10 Integrated Planning

LinTim also contains algorithms to compute multiple planning stages at once or in non-ordinary order.

2.10.1 Algorithms

Timetabling and Passenger Routing: Run

```
R make int-tim-pass
```

to solve the integrated timetabling and passenger routing problem. More information can be found in Section 3.10.1.

Timetabling and Vehicle Scheduling: Run

```
R make int-tim-veh
```

to solve the integrated timetabling and aperiodic vehicle scheduling problem. The passenger routes are fixed in this model. More information can be found in Section 3.10.2.

Line Planning and Timetabling: Run

```
R make int-lin-tim-pass
```

to solve the integrated line planning and timetabling problem. This also includes passenger routing in the timetabling stage. More information can be found in Section 3.10.3.

Timetabling and Vehicle Scheduling: Run

```
R make int-lin-tim-pass-veh
```

to solve the integrated line planning, timetabling and aperiodic vehicle scheduling problem. This also includes passenger routing in the timetabling stage. More information can be found in Section 3.10.4.

Robust Timetabling and Vehicle Scheduling using Machine Learning Run

```
R make int-rob-ml-algo
```

to solve the problem of finding a robust timetable and vehicle schedule based on the current solution. More information can be found in Section 3.10.5.

2.10.2 The Eigenmodel

The eigenmodel is an iterative approach to integrated public transport planning, re-organizing the sequential planning approach to allow new optimization models, solving the original problem in different orderings. For more details, see Section 3.10.6.

Chapter 3

Detailed Description of Algorithms

3.1 Stop Location

3.1.1 Without a given infrastructure network

Running

R `make sl-stop-location`

will create a new PTN with respect to the given demand points. Here, all demand points have to be *covered* by at least one station, i.e., the distance between the demand point an d the nearest station has to be less than a given radius.
The parameters used for adjusting the model are the following:

- CK `sl_distance` norm used for measuring the distance between demand points, stations etc. Currently the only option is `euclidean_norm`.
- CK `sl_radius` maximal distance a demand point may have from a station to be covered.
- CK `sl_destruction_allowed` whether it is allowed to remove station that are not covering any demand points.
- CK `sl_new_stop_default_name` name prefix to be given to new stops.

Fixed travel time on edges

The first step of the classical stop location problem which uses fixed travel times on the edges is to compute a finite dominating set of candidates for new stations. When using the euclidean norm for measuring distance this finite dominating set can easily computed as the intersection of the tracks and circles around the demand point with the given radius and the already existing stops.

Optimization model For the optimization model define the constants

$$a_{ps} = \begin{cases} 1 & \text{if demand point } p \text{ is covered by candidate } s \\ 0 & \text{otherwise} \end{cases}$$

and the variables

$$x_s = \begin{cases} 1 & \text{if candidate } s \text{ is established as station} \\ 0 & \text{otherwise} \end{cases}.$$

The objective is to minimize the number of established stations such that all demand points are covered. The following optimization model is solved to find an optimal solution for the stop location problem.

$$(DSL) \quad \min \sum_{s \in \mathcal{S}} x_s$$
$$\text{s.t.} \sum_{s \in \mathcal{S}} a_{ps} x_s \geq 1 \quad \forall p \in \mathcal{P}$$
$$x_s \in \{0,1\} \quad \forall \mathcal{S}$$

For more information, see [34, 38].

Greedy heuristic The greedy heuristic find a feasible solution to the stop location problem by successively adding the candidate which covered most uncovered demand points at this point in time.
For more information, see [34, 38].

Travel time considering acceleration/deceleration

When considering the acceleration and deceleration phases of vehicles, the following parameters have to be set:

- CK `sl_acceleration`
- CK `sl_deceleration`
- CK `sl_waiting_time`

For more information, see [4].

3.1.2 For a given infrastructure network

If a complete infrastructure network, i.e., an infrastructure network with walking and node-based od-information, is given, the stop location models CK `sl_model` CV `tt` and CV `all` can be used. For CV `tt`, a selection of stops is chosen such that the overall travel time of the passengers (containing public transport use as well as walking) is minimized. Additionally, creating stops is penalized by CK `sl_cost_of_stop`. For CV `all`, all possible stop points are converted to stops in the PTN.
Given forbidden edges in the infrastructure (CK `sl_forbidden_edges`) and given restricted turns in the infrastructure (CK `sl_restricted_turns`) can be converted into the resulting ptn information as well when their corresponding config parameter is set to CV `true`.

3.2 Line Pool Generation

A new line pool is created by running

R `make lpool-line-pool`

There are four main approaches implemented in LinTim.

1. Different variants of a tree-based heuristic. These are invoked by setting CK `lpool_model` to either CV `tree_heuristic` or CV `restricted_line_duration` and are described in Sections 3.2.1 and 3.2.2, respectively.
2. A center-periphery heuristic, which is called with the parameter setting CK `lpool_model` CV `center-periphery` and described in Section 3.2.5.
3. k-shortest paths. This is called by setting CK `lpool_modelto` CV `k_shortest_paths` and is described in Section 3.2.3.
4. All paths between pairs of a given set of terminal stations. This is executed when setting CK `lpool_model` to CV `terminal-to-terminal` and described in Section 3.2.4.

3.2.1 Creating a new line pool with the tree based heuristic

For an undirected PTN a line pool $\mathfrak{L}$ may be created from an existing PTN (CK `default_edges_file` (Fi `basis/Edge.giv`), CK `default_stops_file` (Fi `basis/Stop.giv`)), a given CK `default_loads_file` (Fi `basis/Load.giv`) (see Chapter 8), and a given CK `default_od_file` (Fi `basis/OD.giv`) by running

R `make lpool-line-pool`

with CK `lpool_model` CV `tree_based`, which creates a line pool CK `default_pool_file` (Fi `basis/Pool.giv`) and a corresponding CK `default_pool_cost_file` (Fi `basis/Pool-Cost.giv`). How the line costs are computed can be seen in Section 3.2.6.
The algorithm iteratively creates minimum spanning trees, on which lines are created in three different possible ways:

- as a path from a leaf of the MST to another leaf,
- as a path from a leaf of the MST to a *terminal* or
- as a path from a terminal to another terminal.

Here *terminals* are nodes of a high node degree. Each of the three classes of lines has to fulfill different requirements, which can be seen in the discussion of the configuration parameters. Lines are created until a feasible line concept can be found within the line pool or until the maximal number of iterations is reached. One iteration consists of the following steps:

1. Determine a set of preferred edges.
2. Compute minimum spanning trees and create lines until all preferred edges are covered sufficiently often or no other line can be added.
3. Test whether a feasible line concept can be found in the constructed pool.

In the first iteration preferred edges are chosen from the usage rate in the shortest paths of the OD pairs. Later, the lower frequencies given in the loads file are lowered until a feasible line concept can be found for the new frequencies, and the edges for which the original frequencies are not met are chosen as preferred edges.
The edge weight used to compute the minimum spanning tree is zero if the edge is preferred and the physical length of the edge otherwise.
The configuration parameters are:

- CK `lpool_max_iterations`: the maximal number of iterations.
- CK `lpool_ratio_od`: the ratio of the most frequently used edges in shortest paths of the passengers, which are preferred in the first iteration.
- CK `lpool_node_degree_ratio`: the percentage of the maximal node degree, which has to be attained to qualify a node as a terminal. In the first iteration the node degree depends on the incident edges in the PTN, later it depends on the lines passing the node.
- CK `lpool_min_cover_factor`: a preferred edge has to be covered $\lceil \frac{f_e^{\min}}{\texttt{lpool\_min\_cover\_factor}} \rceil$ times in order to be sufficiently covered.
- CK `lpool_max_cover_factor`: if a new line covers an edge more than $f_e^{\max} \cdot \texttt{lpool\_max\_cover\_factor}$ it cannot be used in the line pool.
- CK `lpool_min_edges`: the minimal number of edges in a line from a leaf to a terminal or from a terminal to another terminal.

- CK lpool_min_distance_leaves: the minimal euclidean distance between two leaves to allow for a line between them.
- CK lpool_add_shortest_paths: determines whether shortest paths are to be added as additional lines to the line pool.
- CK lpool_ratio_shortest_paths: the percentage of the maximal number of passengers in an OD pair which has to be attained in order to add the shortest path for an OD pair as a line. This parameter is only relevant if CK lpool_add_shortest_paths is set to true.
- CK lpool_append_single_edges: Add all links as separate lines to the line pool.

Note that all lines which are created here are cycle-free, as they are either a path in a minimal spanning tree or a shortest path in a network with non-negative edge-lengths.
Possible additional restrictions on the created lines are

- CK lpool_restrict_terminals Only allow lines that start or end at terminals given in CK filename_terminals_file (Fi basis/Terminals.giv)
- CK lpool_restrict_turns Only allow lines that do not contain a restricted turn given in CK filename_turn_restrictions (Fi basis/Restricted-Turns.giv)
- CK lpool_restrict_forbidden_edges Do not allow the forbidden links in CK filename_forbidden_links_file (Fi basis/Edge-forbidden.giv) to be contained in lines

For more information, see [10].

3.2.2 Creating a line pool while restricting the duration of the lines

When running

R make lpool-line-pool

with the parameter CK lpool_model set to CV restricted_line_duration the tree based heuristic (see 3.2.1) is performed with additional constraints on the duration of lines. This is influenced by the following parameters:

- CK ean_model_weight_drive to decide how the duration of a line is computed
- CK ean_model_weight_wait to decide how much waiting time is added in each station
- CK period_length used to determine the feasible duration interval
- CK vs_turn_over_time used to determine the feasible duration interval
- CK lpool_restricted_maximum_buffer_time used to determine the feasible duration interval
- CK lpool_restricted_allow_half_period determines if lines which fit into the interval at exactly half a period minus the corresponding buffer times are allowed to be added

The feasible interval for the duration of a line mod CK period_length is defined as

[CK period_length − CK vs_turn_over_time
−CK lpool_restricted_maximum_buffer_time,
CK period_length − CK vs_turn_over_time].

Note: There will be no shortest paths added to line pools created by this heuristic, i.e.,
CK lpool_add_shortest_paths has no influence.
For more information, see [24].

3.2.3 Creating a line pool by k shortest paths

Another possibility is to create a line pool with corresponding line costs by using the k shortest paths for each OD pair as lines and then deleting lines which are nested in other lines. To do so run

R `make lpool-line-pool`

with the parameters

- CK `lpool_model` CV `k_shortest_paths`
- CK `lpool_number_shortest_paths`, which gives the number of shortest paths which are to be computed for each OD pair.

3.2.4 Terminal-to-terminal

When terminals are given, i.e., CK `filename_terminals_file` (Fi `basis/Terminals.giv`), running

R `make lpool-line-pool`

with the parameters

- CK `lpool_model` CV `terminal-to-terminal`

will result in the enumeration of all possible lines starting and ending at a terminal and therefore finding all possible lines respecting the terminal restrictions. Note that this may result in large computation times and a large number of lines in the linepool, depending on your PTN.

3.2.5 Center-Periphery

Another method to create a line pool is running

R `make lpool-line-pool`

with the parameter

- CK `lpool_model` CV `center-periphery`

This algorithm identifies some nodes as centers and some nodes as periphery to construct lines between those different pairs of nodes. It is a heuristic that tries to identify natural patterns in the PTN and the OD data. The following parameters have to be specified:

- CK `lpool_centers_fraction` Fraction of nodes that can become centers
- CK `lpool_periphery_radius_factor` Factor for the mean distance of two nodes in the PTN to choose periphery nodes
- CK `lpool_direct_periphery_lines_factor`
- CK `lpool_center_radius_factor` Percentage of the mean distance in the PTN determining the radius of the centers
- CK `lpool_concatenate_lines_factor` Factor for the mean OD value to choose node pairs for which direct lines are created by concatenating existing lines
- CK `lpool_min_degree_center` Minimal node degree that a center node must have
- CK `lpool_min_times_edge_covered` Factor for the minimal frequency of each edge to determine how many times it should be covered by a line
- CK `lpool_max_iter_postprocessing` Maximal number of iterations for each postprocessing step

- CK `lpool_opt_cost` Determines, if lines are created along shortest paths w.r.t. edge costs or edge lengths
- CK `lpool_plot_centers` Optional parameter. If set to `True`, plots of the PTN are created where the centers are highlighted in red and the periphery nodes are highlighted in blue. The images are written to the Fi `/graphics` directory.

Choice of centers and periphery

Let V be the set of nodes in the PTN and $n = |V|$. For each node v in the PTN we compute the number of *interactions* for this node as a measure of its importance as

$$\sum_{u \in V: u \neq v} OD_{u,v} + OD_{v,u}.$$

Only nodes with a degree of at least CK `lpool_min_degree_center` in the PTN are candidates for centers. We order the set of those nodes non-increasingly by the interactions computed above. The parameter CK `lpool_centers_fraction` states which portion of the candidates with the highest interaction values should remain center candidates. We look in the interval

$$n \cdot \text{CK } \texttt{lpool\_centers\_fraction} + [-0.2 \cdot n, 0.2 \cdot n]$$

for the greatest difference in the interactions between two neighboring nodes in the sorted list to identify the biggest jump in the interactions around the desired portion. Among those candidates we choose the centers in such a way that no centers have a distance less than the mean distance between any two nodes in the PTN multiplied with CK `lpool_center_radius_factor` and the sum of the interactions of the chosen centers is maximized. This is modelled by an IP which is solved using Gurobi.
Now we determine the periphery nodes. All Endstations, i. e. all nodes with degree 1, are defined to be periphery nodes. Furthermore, all non-center nodes with a distance greater than

$$\text{CK } \texttt{lpool\_periphery\_radius\_factor} \cdot \sum_{c \text{ center}} \sum_{u \in V: u \neq c} \frac{\text{distance}(u, c)}{(\text{number of centers})(n-1)}$$

(the mean distance from a node to a center) to it's closest center become periphery nodes.

Line generation

If the parameter CK `lpool_opt_cost` is set to `True` then all shortest paths are computed with respect to the costs of the edges, otherwise with respect to their lengths.
For each pair of centers lines are generated along all shortest paths between them.
For each pair of a center and a periphery node lines are generated along all shortest paths between them, if this line is not contained in another line which was already generated.
For each pair of periphery nodes lines are generated along all shortest paths, if the correspronding OD-value ist greater than the mean OD-value multiplied with CK `lpool_direct_periphery_lines_factor`.
As a next step we concatenate for each node pair with an OD-value greater than the mean OD-value multiplied with CK `lpool_concatenate_lines_factor` all yet generated lines from the start node to it's closest center, the lines between the closest centers of the nodes and the lines from the closest center of the end node to itself. This gives direct connections between the most important OD pairs, but they fit with the center-periphery pattern we want to establish.
If after this procedure there is still a node not covered by a line, we create lines from this node to its closest center along all shortest paths.
In a last step lines along small detours are created. For this we look at all edges that are covered by the smallest number of lines. For those edges, the closest peripheries and centers of both endpoints are determined. Then lines containing the specified edge are created, starting from one of the closest periphery or center nodes of the left node and ending in one of the closest nodes of the right node. This is done along all shortest paths and for all such pairs of closest peripheries or centers. If this procedure creates cylces it is aborted and the single edge is added as a line to the pool.

Postprocessing

The idea of the postprocessing step is to generate lines, that contain edges which are not yet covered by enough lines proportionally to their minimal frequency. In a while loop, we consider a residual network which is initially the same as the PTN. In every iteration the edge weights are updated and lines are created along the shortest paths found in the residual network. We use all line generating algorithms from the previous section. If no new line was found or the maxmimal number of iterations was reached the while loop is aborted. Let $n(e)$ be the number of lines that contain the edge e and let e_{res} denote the copy of the edge e in the residual network. Then we compute for each residual edge it's lower frequency bound as

$$f_{min}(e_{res}) = \max\{0, f_{min}(e) - n(e) \cdot \min_{L:e\in L} \min_{e'\in L} f_{min}(e')\}$$

where $e \in L$ means that the edge e is contained in the line L. The length of the resiudal edge is then

$$l(e_{res}) = \exp(f_{min}(e) - f_{min}(e_{res})) \cdot l(e)$$

with $l(e)$ being the original length of the edge. The shortest paths are now computed with respect to this new length. If we set CK `lpool_opt_cost` to `True` the costs of the edges are used instead of their lengths.
After this frequency based postprocessing step we do a covered based postprocessing step. The only difference is that the new lengths are set to

$$l(e_{res}) = n(e) \cdot l(e).$$

3.2.6 Line costs

The costs of the lines created by

R `make lpool-line-pool-cost`

are of the following form

$$\begin{aligned}\text{cost}_l = {} & \text{CK } \texttt{lpool\_costs\_fixed} \\ & + \sum_{e\in l} \left(\text{CK } \texttt{lpool\_costs\_length} \cdot \text{length}_e + \text{CK } \texttt{lpool\_costs\_edges}\right) \\ & + \text{CK } \texttt{lpool\_costs\_vehicles} \cdot \left\lceil x \cdot \frac{\text{duration}_l + \text{CK } \texttt{vs\_turn\_over\_time}}{\text{CK } \texttt{period\_length}} \right\rceil,\end{aligned}$$

where x is 1 for directed and 2 for undirected lines (since undirected lines need to be traversed in both directions). The duration of a line is computed as described in Section 3.2.2.
For a given line pool CK `default_pool_file` (Fi `basis/Pool.giv`) a corresponding cost file CK `default_pool_cost_file` (Fi `basis/Pool-Cost.giv`) can be created by running

R `make lpool-line-pool-cost.`

3.3 Line Planning

The line planning problem can be solved by running

R `make lc-line-concept.`

The following subsection describe the corresponding algorithms.

3.3.1 Cost

Running

R `make lc-line-concept`

with CK `lc_model` CV `cost`, CV `cost_greedy_1` or CV `cost_greedy_2` results in solving the line planning model such that the operational costs are minimized. Operational costs in line planning are defined as line based costs cost_l for all line $l \in \mathcal{L}$ and are calculated once per frequency. This means the operation costs of a line concept with line frequencies f_l for line $l \in \mathcal{L}$ is

$$\sum_{l \in \mathcal{L}} \text{cost}_l \cdot f_l.$$

Optimal solution

Running

R `make lc-line-concept`

with CK `lc_model` CV `cost` results in solving the classic costs minimizing line planning problem, described in [36], to optimality. The corresponding integer program is

$$\begin{aligned} &\text{(LP-Cost)} \min \sum_{l \in \mathcal{L}} \text{cost}_l \cdot f_l \\ \text{s.t.} \quad & f_e^{\min} \leq \sum_{l \in \mathcal{L}:\, e \in l} f_l \leq f_e^{\max} \quad \forall e \in E \\ & f_l \in \mathbb{Z} \quad \forall l \in \mathcal{L}. \end{aligned}$$

which is solved either by the solver Gurobi or by the solver Xpress, depending on whether CK `lc_solver` is set to CV `GUROBI` or CV `XPRESS`.

System frequency

Running

R `make lc-line-concept`

with CK `lc_model` CV `cost` and CK `lc_common_frequency_divisor` set to a value unequal to 1, will result in solving the problem with a system frequency, i.e., a frequency is only allowed in a solution, if it is the multiple of the system frequency CK `lc_common_frequency_divisor`. A value <= 0 will test any system frequency (except for 1) and output the best solution.
For more information, see [9].

Heuristic solutions

Running

R `make lc-line-concept`

with CK `lc_model` CV `cost_greedy_1` or CV `cost_greedy_2` results in solving a heuristic for the cost model described in this section. Lines are added to the line concept in a greedy way (w.r.t. the costs of the lines) until the lower frequency bounds on the edges are fulfilled. Note that these algorithms ignore the upper frequency bounds and are therefore not guaranteed to find a feasible solution w.r.t. these bounds. The algorithms are described in [33].

Restricting the number of frequencies

Running

R make lc-line-concept

with CK lc_model CV cost_restricting_frequencies results in solving the cost model, while restricting the number of possible frequencies. The resulting model has more variables than the original problem, which may results in much longer running times. Even if the number of possible frequencies is unrestricted (-1) this is still not the same model as cost due to CK lc_maximal_frequency.

- CK lc_solver either CV GUROBI or CV XPRESS, the solver to use to solve the model
- CK lc_number_of_possible_frequencies restrict the number of possible frequencies (-1=infinity)
- CK lc_timelimit the time limit for the solver (-1=infinity)
- CK lc_maximal_frequency the maximal allowed frequency

Fixed Lines

Running

R make lc-line-concept

with CK lc_model CV cost and CK lc_respect_fixed_lines set to CV true, will result in solving the cost model while fixing the line frequencies given by CK filename_lc_fixed_lines (Fi line-planning/Fixed-Lines.lin). Fixed lines will count towards fulfilling the lower frequency bounds for feasibility and need to be included in the line pool, i.e., CK default_pool_file (Fi basis/Pool.giv) and CK default_pool_cost_file (Fi basis/Pool-Cost.giv). The capacities for fixed lines need to be given in CK filename_lc_fixed_line_capacities (Fi line-planning/Line-Capacities.lin).

Forbidding Links

It is possible to forbid the usage of certain links in the PTN by setting CK lc_respect_forbidden_edges to CV true and giving the forbidden links in CK filename_forbidden_links_file (Fi basis/Edge-forbidden.giv). Then, the upper bounds for all the corresponding links will be set to 0 in the optimization problem, guaranteeing that lines using these links will not be used in a feasible line concept. This may be useful when considering a PTN with multiple public transport modes, i.e., having tracks and streets and optimizing a bus network that may not use tracks. Can be combined with setting fixed lines for the forbidden edges.

3.3.2 Direct

Running

R make lc-line-concept

with CK lc_model CV direct results in solving an optimization model which aims to maximize the number of passengers which can travel on a shortest path from their origin to their destination without having to transfer between lines. The shortest path is determined w.r.t. CK ean_model_weight_drive. Upper and lower frequency bounds have to be fulfilled similar to the cost model and additionally capacity constraints on all edges have to be satisfied. Fixing lines and forbidding links is possible here as well, see the documentation for the cost model in Section 3.3.1.
The following parameters control the behavior of the algorithm:

- CK ean_model_weight_drive
- CK gen_passengers_per_vehicle
- CK lc_budget
- CK lc_common_frequency_divisor
- CK lc_direct_optimize_costs
- CK lc_mip_gap
- CK lc_mult_relation
- CK lc_respect_fixed_lines
- CK lc_respect_forbidden_edges
- CK lc_timelimit
- CK period_length

For more information on the model, see [3].

Restricting the number of frequencies

Running

R `make lc-line-concept`

with CK lc_model CV direct_restricting_frequencies results in solving the direct model, while restricting the number of possible frequencies. The resulting model has more variables than the original problem, which may results in much longer running times. Even if the number of possible frequencies is unrestricted (-1) this is still not the same model as direct due to CK lc_maximal_frequency.

- CK gen_passengers_per_vehicle
- CK lc_budget
- CK ean_model_weight_drive
- CK lc_common_frequency_divisor
- CK lc_timelimit
- CK lc_maximal_frequency

System frequency

Running

R `make lc-line-concept`

with CK lc_model CV direct and CK lc_common_frequency_divisor set to a value unequal to 1, will result in solving the problem with a system frequency, i.e., a frequency is only allowed in a solution, if it is the multiple of the system frequency CK lc_common_frequency_divisor. A value <= 0 will test any system frequency (except for 1) and output the best solution. For more information, see [9].

Aggragating the passengers per OD pair

Running

R make lc-line-concept

with CK lc_model CV direct_relaxation results in solving the direct model, while aggregating the passengers per OD pair. This is a relaxation of the original model, see [3].

Multicriteria optimization

Setting CK lc_direct_optimize_costs to CV true will result in solving the direct model with a weighted sum, accounting for the line costs of the resulting line concept as well. As a weight factor, CK lc_mult_relation will be used.

3.3.3 Cost direct weighted sum

Executing

R make lc-line-concept

with CK lc_model set to CV mult_cost_direct or CV mult_cost_direct_relax solve programs which are weighted sums between the cost model (Section 3.3.1) and the direct travelers model (Section 3.3.2). In the relaxed version (i.e.,
CV mult_cost_direct_relax) the vehicle capacity is not considered for each vehicle but only the aggregated capacity for each edge is considered. The capacity consideration can be turned off by setting CK lc_mult_cap_restrict. The weight can be set by CK lc_mult_relation where CV 0 refers to the direct travellers model and CV 1 to the cost model. The tolerance of feasibility, integrality and optimality can be set by CK lc_mult_tolerance. A time limit in seconds can be set by CK lc_timelimit, but it will only stop the computation if a feasible solution was already found. Otherwise the computation will continue until a feasible solution is found and stop then.
Additionally, there is the possibility to consider system frequencies, i.e., a common integer divisor for all frequencies. For this, set CK lc_common_frequency_divisor to something different than CV 1. When setting it to a value smaller or equal to CV 0, different prime values are tested as a system frequency and the best in terms of objective value is used as output. Note that testing prime numbers is enough for finding an optimal solution.

3.3.4 Traveling time without frequencies

Executing

R make lc-line-concept

with CK lc_model CV traveling_time_cg solves the traveltime model as stated in [39, (LPMT1)] under the name (LPMT1). This model does not include line frequencies but only decides which lines are established. It routes all passengers over established lines and minimizes their resulting total travel time. Each established line incurs some cost and the total cost is bounded by a budget. This model is solved by a column generation procedure in which the passenger paths are generated throughout the column generation iterations. It is implemented as part of [17]. Various different method exist in order to compute a feasible starting tableau. That is

- CK lc_traveling_time_cg_cover can be set to true or false and is a method to include passenger paths based on the idea that every edge is covered by at least one line.
- CK lc_traveling_time_cg_k_shortest_paths can be set to an integer value. This adds a number of shortest paths.

- CK lc_traveling_time_cg_add_sol_1 can be set to true or false. The passenger paths which are based on the line concept (a file) given in CK lc_traveling_time_cg_add_sol_1_name are added.

- CK lc_traveling_time_cg_add_sol_2 can be set to true or false. The passenger paths which are based on the line concept (a file) given in CK lc_traveling_time_cg_add_sol_2_name are added.

- CK lc_traveling_time_cg_add_sol_3 can be set to true or false. The passenger paths which are based on the line concept (a file) given in CK lc_traveling_time_cg_add_sol_3_name are added.

Then the actual column generation procedure is started. Four different versions of constraints (corresponding to CV 1, CV 2, CV 3, CV 4) can be used which are set by CK lc_traveling_time_cg_constraint_type. Finally the following parameters are important for execution.

- CK lc_traveling_time_cg_max_iterations: This many column generation iterations are executed at most.

- CK lc_traveling_time_cg_termination_value: This is the gap in percent between lower and upper bound below which the best solution is returned.

- CK lc_traveling_time_cg_weight_change_edge: The weights of the transfer (change) edges in the Change&Go-Graph are determined by this value.

- CK lc_traveling_time_cg_weight_od_edge: The weights of the OD edges in the Change&Go-Graph are determined by this value.

- CK lc_traveling_time_cg_relaxation_constraint: boolean for additional relaxation constraint $y_l \ \forall l \in \mathcal{L}$

- CK lc_traveling_time_cg_solve_ip: if set to true the integer program corresponding to the final linear program should be solved in the last step to approximate an integer solution.

3.3.5 Traveling time with frequencies

Executing

R make lc-line-concept

with CK lc_model CV traveling_time_mip and CK lc_traveling_time_mip_minimize CV "time" solves the traveling time model with line frequencies as stated in [39, (LPMTF)]. In contrast to the formulation presented in this paper, flow variables belonging to the same origin are aggregated, as in [1]. It uses the solver chosen with CK lc_solver to solve the model. The following additional options can be given:

- CK lc_traveling_time_mip_use_loads: If this is set to true, then the upper and lower bounds on the frequency of service on each edge in the PTN given in the CK default_loads_file are respected. This corresponds to constraint (13) from [39, (LPMTF)] and a symmetric constraint for the lower bound. Otherwise, no bounds on the frequency are respected, i.e., the model only incorporates constraints (10)–(12) and (14) from the referenced model.

- CK lc_traveling_time_mip_integer_flow: Boolean to specify whether the computed passenger flows have to be integral.

- CK lc_traveling_time_mip_integer_frequencies: Boolean to specify whether the computed line frequencies have to be integral.

- CK ean_model_weight_drive: Determines the method used to estimate the driving time of a vehicle on an edge, based on the bounds given in the edge file, see Section 7.9.

- CK ean_model_weight_wait, CK ean_default_minimal_waiting_time, and CK ean_default_maximal_waiting_time: Determine the method used to estimate the waiting time of a vehicle at a station, see Section 7.9.
- CK ean_change_penalty, CK ean_default_minimal_change_time: Each transfer is charged with the sum of these two parameters.
- CK lc_budget: Allowed total cost of the chosen line concept. It is assumed that running a line with frequency f incurs a cost of f times the value specified in the CK default_pool_cost_file.
- CK gen_passengers_per_vehicle: Used to determine the total frequency needed to serve all passengers using a line along an edge.

3.3.6 Cost with traveling time bound

Executing

R `make lc-line-concept`

with CK lc_model CV traveling_time_mip and CK lc_traveling_time_mip_minimize CV "cost" solves a variant of the traveling time model with frequencies, in which the traveling time is bounded by CK lc_traveling_time_mip_time_budget and the cost of the line concept is minimized. Apart from that, it uses the same configuration parameters as the traveling time model with frequencies (exept CK lc_budget).

3.3.7 Minchanges

Running

R `make lc-line-concept`

with CK lc_model CV minchanges_ip or CV minchanges_cg results in solving a program to minimize the number of passenger weighted transfers. For further reference see [15].

Integer program

The integer program corresponding to method CV minchanges_ip is

$$\textbf{(IP-LPT)} \qquad \min \sum_{i,j\in V} \sum_{p\in\mathcal{P}_{CG}^{ij}} d_p c_p \tag{3.1}$$

$$\sum_{p\in\mathcal{P}_{CG}^{ij}} d_p \geq C_{ij} \quad \forall i,j \in V \tag{3.2}$$

$$\sum_{i,j\in V} \sum_{\substack{p\in\mathcal{P}_{CG}^{ij} \\ (e,l)\in p}} d_p \leq A f_l \quad \forall l \in \mathcal{L}, \forall e \in l \tag{3.3}$$

$$\sum_{\substack{l\in\mathcal{L} \\ e\in l}} f_l \leq f_e^{max} \qquad \forall e \in E \tag{3.4}$$

$$d_p \in \mathbb{N}_0 \qquad \forall p \in \mathcal{P}_{CG} \tag{3.5}$$

$$f_l \in \mathbb{N}_0 \qquad \forall l \in \mathcal{L} \tag{3.6}$$

Since paths of passengers have to be tracked in order to obtain their transfers, the model is based on the Change&Go-Graph CG proposed in [39]. Paths in the Change&Go-Graph are referred to as $\mathcal{P}_{CG}$. The number c_p then gives the number of transfers on a path $p \in \mathcal{P}_{CG}$. The variables d_p and f_l specify the number of passengers on path p and the frequency of line $l \in \mathcal{L}$, respectively.
The following parameters are used to execute the computation:

- CK lc_minchanges_nr_ptn_paths determines the maximum number of paths in the PTN on which passengers from each OD pair are allowed to travel. This ensures that also $|\mathcal{P}_{CG}|$ is bounded.
- CK lc_minchanges_xpress_miprelstop. This parameters is passed to the execution of Xpress and determines the gap (in percent) between lower and upper bound which has to be reached such that the best solution is returned.
- CK lc_minchanges_nr_max_changes. Since the number of paths in the Change&Go-Graph could become very large this parameter is used to bound them. Only paths which have less or equal transfers (changes) are considered. A value of 0 means that all paths are considered.
- CK gen_passengers_per_vehicle. This parameter corresponds to the A in constraint (3.3) and determines the vehicle capacity.

Column Generation procedure

In the column generation procedure the integer program (IP-LPT) is relaxed and initially only solved for a subset of all possible paths $\mathcal{P}_{CG}$. Throughout the column generation procedure paths which are likely to improve the current solution are determined and added to the program. The column generation procedure ends if no such paths can be found anymore. The problem which is solved in order to determine paths which are likely to improve the current solution is an all pairs shortest path problem. Since the correspondence of the solution of this problem to the primarily determined paths in the PTN, $\mathcal{P}_G$ has to be checked, two different implementations can be used via CK lc_minchanges_pricing_method.

- CV exact: For each path $p \in \mathcal{P}_G$ the corresponding subgraph of CG is constructed and herein the all-pairs shortest path problem is solved.
- CV heuristic: The all-pairs shortest path problem is solved in the entire Change&Go-Graph CG for all pairs of nodes. It may happen that for a pair of nodes the shortest path does not correspond to a path in $\mathcal{P}_G$. In this case a warning is returned because the computation could be wrong. Still, this procedure is much faster since the Change&Go-Graph does not need to be constructed in every iteration.

Additional to the parameters in Section 3.3.7 the following parameters are of relevance.

- CK lc_minchanges_nr_cg_paths_per_ptn_path: For the starting tableau of the column generation procedure a set of initial paths has to be computed. This parameter determines how many paths in the Change&Go-Graph are computed for each path in the PTN.
- CK lc_minchanges_cg_var_per_it: Only at most this many variables are added in each column generation iteration.
- CK lc_minchanges_max_reduced_costs_included_IP: After the column generation only variables which have reduced costs less than or equal to this value are included in the final IP.

For more information on the model, see [15].

3.3.8 Game

Running

R make lc-line-concept

with CK lc_model set to CV game results in solving a game theoretic model where each line acts as a player and aims to minimize its own (expected delay). The delay is dependent on the traffic loads along its edges, i.e, a lines tries to choose less-frequent edges. The algorithm uses a potential function to find a line plan at an equilibrium which is a system optimum. This line plan is computed by an integer program. For more information, see [40].

3.4 Ridepool Generation

A new ridepool is generated by running

R `make rpool-ride-pool`

There are three models to create a ridepool. The following parameters are needed for all models:

- CK `rpool_min_edges` Minimal number of edges in a ridepooling area. The default value is 1.
- CK `rpool_max_edges` Maximal number of edges in a ridepooling area. The default value is 4.
- CK `rpool_costs_fixed` Costs for one vehicle to operate in one ridepooling area.

A ridepooling area r is represented as a set of edges of the PTN. Those edges must form a strongly connected subgraph of the PTN. For each edge $e \in r$ there is a constant $\alpha_{r,e} \in \mathbb{R}_{\geq 0}$ specifying the fraction of the period length for which one single vehicle in r will probably be used on the edge e to serve the demand on this edge. By default, those values are computed by the following formula:

$$\alpha_{r,e} = N_e \cdot \frac{\text{CK}\ \texttt{period\_length}}{\sum_{e \in r} \delta_e \cdot N_e},$$

where δ_e is the minimal duration of each edge and N_e is defined by

$$N_e = \begin{cases} L_e + 1 & \text{if } L_e \text{ is odd,} \\ L_e & \text{else,} \end{cases}$$

and L_e denotes $\lceil \frac{load(e)}{\text{CK}\ \texttt{rc\_passengers\_per\_vehicle}} \rceil$.

Alternatively, one can also set them to an arbitrary value. Similar to lines, to each area will be assigned a number of vehicles operating in the area. The capacity of one ridepooling vehicle is controlled by CK `rc_passengers_per_vehicle`.

3.4.1 All

By setting CK `rpool_model` CV `all`, only one ridepooling area is generated consisting of all edges of the PTN.

3.4.2 Demand heuristic

For the usage of CK `rpool_model` CV `demand_heuristic`, the parameter CK `rpool_load_factor` is needed. The algorithm constructs recursively areas consisting of edges e with

$$load(e) \leq \text{CK}\ \texttt{rpool\_load\_factor} \cdot \text{CK}\ \texttt{gen\_passengers\_per\_vehicle}, \tag{3.7}$$

i.e. the load of the edge is at most a fraction of CK `rpool_load_factor` of the capacity of a line vehicle. Ridepooling can be used to cover edges with small loads, otherwise one had to establish a line for those edges which would come with big costs.

The algorithm iterates over all edges e in the PTN. Condition (3.7) is checked and if it is fulfilled, a new area r is constructed consisting only of edge e. Now we check recursively for all edges that are incident to an edge in r if (3.7) holds and add them to r until the maximal number of edges is reached or no incident edge with (3.7) is found. The area r is added to the ride pool, if it contains at least CK `rpool_min_edges` edges.

3.4.3 Tree based

The tree bases heuristic can be used by setting CK rpool_model CV tree_based. It computes a maximal spanning tree with respect to the loads on the edges. This tree may represent a possible line concept, covering mainly edges with high demand. The algorithm tries to cover all non-tree edges by ride pool areas.

We iterate over all leaf nodes of the spanning tree. For a leaf node v, we create a new area r and add all incident non-tree edges to r. In a next step, we look for non-tree edges connecting the edges in r and add them to r. If the number of edges of r is between the lower and upper bound, r is added to the ride pool.

3.4.4 Adapt Line Pool

For the ridepooling problem it can be reasonable to adapt a given line pool. By running

R rpool_adapt_lpool

we add additional lines to the existing linepool. There are two methods how to trim existing lines in order to create the additional ones. By setting CK lpool_adapt_method to CV trim_all, we add for each existing line the three lines where CK rpool_nb_edges_to_trim are trimmed, respectively, from the left side, from the right side, and from both sides. By setting CK lpool_adapt_method to CV trim_low_demand, we use a heuristic that deletes only deletes edges with low demand from existing lines. CK lpool_trim_capacity_factor defines how many times the vehicle capacity is considered as low demand. Going along a line, we add the edges to the new line until we reach an edge of low demand. Then, the new line is added to the line pool.

3.5 Ride Concept

By running

R make rc-ride-concept

a new ride concept and a new line concept are generated simultaneously.
The following parameter values are needed as input:

- CK rc_solver specifies the solver to use the IP formulations.
- CK rc_timelimit Timelimit in seconds for the solver. Value of -1 means no restriction.
- CK rc_mip_gap MIP Gap for the solver. Value of -1 means no restriction.
- CK rc_threads Number of threads to use for the solver. Value of -1 means no restriction.
- CK rc_write_lp_file Whether to write the IP formulation as a file or not.
- CK rc_passengers_per_vehicle Capacity of a single ridepooling vehicle.

A ride concept consists of ridepooling areas with an assigned number of vehicles operating in this area.

3.5.1 Lineplanning and ridepooling with fixed demand factors

By setting CK rc_model CV LPRP_ALPHA, the following IP formulation is solved:

$$\text{minimize} \quad \sum_{l\in\mathcal{L}^{\text{pool}}} c_l \cdot f_l + \sum_{r\in\mathcal{R}^{\text{pool}}} c_r \cdot v_r \tag{3.8a}$$

$$\text{subject to} \quad t_e^{\min} \leq \sum_{l\in\mathcal{L}^{\text{pool}}:e\in l} A \cdot f_l + \sum_{r\in\mathcal{R}^{\text{pool}}:e\in r} B \cdot \alpha_{r,e} \cdot v_r \leq t_e^{\max} \forall e \in E, \tag{3.8b}$$

$$f_l, v_r \in \mathbb{N} \quad \forall l \in \mathcal{L}^{\text{pool}}, r \in \mathcal{R}^{\text{pool}} \tag{3.8c}$$

c_l and c_r are the costs of establishing the line l, or operationg one vehicle in the area r, respectively. The variable f_l is the frequency of the line l and v_r is the number of vehicles in the area r. The first constraint ensures that the demand which is covered on each edge is at least the load of the edge ($t_e^{\min}$) and at most the upper (line) frequency bound of the edge multiplied by the (line) vehicle capacity ($t_e^{\max}$). A is the capacity of a line vehicle and B is the capacity of a ridepooling vehicle.

3.5.2 Lineplanning and ridepooling including demand factors as variables

By setting CK `rc_model` CV `LPRP_BETA`, the following IP formulation is solved:

$$\text{minimize} \quad \sum_{l \in \mathcal{L}^{\text{pool}}} c_l \cdot f_l + \sum_{r \in \mathcal{R}^{\text{pool}}} c_r \cdot v_r \tag{3.9a}$$

$$\text{subject to} \quad t_e^{\min} \leq \sum_{l \in \mathcal{L}^{\text{pool}}: e \in l} A \cdot f_l + \sum_{r \in \mathcal{R}^{\text{pool}}: e \in r} B \cdot \beta_{r,e} \leq t_e^{\max} \quad \forall e \in E, \tag{3.9b}$$

$$\sum_{e \in r} d_e \cdot \beta_{r,e} \leq T \cdot v_r \forall r \in \mathcal{R}^{\text{pool}}, \tag{3.9c}$$

$$\beta_{r,e} \in \mathbb{R}_{\geq 0} \quad \forall r \in \mathcal{R}^{\text{pool}}, e \in r, \tag{3.9d}$$

$$f_l, v_r \in \mathbb{N} \quad \forall l \in \mathcal{L}^{\text{pool}}, r \in \mathcal{R}^{\text{pool}} \tag{3.9e}$$

In this model, the edge-specific factors $\alpha_{r,e}$ from the first model are now determined as continuous variables $\beta_{r,e}$. The additional constraint links the number of vehicles to the factor $\beta_{r,e}$.

3.6 Timetabling

3.6.1 Modulo network simplex algorithms

There are different ways to use the Modulo Network Simplex Algorithm, depending on how to provide a starting solution:

- CK `tim_model` CV `ns_improve` It is assumed that Timetable-periodic.tim already contains a feasible starting solution; only improvement steps are taken.
- CK `tim_model` CV `csp_ns` A starting solution is found using Abscon; high reliability, small running times, but the starting solution quality is usually bad – see Section 3.6.2.
- CK `tim_model` CV `con_ns` A starting solution is found using constraint propagation; may take too long for some networks, but has good quality when it succeeds – see Section 3.6.3.
- CK `tim_model` CV `ns_cb` It is assumed that Timetable-periodic.tim already contains a feaseible starting solution. It is improved with the network simplex. Afterwards, a cycle based IP is called. CK `tim_use_old_solution` needs to be set to CV `true` such that the network simplex solution is used as a starting solution for the IP.

There are two search procedures that may be further specified, one for local search and one for fundamental search for cuts, see [12]. The first is represented by the parameter CK `tim_nws_loc_search`, the second by CK `tim_nws_tab_search`.
The possible local search algorithms are:

- CV `SINGLE_NODE_CUT`.
  The first improving single node cut that is found will be used. No further parameters have to be specified.

- CV `RANDOM_CUT`.
  Single node cuts are chosen at random, ignoring whether they are improving or not. This will be repeated 10 times. This procedure is likely to give better results than SINGLE_NODE_CUT, but will take longer. No further parameters have to be specified.

- CV `WAITING_CUT`.
  Cuts are chosen along each waiting edge cut. This will only improve
  SINGLE_NODE_CUT if the interval $[l_e, u_e]$ is especially small for waiting activities. No further parameters have to be specified.

- CV `CONNECTED_CUT`.
  Cuts are found using a local search technique. This will be repeated up to 3 times. Usually yields the best results.

These are the possible fundamental search algorithms. Their setting will have the largest impact on the quality and time consumption of the solution.

- CV `TAB_FULL`.
  All possible base exchanges are considered and the best one is chosen. This is usually quite time consuming but gives high quality results. No further parameters have to be specified. This may be considered as the default setting.

- CV `TAB_SIMPLE_TABU_SEARCH`.
  As in TAB_FULL, all base exchanges are considered, but a tabu list gives the possibility to leave local optima again. Parameters are:

  – CK `tim_nws_ts_memory`. The length of the tabu list.
  – CK `tim_nws_ts_max_iterations`. The number of iterations that are allowed before searching for a local cut.

  Because of the tabu list this algorithm is even slower than TAB_FULL but will seldom give better results because of the large number of neighbors in every step.

- CV `TAB_SIMULATED_ANNEALING`.
  Base exchanges are chosen at random and used despite of being non-improving considering a steadily cooling temperature. Parameters are:

  – CK `tim_nws_sa_init`. The starting temperature.
  – CK `tim_nws_sa_cooldown`. The cooling factor < 1.

  This algorithm may improve TAB_FULL significantly. The time consumption is about the same.

- CV `TAB_STEEPEST_SA_HYBRID`.
  A mix of TAB_FULL and TAB_SIMULATED_ANNEALING. This will usually yield the best results but takes longer than TAB_FULL. The same parameters are used as in TAB_SIMULATED_ANNEALING.

- CV `TAB_PERCENTAGE`.
  A fast algorithm that decreases the quality of the solution only slightly. Parameters are:

  – CK `tim_nws_percentage`. An integer < 100 that gives the size of the search space.

- CV `TAB_FASTEST`.
  Similar to TAB_PERCENTAGE. Parameters are

  – CK `tim_nws_min_pivot`. The minimum relative improvement a base exchange has to give.
  – CK `tim_nws_dyn_pivot`. The value by which the first parameter is multiplied if no cut fulfilling the criteria is being found.

For more information, see [13].

3.6.2 Constraint propagation

This is a way to find a feasible solution. The corresponding parameter is:

- C `tim_model; "con_prop"`

A solution is found by fixing any event time, and propagating this information through the network, thus removing infeasible solutions. A backtracking procedure is used to fix times differently, if there is no feasible solution anymore.
Parameters are:

- C `tim_cp_sortmode; "UP", "DOWN", "RANDOM"` Determines how event times are fixed. "UP" tries to tighten them as far as possible, while "DOWN" tries to relax them as far as possible. "RANDOM" chooses randomly from the set of locally feasible times, and often succeeds where the other two settings don't.

- C `tim_cp_check_feasibility; true/false` If set to true, a heuristic check for feasibility is performed before the actual constraint propagation. This takes some time, but may help to determine infeasibility.

3.6.3 Abscon

Currently not included in the release version of LinTim.
To use Abscon, set

- C `tim_model; "csp"`

The problem of finding a feasible timetable is then translated to a generic constraint satisfaction problem, and the third-party solver Abscon is started to find a feasible solution. If the problem is feasible, a feasible solution can be found relatively fast; however, its objective value tend to be worse than the one generated by constraint propagation. No parameters.

3.6.4 MATCH

To use MATCH, set

- C `tim_model; "MATCH"`

A feasible timetable is found by a matching-merge heuristic. The details of this method can be looked up in [25].

3.6.5 PESP-IP

To use the pesp ip formulation, set

- C `tim_model; "ip"`

This will try to solve an integer programming model of the periodic timetabling problem, see [33]. The IP model is implemented in Xpress and Gurobi.
For Gurobi, a solution limit, a best bound stop value, starting solution procedure and a MIPFocus are implemented (see Gurobi documentation):

- C `tim_pesp_ip_solution_limit`
- C `tim_pesp_ip_best_bound_stop`
- C `tim_pesp_ip_mip_focus`
- C `tim_use_old_solution`

For all parameters the default value of 0 will disable the respective option.
For more information on the model, see [41].

3.6.6 Cycle-based IP

To use the cycle based mip formulation, set

- C tim_model; "cb_ip"

This will try to solve a cycle based integer programming model of the periodic timetabling problem, see [33]. You can set a time limit, a thread limit and a desired gap by setting

- C tim_mip_gap
- C tim_timelimit
- C tim_threads.

The following parameter is for a (heuristic) preprocessing step where edges with few passengers are removed:

- C tim_pesp_cb_passenger_cut.

Additionally for Gurobi, a solution limit,a best bound stop value, and a MIPFocus are implemented (see Gurobi documentation):

- C tim_pesp_cb_solution_limit
- C tim_pesp_cb_best_bound_stop
- C tim_pesp_cb_mip_focus_stop.

For all parameters the default value of 0 will disable the respective option.
For more information on the model, see [41].

3.6.7 Phase 1 simplex

To use the phase 1 simplex method, set CK tim_model to CV phase-one. The idea of this model is to construct an auxiliary PESP instance that is easy to solve and a feasible solution can be converted into a feasible solution for the original problem or prove the infeasiblity of the original problem. For more information on this procedure, see [14].

3.6.8 Adaptions

Fixed times

Some timetabling models are able to handle additional restrictions on the events, namely an additional interval for each one. Note that this interval may only include one value, fixing some events to a specific time.
To use this feature, set CK tim_respect_fixed_times to CV true and add
CK filename_tim_fixed_times (Fi timetabling/Fixed-timetable-periodic.tim) for the additonal information.

3.7 Tariff Planning

Running

R make taf-tariff

will determine a new tariff minimizing the deviation from reference prices in a prespecified model. The available models are the following:

- CK taf_model CV flat, optimization model determining a flat tariff

- CK taf_model CV beeline_distance, optimization model determining an affine beeline distance tariff
- CK taf_model CV network_distance, optimization model determining an affine network distance tariff
- CK taf_model CV zone, optimization model determining a zone tariff

All models optimize prices such that the new tariff is close to the reference prices, which can e.g. be obtained from a former tariff, given in CK filename_tariff_reference_price_matrix_file (Fi basis/Reference-Price-Matrix.giv).

3.7.1 General Remarks

In tariff planning we only consider node pairs with different nodes and call them non-trivial OD pairs:

$$D := (V \times V) \setminus \{(v, v) : v \in V\}.$$

Tariff planning always produces a price matrix file CK filename_tariff_price_matrix_file (Fi tariff/Price-Matrix.taf) as output. Prices for trivial OD pairs, i.e. pairs with the same origin and destination node, are set to zero.

Objective function and weight options

In each of the available tariff models there are two options for the objective function. For each option one can chose one out of three possible weight options. The objective function is determined by CK taf_objective and CK taf_weights_objective.
If CK taf_objective has the value

- CV sum_absolute_deviation, the objective function is the weighted sum of absolute deviations between the new prices and the reference prices (see equation (3.10)),
- CV max_absolute_deviation, the objective function is the weighted maximum absolute deviation between the new prices and the reference prices (see equation (3.11)).

If CK taf_weights_objective has the value

- CV od, the price deviations are weighted by the OD data,
- CV unit, the price deviations have weight 1,
- CV reference_inverse, the price deviations are weighted by the inverse of the given reference prices.

This results in one of the two objectives

$$\sum_{d \in D} C_d |r_d - \pi_d|, \tag{3.10}$$

$$\max_{d \in D} C_d |r_d - \pi_d| \tag{3.11}$$

with the above defined set D of all non-trivial OD pairs and reference prices r_d for all non-trivial OD pairs $d \in D$. The new prices that have to be computed are denoted by π_d. The weights C_d refer either to the OD data if CK taf_weights_objective CV od, or is $C_d = 1$ for all $d \in D$ if CV unit, or is $C_d = \frac{1}{r_d}$ for all $d \in D$ if CV reference_inverse. Both objective functions are applied in a linearized form in the programming formulations.

Routing options

In the tariff models CV distance and CV zone the prices are optimized with respect to given passenger paths in the PTN. Which paths are used is determined by CK taf_routing_generation:

- CV fastest-paths, a new routing using fastest paths with respect to the lower time bounds on the edges is created,
- CV read-all, a routing for all non-trivial OD pairs given in CK filename_routing_ptn_input (Fi basis/Routing-ptn.giv) is read and used,
- CV read-partial-fill, a partial routing given in CK filename_routing_ptn_input (Fi basis/Routing-ptn.giv) is read. Unspecified paths for non-trivial OD pairs are filled with fastest paths with respect to the lower time bounds on the edges.

In all cases the used routing is stored to CK filename_routing_ptn_output (Fi basis/Routing-ptn.giv).

Solver options

The following parameters control the behavior of the solver in all models.

- CK taf_solver determines the solver to be used. Note that currently only Gurobi is supported.
- CK taf_threads determines the maximum number of threads to use for the solver (-1=use default value, i.e., no restriction). Note that this will only be used for a possible solver integration of the chosen model, not for the rest of the algorithm.
- CK taf_timelimit sets a time limit for the solver in seconds (-1=use default value).
- CK taf_write_lp_file determines whether to write the lp file of the model to solve.
- CK taf_mip_gap sets the MIP optimization gap for the solver. The solver will terminate with an optimal solution if the gap between lower and upper objective bound is less than this value times the absolute value of the incumbent objective value.

3.7.2 Flat Tariff

Running

R `make taf-tariff`

with CK taf_modelCV flat determines a new flat tariff, i.e. a fixed price f for all paths W in the given PTN.
The flat tariff model solves the following linear program to optimality:

$$\begin{aligned} \min \quad & g(f) \\ \text{s.t.} \quad & f \geq 0, \end{aligned}$$

where the objective function g is determined by CK taf_objective and CK taf_weights_objective as described in Section 3.7.1, i.e.

$$g(f) = \begin{cases} \sum_{d \in D} C_d |r_d - f| & \text{for CV sum\_absolute\_deviation,} \\ \max_{d \in D} C_d |r_d - f| & \text{for CV max\_absolute\_deviation.} \end{cases}$$

The computed flat price is written as CK taf_flat_price to the State-Config-file CK filename_state_config (Fi basis/State-Config.cnf). The following file is produced as output:

- CK filename_tariff_price_matrix_file (Fi tariff/Price-Matrix.taf), price matrix containing the price f for each OD pair.

3.7.3 Distance Tariffs

Running

R make taf-tariff

with CK taf_model CV beeline_distance or CV network_distance results in determining a new distance-based tariff, i.e. the price $p(W)$ for travelling along the path W in the given PTN is determined by $p(W) = f + l(W) \cdot p$ where $f \geq 0$ are fixed costs and $p \geq 0$ is a price factor that is multiplied with the distance $l(W)$, which is either the Euclidean distance between the start and the end station of the path or the sum of all edge lengths of the path.
The distance-based model solves the following program to optimality:

$$\begin{array}{ll} \min & g(f,p) \\ \text{s. t.} & f,p \geq 0, \end{array}$$

where the objective function g is determined by CK taf_model, CK taf_objective and CK taf_weights_objective. This results in one of the two objectives:

$$g(f) = \begin{cases} \sum\limits_{d \in D} C_d |r_d - (f + l_d \cdot p)| & \text{for CV sum\_absolute\_deviation,} \\ \max\limits_{d \in D} C_d |r_d - (f + l_d \cdot p)| & \text{for CV max\_absolute\_deviation.} \end{cases}$$

as described in Section 3.7.1. The value of l_d is determined by CK taf_model:

- CV beeline_distance, the distance l_d of a non-trivial OD pair $d \in D$ is calculated as the Euclidean distance in km in the plane,
- CV network_distance, the distance l_d of a non-trivial OD pair $d \in D$ is the length (in terms of edge length) of the path of this node pair in the routing. The parameter CK taf_routing_generation specifies how the routing is determined as described in Section 3.7.1.

The computed fixed price f and the price factor p are written as CK taf_fixed_costs and CK taf_factor_costs to the State-Config-file CK filename_state_config (Fi basis/State-Config.cnf). The following file is produced as output:

- CK filename_tariff_price_matrix_file (Fi tariff/Price-Matrix.taf), price matrix containing the prices for each OD pair.

3.7.4 Zones

Running

R make taf-tariff

with CK taf_model CV zone determines a new zone tariff by determining zones and a price list. The set of zones is a partition of the set of nodes of the PTN. Prices are given as a price list that assigns a price to the number of traversed zones. The price of a path depends on the number of traversed zones on that path. We say that a zone is traversed if a node of this zone is part of the path, in particular the zones of the start node and end node of a path are traversed. The price for traversing more zones than the maximal specified number in the price list is just the price for traversing the maximal specified number of zones.
It is also possible to determine only a price list for given zones or only zones for a given price list.
The parameters are

- CK taf_zone_counting: Specifies how the number of traversed zones is counted. If CV single, then each different zone is only counted once. If CV multiple, then a zone is counted each time that it is entered. For example consider the path from station 1 to station 3 in the PTNs with zones given in Figure 3.1. In the multiple counting case, the number of traversed zones is 3 in both PTNs. In the single counting case, the number of traversed zones is 3 in the left PTN and 2 in the right PTN because there are only two different zones on the path and the reentry is not counted.

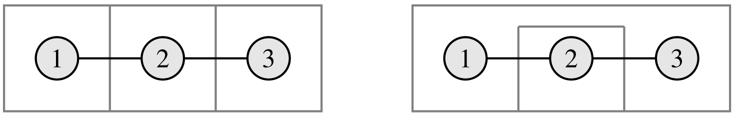


Figure 3.1: PTNs with zones

- CK `taf_zone_n_zones` integer number specifying the maximum number of zones,
- CK `taf_zone_enforce_all_zones` boolean, determines whether exactly CK `taf_zone_n_zones`-many zones (CV `true`) or at most that many zones (CV `false`) must be determined,
- CK `taf_zone_connected` boolean, specifies whether the subgraph of a zone, induced by the nodes assigned to the zone, needs to be connected (in case of a directed graph it is weakly connected),
- CK `taf_zone_enforce_no_elongation` boolean, determines whether the no-elongation property must be satisfied. This property ensures, that it is never cheaper for passengers to buy a ticket for more zones than they actually want to travel through. Let p_k be the price of a path that traverses k zones. The no-elongation property is satisfied if it holds that

  $$p_k \leq p_{k+1} \qquad \text{for all } k \in \{1, ..., (\text{CK } \texttt{taf\_zone\_n\_zones}) - 1\}.$$

- CK `taf_zone_enforce_no_stopover` boolean, determines whether the no-stopover property must be satisfied. This property ensures that it is never cheaper for a passenger to buy two seperate tickets for one journey and combine them instead of buying one ticket for the whole journey. Let again p_k be the price of a path that traverses k zones. The no-stopover property in the case of single counting is

  $$p_k \leq p_i + p_j \qquad \text{for all } k \in \mathbb{N}_{\geq 1}, i, j \in \{1, ..., k\} \text{ with } i + j \geq k + 1.$$

  In the case of multiple counting the property holds if

  $$p_k \leq p_i + p_j \qquad \text{for all } k \in \mathbb{N}_{\geq 1}, i, j \in \{1, ..., k\} \text{ with } i + j = k + 1.$$

- CK `taf_zone_symmetry_breaking`, determines which symmetry breaking model (see below) should be used. Possible values are CV `A`, CV `B` and CV `NONE`.
- CK `taf_routing_generation`, determines which routing should be used, see Section 3.7.1.
- CK `taf_zone_only_zones` boolean, specifies whether only zones based on given prices must be computed,
- CK `taf_zone_only_prices` boolean, specifies whether only prices based on given zones must be computed.

The objective in the zone model is to minimize the objective function $g(\pi)$ such that the above mentioned constraints are satisfied. The objective function is determined by CK `taf_objective` and CK `taf_weights_objective` as described in Section 3.7.1, i.e.

$$g(\pi) = \begin{cases} \sum_{d \in D} C_d |r_d - \pi_d| & \text{for CV } \texttt{sum\_absolute\_deviation}, \\ \max_{d \in D} C_d |r_d - \pi_d| & \text{for CV } \texttt{max\_absolute\_deviation}. \end{cases}$$

Here π_d refers to the price of OD pair d for travelling along the path given in the routing determined by CK `taf_routing_generation` (see Section 3.7.1).
The results are written to:

- CK filename_tariff_price_matrix_file (Fi tariff/Price-Matrix.taf), containing the prices,
- CK filename_tariff_zone_file (Fi tariff/Zones.taf), containing the assignment of stops to zones and
- CK filename_tariff_zone_price_file (Fi tariff/Zone-Prices.taf), price list of the zone tariff.

Symmetry Breaking

When determining a zone tariff some feasible solutions may only vary in name. Therefore symmetry breaking constraints can be introduced to the MILP solving the problem.
Let $x_{vz} = 1$ if and only if the stop with stop-ID v is allocated to the zone with zone-ID z and 0 else. Then the following constraints can be introduced:

$$x_{vz} = 0 \qquad \text{for all } v \in \{1, \ldots, \min\{n, N\}\}, z \in \{v+1, \ldots, N\} \tag{3.12}$$

$$x_{vz} \leq \sum_{k=1}^{v-1} x_{k,z-1} \qquad \text{for all } v \in \{3, \ldots, n\}, z \in \{3, \ldots, N\} \tag{3.13}$$

$$\sum_{v \in V} x_{vz} \geq \sum_{v \in V} x_{v,z+1} \qquad \text{for all } z \in \{1, \ldots, N-1\} \tag{3.14}$$

Here, N denotes the maximum number of zones (CK taf_zone_n_zones) and n is the number of nodes in the PTN.
The first constraint (3.12) ensures that the i-th stop can only be in the first i zones. The second one (3.13) ensures that a stop is only allowed in a zone if a node with a smaller stop-ID is in the zone with the next smaller zone-ID. The third one (3.14) orders the zones descending by size such that the one with the smalles zone-ID is the biggest in terms of number of nodes.
The parameter CV taf_zone_symmetry_breaking determines which one of them will be used. Three options are available:

- CK A, implementing (3.12) and (3.13). This seems to be the fastest.
- CK B, implementing (3.14). This seems to be slower.
- CK NONE, no symmetry breaking constraints are applied.

Only Prices

If only prices should be optimized for given zones, CK taf_zone_only_prices (boolean) must be set to CV true. By default it is CV false.
If CV true, the same MILP as described above is solved, but the zones are fixed. Therefore a zone file CK filename_tariff_zone_file (Fi tariff/Zones.taf) must be given, otherwise the algorithm fails.

Only Zones

If only zones should be optimized for given prices, CK taf_zone_only_zones (boolean) must be set to CV true. By default it is CV false.
If CV true, the same MILP as described above is solved, but the prices for travelling through a certain number of zones are fixed. Therefore a zone-price file CK filename_tariff_zone_price_file (Fi tariff/Zone-Prices.taf) must be given, otherwise the algorithm fails.

3.8 Vehicle Scheduling

The vehicle scheduling step can be invoked via

R `make vs-vehicle-schedules`

It assumes that there is an aperiodic Event-Activity Network with a given timetable for the aperiodic events and a set of trips to cover, which can be generated from a periodic timetable by the auxiliary rollout algorithm (see Section 4.9).

3.8.1 Mdm1

Running

R `make vs-vehicle-schedules`

with the CK `vs_model` set to CV `MDM1` will result in running a model minimizing the number of vehicles used in the vehicle schedule. For two consecutive trips the last station of the first trip has to be equal to the first station of the second trip. A depot, given by CK `vs_depot_index`, is considered. For more information on the model, see [2].

3.8.2 Mdm2

Running

R `make vs-vehicle-schedules`

with the CK `vs_model` set to CV `MDM2` will result in running a model that is equivalent to CV `MDM1`, except that no depot is considered. For more information on the model, see [2].

3.8.3 Assignment model

Running

R `make vs-vehicle-schedules`

with the CK `vs_model` set to CV `ASSIGNMENT_MODEL` will result in running a model minimizing the overall costs, considering vehicle costs(CK `vs_vehicle_costs`) and empty meters costs (given by the respective distance in time). A depot, given by CK `vs_depot_index`, can be considered.
Two consecutive trips can have different end and start stations respectively. Whether they can be connected relies on the end time of trip on, the start time of trip two, the distance between the two respective stations (in terms of minimal running times on shortest path) and a minimal turnover time (CK `vs_turn_over_time`). Note that the turnover time is not a simple restriction on the time between two connected consecutive trips, but includes the time needed to travel to the later station, i.e., it is the designated time the vehicle needs to be available at the later station before departing again.
For more information on the model, see [2].

3.8.4 Transportation model

Running

R `make vs-vehicle-schedules`

with the CK `vs_model` set to CV `TRANSPORTATION_MODEL` will result in running a model minimizing the overall costs, considering vehicle costs by driving to/from the depot, given by CK `vs_depot_index`, and (fixed) penalty costs CK `vs_penalty_costs` for not giving service on a trip. For more information on the model, see [2].

3.8.5 Network flow model

Running

R `make vs-vehicle-schedules`

with the CK `vs_model` set to CV `NETWORK_FLOW_MODEL` will result in running a model minimizing the overall costs considering both vehicle and empty meters costs. A depot, given by CK `vs_depot_index`, is considered. The number of vehicles can be bounded. For more information on the model, see [2].

3.8.6 Canal model

Running

R `make vs-vehicle-schedules`

with the CK `vs_model` set to CV `CANAL_MODEL` will result in running a more detailed version of CV `ASSIGNMENT_MODEL` incorporating the actual waiting times at every node and furthermore the considered period can be extended. Thus, each station can be regarded as a depot if trains from one day wait at the station for a service from that station the next day. Also, depot and maintenance decisions for locations which are farther away from the actual station can be taken. The minimal turnover time (CK `vs_turn_over_time`) will be respected. For more information on the model, see [43].

3.8.7 Line-based

Running

R `make vs-vehicle-schedules`

with the CK `vs_model` set to CV `LINE-BASED` will result in running a model based on line planning only. This model runs with the CK `vs_line_based_method` set to CV `4`, CV `3` or CV `2` and CK `vs_line_based_alpha` set to CV `0.3`. Here the CK `vs_line_based_method` describes the program type and the CK `vs_line_based_alpha` describes the value of α. For more information on the model, see [18].

3.8.8 Simple

Running

R `make vs-vehicle-schedules`

with the CK `vs_model` set to CV `SIMPLE` will result in a homogeneous vehicle schedule, i.e., all vehicles will serve only one line, back and forth.

3.8.9 IP model

Running

R `make vs-vehicle-schedules`

with the CK `vs_model` set to CV `IP` will result in a simple ip model to determine a cost efficient vehicle schedule. Trips are determined compatible, if the shortest path w.r.t. the lower bounds is sufficient to serve the trips after each other. A depot, given by CK `vs_depot_index` can be considered. Currently, only CV `GUROBI` is allowed as CK `vs_solver`. A time limit for the ip model can be set via CK `vs_timelimit`, where CV `-1` disables this option. The cost of a vehicle is determined using CK `vs_vehicle_costs` and the cost of an empty trip by CK `vs_eval_cost_factor_empty_trips_length` and CK `vs_eval_cost_factor_empty_trips_duration`. The minimal turnover time (CK `vs_turn_over_time`) will be respected. For more information on the model, see [2].

3.9 Delay Management

The delay-management step can be invoked via

R `make dm-disposition-timetable`

It assumes that there is an aperiodic Event-Activity Network with a given timetable for the aperiodic events, which can be generated from a periodic timetable by the auxiliary rollout algorithm (see Section 4.9), and some primary delays on events and/or activities (see Section 4.10). The lower bounds on the drive, wait (dwell) and fixed-circulation activities can be automatically reduced to account for a globally applied buffer that is contained in the lower bounds but may be exploited in case of delays. To this end, the parameter CK `DM_lower_bound_reduction_factor` can be set to a value below CV `1.0`.
Note that during all these steps – in contrast to preceding planning steps like line planning or periodic timetabling – time intervals are measured in seconds, points in time in seconds since 0:00. E.g., if an activity has a lower bound of 60, this means 60 seconds, and if the time of an event is 28 800, this means 08:00 a.m.

The following parameters are used by all methods:

- CK `DM_verbose`: enable verbose output
- CK `DM_enable_consistency_checks`: enable (time-consuming) consistency checks of input data and results
- CK `DM_debug`: enable debugging output (also enables verbose output and consistency checks)

3.9.1 Propagate

The mere propagation of delays to produce a feasible disposition timetable is done when CK `DM_method` is set to CV `propagate`. After applying the given delays on events and on the lower bounds on activity durations, the (rolled-out) events are traversed in a topological sorting. Upon visit of each event, its time becomes fixed (since, due to the topological sorting, all events taking place earlier have been fixed before) and its successor events (targets of outgoing activities) are delayed as much as necessary to fulfill the lower bound on the duration of the respective activity.
During this propagation procedure, change activities can be cut off (so that delays will not propagate along them) based on a maximum waiting time: If the target event of a change activity would be delayed by more than CK `DM_propagate_maxwait` seconds, then this change activity is not respected at all. If all change activities shall be maintained, this parameter must be set to a very large value (e.g. the duration of the time horizon according to the rollout parameters, in seconds).
Furthermore, the headway constraints can be swapped around in those cases where the train that was originally scheduled first is so late that the train that was originally scheduled to go second can actually go first without affecting the train originally scheduled first. To enable this swapping of headways, CK `DM_propagate_swapHeadways` must be set to CV `true`.

3.9.2 Integer-Linear-Programming based methods

The aim of delay management is to react to delays in such a way that the effect on the passengers is minimal. To this end, one has to decide for each connection whether it should be maintained or not (i.e., if a connecting train waits for a delayed feeder train or not) and for each pair of trains using the same piece of track which train should go first. The delay management problem is for example described in [27]. The following parameters are used by all delay management algorithms:

- CK `DM_solver`: Defines which MIP solver should be used. Possible choices are `Gurobi` and `Xpress`. Please note that your environment (e.g. the `CLASSPATH` variable) has to be set up properly.
- CK `DM_solver_time_limit`: Time limit for the MIP solver in seconds – after this time, the solver is interrupted and the best solution found so far is used. If set to 0, no time limit is used.

- CK DM_lower_bound_reduction_factor: Describes how much buffer time is included in the minimal duration of the activities in the event-activity network. The lower bounds read from the input are multiplied with this number, so setting CK DM_lower_bound_reduction_factor to 1 does not change the lower bounds, while setting it to a value in $]0, 1[$ reduces the lower bound of all activities.

The variable CK DM_method defines which algorithm should be used to solve the delay management problem:

CV **DM1:** Computes an optimal solution of the MIP formulation (DM1) presented in [34, 35]. This is the slowest algorithm provided. To perform the calculation, the rollout must have been done where the parameter CK rollout_passenger_paths is set to CV true since the algorithm minimizes the delays on the passenger paths given in CK default_passenger_paths_file.

CV **DM2:** Computes an optimal solution of the MIP formulation (DM2) presented in [34, 35]. This is an approximation for (DM1) and a bit faster but still far slower than the other algorithms.

CV **DM2-pre:** The same as CV DM2, but with a preprocessing step. Computes an optimal solution of the MIP formulation (DM2) after applying Algorithm 3.2 from [27, p. 38] for reducing the size of the event-activity network. For more information, see [34, 35].

CV **FSFS:** "First scheduled, first served" – fixes the forward headways, deletes the backward headways, and solves the resulting uncapacitated delay management problem with fixed headways to optimality using DM1 or DM2, as specified in CK DM_opt_method_for_heuristic, see Algorithm 4.1 in [27, p. 56]. For more information, see [27, 28]. *Heuristic algorithm – might not find the global optimum.*

CV **FRFS:** "First rescheduled, first served" – fixes the headways according to the optimal solution of the corresponding uncapacitated delay management problem, then solves the resulting uncapacitated delay management problem with fixed headways to optimality using CV DM1 or CV DM2, as specified in CK DM_opt_method_for_heuristic, see Algorithm 4.2 in [27, p. 57]. For more information, see [27, 28]. *Heuristic algorithm – might not find the global optimum.*

CV **EARLYFIX:** Similar to CV FRFS – but also fixes the changing activities according to the solution of the corresponding uncapacitated delay management problem by using CV DM1 or CV DM2, as specified in CK DM_opt_method_for_heuristic, see Algorithm 4.3 in [27, p. 57]. For more information, see [27, 28]. *Heuristic algorithm – might not find the global optimum. Note that* CV *FRFS is always at least as good as this method* [27, Lemma 4.5]*, while this method might be faster on instances with many changing activities.*

CV **PRIORITY:** Similar to CV FSFS – but also fixes the "most important" connections (the variable CK DM_method_prio_percentage defines how many percent of all connections should be maintained), see Algorithm 4.4 in [27, p. 57]. For more information, see [27, 28]. *Heuristic algorithm – might not find the global optimum. Uses* CV *DM1 or* CV *DM2, as specified in* CK *DM_opt_method_for_heuristic for optimization. Note that* CV *FSFS is always at least as good as this method* [27, Lemma 4.6]*, while this method might be faster on instances with many changing activities.*

CV **PRIOREPAIR:** Fixes the connections according to their weights like CV PRIORITY, relaxes the headway constraints, and solves the resulting problem using CV DM1 or CV DM2, as specified in CK DM_opt_method_for_heuristic. Then it uses this solution to fix the headways and solves the problem again (again CV DM1 or CV DM2) (see Algorithm 4.7 in [27, p. 68]). For more information, see [27, 28]. *Heuristic algorithm – might not find the global optimum.*

CV **best-of-all:** Runs CV FSFS, CV FRFS and CV PRIOREPAIR consecutively and takes the best solution. Due to [27, Lemma 4.5] and [27, Lemma 4.6], it's sufficient to run CV FSFS, CV FRFS, and CV PRIOREPAIR and to ignore CV EARLYFIX and CV PRIORITY. Uses CV DM1 or CV DM2, as specified in CK DM_opt_method_for_heuristic for optimization. If

CK DM_best_of_all_write_objectives is set to CV true, this will output all objective values of the different methods into CK filename_dm_best_of_all_objectives (Fi statistic/dm_objectives.sta). For more information, see [28]. *Heuristic algorithm – might not find the global optimum.*

CV **PASSENGERFIX:** Uses a IP to fix the headways of passenger paths with the most passenger weight sum possible without contradictions and solves the following smaller problem with CV DM1. Note that all headways on a path get fixed if and only if no headway contradicts the earlier decisions. Otherwise no headway gets fixed. Same requirement as CV DM1. The IP is very big and slow!

CV **PASSENGERPRIOFIX:** A heuristic for the IP of CV PASSENGERFIX, fixes the headways of the first CK DM_method_prio_percentage percent of the passenger paths sorted by weight. Fixes any headway for a path only if this is possible without contradiction to the previous paths. After that, it solves the smaller problem with CV DM1. Same requirement as this method.

CV **FIXFSFS:** First uses the fixing method of CV PASSENGERPRIOFIX on as many paths as possible, again sorted by weight. After that it uses the fixing method of CV FSFS to fix the remaining headways. After that, it solves the reduced problem with CV DM1 with the same requirement.

CV **FIXFRFS:** Like CV FIXFSFS, just uses the fixing method of CV FRFS instead of CV FSFS

3.10 Integrated Planning

The common parameters for all integrated programs are the following. Whether these parameters are used is dependent on the specific problems.

CK **int_max_threads** The maximal number of cpu threads used for optimization

CK **int_factor_travel_time** The objective factor for the travel time

CK **int_factor_drive_time** The objective factor for the drive time of the passengers

CK **int_factor_transfer_time** The objective factor for the transfer time of the passengers

CK **int_factor_wait_time** The objective factor for the waiting time of the passengers

CK **int_factor_penalty_time_slice** The penalty for changing time slices for the passengers. Only applicable on models respecting time slices. Only applicable for models with passenger routing.

CK **int_time_slices** The number of time slices to use. Only applicable for models with passenger routing.

CK **int_number_of_periods** The number of periods to consider the vehicle schedule for. Lines will not be cut off at the end of the planning period. Only applicable for models with vehicle scheduling.

CK **int_restrict_to_system_frequency** Whether to use a system frequency, i.e., a common divisor for all frequency values. Only applicable for models with line planning.

CK **int_system_frequency** The value for the system frequency, i.e., the common divisor for all frequency values. Only applicable for models with line planning.

CK **int_check_lower_frequencies** Whether the model should respect the lower frequency bounds, i.e., the minimal number of times edges in the public transport network need to be covered. Only applicable for models with line planning.

CK **int_check_upper_frequencies** Whether the model should respect the upper frequency bounds, i.e., the maximal number of times edges in the public transport network may be covered. Only applicable for models with line planning.

CK **int_set_starting_timetable** Whether to set the starting values for timetabling. Only applicable for models not containing line planning.

CK **int_solver_type** The solver to use.

3.10.1 Integrated timetabling and passenger routing

An implementation of the integrated periodic timetabling and passenger routing problem. For details on the model, see [30].

CK **tim_pass_use_preprocessing** Whether to use an exact preprocessing method to reduce the problem size before optimization.

CK **tim_pass_use_cycle_base** Whether to use a cycle-base formulation. This is normally much faster.

CK **tim_pass_restrict_transfer_stations** Whether to use an auxiliary IP to restrict the transfer stations. This method is only exact if all drive- and wait-activities are fixed.

CK **tim_pass_add_fixed_passenger_paths** Whether to add the non-routed passengers as fixed weigths to the model.

CK **tim_pass_number_of_routed_od_pairs** The number of routed od pairs.

CK **tim_pass_choose_routed_od_pairs** How to choose the routed od pairs. The following methods are possible:

- CV **POTENTIAL** Choose the od pairs with the most potential, i.e., compute the shortest path w.r.t. lower bounds on the EAN, evaluate these paths w.r.t. the difference of upper and lower bound on each activity and weight the result by the number of passengers of the od pair.
- CV **LARGEST_WEIGHT** Choose the od pairs with the largest weight.
- CV **SMALLEST_WEIGHT** Choose the od pairs with the smallest weight.
- CV **LARGEST_WEIGHT_WITH_TRANSFER** Choose the od pairs with the largest weight that additionally have at least one transfer in their shortest path w.r.t. the lower bounds on the EAN.
- CV **LARGEST_DISTANCE** Choose the od pairs with the largest euclidian distance.
- CV **DIFF** Similar to CV POTENTIAL but without the additional scaling by the number of passengers.
- CV **RANDOM** Random.

CK **tim_pass_time_limit** The time limit for the optimization.

CK **tim_pass_mip_gap** The mip gap for the optimization.

CK **tim_pass_write_lp_output** Whether to write the lp output. Will additionally compute an IIS for infeasible programs.

3.10.2 Integrated timetabling and aperiodic vehicle scheduling

Solve the integrated periodic timetabling and aperiodic vehicle scheduling problem. Includes passenger routing for the timetabling step. For more information, see [30].

CK **tim_veh_allow_empty_trips** Whether to allow empty trips in the vehicle schedule.

CK **tim_veh_use_lower_bound** Whether to include an additional lower bound on the objective function.

CK **tim_veh_time_limit** The time limit for the optimization.

CK **tim_veh_mip_gap** The mip gap for the optimization.

CK **tim_veh_write_lp_output** Whether to write the lp output. Will additionally compute an IIS for infeasible programs.

3.10.3 Integrated line planning and timetabling

Solve the integrated line planning and periodic timetabling problem. Includes passenger routing for the timetabling stage. For more information, see [30].

- CK **lin_tim_pass_use_preprocessing** Whether to use an exact preprocessing method to reduce the problem size before optimization.
- CK **lin_tim_pass_add_fixed_passenger_paths** Whether to add the non-routed passengers as fixed weigths to the model.
- CK **lin_tim_pass_number_of_routed_od_pairs** The number of routed od pairs.
- CK **lin_tim_pass_factor_line_cost** The factor for the line costs.
- CK **lin_tim_pass_time_limit** The time limit for the optimization.
- CK **lin_tim_pass_mip_gap** The mip gap for the optimization.
- CK **lin_tim_pass_write_lp_output** Whether to write the lp output. Will additionally compute an IIS for infeasible programs.
- CK **lin_tim_pass_choose_routed_od_pairs** How to choose the routed od pairs. The following methods are possible:
  - CV **LARGEST_WEIGHT** Choose the od pairs with the smallest weight.
  - CV **SMALLEST_WEIGHT** Choose the od pairs with the smallest weight.
  - CV **LARGEST_DISTANCE** Choose the od pairs with the largest euclidian distance.
  - CV **RANDOM** Random.

3.10.4 Integrated line planning, timetabling and vehicle scheduling

Solve the integrated line planning, periodic timetabling and aperiodic vehicle scheduling problem. Includes passenger routing for the timetabling stage. For more information, see [30].

- CK **lin_tim_pass_veh_use_preprocessing** Whether to use an exact preprocessing method to reduce the problem size before optimization.
- CK **lin_tim_pass_veh_add_fixed_passenger_paths** Whether to add the non-routed passengers as fixed weigths to the model.
- CK **lin_tim_pass_veh_number_of_routed_od_pairs** The number of routed od pairs.
- CK **lin_tim_pass_veh_time_limit** The time limit for the optimization.
- CK **lin_tim_pass_veh_mip_gap** The mip gap for the optimization.
- CK **lin_tim_pass_veh_write_lp_output** Whether to write the lp output. Will additionally compute an IIS for infeasible programs.
- CK **lin_tim_pass_choose_routed_od_pairs** How to choose the routed od pairs. The following methods are possible:
  - CV **LARGEST_WEIGHT** Choose the od pairs with the smallest weight.
  - CV **SMALLEST_WEIGHT** Choose the od pairs with the smallest weight.
  - CV **LARGEST_DISTANCE** Choose the od pairs with the largest euclidian distance.
  - CV **RANDOM** Random.

3.10.5 Robust Timetabling and Vehicle Scheduling Using Machine Learning

This algorithm tries to improve the robustness of the given timetable and vehicle schedule by using a machine-learned oracle and meta-heuristics for robustness prediction and determining possible improvement steps. For more information, see [22].
For this model to work, a machine-learned oracle needs to be trained first. This step is not part of LinTim. For more information on the training process, see [21]. To compute the key features described there and in the publication above, use

R `make int-rob-ml-key-features`

This will create CK `filename_robustness_tensor_file_name` (Fi `statistic/data.tensor`) which can then be used for training externally.
The following configuration parameters determine the behavior of the algorithm.

CK **`ean_change_penalty`** the change penalty to respect when routing passengers

CK **`ean_default_maximal_change_time`** the maximal change time. Will be used when CK `rob_create_missing_changes` is set to CV `true`

CK **`ean_default_minimal_change_time`** the minimal change time. Will be used when CK `rob_create_missing_changes` is set to CV `true`

CK **`filename_robustness_ml_model`** the filename of the machine-learned model to consider. Will only be used when CK `rob_use_api_for_prediction` is set to CV `false`.

CK **`gen_passengers_per_vehicle`** the vehicle capacity

CK **`rob_max_changes`** the maximal changes allowed in the key feature vector used for rbustness prediction

CK **`rob_max_group_size`** the maximal passenger group size to route. Grouping passengers may improve routing runtime.

CK **`rob_max_iteration`** the maximal number of iterations the algorithm is allowed to perform before aborting

CK **`rob_max_travel_time`** the maximal travel time in the key feature vector used for robustness prediction

CK **`rob_max_turnaround_time`** the maximale turnaround time allowed in the key feature vector used for rbustness prediction

CK **`rob_output_every_solution`** whether every solution should be written to disk. If set to CV `true`, a subfolder CK `rob_debug_output_path` will be used to store the result of every iteration. Note that this may take up a large amount of disk space when used on large datasets with many iterations.

CK **`rob_reroute_interval`** the interval to reroute, i.e., setting this to CV 5 will result in rerouting taking place every fifth iteration. Inceasing this value may improve the runtime but decrease the prediction quality.

CK **`rob_routing_end_time`** the time when the routing of the passengers should stop. You should allow enough time for your transportation system to settle after ending the routing of passengers. Events outside of the routing window will not be considered for the key features. Note that we will consider at most 4 hours, i.e., setting this higher will have no effect.

CK **`rob_routing_start_time`** the time when the routing of the passengers should start. You should allow enough "startup" time for your transportation system to settle before starting the routing of passengers.

CK **rob_start_solutions_file** the start solution file to read. Start solutions are read for the genetic algorithm (i.e. CK rob_use_genetic_algorithm CV true) or when a specific start solution should be used for the local search (i.e. CK rob_local_search_start_solution$\neq -1$). The file should be a zip file containing the possible start solutions each in a seperate folder, named e.g. Fi A_10 for start solution with index 10. In this folder should be a valid LinTim dataset.

CK **rob_use_api_for_prediction** will not read the model directly but use an api provided on port CK rob_api_port. The algorithm will send the key feature vector seperated with ";" and expect the resulting values as a ";" seperated vector as well, followed by "
n". The average of the received vector will be used as the predicition value for the given key feature vector.

CK **rob_use_single_ann_models** will not read a single neural network model but one for each of the four robustness objectives. Will insert "_1", ..., "_4" into the filename, i.e., for CV model.h5 in CK filename_robustness_ml_model, this will try to read Fi model_1.h5,... Fi model_4.h5.

Specific for the local search, i.e., with CK rob_use_genetic_algorithm set to CV false

CK **rob_ls_allowed_travel_time_increase** the allowed travel time increase of the passengers, i.e., when this is set to CV 1.1 the algorithm allows an average travel time increase of 10% before aborting

CK **rob_ls_buffer_increase_per_step** the amount of buffer to add in each step, in seconds.

CK **_ls_candidates_per_type** determines how many candidates per activity type should be added in each neighboorhood

CK **rob_ls_change_weight** the weight factor for change activities in the neighborhood selection process

CK **rob_ls_drive_weight** the weight factor for drive activities in the neighborhood selection process

CK **rob_ls_propagate_slack_use_percentage** determindes the propagation of slack on activities. When set to CV true, CK rob_ls_propagate_slack_percentage gives the ratio of the activity slack to reduce in each step. When set to CV false, a minimal slack time of CK rob_ls_propagate_slack_min_time will be used instead.

CK **rob_ls_select_by_ratio** when set to true, not the absolute robustness improvement but the robustness improvement divided by the lost passenger travel time will be used to determine the best solution in each neighboorhood.

CK **rob_ls_turn_weight** the weight factor for turnover activities in the neighborhood selection process

CK **rob_ls_use_periodic_timetabling** whether to maintain a periodic timetable in every step or not

CK **rob_ls_wait_weight** the weight factor for wait activities in the neighborhood selection process

Specific for the genetic algorithm, i.e., CK rob_use_genetic_algorithm set to CV true

CK **rob_ga_breedings_per_iteration** the number of breedings to perform per generation

CK **rob_ga_mutation_amount** the maximal amount of mutation to use in each mutation

CK **rob_ga_number_mutations_at_breeding** the number of vector entries to mutate during the breeding process

CK **rob_ga_number_mutations_at_start** the number of vector entries to mutate in the start solutions

CK **rob_ga_number_start_solutions** the number of start solutions to use.

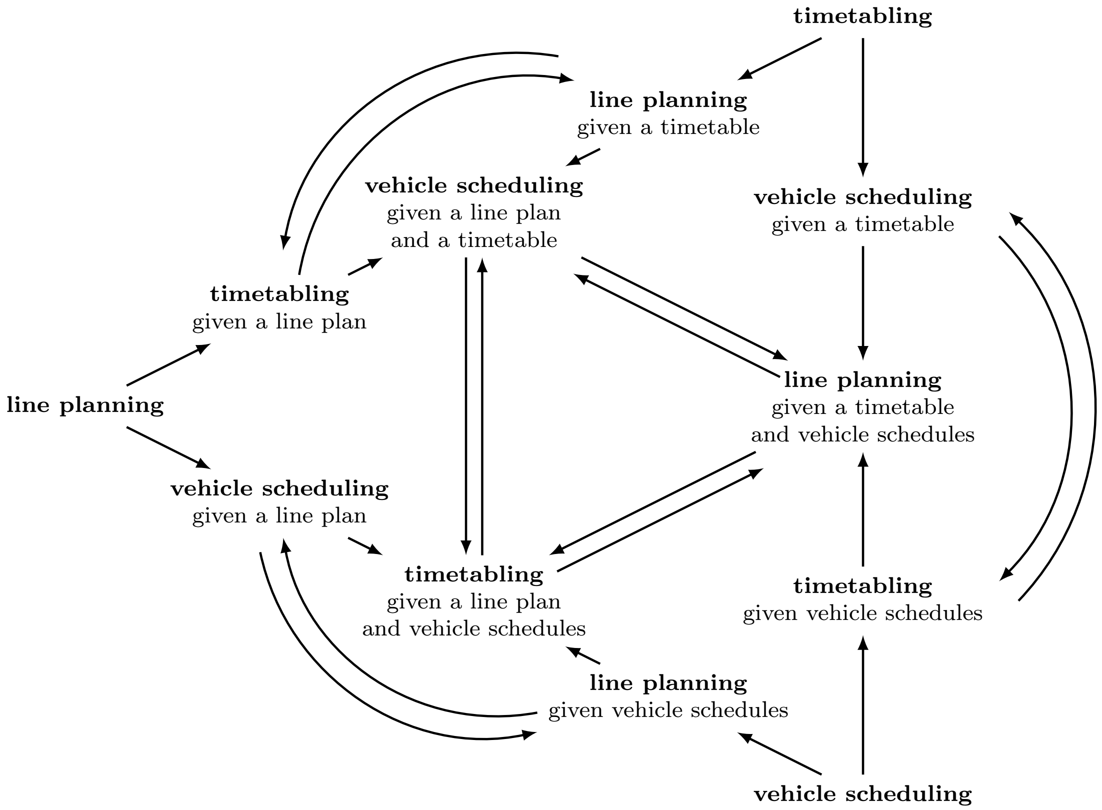


Figure 3.2: Depiction of the eigenmodel described in [37].

CK **rob_ga_only_best_breeding** whether to use only the best/fittest or all of the population for breeding

CK **rob_ga_seed** the random seed

CK **rob_ga_selection** determines how to choose the next generation. While CV `QUALITY` will only keep the best/fittest solutions, CV `PARETO` will keep all non-dominated (w.r.t. predicted robustness and travel time) individuals and add the best/fittest solutions if those are not enough (compared to CK `rob_genetic_solution_pool_size`).

CK **rob_ga_solution_pool_size** the number of solutions in each generation

CK **rob_mip_gap** the mip gap for the vehicle scheduling subproblem. Set to -1 to disable.

CK **rob_threads** the thread limit for the vehicle scheduling subproblem. Set to -1 to disable.

CK **rob_timelimit** the time limit for the vehicle scheduling subproblem. Set to -1 to not set a time limit.

3.10.6 Eigenmodel

The eigenmodel is a theoretical model for iteratively solving the integrated public transport model. A representation can be seen in Figure 3.2. For more information, see [37].

Tim-Veh-To-Lin

Implementation of one of the steps of the inner circle of the eigenmodel. For a fixed line plan and vehicle schedule, compute a new periodic timetable. For more information, see [29]. Note that this model will only work for line frequencies of 1.

CK **tim_veh_to_lin_time_limit** The time limit for the optimization.

CK **tim_veh_to_lin_mip_gap** The mip gap for the optimization.

CK **tim_veh_to_lin_write_lp_output** Whether to write the lp output. Will additionally compute an IIS for infeasible programs.

CK **DM_earliest_time_EM** The earliest time for events to consider for this model. Should be large enough that the time between CK DM_earliest_time_EM and CK DM_latest_time_EM is free of any aperiodic side effects.

CK **DM_latest_time_EM** The latest time for events to consider for this model. Should be small enough that the time between CK DM_earliest_time_EM and CK DM_latest_time_EM is free of any aperiodic side effects.

Chapter 4

Auxiliary Algorithms

4.1 Dataset Generation

With the dataset generator it is possible to create new artificial datasets. To use it, navigate into the Fo `/datasets` directory and run

R `make dg-generate-dataset`

This will create a new subdirectory in Fo `/datasets`.

4.1.1 Input

As input, only some parameters in the file Fi `/dataset-generation/basis/Config.cnf` are needed.

CK `ptn_name` The name for the new dataset. As default, this is set to be `new_generic_dataset`.

CK `dg_model` specifies the method by which the new dataset is created.

Depending on the chosen CK `dg_model`, some more config parameters are required; see below.

4.1.2 Output

As output a new directory Fo `/datasets/`CK `ptn_name` is created. The config file from the directory Fo `dataset-generation` is copied into the new dataset. This is then ready to be used as a dataset with all functionalities of LinTim.

4.1.3 Algorithms

Parametrized City

CK `dg_model` CV `parametrized_city`

The model is based on a paper by Fielbaum et al. [5]. This model divides a city into various zones. The authors state, that most big cities consist of one Central Business District (CBD) sourrounded by some subcenters. As output, the files Fi `Stop.giv`, Fi `Edge.giv` and Fi `OD.giv` are created in the new directory. The PTN is generated by the following procedure:
First, the CBD is represented by a node in the center of the PTN. The CBD is surrounded by n zones, each of which consists of a subcenter-node and a periphery node. All the subcenters are then connected to the CBD and their neighboring subcenters. The periphery nodes are only connected to their own subcenter.
The distance between a subcenter and the geometrical center C of the graph is L. It is not necessary that the CBD is located at C, but it can have an offset to C by ηL along an axis CBD-subcenter. The distance

between a periphery and its subcenter is gL.
Considering the creation of the OD-Matrix, the parameter Y states how many trips are generated in total. They are evenly splitted among the n zones, such that exactly $\frac{Y}{n}$ trips start in each zone. A fraction a of those trips start in the subcenter and a fraction of $b = 1 - a$ depart from the periphery. Usually we have $b < a$. A fraction of α of all trips generated in a periphery goes to the CBD and a fraction of β goes to it's own subcenter. The rest ($\gamma = 1 - \alpha - \beta$) goes evently splitted to all other (foreign) subcenters.
To use the Parametrized city model, specify the following parameters:

- CK `gen_vehicle_speed` Speed of the vehicles in km/h.
- CK `dg_param_city_number_subcenters` Number of subcenters sourrounding the CBD. The PTN has $2n + 1$ nodes.
- CK `dg_param_city_alpha` Trips proportion from periphery that go to the CBD.
- CK `dg_param_city_beta` Trips proportion from periphery to own subcenter. From α and β we calculate the value of $\gamma = 1 - \alpha - \beta$ representing the trips proportion from periphery to foreign subcenters.
- CK `dg_param_city_eta` Portion of displacement of the CBD from the center of the city in an axis CBD-subcenter.
- CK `dg_param_city_Y` Total number of trips generated.
- CK `dg_param_city_L` Distance from any subcenter to the geometrical center of the city.
- CK `dg_param_city_g` Distance periphery-subcenter / Distance subcenter-CBD.
- CK `dg_param_city_a` Trips proportion that depart from the periphery. From this we calculate the value of $b = 1 - a$ representing the trips proportion that depart from a subcenter.

According to [5] the parameters in table 4.1.3 should give a reasonable model of the corresponding cities.

	Santiago	Bordeaux	Los Angeles
n	7	3	7
α	0.25	0.18	0.0033
β	0.22	0.72	0.287
γ	0.53	0.1	0.68
η	0	0	0
Y	2,565,622	250,000	4,500,000
L	10	6.6	11.65
g	0.85	1.2	0.79
a	0.78	0.3	0.91
b	0.22	0.7	0.09

Table 4.1: These parameters should reproduce a reasonable model for the correpsonding cities.

Ring

CK `dg_model` CV `ring`

This model creates an undirected PTN consisting of some concentric rings and a center node. For each edge the lower bound is set to 1 and the upper bound is set to 20. The following parameters control the layout:

- CK `dg_ring_number_of_rings` Number of concentric rings that are generated.
- CK `dg_ring_nodes_per_ring` Number of nodes that each ring consists of

CK dg_ring_length_1 If this boolean parameter is set to true, the lengths of all edges are equal to 1.

CK dg_ring_radius specifies the radius of the inner ring, i.e. the lengths of the edges from the center to the nodes of the inner ring. The lengths of all other edges are set according to the euclidean distance in the plane. Only used, if CK dg_ring_length_1 is false.

There are different methods for the creation of the OD data, specified by CK dg_ring_demand_type. OD-values are always created symmetrically and they are equal to zero if both nodes are identical. Available options are:

CV UNIFORM All OD-values are set to 1.

CV UNIFORM_CENTRE If one of the nodes is the center node, the OD value is 100, otherwise it is 10.

CV RANDOM All OD-values are set to random integers between 1 and 100.

CV RANDOM_NEIGHBOUR_CENTRE If one of the nodes is the center node, then the OD-value is a random integer between 40 and 150. If there exists an edge between both nodes, the OD-value is a random integer between 20 and 50 and otherwise it is a random integer between 0 and 30.

CV SPOKE_RING For each pair of nodes we compute a shortest path in the PTN with respect to the euclidean distance (even if the edge lengths are set to 1). Along this path we count the number of ring edges and the number of spoke edges. A spoke edge is an edge between two nodes in different rings. The following parameters need to be specified:

CK dg_ring_spoke_edge_demand

CK dg_ring_ring_edge_demand

CK dg_ring_demand_scaling_factor

The OD value is then computed as

$$\texttt{scaling\_factor} \cdot \left(\frac{\texttt{spoke\_edge\_demand}}{\#\text{spoke edges} + 1} + \frac{\texttt{ring\_edge\_demand}}{\#\text{ring edges} + 1} \right)$$

rounded to the nearest integer.

4.2 OD Matrix Creation

In the OD matrix creation step, an OD matrix is calculated using a given demand and a PTN.

4.2.1 Input

The following files are needed as input:

- CK default_stops_file (Fi basis/Stop.giv) stops of the PTN
- CK default_edges_file (Fi basis/Edge.giv) edges of the PTN
- CK default_demand_file (Fi basis/Demand.giv) demand at geographical positions

4.2.2 Output

The following file is produced as output:

- CK default_od_file (Fi basis/OD.giv) OD matrix for one planning period

4.2.3 Algorithms

To compute an OD matrix run

R make od-create

For all pairs of demand point a shortest path is computed, which includes the path to and from the PTN and might also not use any PTN edges. The demand at one demand point is distributed randomly to all other demand points with probabilities proportional to

$$\frac{\text{demand at other demand point}}{(\text{distance between demand points})^2}.$$

The passengers which are computed to travel between to demand points are attributed to the OD pairs consisting of the first and last station on the shortest path. If the shortest path does not contain any stations, the passengers are not counted towards the OD matrix.
The following parameters can be used to influence the OD matrix which is created:

- CK od_use_network_distance: if set to true, the distance between demand point which is used for distributing passengers to destination demand points is the travel time between the demand points on the shortest paths. Otherwise it is proportional to the geographical distance between the demand point depending on the norm
CK sl_distance.
- CK od_remove_uncovered_demand_points: if set to true, demand points which are more than CK sl_radius away from the nearest station are not included in the computation.
- CK od_network_acceleration: speed up factor for driving in the PTN compared to traveling directly, also used for driving to and from the network.
- CK ptn_stop_waiting_time: the time (in minutes) a vehicle has to stop at each station which is considered during the computation of the shortest path.

4.2.4 Distribute from node demand

If an od demand based on an infrastructure is given, i.e., CK filename_od_nodes_file (Fi basis/OD-Node.giv), an od distribution algorithm can be used to create a stop based od matrix. For this, run

R make od-distribute-from-nodes

to obtain CK default_od_file (Fi basis/OD.giv). This will find travel-time-minimal paths for all passengers and create a stop od matrix based on their chosen route, i.e., the first boarding station and the last alighting station will determine the new od matrix. For this, the walking edges provided in CK filename_walking_edge_file (Fi basis/Edge-Walking.giv) and a penalty factor for walking, i.e., CK gen_walking_utility, will be considered. The drive time on infrastructure edges is based on CK ean_model_weight_drive and the waiting time at stations is calculated based on CK ean_model_weight_wait. Additionally, the obtained assignment from node od pair to stop od pair can be written to CK filename_od_node_assignment_file (Fi basis/OD-Node-Assignment.giv) by setting CK od_node_write_assignment to CV true.

4.3 Load distribution

This step takes the OD matrix and distributes the passengers to the PTN. The resulting edge loads are used as an input for following steps, e.g. most line planning algorithms. This section first handles the setting of CK load_generator_model to CV LOAD_FROM_PTN. For the case of setting CK load_generator_model to CV LOAD_FROM_EAN, see 4.3.4, and for the case of setting it to CV spanners, see 4.3.5.

4.3.1 Input

The following files are needed as input:

- CK default_stops_file (Fi basis/Stop.giv)
- CK default_edges_file (Fi basis/Edge.giv)
- CK default_od_file (Fi basis/OD.giv)

When parameter CK load_generator_use_cg is set to CV true, the line pool is needed as well to build the Change&Go-network, i.e.,

- CK default_pool_file (Fi basis/Pool.giv)
- CK default_pool_cost_file (Fi basis/Pool-Cost.giv)

4.3.2 Output

The following file is produced as output:

- CK default_loads_file (Fi basis/Load.giv)

4.3.3 Algorithms

To compute a new load, run

R make ptn-regenerate-load

There are different objective functions to distribute the passengers, namely

- CK load_generator_type CV SP
- CK load_generator_type CV REWARD
- CK load_generator_type CV REDUCTION

CV SP distributes the passengers on shortest paths. For determining the length of a PTN edge, parameter CK ean_model_weight_drive is used.
The load generators CV REWARD and CV REDUCTION are iterative and include an additional term, rewarding in different ways the bundling of passengers. The weight of the additional terms is determined by CK load_generator_scaling_factor. CV REDUCTION adds a penalty depending on the usage of the edge in PTN (high penalty for low usage) and CV REWARD rewards an edge more if less passengers are needed to fill the next vehicle on the edge. For a more detailed description of the models, see [8].
There are two other parameters to determine the behavior of the algorithm:

CK **load_generator_use_cg** When this is set to CV true, a Change&Go-network is used for routing the passengers. This includes the knowledge of the line pool, allowing to consider transfers. The cost of a transfer will be the estimated change time (CK load_generator_min_change_time_factor times CK ean_default_minimal_change_time; at most CK ean_default_maximal_change_time) plus CK ean_change_penalty. For waiting at a stop, the behavior of CK ean_model_weight_wait is adopted. For a more detailed description of the Change&Go-network see [39]. Since the network to route in is much larger, this increases the runtime, especially for bigger pools. But the resulting load is often more realistic.

CK **load_generator_number_of_shortest_paths** This determines the number of shortest paths the passenger are distributed to, i.e., if this is set to K, the K shortest paths are computed in each step. This increases the runtime! To distribute the passengers on the different paths, a logit model with parameter CK load_generator_sp_distribution_factor is used.

For an undirected PTN the algorithm does not distinguish the direction in which an edge is traversed, i.e., the load on an edge is the sum of the numbers of passengers traversing it in each direction. To determine the lower and upper frequency values in the CK `default_loads_file` (Fi `basis/Load.giv`), the resulting load is divided by the vehicle capacity CK `gen_passengers_per_vehicle`. Overall, the following parameters determine the behavior of the algorithm:

CK **ean_change_penalty**

CK **ean_default_maximal_change_time**

CK **ean_default_maximal_waiting_time**

CK **ean_default_minimal_change_time**

CK **ean_default_minimal_waiting_time**

CK **ean_model_weight_drive**

CK **ean_model_weight_wait**

CK **gen_passengers_per_vehicle**

CK **load_generator_add_additional_load**

CK **load_generator_fixed_upper_frequency**

CK **load_generator_fix_upper_frequency**

CK **load_generator_lower_frequency_factor**

CK **load_generator_max_iteration**

CK **load_generator_min_change_time_factor**

CK **load_generator_model**

CK **load_generator_number_of_shortest_paths**

CK **load_generator_scaling_factor**

CK **load_generator_sp_distribution_factor**

CK **load_generator_type**

CK **load_generator_use_cg**

CK **load_generator_upper_frequency_factor**

4.3.4 Using the EAN

If CK `load_generator_model` is set to CV `LOAD_FROM_EAN`, the EAN is used to determine the load of the PTN edges. Therefore the EAN is read and has to be present, i.e., the files

- CK `default_events_periodic_file` (Fi `timetabling/Events-periodic.giv`)
- CK `default_activities_periodic_file` (Fi `timetabling/Activities-periodic.giv`)

4.3.5 Using spanner MIPs

If CK load_generator_model is set to CV spanners, a mixed-integer formulation of the problem is used to distribute the load and find an optimal subgraph of the PTN. The three measures of quality considered in this model are:

Building cost The cost of the subgraph (i.e., total length of the edges)

Total travel time The sum of travel times for each OD pair, weighted by the demand

Maximum detour factor The maximum (over all OD pairs) ratio between shortest path in subgraph and shortest path in full graph.

The model is described in detail in Heinrich et al. (2023) [16]. The goal of this model is to find a subgraph of the PTN satisfying the given constraints and minimizing an objective. This method gives flexibility to the user in choosing which measures should be considered as constraints and which one as the objective. The model assumes that all passengers choose the shortest path available to them.
The main parameters determining the behavior of the algorithm are:

CK **load_generator_building_cost**

CK **load_generator_travel_time**

CK **load_generator_detour_factor**

CK **load_generator_max_building_cost**

CK **load_generator_max_travel_time**

CK **load_generator_max_detour**

The first three parameters are allowed to take values CV obj, CV cons and CV none. Only one of them should be set to CV obj, thus setting the objective of the MIP to minimize building costs of the spanner, total travel time of passengers, or the maximum detour factor. The rest of the parameters can be set to CV cons, adding a constraint determining an upper bound for the respective expression (determined by the latter three parameters above), or CV none, if, e.g., total travel time should not be considered as either the objective function or a constraint.
Other parameters affecting the MIP are:

CK **load_generator_gen_cuts** When this is set to CV true, so-called valid inequalities are added to the MIP. These have no effect on the solution, but tend to greatly improve computational performance.

CK **load_generator_remove_unused_edges** When this is set to CV true, after solving the model, the unused edges are removed from CK default_edges_file (Fi basis/Edge.giv).

Finally, the solver parameters can be set using:

CK **load_generator_solver** determine the solver to be used, the supported solvers are Gurobi, CPLEX and Xpress.

CK **load_generator_threads** determine the maximal number of threads to use for the solver (-1=use default value, i.e., no restriction).

CK **load_generator_timelimit** the time limit for the solver in seconds (-1=use default value).

CK **load_generator_mip_gap** the mip optimization gap for the solver, 0.1 equals a gap of 10 % (-1=use default value).

CK **load_generator_write_lp_file** whether to write the lp file of the model to solve.

4.4 Headway creation

This is a small helper script to create a headway file for the current dataset. Some older methods still need a headway file present, even if the content is not used.

4.4.1 Input

The following file is needed as input

- CK `default_edges_file` (Fi `basis/Edge.giv`) edges of the PTN

4.4.2 Output

The following file is produced as output:

- CK `default_headways_file` (Fi `basis/Headway.giv`) a file containing a default headway value for each edge

4.4.3 Algorithm

To create the headways, run

R `make ptn-headways`

This will create a new headway file, using CK `ptn_default_headway_value` as a value for each edge.

4.5 PTN to EAN

4.5.1 Input

The following files are required as input

- CK `default_stops_file` (Fi `basis/Stop.giv`) edges of the PTN
- CK `default_edges_file` (Fi `basis/Edge.giv`) edges of the PTN
- CK `default_lines_file` (Fi `line-planning/Line-Concept.lin`) a line concept on the PTN

4.5.2 Output

This procedure gives the following output

- CK `default_events_periodic_file` (Fi `timetabling/Events-periodic.giv`)
- CK `default_activities_periodic_file` (Fi `timetabling/Activities-periodic.giv`)

4.5.3 Algorithm

To create the Event-Activity-Network (required as input for Timetabling etc.), run

R `make ean`

The event-activity-network is then created. To this end for every line departure and arrival events for every station the line passes (every line is executed in both directions, depending on CK `ptn_is_undirected`) will be created. These events are then connected either with drive or wait activities (respecting the bounds given by the configuration of CK `ean_default_minimal_waiting_time` etc.). Furthermore it will assign each arc with some weight, corresponding to the amount of passengers driving on it. To this end, the

passengers are routed along shortest paths in the EAN, where the calculation of the edge lengths assumes that the times for each activity are given by CK ean_model_weight_drive (resp. wait/change/etc.). Note that this computation ignores the loads of the edges in the PTN.
Per default CK ean_construction_target_model_frequency is set to CV FREQUENCY_AS_MULTIPLICITY, which additionally creates synchronisation activities between every repetition of each line. This ensures that in the EAN the frequency of each line is indeed respected. Note, that such synchronisation activities have fixed upper and lower bounds, that are equal. If the frequency of a line does not divide the period length, this routine will distribute the remaining time buffer evenly to the different activities.
If headways exist, they can also be created for the EAN by setting CK ean_construction_target_model_headway to something different than CV NO_HEADWAYS (which is the default), e.g. to CV SIMPLE.
Individual station limits can be provided by CK filename_station_limit_file (Fi basis/Station-Limits.giv) when CK ean_individual_station_limits is set to CV true. For every station in the station limit file, the given individual limits will be used. For stops not in the limit file or entries of -1 the global default values will be used.
Additionally, it is possible to restrict the set of stations where transfers may take place. For this, set CK ean_respect_change_stations to CV true and provide a list of possible transfer stations in CK filename_change_station_file (Fi basis/Change-Stations.giv). Transferring in other stations will be forbidden, i.e., no transfer activities will be created there.
It is also possible to enable walking, i.e., transferring between different stops connected by walking edges. For this, CK ean_use_walking must be set to CV true and an infrastructure network with corresponding walking edges needs to be provided that is consistent with the PTN used, i.e., we assume that the node id of the corresponding node is stored in the long name of the stops. Additionally, a total maximal walking time (CK sl_max_walking_time) can be provided, only allowing walking transfers with the given maximal length.
The following parameters control the behavior of the algorithm:

CK **debug_paths_in_ptn**

CK **debug_paths_in_ean**

CK **ean_algorithm_shortest_paths**

CK **ean_change_penalty**

CK **ean_construction_skip_passenger_distribution**

CK **ean_construction_target_model_frequency**

CK **ean_construction_target_model_headway**

CK **ean_default_maximal_change_time**

CK **ean_default_maximal_waiting_time**

CK **ean_default_minimal_change_time**

CK **ean_default_minimal_waiting_time**

CK **ean_discard_unused_change_activities**

CK **ean_dump_initial_duration_assumption**

CK **ean_individual_station_limits**

CK **ean_initial_duration_assumption_model**

CK **ean_model_weight_change**

CK **ean_model_weight_drive**

CK **ean_model_weight_wait**

CK **ean_random_shortest_paths**

CK **ean_use_walking**

CK **period_length**

CK **sl_max_walking_time**

4.6 EAN buffer activities

There are several algorithms to add buffer times to the EAN. All methods are called using

R `make ean-buffer-activities`

and the implementation used is determined by the config parameter CK `rob_buffer_generator` with the following choices:

- CV `exponential`: Exponential distribution
- CV `reverse-exponential`: Reverse exponential distribution
- CV `uniform-random`: Uniform random buffer distribution
- CV `exceed-random`: Uniform random distribution with an additonal upper bound
- CV `proportional`: Add a fixed buffer to all activities
- CV `proportional-restricted`: Buffer all activities with a fixed term, but restrict the number of events or activities to buffer

For CV `proportional-restricted`, the following config parameters determine the behavior:

- CK `rob_buffer_link_list`: A given list of link ids to buffer. All activities belonging to the given links will be buffered
- CK `rob_buffer_on_wait_activity`: The buffer to add to wait activities, only activities determined by the CK `rob_buffer_stop_percentage` will be buffered.
- CK `rob_buffer_on_drive_activity`: The buffer to add to drive activities, only activities determined by the CK `rob_buffer_link_percentage` or CK `rob_buffer_link_list`: will be buffered.
- CK `rob_proportional_drive_activity_buffer`: An additional percentage based buffer for the drive activities, should be between 0 and 1
- CK `rob_buffer_link_percentage`: The percentage of links to buffer. Will buffer all drive activities on the most used links, i.e., the links with the most drive activities. Should be between 0 and 1.
- CK `rob_buffer_stop_percentage`: The percentage of stops to buffer. Will buffer all wait activities at the most used stops, i.e., the stops with the most changing passengers. Should be between 0 and 1.

The buffered activities will be written to CK `default_activity_buffer_file` and
CK `use_buffered_activities` will be set to CV `true`. Reading
CK `default_activities_periodic_file` should always return the value for
CK `default_activity_buffer_file` when CK `use_buffered_activities` is set to CV `true`.

4.7 EAN reroute passengers

R `make ean-reroute-passengers`

This generates a passenger distribution (i.e., new weights on the activities) by rerouting the passengers (i.e., the OD pairs) through the periodic EAN on shortest paths with respect to the timetable derived durations. Note that the passengers of the same OD pair will not be split up, but will all use the same shortest path in the EAN.

4.8 Tariff (Reference) Price Matrix

Running

R `make taf-tariff-price-matrix`

creates a price matrix for a specified tariff with given tariff information. No optimization is done.
Running

R `make taf-tariff-reference-price-matrix`

creates a reference price matrix for a specified tariff with given tariff information.

4.8.1 Input

The following files are needed as input:

- CK `default_stops_file` (Fi `basis/Stop.giv`), stops of the PTN,
- CK `default_edges_file` (Fi `basis/Edge.giv`), edges of the PTN.

If CK `taf_model` CV `flat`, the following configuration value is needed:

- CK `taf_flat_price`, the constant price for all paths in the tariff.

If CK `taf_model` CV `beeline_distance` or CV `network_distance`, the following configuration values are needed:

- CK `taf_fixed_costs`, the fixed costs of an affine beeline distance or network distance tariff,
- CK `taf_factor_costs`, the factor costs of an affine beeline distance or network distance tariff.

If CK `taf_model` CV `zone`, then the two following files are additionally needed as input:

- CK `filename_tariff_zone_file` (Fi `tariff/Zones.taf`), assignment of stops to zones,
- CK `filename_tariff_zone_price_file` (Fi `tariff/Zone-Prices.taf`), price list of the zone tariff.

If CK `taf_model` is CV `network_distance` or CV `zone` and CK `taf_routing_generation` is CV `read-all` or CV `read-partial-fill`, then the following file is additionally needed as input:

- CK `filename_routing_ptn_input` (Fi `basis/Routing-ptn.giv`).

4.8.2 Output

Running

R make taf-tariff-price-matrix

yields as output

- CK filename_tariff_price_matrix_file (Fi tariff/Price-Matrix.taf), prices for each OD pair,

and

R make taf-tariff-reference-price-matrix

yields as output

- CK filename_tariff_reference_price_matrix_file (Fi basis/Reference-Price-Matrix.giv), reference prices for each OD pair.

If CK taf_model is CV network_distance or CV zone, the routing file is produced as output as well:

- CK filename_routing_ptn_output (Fi basis/Routing-ptn.giv), routing in the PTN.

In all cases, the following statistic file is also produced as output:

- CK filename_tariff_properties_file (Fi statistic/tariff-properties.sta), statistic file containing information whether the no-elongation property and the no-stopover property (see Section 3.7.4) are fulfilled for the computed price matrix.

4.8.3 Algorithms

Price Matrix

Run

R make taf-tariff-price-matrix

to create a price matrix for all OD pairs for a specified tariff with given tariff information. The following models CK taf_model are available:

- CV flat, write price matrix for a flat tariff,
- CV beeline_distance, write price matrix for an affine beeline distance tariff,
- CV network_distance, write price matrix for an affine network distance tariff,
- CV zone, write price matrix for a zone tariff.

If CK taf_model CV flat, CK taf_flat_price is read and for each non-trivial OD pair (i.e. for all $d \in D := (V \times V) \setminus \{(v, v) : v \in V\}$) this flat price is written to CK filename_tariff_price_matrix_file (Fi tariff/Price-Matrix.taf). For trivial OD pairs $d = (v, v)$ for $v \in V$ the price is set to 0.
Be aware that entries for CK taf_flat_price in the private configuration file overwrite those specified in the state configuration file (see Section 8.1).

If CK taf_model CV beeline_distance, the Euclidean distances l_d between the start and end station of each OD pair d are determined. Then the fixed costs f (CK taf_fixed_costs) and factor costs p (CK taf_factor_costs) are read from the configuration file and the prices are determined for each non-trivial OD pair d by $f + l_d \cdot p$ and written to CK filename_tariff_price_matrix_file (Fi tariff/Price-Matrix.taf). For trivial OD pairs $d = (v, v)$ for $v \in V$ the price is set to 0.

Be aware that entries for CK taf_fixed_costs and CK taf_factor_costs in the private configuration file overwrite those specified in the state configuration file (see Section 8.1).

If CK taf_model CV network_distance, for each OD pair the summed edge lengths of the routing specified by CK taf_routing_generation with the following possible values are used as distances:

- CV fastest-paths, a new routing using fastest paths with respect to the lower time bounds is created,
- CV read-all, a routing given in CK filename_routing_ptn_input (Fi basis/Routing-ptn.giv) is read and used to determine distances,
- CV read-partial-fill, a partial routing given in CK filename_routing_ptn_input (Fi basis/Routing-ptn.giv) is read and used to determine distances. Unspecified paths are filled with fastest paths with respect to the lower time bounds.

From this the paths lengths l_d are calculated by summing up the edge lengths on the path. For all non-trivial OD pairs $d \in D$ the prices are calculated by $f + l_d \cdot p$ with fixed costs f (CK taf_fixed_costs) and factor costs p (CK taf_factor_costs). For trivial OD pairs $d = (v, v)$ for $v \in V$ the price is set to 0. The prices are written to CK filename_tariff_price_matrix_file (Fi tariff/Price-Matrix.taf).
Be aware that entries for CK taf_fixed_costs and CK taf_factor_costs in the private configuration file overwrite those specified in the state configuration file (see Section 8.1).

If CK taf_model CV zone, then CK filename_tariff_zone_file (Fi tariff/Zones.taf), CK filename_tariff_zone_price_file (Fi tariff/Zone-Prices.taf) and CK taf_zone_counting are read. From this for each non-trivial OD pair a path specified by CK taf_routing_generation (as explained above and in Section 3.7.1) is used to determine the price by counting the number of traversed zones respecting CK taf_zone_counting (as explained in Section 3.7.4). For trivial OD pairs $d = (v, v)$ for $v \in V$ the price is set to 0. The prices are written to CK filename_tariff_price_matrix_file (Fi tariff/Price-Matrix.taf).

Reference Price Matrix

Run

R make taf-tariff-reference-price-matrix

to create a reference price matrix for all OD pairs for a specified tariff with given tariff information.
This command follows the same logic as

R make taf-tariff-price-matrix

with the difference being that the prices are written to CK filename_tariff_reference_price_matrix_file (Fi basis/Reference-Price-Matrix.giv) instead of CK filename_tariff_price_matrix_file (Fi tariff/Price-Matrix.taf).

4.9 Rollout

The periodic event-activity network and the periodic timetable have to be converted to a nonperiodic event-activity network that can be used in the operational phase of public transport.

4.9.1 Input

The following files are needed as input

- CK default_edges_file (default: basis/Edge.giv)
- CK default_headways_file (default: basis/Headway.giv)
- CK default_events_periodic_file (Fi timetabling/Events-periodic.giv)
- CK default_activities_periodic_file (Fi timetabling/Activities-periodic.giv)

4.9.2 Output

The following files are produced as output:

- CK default_events_expanded_file (Fi delay-management/Events-expanded.giv) a file containing the aperiodic events
- CK default_activities_expanded_file (Fi delay-management/Activities-expanded.giv) a file containing the aperiodic activities
- CK default_timetable_expanded_file (Fi delay-management/Timetable-expanded.tim) a file containing the aperiodic timetable

4.9.3 Algorithm

To roll out, all (nonperiodic) events that take place in the time interval [CK DM_earliest_time, CK DM_latest_time] (given in seconds since 0:00) as well as all (nonperiodic) activities connecting those events are taken into account. If CK rollout_whole_trips is set to CV true, all trips whose start event or end event are not contained in [CK DM_earliest_time, CK DM_latest_time] are deleted. If CK rollout_discard_unused_change_edges is set to CV true, changing activities with weight 0 are ignored (this might significantly reduce the size of the nonperiodic event-activity network, speeding up the delay management step). The parameter CK rollout_for_nonperiodic_timetabling influences the output: if set to CV true, only forward headways are contained in the output, and for each activity, the output also contains an upper bound on its duration (note that this parameter always should be set to false unless you really know what you are doing!).

Delay Management and Vehicle Scheduling When rolling out for vehicle scheduling, usually a long time period (e.g. a whole day) is considered and CK rollout_whole_trips *must* be set to CV true. When rolling out for delay management, usually a short time period (e.g. two hours) is considered and CK rollout_whole_trips should be set to CV false. Typically, the combination of vehicle scheduling and delay management could be like this:

1. Set [CK DM_earliest_time, CK DM_latest_time] to a “large” time interval, e.g. one day, and CK rollout_whole_trips to CV true.
2. Run

   R make ro-rollout && make ro-trips

3. Run

   R make vs-vehicle-schedules

   to generate the vehicle schedules.

4. Set [CK DM_earliest_time, CK DM_latest_time] to the time interval needed for delay management, e.g. two hours, and CK rollout_whole_trips to CV false.

5. Run

   R `make ro-rollout && make vs-add-circulations-to-ean`

   to roll out for delay management and to add the circulations to the rolled-out event-activity network.

Generating passenger paths For more precise methods of delay management, OD pairs may be rolled out over the delay management period into distinct paths in the aperiodic EAN. As this takes quite some time in the rollout and in the evaluation of the delay management, this has to be explicitly enabled by setting the `rollout_passenger_paths` parameter to `true`. A new file determined by `default_passenger_paths_file` will be created containing in each line a departure event, an arrival event, the source and target station id, an integral passenger weight and a comma-separated list of change activities. The weights are distributed from the original OD file, where passengers are equally distributed over the time between `DM_earliest_time` and the departure time of their last connection. Every passenger gets assigned to the next possible departure event. If there exists multiple paths with the same arrival time, among them only those with a minimal number of changes and with the latest possible departure time will be kept and considered. A path for which another path with the same or a later departure time but an earlier arrival time exists will not be considered either. If there still are multiple paths for one departure time, the passengers will be divided between them equally but integrally (such that some of them may have 1 passenger less than others). If passenger paths are rolled out, there will be an additional file according to `default_od_expanded_file` will be created. This file contains a timestamped OD demand according to the path-distribution of the passengers.

4.9.4 Requirements and caveats

- If CK `DM_enable_consistency_checks` is set to CV `true`, IDs in files are checked to be consecutively numbered beginning from 1.

4.9.5 Generating trips

For vehicle scheduling, it is necessary to additionally create the trips after rolling out, i.e., after

R `make ro-rollout`

with CK `rollout_whole_trips` set to CV `true`,

R `make ro-trips`

should be run as well. This method uses the files

- CK `default_activities_expanded_file` (Fi `delay-management/Activities-expanded.giv`)
- CK `default_events_expanded_file` (Fi `delay-management/Events-expanded.giv`)

to create

- CK `default_trips_file` (Fi `delay-management/Trips.giv`)

The file CK `default_trips_file` (Fi `delay-management/Trips.giv`) will then contain all information regarding line trips that need to be covered of a feasible vehicle schedule.

4.10 Delay generation

To simulate source delays during the operational phase, different delay generators are included in LinTim. The following parameters are used by all delay generators:

- The interval [CK delays_min_time, CK delays_max_time] defines which events and/or activities might be delayed (only events taking place in this time interval or activities connecting two such events might be delayed). Note that [CK delays_min_time, CK delays_max_time] ⊆ [CK DM_earliest_time, CK DM_latest_time] is required.
- The parameters CK delays_min_delay and CK delays_max_delay define lower and upper bounds on the amount of a source delay. If CK delays_absolute_numbers is set to CV true, the bounds are in seconds, otherwise the bounds are in % of the nominal duration of a delayed activity (this is needed for delays on activities only).
- If CK delays_append is set to CV true, the generated source delays are appended to already existing files containing source delays (to allow a combination of delays, generated by different delay generators); if set to CV false, existing files containing source delays are replaced. Please note that several source delays of the same event (activity) are not additive: newly generated source delays are simply appended to the file containing the source delays, and this file is read sequentially – so for each event (activity), only the last source delay contained in the file is taken into account.

Which generator is going to be used is controlled by the CK delays_generator parameter.

CV **uniform_distribution:** Adds random source delays to randomly chosen events and/or activities. Its behavior can be controlled by the following parameters:

- CK delays_events: If set to CV true, source delays on events are generated (can be combined with CK delays_activities).
- CK delays_activities: If set to CV true, source delays on driving activities are generated (can be combined with CK delays_events).
- CK delays_count: Number of source delays that will be generated. If CK delays_count_is_absolute is set to CV true, CK delays_count is an absolute number; otherwise it defines how many events of all events taking place in [CK delays_min_time, CK delays_max_time] (in %) and/or how many driving activities of all driving activities with start event and end event in [CK delays_min_time, CK delays_max_time] (in %) will be delayed.

CV **events_in_station:** Delays *all* events in the station defined by CK delays_station_id_for_delays. If CK delays_station_id_for_delays is CV -1, the station is chosen randomly. If you want to delay all events in several different stations, you have to run the delay generator several times with different values of CK delays_station_id_for_delays and CK delays_append set to CV true.

CV **activities_on_track:** Delays *all* driving activities on the track defined by CK delays_edge_id_for_delays. If CK delays_edge_id_for_delays is CV -1, the track is chosen randomly. If you want to delay all driving activities on several different tracks, you have to run the delay generator several times with different values of CK delays_edge_id_for_delays and CK delays_append set to CV true.

CV **uniform_background_noise:** Adds random source delays to every event and/or activity. Its behavior can be controlled by the following parameters:

- CK delays_seed: For reproducible purpose a seed for generating random delay amount is introduced. If delays seed is set to CV 0, no seed will be set and thus the experiment in general is not reproducible.

- CK delays_events: If set to CV true, source delays on events are generated (can be combined with CK delays_activities).
- CK delays_activities: If set to CV true, source delays on driving activities are generated (can be combined with CK delays_events).
- CK delays_append: If this is set to CV true, the already delayed events and activities are not further manipulated.

4.11 Visualization

LinTim offers algorithms for drawing several states of the public transportation system. The output files can be found in Fo graphics.

4.11.1 PTN

To create an illustration of the PTN run

R `make ptn-draw`

The result for dataset toy is depicted in Figure 4.1.

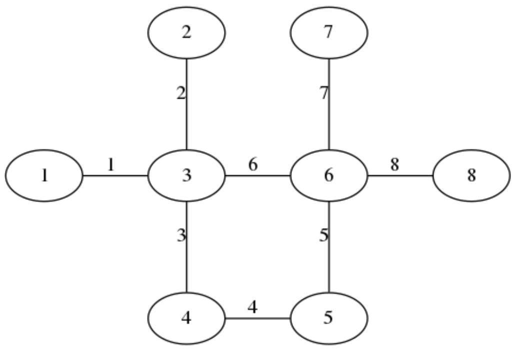


Figure 4.1: The PTN of the toy dataset

The graph can be scaled by adapting CK ptn_draw_conversion_factor.
Setting CK ptn_draw_use_coordinates to CV false results in disregarding the stop-coordinates. Instead, the stops are arranged automatically. The result for dataset toy is depicted in Figure 4.2.

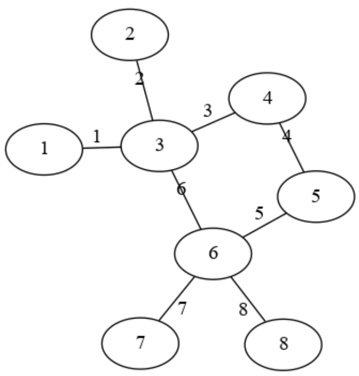


Figure 4.2: The PTN of the toy dataset with automatically arranged stops

To create an illustration of the PTN that is readable even for larger datasets, run

R `make ptn-draw-interactive`

The resulting html-script allows for some interaction, like changing node sizes or viewing network information when tracing over the graph. One possible output for dataset bahn-01 is depicted in Figure 4.3. Edge labels can be enabled with CK `ptn_draw_interactive_graph_edge_labels`.

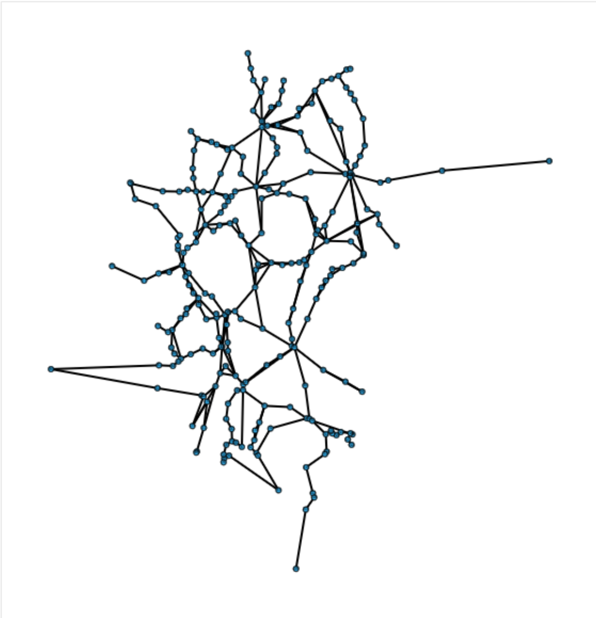


Figure 4.3: The PTN of the bahn-01 dataset

4.11.2 OD

To create an illustration of the OD data run

R `make od-draw`

The result for dataset toy is depicted in Figure 4.4. The graph displays only those OD pairs whose

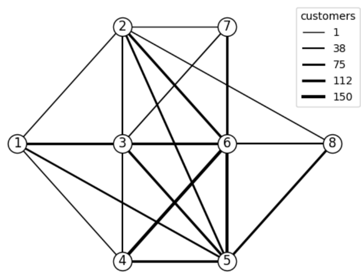


Figure 4.4: The OD data of the toy dataset where the edge width indicates the number of passengers traveling

fractional value in relation to the maximal value of the OD pairs lies within the closed interval given by CK `od_visualization_lower_bound` and CK `od_visualization_upper_bound`. Datasets with symmetric OD data will be illustrated using undirected graphs. Otherwise a directed graph will be used. The output is saved in CK `filename_od_visualization_file`. The graph can visualize the logarithm of the number of passengers traveling with CK `od_visualization_use_log_scale`. The graphs maximal edge width can be adjusted with CK `od_visualization_max_edge_width`. The number of passengers traveling can be indicated with edge color instead of edge width using the parameter CK `od_visualization_use_edge_color`. The result for dataset toy is depicted in Figure 4.5. Either graph can be scaled by adapting CK `od_draw_conversion_factor`.

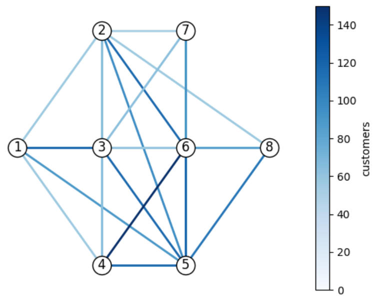


Figure 4.5: The OD data of the toy dataset where the edge color indicates the number of passengers traveling

Alternatively, a heatmap visualization can be used with CK `od_visualization_use_heatmap`. It can be annotated using CK `od_visualization_use_annotations`. As with the graph visualization, the heatmap can visualize the logarithm of the number of passengers traveling with CK `od_visualization_use_log_scale` The result for dataset toy is depicted in Figure 4.6.

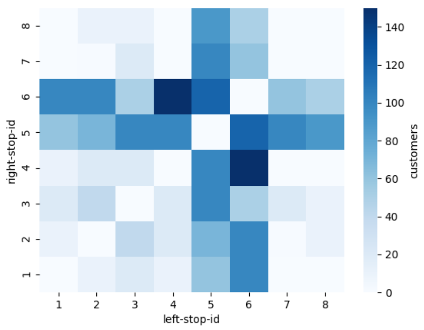


Figure 4.6: The OD data of the toy dataset visualized as a heatmap

4.11.3 Loads

To create an illustration of the traffic loads in the PTN run

R `make ptn-load-draw`

Displayed are the links whose traffic load in relation to the maximal traffic load in the network is within the interval given by the fractions CK `loads_graph_lower_bound` and CK `loads_graph_upper_bound`. The traffic loads can be illustrated using the edge color or the edge width of the PTN. This can be chosen using CK `loads_graph_use_edge_color`. The result of the former for dataset toy is depicted on the left hand side of Figure 4.7, whereas the result of the latter is depicted on the right hand side of Figure 4.7. In the latter case, the maximal edge width can be scaled by adapting CK `loads_graph_max_edge_width`.The entire figure can be scaled by adapting CK `loads_draw_conversion_factor`

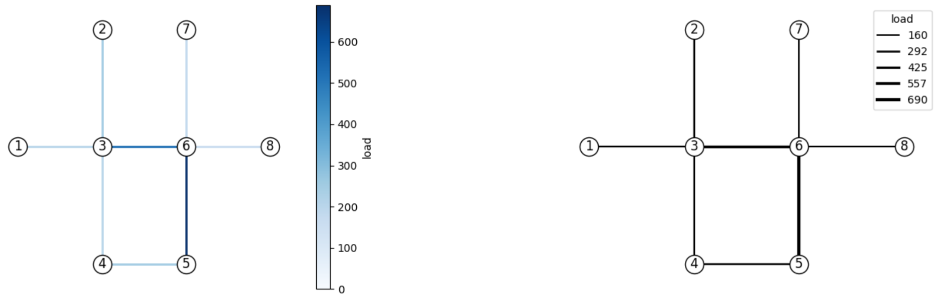


Figure 4.7: The traffic loads of the toy dataset. On the left hand side, the load of an edge is indicated by its width, on the right hand side by its color

4.11.4 Line pool

To create an illustration of the line pool run

R `make lpool-line-pool-draw`

The result for dataset toy is depicted in Figure 4.8.

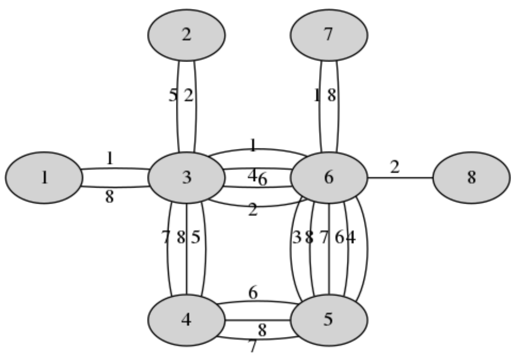


Figure 4.8: The line pool of the toy dataset

The graph can be scaled by adapting CK `lpool_coordinate_factor`.

4.11.5 Line concept

To create an illustration of the line concept run

R `make lc-line-concept-draw`

The result for dataset toy is depicted in Figure 4.9.
The graph can be scaled by adapting CK `lpool_coordinate_factor`.

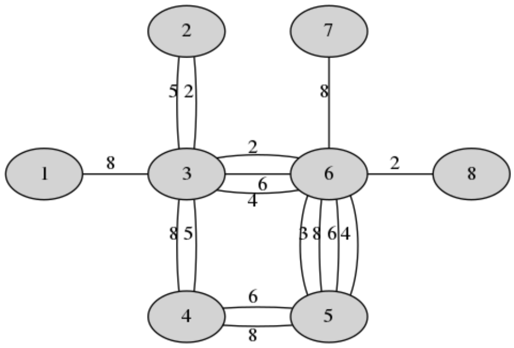


Figure 4.9: One possible line concept of the toy dataset

4.11.6 Timetable

To create an illustration of the timetable, run

```
make tim-timetable-draw
```

The result for dataset toy is depicted in Figure 4.10. Note, that this command will draw only the ean, if no timetable is present.

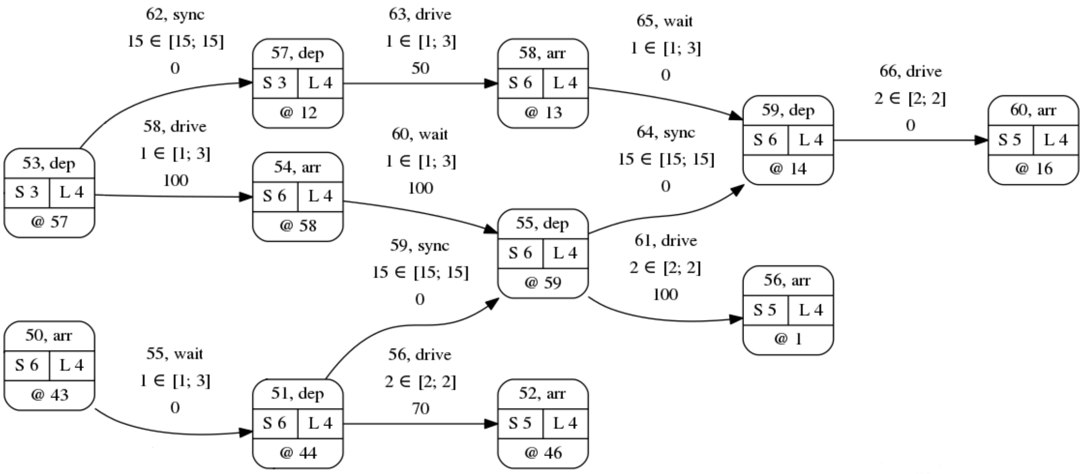


Figure 4.10: Extract of one possible timetable of the toy dataset

4.11.7 Disposition timetable

To create an illustration of the disposition timetable, run

```
make dm-disposition-timetable-draw
```

The result for dataset toy is depicted in Figure 4.11. Delayed events will be displayed in red (more delay results in more saturation). Note, that this command will draw only the extended timetable, if no disposition timetable is present.

4.11.8 Tariff

Running

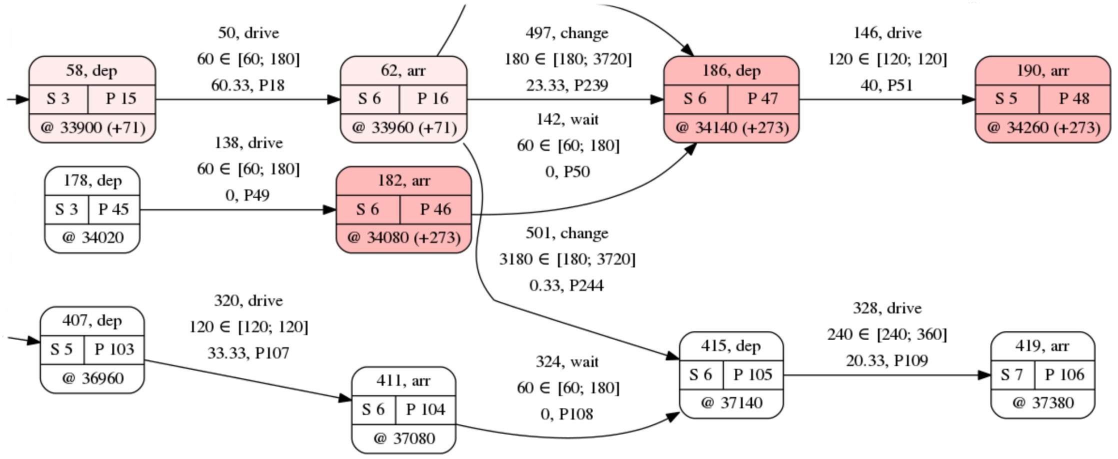


Figure 4.11: Extract of one possible disposition timetable of the toy dataset

R `make taf-tariff-draw`

yields a heatmap of prices or of price differences between two specified price matrices and can draw the PTN with nodes assigned to their zones in case of a zone tariff.

Heatmap

If CK `taf_draw_heatmap` CV `true` (which is the default), executing the above make command generates a heatmap of prices or of price differences for all OD pairs and stores it to CK `filename_tariff_heatmap` (Fi `graphics/tariff-heatmap.png`).
Which price matrix or which comparison of price matrices is visualized, is controlled by CK `taf_heatmap_mode` with the following possible values:

- CV `old`, the price matrix specified by CK `taf_evaluate_old_prices` (default: CV `basis/Reference-Price-Matrix` is visualized,
- CV `new`, the price matrix specified by CK `taf_evaluate_new_prices` (default: CV `tariff/Price-Matrix.taf`) is visualized and
- CV `compare`, the price differences between the price matrices specified by CK `taf_evaluate_new_prices` and CK `taf_evaluate_old_prices` are visualized such that the heatmap shows the change from the old prices to the new prices, i.e. the old prices are subtracted from the new prices.

By default CK `taf_heatmap_mode` is CV `old`.
Furthermore the following features of the heatmap can be controlled:

- CK `taf_heatmap_log_scale` boolean, whether or not the heatmap should use a logarithmic scale. By default it is CV `false`.
- CK `taf_heatmap_use_annotations`, whether or not the heatmap should be annotated with the calculated differences in each square. By default it is CV `false`.

Zones

If CK `taf_draw_zones` is CV `true` (default: CV `false`), then a PTN with stops colored fittingly to their zones is drawn (see Section 4.11.1), which is outputed to CK `filename_tariff_ptn_zone_graph` (Fi `graphics/ptn-zones.png`).
An example for the dataset toy with four zones is depicted in Figure 4.12.

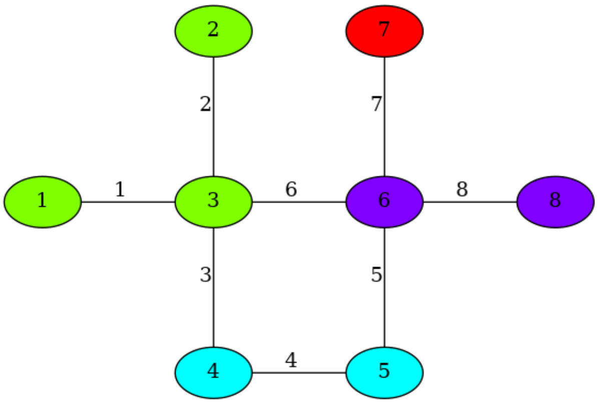


Figure 4.12: The graph of the toy dataset with zones in different colors

4.11.9 mapgui

Additionally, there is an interactive tool for displaying public transportation systems on a map which is used by running

R `make mapgui`

To decide which step is displayed, set the parameter CK `mapgui_show_step` to CV `ptn`, CV `linepool`, CV `lineconcept`, CV `timetable` or CV `dispotimetable`, respectively. The speed of the visualization is controlled by CK `mapgui_visual_speed`.

4.12 Interaction with VISUM

During the work on DFG FOR 2083[1], a basic interface to PTV VISUM [26] was created. For this, LinTim gained the ability to write the periodic timetable in a format that can be easily read by VISUM, as well as reading different infrastructure and solution information from VISUM-net-files. In this section, we will describe the different interfaces and their file requirements. Note that the name of the transport system to read can be set by CK `visum_tsyscode`, which defaults to "B". In this documentation, all attributes will include this default in their name when necessary but the read attributes are dependent on the config key.

4.12.1 Writing files for VISUM

By calling

R `make tim-transform-to-visum`

LinTim will create a timetable file based on stops (or stop points in VISUM) at CK `default_timetable_visum_file` (Fi `timetabling/Timetable-visum-nodes.tim`), that can be read easier by VISUM.

4.12.2 Reading a config file

By calling

R `make config-fill-config-from-visum`

LinTim will read a visum configuration file provided for LinTim and set the contained config parameters in the LinTim config file Fi `basis/Config.cnf`. It will read CK `filename_visum_config_file` (Fi `config.net`). The following parameters will be read

[1] https://for2083.mathematik.uni-kl.de

LINTIM_BASE_UNIT_FOR_HEADWAY : The system frequency to use, i.e., the common frequency divisor for all line freuqencies. Will set CK `lc_common_frequency_divisor`.

LINTIM_DEFDWELLTIME the default minimal waiting time at each station, will set CK `ean_default_minimal_waiting_time`.

LINTIM_MIN_TRANSFERTIME the default minimal transfer time at each station, will set CK `min_change_time`

LINTIM_PERIOD_LENGTH the period length in time units to use. Will set CK `period_length`.

LINTIM_POSTPREPTIME the turnover time after each line serving. One part of CK `vs_turn_over_time`, i.e., the values of LINTIM_POSTPREPTIME and LINTIM_PREPREPTIME will be summed up.

LINTIM_PREPREPTIME the turnover time before each line serving. One part of CK `vs_turn_over_time`, i.e., the values of LINTIM_POSTPREPTIME and LINTIM_PREPREPTIME will be summed up.

LINTIM_TIME_UNITS_PER_MINUTE the time units per minute to use. Will set CK `time_units_per_minute`.

LINTIM_TRANSFER_UTILITY the change penalty to use, i.e., the additional penalty to add for each transfer. Will set CK `ean_change_penalty`.

LINTIM_TSYS_FOR_ADAPTING the public transport mode to adapt. Will determine, which set of ptn links/infrastructure edges from Visum will be set to usable/forbidden in LinTim. Will set CK `visum_tsyscode`.

LINTIM_WALKTIME_UTILITY the walk time utility, i.e., the penalty factor for time spend walking. Will set CK `gen_walking_utility`.

SCENARIO_NAME the name of the dataset, will set CK `ptn_name`.

4.12.3 Reading the infrastructure

By calling

R `make ptn-read-infrastructure-from-visum`

LinTim will read the infrastructure information on node-level from the provided VISUM-net-file and the corresponding walking information. Note that this will not create a PTN but the underlying infrastructure, i.e., you need to compute the PTN yourself. Whether the walking information is assumed to be symmetric is dependent on CK `sl_walking_is_directed`. The following files and contents will be read:

CK **`filename_net_file`** (Fi **`infrastructure.net`**) the infrastructure file with the following objects and attributes

$ NODE: NO, XCOORD, YCOORD

$ LINK: FROMNODENO, LENGTH, NO, TONODENO, TSYSSET, T_PUTSYS(B)

CK **`filename_visum_walk_file`** (Fi **`walk_times.att`**) the walking file with the following objects and attributes

$ ODPAIR: FROMZONENO, TOZONENO, WALK_TIME (note that any third attribute will be interpreted as the walk time and only three attributes are allowed here!)

The following files will be written:

- CK filename_node_file (Fi basis/Node.giv): The nodes will contain the original visum node number as name.
- CK filename_infrastructure_edge_file (Fi basis/Edge-Infrastructure.giv)
- CK filename_walking_edge_file (Fi basis/Edge-Walking.giv)

4.12.4 Reading the PTN

By calling

R make ptn-read-ptn-from-visum

LinTim will read the infrastructure information regarding the PTN from the provided VISUM-net-file. Note that the read infrastructure needs to represent a valid LinTim PTN, i.e., links may only include nodes that are stop points. The following files and contents will be read:

CK filename_net_file (Fi infrastructure.net) the infrastructure file with the following objects and attributes

$ NODE: NO, XCOORD, YCOORD

$ STOPPOINT: NO, NODENO

$ LINK: FROMNODENO, LENGTH, NO, TONODENO, TSYSSET, T_PUTSYS(B)

The following files will be written:

- CK default_stops_file (Fi basis/Stop.giv): The stops will contain the original visum node number as short and long name.
- CK default_edges_file (Fi basis/Edge.giv)

4.12.5 Reading the demand

Reading stop demand

By calling

R make od-read-stop-od-from-visum

LinTim will read the demand data for the current stops from the provided VISUM-net-file. This step will assume that all zones in the demand matrix are located and named by their corresponding stopping point, which should be present in the short name of the LinTim stops. Demand from and to the same zone will be ignored and set to 0. The following files and contents will be read:

CK filename_visum_od_file (Fi od.att) the demand file with the following objects and attributes

$ ODPAIR: FROMZONENO, TOZONENO, DEMAND (note that any third attribute will be interpreted as the demand and only three attributes are allowed here!)

The following file will be written:

- CK default_od_file (Fi basis/OD.giv)

Reading node demand

By calling

R make od-read-node-od-from-visum

LinTim will read the demand data for the nodes from the provided VISUM-net-file. This step will assume that all zone numbers correspond to the original visum node numbers which should be stored in the names of the LinTim nodes. The following files and contents will be read:

CK **filename_visum_od_file** (Fi **od.att**) the demand file with the following objects and attributes

$ ODPAIR: FROMZONENO, TOZONENO, DEMAND (note that any third attribute will be interpreted as the demand and only three attributes are allowed here!)

The following file will be written:

- CK filename_od_nodes_file (Fi basis/OD-Node.giv)

4.12.6 Reading stops and lines

For a given infrastructure network and demand, i.e., nodes, infrastructure edges and a node-based demand, given VISUM stops and lines can be read by calling

R make lc-read-stops-and-lines-from-visum

This step will assume that the original visum node numbers are stored in the names of the LinTim nodes and that the read lines are undirected. The following files and contents will be read:

CK **filename_visum_timetable_file** (Fi **vehicle_journeys.att**) the vehicle journey file with the following objects and attributes:

$ VEHJOURNEYITEM: DEP, DIRECTIONCODE, INDEX, LINENAME, TIMEPROFILEITEM\LINEROUTEITEM\STOPPOINT\NO, VEHJOURNEYNO

The following files will be written:

- CK default_stops_file (Fi basis/Stop.giv): The stops will contain the original visum node number as short and long name.
- CK default_edges_file (Fi basis/Edge.giv)
- CK default_pool_file (Fi basis/Pool.giv)
- CK default_pool_cost_file (Fi basis/Pool-Cost.giv)
- CK default_lines_file (Fi lineplanning/Line-Concept.lin)

4.12.7 Reading a timetable

For a given line concept a timetable for the same lines can be read from provided VISUM-net-files by calling

R make tim-read-timetable-from-visum

This step will assume that the lines for the VISUM timetable are the same as in the current line concept, including the frequencies but excluding the direction, i.e., LinTim and VISUM may have the same lines noted in different directions, since lines are assumed to be undirected. The original VISUM node numbers are assumed to be present in the short names of the stops. This method will read the timetable in one specific hour, given by CK visum_hour_to_consider. The corresponding periodic EAN is assumed to be present. The following files will be read:

CK **filename_visum_timetable_file** (Fi **vehicle_journeys.att**) the vehicle journey file with the following objects and attributes:

$ VEHJOURNEYITEM: ARR, DEP, DIRECTIONCODE, INDEX, LINENAME, TIMEPROFILEITEM\LINEROUTEITEM\STOPPOINT\NO, VEHJOURNEYNO

The following file will be written:

- CK `default_timetable_periodic_file` (Fi `timetabling/Timetable-periodic.tim`)

4.12.8 Reading fixed lines

By calling

R `make lc-read-fixed-lines-from-visum`

LinTim will read lines to fix from the provided VISUM-net-file. For this, we assume that there is a transportation system that should be optimized (given by CK `visum_tsyscode`) and other fixed transportation systems. All fixed lines are read and added to the line pool as well as a fixed line file with their respective frequency and the corresponding capacities. Note that this will change the line pool, i.e., running this multiple times in a row without resetting the pool may lead to unintended consequences. We assume that the short name of the LinTim stops contains the original VISUM node number.
Afterwards, setting CK `lc_respect_fixed_lines` to CV `true` will respect these lines for the line planning problem. This is not supported for all line planning problems, see the corresponding line planning documentation in Section 3.3.
The following file and contents will be read:

CK **filename_net_fixed_lines_file** (Fi **visum-fixed-lines.net**) the infrastructure file with the following objects and attributes

$ LINE: NAME, TSYSCODE

$ LINEROUTEITEM: DIRECTIONCODE, LINENAME, NODENO

$ LINK: FROMNODENO, NO, TONODENO

$ VEHJOURNEY: DEP, LINENAME

$ VEHUNIT: TOTALCAP, TSYSSET

The following files will be written:

- CK `filename_lc_fixed_lines` (Fi `line-planning/Fixed-Lines.lin`) the fixed lines
- CK `filename_lc_fixed_line_capacities` (Fi `line-planning/Line-Capacities.lin`) the capacities of the fixed lines

4.12.9 Reading fixed times

By calling

R `make tim-read-fixed-times-from-visum`

LinTim will read the timetable of some fixed lines from the provided VISUM-net-file. For this, we assume that there is a transportation system that should be optimized (given by CK `visum_tsyscode`) and other fixed transportation systems. The fixed lines are assumed to be included in the event-activity-network and the corresponding times will be read.
Afterwards, setting CK `tim_respect_fixed_times` to CV `true` will respect these times for the timetabling problem. For more information, see Section 3.6.8.
The following file and contents will be read:

CK **`filename_net_fixed_lines_file`** (Fi **`visum-fixed-lines.net`**) the infrastructure file with the following objects and attributes

$ LINK: FROMNODENO, NO, TONODENO

$ LINEROUTEITEM: DIRECTIONCODE, LINENAME, NODENO

$ TIMEPROFILE: ARR, DEP, DIRECTIONCODE, LINENAME

The following file will be written:

- CK `filename_tim_fixed_times` (Fi `timetabling/Fixed-timetable-periodic.tim`) the fixed times

Chapter 5

Evaluation

5.1 Evaluation of the PTN Created by Stop Location

To evaluate the properties of the public transportation network created by stop location, you can use the makefile target

R `make sl-evaluate`

The following parameters will be evaluated and written to
CK `default_statistic_file` (Fi `statistic/statistic.sta`):

SK **ptn_feasible_od** For every OD pair exists a path through the PTN. (Only evaluated if an OD matrix exists.)

SK **ptn_feasible_sl** Every demand point that is no more than CK `sl_radius` away from the PTN is covered by a stop, i.e., it is no more than CK `sl_radius` away from a stop.

SK **ptn_time_average** Average travel-time of all passengers on shortest path through the PTN in seconds. (Only evaluated if an OD matrix exists.)

SK **ptn_obj_stops** Number of stops.

SK **ptn_prop_edges** Number of undirected edges for an undirected PTN, number of directed edges for a directed PTN.

SK **ptn_prop_existing_stops** Number of stops before a stop location algorithm was executed.

SK **ptn_prop_existing_edges** Number of undirected edges for an undirected PTN, number of directed edges for a directed PTN before a stop location algorithm was executed.

SK **ptn_prop_demand_point** Number of demand points.

SK **ptn_prop_relevant_demand_point** Number of demand points that are no more than CK `sl_radius` away from the PTN.

SK **ptn_travel_time_realistic** Sum of the realistic travel-travel time on all edges of the PTN in seconds considering the acceleration (CK `sl_acceleration`) and deceleration (CK `sl_deceleration`) of the vehicles.

SK **ptn_travel_time_const** Sum of the travel-travel time on all edges of the PTN in seconds assuming the vehicles would maintain a constant speed of CK `gen_vehicle_speed`.

If

C sl_eval_extended; true

is set, the following parameters will additionally be evaluated:

SK **ptn_max_distance** Maximal distance any demand point has to the stop nearest to it.

SK **ptn_candidates** Number of candidates considered as new stops during the stop location algorithm.

5.2 Evaluation of the PTN

To evaluate the properties of the public transportation network, you can use the makefile target

R make ptn-evaluate

The following parameters will be evaluated and written to
CK default_statistic_file (Fi statistic/statistic.sta):

SK **ptn_feasible_od** For every OD pair exists a path through the PTN. (Only evaluated if an OD matrix exists.)

SK **ptn_time_average** Average travel-time of all passengers on shortest path through the PTN. (Only evaluated if an OD matrix exists.)

SK **ptn_obj_stops** Number of stops.

SK **ptn_prop_edges** Number of undirected edges for an undirected PTN, number of directed edges for a directed PTN.

5.3 Evaluation of the OD Matrix

To evaluate the properties of the origin destination matrix, you can use the makefile target

R make od-evaluate

The following parameters will be evaluated and written to
CK default_statistic_file (Fi statistic/statistic.sta):

SK **od_prop_entries_greater_zero** Number of entries greater than zero, i.e., of OD pairs (A, B) where more than zero passengers want to travel from A to B.

SK **od_prop_overall_sum** Sum over all entries in the matrix, i.e., all passengers who want to travel in the network.

5.4 Evaluation of the Line Pool

To evaluate the properties of the line pool, you can use the makefile target

R make lpool-line-pool-evaluate

The following parameters will be evaluated and written to
CK default_statistic_file (Fi statistic/statistic.sta):

SK **lpool_cost** $\sum_{l\in\mathfrak{L}} cost_l$ - sum over costs per line.

SK **lpool_feasible_circles** No line is containing a circle.

SK **lpool_feasible_od** For every passenger there exists a path through the PTN that is only using edges occurring in the line pool.

SK **lpool_prop_directed_lines** Number of directed lines.

SK **lpool_time_average** Average travel-time of all passengers on shortest path through the PTN where only edges occurring in the line pool are used.

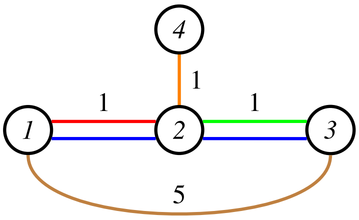
5.5 Evaluation of the Line Concept

To evaluate the properties of the line concept, you can use the makefile target

R `make lc-line-concept-evaluate`

The following parameters will be evaluated and written to
CK `default_statistic_file` (Fi `statistic/statistic.sta`). Some of them (travel times, changes) depend on the routes of the passengers, which can be computed in different ways. Their meaning is illustrated with the following example.

Example.



CK *period_length:* CV *60*
CK *ean_default_minimal_waiting_time:* CV *1*
CK *ean_default_maximal_waiting_time:* CV *3*
CK *ean_default_minimal_change_time:* CV *3*
CK *ean_model_weight_wait:* CV *AVERAGE_WAITING_TIME*
CK *ean_model_weight_change:* CV *FORMULA_1*
CK *ean_change_penalty:* CV *5*
CK *gen_passengers_per_vehicle:* CV *1*

The numbers at the arcs indicate the driving times. Assume that the red line is operated at frequency 2 *and all other lines have a frequency of* 1*. Let there be a demand of* 2 *between nodes 1 and 3 and of* 1 *between nodes 1 and 4.*

SK **lc_average_distance** Average Euclidean distance between the two endpoints of a line.

SK **lc_average_edges/length** Average number of edges/length of the lines in the line concept.

SK **lc_cost** Sum over costs of line times frequency: $\sum_{l \in \mathcal{L}} cost_l f_l$.

SK **lc_feasible** Lower and upper bounds on frequency on every edge respected: $f_e^{min} \leq \sum_{\substack{l \in \mathcal{L} \\ e \in l}} f_l \leq f_e^{max}$.

SK **lc_min_distance/edges/length** Minimal distance/number of edges/length of the lines in the line concept.

SK **lc_obj_game** Sum of the squared frequencies on all edges: $\sum_{e \in E} f_e^2$.

SK **lc_prop_directed_lines** number of directed lines. If a line is undirected, it is counted twice.

SK **lc_prop_freq_max** The maximal frequency: $\max_{l \in \mathcal{L}} f_l$.

SK **lc_var_distance/edges/length** Variance of the distance/number of edges/length of the lines in the line concept.

For the following two properties, passengers are routed along shortest paths in the network consisting of all edges that are covered with frequency at least one. Every driving edge contributes its driving time (computed according to CK `ean_model_weight_drive`) and every intermediate station contributes the waiting time (computed according to CK `ean_model_weight_wait`), independently of whether a change is necessary. Hence, the passenger routing respects neither vehicle capacities nor changes.

SK **lc_time_average_without_transfers** Average travel-time of all passengers in the routing described above, where every driving edge contributes its driving time and every intermediate station contributes the waiting time, independently of whether a change is necessary.

SK **lc_uncapacitated_direct_travelers** Number of travelers that have a shortest path in the routing described above that does not require a transfer. Does not respect the changing times, change penalty, and the capacity of lines, so this is not the same as the objective of the direct travelers model, see Section 3.3.2. For this, please check SK lc_obj_direct_travelers in the extended evaluation.

In the example the path 1,2,3 is has cost $1 + 2 + 1 = 4 < 5$. Therefore, both passengers between 1 and 3 are routed along this path and are counted as direct passengers. Moreover, the passenger from 1 to 4 is routed along the path 1,2,4, which also has travel time 4 but cannot be followed without changing. Therefore, SK lc_time_average_without_transfers is SV 4 and SK lc_uncapacitated_direct_travelers is SV 2.
When setting the config-parameter CK lc_eval_extended to *true*, additionally the following properties will be evaluated and written to CK default_statistic_file (Fi statistic/statistic.sta). Note that an IP solver is necessary for that. The IP solver used is selected via CK lc_solver.
For the following two properties, passengers are routed along shortest paths in the Change&Go graph, where every change contributes the transfer penalty (CK ean_change_penalty) plus the transfer time calculated according to CK ean_model_weight_change. Hence the vehicle capacities are not respected.

SK **lc_perceived_time_average** Average travel time including a penalty for each transfer (CK ean_change_penalty) in the passenger routing described above.

SK **lc_prop_changes** Total number of transfers in the passenger routing described above.

In the example, the transfer time between the red and one of the other two lines is computed according to Formula 1 (see Section 7.9) as $\frac{60}{2} + \frac{60}{1} + 5 = 95$. Since capacities are ignored, the passengers between 1 and 3 are routed via the blue line with perceived travel time 4, while the passenger between 1 and 4 has a perceived travel time of 97. Therefore, SK lc_perceived_time_average is $\frac{2\cdot4+97}{3}$ = SV 35 and SK lc_prop_changes is SV 1.
For the following property, passengers are routed according to a minimum-cost multi-commodity flow in the Change&Go graph, respecting the capacities and the transfer times, taken as CK ean_default_minimal_change_time plus CK ean_change_penalty.

SK **lc_obj_travel_time** Average travel time including a transfer penalty (CK ean_change_penalty) in the routing described above. This is the objective value of the current solution the travel time model, see Section 3.3.4.

In the example the transfer times are estimated as $3 + 5 = 8$. Only one passenger between 1 and 3 can be routed via the blue line. The other will take the brown line with travel time $5 < 10 = 1 + 8 + 1$. The passenger between 1 and 4 has a travel time of 10. Therefore, the SK lc_obj_travel_time is $\frac{4+5+10}{3}$ = SV 6.33333.
For the following two properties, passengers are routed according to a minimum-cost multi-commodity flow in the Change&Go graph, respecting the capacities and the transfer times, computed according to CK ean_model_weight_change plus CK ean_change_penalty.

SK **lc_capacitated_perceived_time_average** Average travel time including a transfer penalty (CK ean_change_penalty) in the routing described above.

SK **lc_capacitated_prop_changes** Total number of transfers in the passenger routing described above.

In the example the transfer time between the red line and another line is again estimated as 95. Therefore, SK lc_capacitated_perceived_time_average is $\frac{4+5+97}{3}$ = SV 35.33333 and SK lc_capacitated_prop_changes is SV 1.
For the following property, the passengers are routed according to a minimum-cost path-based multi-commodity flow in the Change&Go graph, where every origin-destination pair (commodity) is restricted to paths that correspond to a shortest path in the PTN (with the length of a path including the waiting time at every intermediate station).

SK **lc_obj_direct_travelers** Total number of passengers travelling without transfer in the passenger routing described above. This is the objective of the direct travelers model for the current solution, see Section 3.3.2.

In the example, the passengers between 1 and 3 can only be routed along the path 1,2,3, which has length $4 < 5$ in the PTN. Therefore, one passenger uses the blue line and the other changes at 2. Also the passenger between 1 and 4 changes at 2. Therefore, SK `lc_obj_direct_travelers` is only SV 1.

5.6 Evaluation of the EAN

To evaluate the properties of the event activity network, you can use the makefile target

R `make ean-evaluate`

The following parameters will be evaluated and written to
CK `default_statistic_file` (Fi `statistic/statistic.sta`):

SK **ean_prop_events** $|\mathscr{E}|$ - number of events.

SK **ean_prop_events_arrival** $|\{e \in \mathscr{E} : e \text{ is } \textit{arrival}\}|$ - number of arrival events.

SK **ean_prop_events_departure** $|\{e \in \mathscr{E} : e \text{ is } \textit{departure}\}|$ - number of departure events.

SK **ean_prop_activities** $|\mathscr{A}|$ - number of activities.

SK **ean_prop_activities_change** $|\mathscr{A}_{change}|$ - number of change activities.

SK **ean_prop_activities_drive** $|\mathscr{A}_{drive}|$ - number of drive activities.

SK **ean_prop_activities_wait** $|\mathscr{A}_{wait}|$ - number of wait activities.

SK **ean_prop_activities_headway** $|\mathscr{A}_{headway}|$ - number of headway activities.

SK **ean_prop_activities_od** $|\{a \in \mathscr{A} : c_a > 0\}|$ - number of activities with more than 0 passengers.

SK **ean_prop_activities_od_change** $|\{a \in \mathscr{A}_{change} : c_a > 0\}|$ - number of change activities with more than 0 passengers.

SK **ean_prop_activities_od_drive** $|\{a \in \mathscr{A}_{drive} : c_a > 0\}|$ - number of drive activities with more than 0 passengers.

SK **ean_prop_activities_od_wait** $|\{a \in \mathscr{A}_{wait} : c_a > 0\}|$ - number of wait activities with more than 0 passengers.

SK **ean_time_average** $\frac{1}{\sum_{a \in \mathscr{A}} c_a} \sum_{a \in \mathscr{A}} c_a \cdot$ "duration assumption" - estimated average travel time. For duration assumption see 4.5.

Furthermore by setting config-parameter CK `ean_eval_extended` to *true* additionally the following parameter will be evaluated and written to CK `default_statistic_file` (Fi `statistic/statistic.sta`):

SK **ean_prop_activities_feas** $|\{a \in \mathscr{A} : U_a - L_a < T - 1\}|$ - number of activities that impose constraints.

SK **ean_prop_activities_objective** $|\{a \in \mathscr{A} : c_a > 0 \text{ or } U_a - L_a < T - 1\}|$ - number of activities that have an influence on the objective value.

SK **ean_prop_changes_od_max** $\max_{\substack{a \in \mathscr{A}_{change} \\ c_a > 0}}$ "duration assumption of a" - maximal used change duration.

SK **ean_prop_changes_od_min** $\min_{\substack{a \in \mathcal{A}_{change} \\ c_a > 0}}$ "duration assumption of a" - minimal used change duration.

SK **ean_prop_headways_dep** Are headways between departures only.

SK **ean_prop_headways_interstation** Do headways exist between different stations.

Additionally, the loads on the ean will be evaluated and compared to the maximal feasible load on the ptn edges given by the line concept. If the load on the ptn is invalid, i.e., too high, the respective ptn edges and their load will be written to CK `filename_invalid_loads` (Fi `statistic/Invalid-Loads.sta`). Additionally, the maximal load factor will be written as SK `ean_max_load_factor` to CK `default_statistic_file` (Fi `statistic/statistic.sta`).

5.7 Evaluation of the Timetable

To evaluate the properties of the timetable, you can use the makefile target

R `make tim-timetable-evaluate`

The following parameters will be evaluated and written to CK `default_statistic_file` (Fi `statistic/statistic.sta`):

SK **tim_feasible** $L_a \leq ((\pi_j - \pi_i - L_a) \bmod T) + L_a \leq U_a$ for all $(i, j) = a \in \mathcal{A}$ - Are lower and upper bounds on travel time on each activity respected.

SK **tim_obj_ptt1** $\sum_{(i,j)=a \in \mathcal{A}} c_a \left(\left(\left(\pi_j - \pi_i - L_a\right) \bmod T\right) + L_a\right)$ - Sum of weighted travel time. Weights correspond to the number of passengers specified in activity file.

SK **tim_obj_slack_average** $\frac{1}{|\mathcal{A}|} \sum_{(i,j)=a \in \mathcal{A}} \left(\pi_j - \pi_i - L_a\right) \bmod T$ - Average of slacks.

SK **tim_time_average** - Average travel time per passenger. The travel time for every OD pair is calculated according to its shortest path in the EAN.

SK **tim_perceived_time_average** - Average travel time per passenger. The travel time for every OD pair is calculated according to its shortest path in the EAN with additionally CK `ean_change_penalty` on change activities.

Furthermore by setting config-parameter CK `tim_eval_extended` to *true* additionally the following parameter will be evaluated and written to CK `default_statistic_file` (Fi `statistic/statistic.sta`):

SK **tim_obj_slack_drive_average** $\frac{1}{|\mathcal{A}_{drive}|} \sum_{(i,j)=a \in \mathcal{A}_{drive}} \left(\pi_j - \pi_i - L_a\right) \bmod T$ - average slack on drive activities.

SK **tim_obj_slack_wait_average** $\frac{1}{|\mathcal{A}_{wait}|} \sum_{(i,j)=a \in \mathcal{A}_{wait}} \left(\pi_j - \pi_i - L_a\right) \bmod T$ - average slack on wait activities.

SK **tim_obj_slack_change_average** $\frac{1}{|\mathcal{A}_{change}|} \sum_{(i,j)=a \in \mathcal{A}_{change}} \left(\pi_j - \pi_i - L_a\right) \bmod T$ - average slack on change activities.

SK **tim_obj_slack_headway_average** $\frac{1}{|\mathcal{A}_{headway}|} \sum_{(i,j)=a \in \mathcal{A}_{headway}} \left(\pi_j - \pi_i - L_a\right) \bmod T$ average slack on headway activities.

SK **tim_overcrowded_time_average** the average time that passengers are overcrowded in the vehicles. Does not take any rerouting into account, i.e., will use the passenger weights currently stored in the EAN. A drive or wait activity is overcrowded, if the number of passengers using the activity is larger than CK `gen_passengers_per_vehicle`.

SK **tim_prop_changes_od_max** $\max\limits_{\substack{(i,j)=a\in\mathcal{A}_{change}\\ c_a>0}} \left(\pi_j - \pi_i\right) \mathrm{mod} T$ - maximal used change duration.

SK **tim_prop_changes_od_min** $\min\limits_{\substack{(i,j)=a\in\mathcal{A}_{change}\\ c_a>0}} \left(\pi_j - \pi_i\right) \mathrm{mod} T$ - minimal used change duration.

SK **tim_number_of_transfers** Weighted number of transfers.

5.7.1 Capacitated Routing

There is also the possibility to do a capacitated routing of the passengers in the EAN by solving a multicommodity flow problem. To do so, set the config-parameter

CK `tim_cap_eval` CV `true`.

First, the standard timetable evaluation is executed and written to the statistics-file. The algorithm then creates a routing network from the EAN. For each Stop $s \in \mathcal{S}$ we add two additional nodes to the EAN, one origin event and one destinantion event. We add arcs from the origin event to all departure events corresponding to stop s and from all arrival events of s to the destination event, those activities are added to $\mathcal{A}$. Let origin(s) be the origin event to the stop s and dest(s) the correspronding destination event. The set of all origin events is denoted by $\mathcal{E}_{origin}$ and the set of all destination events by $\mathcal{E}_{dest}$.
Let duration(a) be the duration of activity a, wich comes from the existing timetable. If the config-parameter CK `tim_cap_eval_tt` is `false`, we don't use the timetable but the lower bound of the activity. If a is a change activity, we add a penalty time CK `ean_change_penalty`. Furthermore we need the capacity C of each vehicle, given by CK `gen_passengers_per_vehicle`.
For each activity a and each stop s there is a variable $x_{s,a}$ that states how many passengers starting from stop s use activity a. Depending on CK `tim_cap_eval_integer_flow` they are integer or continuous variables. We set up the following optimization problem and solve it with Gurobi.

$$\min \quad \sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}} \mathrm{duration}(a)\cdot x_{s,a} \tag{5.1a}$$

$$\text{s.t.} \quad \sum_{\substack{a\in\mathcal{A}\\ a\in\delta^+(e)}} x_{s,a} - \sum_{\substack{a\in\mathcal{A}\\ a\in\delta^-(e)}} x_{s,a} = \sum_{u\in\mathcal{S}} C_{s,u} \qquad \forall e\in\mathcal{E}_{origin}, \forall s\in\mathcal{S}, \tag{5.1b}$$

$$\sum_{\substack{a\in\mathcal{A}\\ a\in\delta^-(e)}} x_{s,a} - \sum_{\substack{a\in\mathcal{A}\\ a\in\delta^+(e)}} x_{s,a} = C_{s,t} \qquad \forall e\in\mathcal{E}_{dest}, \forall s\in\mathcal{S} : e = \mathrm{dest}(t), \tag{5.1c}$$

$$\sum_{\substack{a\in\mathcal{A}\\ a\in\delta^+(e)}} x_{s,a} - \sum_{\substack{a\in\mathcal{A}\\ a\in\delta^-(e)}} x_{s,a} = 0 \qquad \forall e\in\mathcal{E}_{arr}\cup\mathcal{E}_{dep}, \forall s\in\mathcal{S}, \tag{5.1d}$$

$$\sum_{s\in\mathcal{S}} x_{s,a} \le C \qquad \forall a\in\mathcal{A}_{drive}, \tag{5.1e}$$

$$x_{s,a}\in\{0,...,C\}\subseteq\mathbb{N} \qquad \forall s\in\mathcal{S}, a\in\mathcal{A} \tag{5.1f}$$

Here, $\delta^+(e)$ is the set of all outgoing arcs of the node e and $\delta^-(e)$ is the set of all ingoing arcs. $C_{u,v}$ denotes the OD-value from stop u to stop v, i.e. the number of passengers that start from the stop u and end at stop v according to the given OD-Matrix. If CK `tim_cap_eval_integer_flow` is `false`, constraint (5.1f) is replaced by $x_{s,a}\in[0,C]\subseteq\mathbb{R}$.
If CK `tim_cap_eval_accumulate_on_links` is `true`, constraint (5.1e) is replaced by

$$\sum_{\substack{s\in\mathcal{S}\\ a\in\mathrm{activities}(l)}} x_{s,a} \le C\cdot|\mathrm{activities}(l)| \qquad \forall l \text{ Link in PTN}$$

where activities(l) is the set of all activities a that belong to the link l of the PTN. This gives a weaker version of the IP.

The following parameters will be evaluated and written to
CK default_statistic_file (Fi statistic/statistic.sta):

SK **tim_capacitated_travel_time** $\sum_{a \in \mathcal{A}}$ passengers(a) · duration(a) - sum of all travel times, where passengers(a) denotes the number of passengers using activity a according to the optimal solution of the IP.

SK **tim_capacitated_travel_time_average** travel time divided by the total number of passengers - average travel time per passenger.

SK **tim_capacitated_percieved_travel_time** $\sum_{a \in \mathcal{A}}$ passengers(a) · percieved_duration(a) - sum of all percieved travel times. percieved_duration(a) = duration(a)+ CK ean_change_penalty if $a \in \mathcal{A}_{wait}$ and percieved_duration(a) = duration(a) for all other activities a.

SK **tim_capacitated_percieved_travel_time_average** percieved travel time divided by the total number of passengers - average percieved travel time per passenger.

SK **tim_capacitated_max_load** $\max_{a \in \mathcal{A}} \frac{\text{passengers}(a)}{C}$ - maximal percentage load. Could be greater than 1, if CK tim_cap_eval_accumulate_on_links is true.

5.8 Evaluation of the Tariff created by Tariff Planning

To evaluate the properties of a tariff created by

R make taf-tariff

or the price matrices created by

R make taf-tariff-price-matrix

run

R make taf-tariff-evaluate

The following configuration parameters control the evaluation:

- CK taf_evaluate_old_prices, points towards a price matrix. By default this is the reference price matrix CV basis/Reference-Price-Matrix.giv.
- CK taf_evaluate_new_prices, points towards a price matrix. By default this is the tariff price matrix CV tariff/Price-Matrix.taf.

The following parameters will be evaluated and written to
CK default_statistic_file (Fi statistic/statistic.sta), where they are sorted alphabetically:

SK **taf_revenue_old** The revenue genereated by the tariff if all passengers pay the prices in CK taf_evaluate_old_prices.

SK **taf_revenue_new** The revenue genereated by the tariff if all passengers pay the prices in CK taf_evaluate_new_prices.

SK **taf_od_pairs_increased_prices** The absolute number of OD pairs for which the price increases comparing the prices in CK taf_evaluate_old_prices to the prices in CK taf_evaluate_new_prices.

SK **taf_od_pairs_decreased_prices** The absolute number of OD pairs for which the price decreases comparing the prices in CK taf_evaluate_old_prices to the prices in CK taf_evaluate_new_prices.

SK **taf_passengers_increased_prices** The absolute number of passengers for which the price increases comparing the prices in CK taf_evaluate_old_prices to the prices in CK taf_evaluate_new_prices.

SK **taf_passengers_decreased_prices** The absolute number of passengers for which the price decreases comparing the prices in CK taf_evaluate_old_prices to the prices in CK taf_evaluate_new_prices.

SK **taf_objective_sum_unit** The sum of absolute deviations between the new prices (CK taf_evaluate_new_prices) and the old prices (CK taf_evaluate_old_prices) all weighted with one.

SK **taf_objective_sum_od** The sum of absolute deviations between the new prices (CK taf_evaluate_new_prices) and the old prices (CK taf_evaluate_old_prices) weighted with the od values.

SK **taf_objective_sum_reference_inverse** The sum of absolute deviations between the new prices (CK taf_evaluate_new_prices) and the old prices (CK taf_evaluate_old_prices) weighted with the inverse of the old prices.

SK **taf_objective_max_unit** The maximum of absolute deviations between the new prices (CK taf_evaluate_new_price and the old prices (CK taf_evaluate_old_prices) all weighted with one.

SK **taf_objective_max_od** The maximum of absolute deviations between the new prices (CK taf_evaluate_new_prices) and the old prices (CK taf_evaluate_old_prices) weighted with the od values.

SK **taf_objective_max_reference_inverse** The maximum of absolute deviations between the new prices (CK taf_evaluate_new_prices) and the old prices (CK taf_evaluate_old_prices) weighted with the inverse of the old prices.

For the calculation of the objective values only non-trivial OD pairs are considered as described in section 7.12. If there are zero prices for non-trivial OD pairs, SK taf_objective_sum_reference_inverse and SK taf_objective_max_reference_inverse are None.

5.9 Evaluation of the Trips

To evaluate the properties of the trips, you can use the makefile target

R make ro-trips-evaluate

The following parameters will be evaluated and written to
CK default_statistic_file (Fi statistic/statistic.sta):

SK **ro_trips_feasible** whether the trips are feasible. The trips are considered feasible if they cover every event in the aperiodic event activity network and no event is used in multiple trips.

SK **ro_prop_trips** $|\mathcal{T}|$ - number of trips.

SK **ro_prop_stops_at_begin_or_end** Number of stations that are start or end station of a trip.

5.10 Evaluation of the Vehicle Schedules

To evaluate the properties of the vehicle scheduling, you can use the makefile target

R make vs-vehicle-schedules-evaluate

This evaluation will read the following parameters from the config-files:

CK **vs_vehicle_costs** The cost of a vehicle, needed to determine the costs

CK **vs_eval_cost_factor_empty_length** the cost of a kilometer on an empty trip

CK **vs_eval_cost_factor_empty_duration** the cost for the vehicle driving on an empty trip for an hour

CK **vs_eval_cost_factor_full_length** the cost of a kilometer serving a line

CK **vs_eval_cost_factor_full_duration** the cost for the vehicle driving for an hour while serving a line

The following parameters will be evaluated and written to
CK `default_statistic_file` (Fi `statistic/statistic.sta`):

SK **vs_cost** The cost of the vehicle schedule, weighted according to the parameters above.

SK **vs_feasible** Whether the current vehicle schedule is feasible. This only checks, whether the time for the empty trips is sufficient, not the viability of the covered lines.

SK **vs_circulations** The number of circulations in the vehicle schedule.

SK **vs_vehicles** The number of used vehicles in the vehicle schedule.

SK **vs_empty_distance** The distance a vehicle drives without passengers in the current vehicle schedule, given in kilometers.

SK **vs_empty_distance_with_depot** The distance a vehicle drives without passengers in the current vehicle schedule including driving from and to the depot, given in kilometers. Will be the same as above if the depot index is not set.

SK **vs_empty_duration** The time needed for empty trips in the current vehicle schedule, given in minutes. Does not include waiting in stations.

SK **vs_empty_duration_with_depot** The time needed for empty trips in the current vehicle schedule including driving from and to the depot, given in minutes. Does not include waiting in stations. Will be the same as above if the depot index is not set.

SK **vs_empty_trips** The number of empty trips in the current vehicle schedule. Does not include waiting in stations.

SK **vs_emtpy_trips_depot** The number of empty trips to and from the depot.

SK **vs_minimal_waiting_time** The minimal waiting time in a station between two consecutive trips, served by the same vehicle. Only if the station is not changed in the empty trip.

SK **vs_maximal_waiting_time** The maximal waiting time in a station between two consecutive trips, served by the same vehicle. Only if the station is not changed in the empty trip.

SK **vs_average_waiting_time** The average waiting time in a station between two consecutive trips, served by the same vehicle. Only if the station is not changed in the empty trip.

SK **vs_full_distance** The distance a vehicle drives with passengers in the current vehicle schedule, given in kilometers.

SK **vs_full_duration** The time needed for serving trips in the current vehicle schedule, given in minutes.

5.11 Evaluation of the Disposition Timetable

To evaluate the properties of the delay management, you can use the makefile target

R `make dm-disposition-timetable-evaluate`

The following parameters will be evaluated and written to
CK `default_statistic_file` (Fi `statistic/statistic.sta`):

SK **dm_feasible** Whether the disposition timetable is feasible according to the lower bounds of the activities.

SK **dm_obj_changes_missed_od** The number of missed used connections in the disposition timetable.

SK **dm_obj_delay_events_average** The average delay of the events in the disposition timetable.

SK **dm_obj_dm2** The objective value of the `DM_method DM2`.

SK **dm_obj_dm2_average** The objective value of `DM_method DM2`, divided by the number of passengers.

SK **dm_prop_events_delayed** The number of delayed events in the disposition timetable.

SK **dm_prop_headways_swapped** The number of headways swapped in the disposition timetable, compared to the original timetable.

SK **dm_time_average** The average travel time of the passengers according to the disposition timetable.

Furthermore by setting config-parameter CK `DM_eval_extended` to *true* additionally the following parameters will be written to CK `default_statistic_file` (Fi `statistic/statistic.sta`). Note, that the rollout must have been done with the parameter `ro_rollout_passenger_paths` set to *true*.

SK **dm_obj_dm1** The objective value of `DM_method DM1`.

SK **dm_obj_dm1_average** The objective value of `DM_method DM1`, divided by the number of passengers.

SK **dm_passenger_delay** The delay of the passenger after rerouting, given the distribution of `DM_passenger_routing_arrival_on_time`.

SK **dm_passenger_delay_average** The average delay of the passenger after rerouting, given the distribution of `DM_passenger_routing_arrival_on_time`.

Additionally, when the config-parameter CK `DM_eval_extended` is set to *true*, the following distributions will be written to Fi `./statistic/statistic_dist.sta`:

SK **dm_dist_delays_events** For each possible delay (in seconds) there is one entry giving the number of events with this delay in the disposition timetable.

SK **dm_dist_delays_od** For each possible delay (in seconds) there is one entry giving the number of passengers with this delay in the disposition timetable.

Chapter 6

Overview of Supported Integer Programming Solvers

Different algorithms in LinTim use integer programm solvers. Altogether, the following solvers are currently used in LinTim

- Gurobi
- Xpress
- CPLEX
- SCIP
- COIN
- CBC
- GLPK

For an overview, which algorithms support which solver choice, see Table 6.1. For information on how to combine LinTim with one of the solvers above, see Section 1.2.1.

Algorithm	Supported Solvers	Config-Key	Reference
Stop Location		CK sl_solver	Section 3.1
dsl, dsl-tt, dsl-tt-2	Xpress		
tt	Gurobi, Xpress, CPLEX, SCIP, GLPK		
Line Pool Generation		CK lc_solver	Section 3.2
all	Gurobi, Xpress, CPLEX, SCIP, GLPK		
Line Planning		CK lc_solver	Section 3.3
Cost	Gurobi, Xpress, CPLEX, SCIP, GLPK		
CostRestricted	Gurobi, Xpress, CPLEX, SCIP, GLPK		
CostExtended	Gurobi, Xpress, CPLEX, SCIP, GLPK		
Direct	Gurobi, Xpress, CPLEX, SCIP, GLPK		
DirectRelaxation	Xpress		
DirectRestricted	Gurobi, Xpress, CPLEX, SCIP, GLPK		
CostDirect	Xpress		
Game	Xpress		
Min-Changes	Xpress		
Travelling-Time-CG	Xpress		
Traveling Time IP	Gurobi		
Timetabling		CK tim_solver	Section 3.6
IP	Gurobi, Xpress, CPLEX, SCIP, GLPK		
Aperiodic-robust	Xpress		
Cycle-base	Gurobi, Xpress, CPLEX, SCIP, COIN, CBC, GLPK		
Tariff Planning		CK taf_solver	Section 7.12
Distance	Gurobi		
Beeline	Gurobi		
Zone	Gurobi		
Vehicle Scheduling		CK vs_solver	Section 3.8
Canal-based	Xpress		
IP	Gurobi, Xpress, CPLEX, SCIP, GLPK		
Line-Based	Xpress		
Delay Management		CK DM_solver	Section 3.9
IP	Gurobi, Xpress		
Integrated Models		CK int_solver	Section 3.10
all	Gurobi, Xpress, CPLEX, SCIP, COIN, CBC, GLPK		
Tools			
Line-Rearrange	Xpress		
Regenerate-load (spanner-based)	Gurobi, CPLEX, Xpress	CK load_generator_solver	Section 4.3.5

Table 6.1: Table of all algorithms using an integer programming solver

Chapter 7

Configuration Parameters

This section describes the configuration parameter available in LinTim. For a detailed description of the different algorithms, see Section 3. There, you find a list of corresponding parameters for the different algorithms.

7.1 General

CK **console_log_level** the log level to use, determines the amount of output on the console. The possible log levels are:

- CV ERROR: Only write error messages
- CV WARN: Additionally write warnings
- CV INFO: The default. Will give general information about the current step of the algorithm used.
- CV DEBUG: This includes many information to better understand the behavior of the algorithm, e.g., information about substeps of the algorithm, the read configuration values, the read input files, solver output, ...

CK **gen_passengers_per_vehicle** the capacity of the vehicles.

CK **gen_walking_utility** the penalty factor for walking.

CK **period_length** the length of the periodic planning period.

7.2 Stop Location

CK **sl_destruction_allowed** whether it is allowed to destroy existing stops

CK **sl_distance** the distance function to use

CK **sl_eval_extended** activate the extended evaluation

CK **sl_max_walking_time** the maximal walking time allowed for passengers

CK **sl_mip_gap** the mip optimization gap for the solver, 0.1 equals a gap of 10 % (-1=use default value).

CK **sl_model** the model to use. For an overview on all models, see Section 3.1.

CK **sl_radius** the covering radius of a stop

CK **sl_solver** determine the solver to be used. Note that not all solvers are supported by all models.

CK **sl_threads** determine the maximal number of threads to use for the solver (-1=use default value, i.e., no restriction). Note that this will only be used for a possible solver integration of the chosen model, not for the rest of the algorithm.

CK **sl_timelimit** the time limit for the solver in seconds (-1=use default value).

CK **sl_write_lp_file** whether to write the lp file of the model to solve

7.3 OD

CK **od_draw_conversion_factor** scaling factor for the graph visualization of the OD data.

CK **od_visualization_lower_bound** percentage of the maximum OD pair value. Lower valued OD pairs will not be displayed in the graph visualization of the OD data.

CK **od_visualization_upper_bound** percentage of the maximum OD pair value. Higher valued OD pairs will not be displayed in the graph visualization of the OD data.

CK **od_visualization_use_annotations** whether to use annotations in the heatmap visualization of the OD data.

CK **od_visualization_use_edge_colors** whether to use the edge color (instead of the edge width) to indicate passenger numbers in the graph visualization of the OD data.

CK **od_visualization_use_heatmap** whether to use a heatmap (instead of a graph) for the visualization of the OD data

CK **od_visualization_use_log_scale** whether to visualize the logarithmic values of the passenger numbers.

CK **od_visualization_max_edge_width** the maximal edge width in points in the graph visualization where the edge width is used to visualize passenger numbers.

7.4 PTN

CK **ptn_draw_use_coordinates** whether stop-coordinates are used for plotting the PTN or stops are arranged automatically.

CK **ptn_draw_interactive_graph_edge_labels** whether edge labels are displayed in the interactive PTN visualizations

7.5 Line Planning

CK **lc_budget** the budget for the line concept, i.e., the maximal weighted sum of the line costs and the computed frequencies.

CK **lc_common_frequency_divisor** the common divisor of the frequencies, i.e., a frequency is only allowed if it is a multiple of this value. A value <= 0 will test any system frequency (except for 1) and output the best solution.

CK **lc_direct_optimize_costs** whether to additionally optimize the costs in the direct model, see Section 3.3.2. When set to CV `true`, the model will optimize a weighted sum of line costs and direct travelers and will use CK `lc_mult_relation` as a weight.

CK **lc_eval_extended** enables the extended evaluation. Needs an IP solver present. For more information, see Section 5.5.

CK **lc_maximal_frequency** the maximal frequency value allowed

CK **lc_mult_relation** weighting factor in a convex combination of costs and direct travelers. A value of 0 is equivalent to solving the direct travelers model while a value of 1 is equivalent to solving the cost model, therefore the value should be in [0, 1].

CK **lc_mip_gap** the mip optimization gap for the solver, 0.1 equals a gap of 10 % (-1=use default value).

CK **lc_model** the line planning model to use. For an overview of all models, see Section 2.3.

CK **lc_number_of_possible_frequencies** the maximal number of different frequency values allowed to use.

CK **lc_respect_fixed_lines** whether to respect fixed lines, i.e., lines with a given frequency

CK **lc_respect_forbidden_edges** whether to respect forbidden links, i.e., links in the PTN that may not be used by the public transport model currently optimized. This may e.g. be the case when optimizing a bus network and considering a PTN containing train tracks.

CK **lc_solver** determine the solver to be used. Note that not all solvers are supported by all models.

CK **lc_threads** determine the maximal number of threads to use for the solver (-1=use default value, i.e., no restriction). Note that this will only be used for a possible solver integration of the chosen model, not for the rest of the algorithm.

CK **lc_timelimit** the time limit for the solver in seconds (-1=use default value).

CK **lc_write_lp_file** whether to write the lp file of the model to solve

7.6 Ridepooling

CK `rpool_model` Model to use for ride pool generation. Available options are CV `all`, configvalue-demand_heuristic and CV `tree_based`.

CK `rpool_min_edges` Minimal number of edges in a ridepooling area. The default value is 1.

CK `rpool_max_edges` Maximal number of edges in a ridepooling area. The default value is 4.

CK `rpool_costs_fixed` Costs for one vehicle to operate in one ridepooling area.

CK `rpool_load_factor` Used by the demand heuristic. Only edges where the load is at most this parameter times the line vehicle capacity are added to ridepooling areas.

CK `rpool_nb_edges_to_trim` Number of edges to trim the lines in the existing linepool.

CK `rc_model` model to use to ride concept determination.

CK `rc_solver` specifies the solver to use the IP formulations.

CK `rc_timelimit` Timelimit in seconds for the solver. Value of -1 means no restriction.

CK `rc_mip_gap` MIP Gap for the solver. Value of -1 means no restriction.

CK `rc_threads` Number of threads to use for the solver. Value of -1 means no restriction.

CK `rc_write_lp_file` Whether to write the IP formulation as a file or not.

CK `rc_passengers_per_vehicle` Capacity of a single ridepooling vehicle.

7.7 Load Generation

CK **load_generator_add_additional_load** whether to add additional load per link, given in CK filename_additional_load_file (Fi basis/Additional-Load.giv).

CK **load_generator_fixed_upper_frequency** the fixed upper frequency bound of a link after load generation. Whether this or a factor of the lower frequency bound is used is determined by CK load_generator_fix_upper_frequency

CK **load_generator_fix_upper_frequency** whether a fixed upper frequency bound (CV true) or a multiple of the lower bound should be used for a link after load generation.

CK **load_generator_lower_frequency_factor** the factor to multiply the minimal lower frequency bound (given by the capacity of the vehicle) to obtain the new lower frequency bound. The result is rounded up.

CK **load_generator_max_iteration** determines the number of iterations allowed before the algorithms terminates, if no convergence is observed

CK **load_generator_min_change_time_factor** the factor to weight the minimal change time (CK ean_default_minimal_change_time) to obtain the change objective function for routing. The change objective function will never be higher than the maximal change time (CK ean_default_maximal_change_time)

CK **load_generator_model** how to route the passengers

- CV **LOAD_FROM_EAN** use the current weights in the EAN to determine the weights on the PTN links. The EAN has to be present.
- CV **LOAD_FROM_PTN** determine new passenger routes based on the other parameters given.
- CV **spanners** determine new passenger routes using a MIP formulation considering building and routing costs and maximum detour factors.

CK **load_generator_number_of_shortest_paths** the number of shortest paths to use. For every passenger, the given number of shortest paths are computed and the passengers are distributed with a logit model using CK load_generator_sp_distribution_factor on their different paths

CK **load_generator_scaling_factor** the factor for the reward or reduction cost factor in the objective function when CK load_generator_type is set to CV REWARD or CV REDUCTION. A higher value will result in larger detours for the passengers.

CK **load_generator_sp_distribution_factor** the parameter for the logit model used to distribute the passenger when CK load_generator_number_of_shortest_paths is bigger than 1.

CK **load_generator_type** the different ptn load generator types.

- CV **SP** use the travel time shortest paths for the passengers, depending on the travel time approximation used.
- CV **REDUCTION** adds a penalty depending on the usage of the edge in the PTN (high penalty for low usage)
- CV **REWARD** reward an edge more, if less passengers are needed to fill the next vehicle on the edge

For a more detailed description of the different models, see [8].

CK **load_generator_use_cg** whether to use a change and go network for routing. This includes knowledge of the line pool, allowing to consider transfers. The line pool needs to be present!

CK **load_generator_upper_frequency_factor** the factor to multiply the lower frequency bound to obtain the new upper frequency bound. The result is rounded up. Whether this or a fixed bound is used depends on CK load_generator_fix_upper_frequency.

CK **load_generator_building_cost** sets building cost as the objective (CV obj) or a constraint (CV cons). If building cost should not be considered in the model, set this to CV none.

CK **load_generator_travel_time** sets total travel time as the objective (CV obj) or a constraint (CV cons). If total travel time should not be considered in the model, set this to CV none.

CK **load_generator_detour_factor** sets maximum detour factor as the objective (CV obj) or a constraint (CV cons). If maximum detour factor should not be considered in the model, set this to CV none.

CK **load_generator_max_building_cost** upper bound for building cost used if CK load_generator_building_cost is set to CV cons. Note that this value is converted according to CK gen_conversion_length.

CK **load_generator_max_travel_time** upper bound for total travel time used if CK load_generator_travel_time is set to CV cons. Note that this value is converted according to CK gen_conversion_length.

CK **load_generator_max_detour** upper bound for maximum detour factor used if CK load_generator_detour_factor is set to CV cons.

CK **load_generator_gen_cuts** when this is set to CV true, so-called valid inequalities are added to the spanner MIP. These have no effect on the solution, but tend to greatly improve computational performance.

CK **load_generator_remove_unused_edges** when this is set to CV true, after solving the model, the unused edges are removed from CK default_edges_file (Fi basis/Edge.giv).

CK **load_generator_solver** determine the solver to be used.

CK **load_generator_threads** determine the maximal number of threads to use for the solver (-1=use default value, i.e., no restriction). Note that this will only be used for a possible solver integration of the chosen model, not for the rest of the algorithm.

CK **load_generator_timelimit** the time limit for the solver in seconds (-1=use default value).

CK **load_generator_mip_gap** the mip optimization gap for the solver, 0.1 equals a gap of 10 % (-1=use default value).

CK **load_generator_write_lp_file** whether to write the lp file of the model to solve.

7.8 Load Visualization

CK **filename_loads_graph_file** filename under which the load visualization is saved.

CK **loads_draw_conversion_factor** scaling factor for the load visualization.

CK **loads_graph_lower_bound** percentage of the maximum load. Lower loads will not be displayed in the load visualization.

CK **loads_graph_max_edge_width** maximum edge width used in the load visualization which uses edge width as a medium to display the traffic load.

CK **loads_graph_upper_bound** percentage of the maximum load. Higher loads will not be displayed in the load.

CK **loads_graph_use_edge_color** whether to use the edge color to represent the traffic loads. Otherwise edge width will be used.

7.9 Periodic EAN

CK **ean_algorithm_shortest_paths** the algorithm to use to compute the shortest paths in the ean. Choices are CV JOHNSON, CV FLOYD, CV FIBONACCI_HEAP and CV TREE_MAP_QUEUE.

CK **ean_change_penalty** the change penalty for routing, i.e., a penalty for each transfer a passenger needs to take during their journey. Given in time units.

CK **ean_construction_skip_passenger_distribution** whether to skip the initial passenger distribution during ean construction.

CK **ean_construction_target_model_frequency** whether to include the frequency of lines only as attributes in the ean (CV FREQUENCY_AS_ATTRIBUTE) or include multiple frequency repetitions, connected by synchronization activities (CV FREQUENCY_AS_MULTIPLICITY). Note that CV FREQUENCY_AS_ATTRIBUTE can not be handled by all timetabling algorithms.

CK **ean_construction_target_model_headway** how to model headways in the ean. The following options are available:

- CV **NO_HEADWAYS** create new headway activities
- CV **SIMPLE** creates a headway between every two departures from the same station using the same link
- CV **PRODUCT_OF_FREQUENCIES** When using CV FREQUENCY_AS_ATTRIBUTE as a CK ean_construction_target_model_frequency, this will create all the necessary headway activities between the corresponding departure events, i.e., will include multiple headway activities for lines with frequency > 1. Note that this is not necessary when CK ean_construction_target_model_frequency is set to CV FREQUENCY_AS_MULTIPLICITY, in this case this model is the same as CV SIMPLE.
- CV **LCM_OF_FREQUENCIES** This will create as many headway activities as the least commom multiple of the two corresponding line frequencies between all departures from the same station using the same link.
- CV **LCM_REPRESENTATION** For headway creation, this behaves the same as CV SIMPLE but some timetabling models will respect these headways the same as CV LCM_OF_FREQUENCIES later on.

CK **ean_default_maximal_change_time** the default maxmimal change time at a station

CK **ean_default_maximal_waiting_time** the default maximal waiting time at a station

CK **ean_default_minimal_change_time** the default minimal change time at a station

CK **ean_default_minimal_waiting_time** the default minimal waiting time at a station

CK **ean_discard_unused_change_activities** when set to CV true, this will remove all change activities from the ean that do not have a positive weight after the initial passenger distribution

CK **ean_dump_initial_duration_assumption** when set to CV true, this will output the initial duration assumption of every activity, i.e., the computed duration of every activity in the initial passenger distribution. CK filename_initial_duration_assumption (Fi timetabling/Initial-duration-assumption-periodic.giv) will be used for output.

CK **ean_individual_station_limits** when set to CV true, individual station limits for change and waiting time will be used. For information on how to give these limits, see the documentation for CK filename_station_limit_file (Fi basis/Station-Limits.giv).

CK **ean_initial_duration_assumption_model** How to compute the initial duration assumption. The following options are available:

- CV **AUTOMATIC** fully automated initial durations, based on CK `ean_model_weight_change`, CK `ean_model_weight_drive` and CK `ean_model_weight_wait`.
- CV **SEMI_AUTOMATIC** the initial duration for individual activities can be given. For information on how to give these durations, see the documentation for CK `filename_initial_duration_assumption` (Fi `timetabling/Initial-duration-assumption-periodic.giv`).

CK **ean_model_weight_change** determines how to estimate the transfer time between two lines without a given timetable. For a transfer between the lines l_1 and l_2, let f_1 and f_2 be the respective frequencies and T the CK `period_length`. The following options are available:

- CV **FORMULA_1** $\frac{T}{f_1} + \frac{T}{f_2} +$ CK `ean_change_penalty`.
- CV **FORMULA_2** $\frac{T}{2 \cdot f_1 \cdot f_2} +$ CK `ean_change_penalty`.
- CV **FORMULA_3** $\frac{T}{2 \cdot f_2} +$ CK `ean_change_penalty`.
- CV **MINIMAL_CHANGING_TIME** CK `ean_default_minimal_change_time` + CK `ean_change_penalty`.

CK **ean_model_weight_drive** determines how to estimate the drive time on an infrastructure edge without a given timetable. The following options are available:

- CV **AVERAGE_DRIVING_TIME** using the average between minimal and maximal travel time of the infrastructure edge.
- CV **EDGE_LENGTH** using the edge length of the infrastructure edge.
- CV **MINIMAL_DRIVING_TIME** using the minimal travel time of the infrastructure edge.
- CV **MAXIMAL_DRIVING_TIME** using the maximal travel time of the infrastructure edge.

CK **ean_model_weight_wait** determines how to estimate the waiting time when traversing a stop in a line without a given timetable. The following options are available:

- CV **AVERAGE_WAITING_TIME** using the average of CK `ean_default_minimal_waiting_time` and CK `ean_default_maximal_waiting_time`.
- CV **MAXIMAL_WAITING_TIME** using CK `ean_default_maximal_waiting_time`.
- CV **MINIMAL_WAITING_TIME** using CK `ean_default_minimal_waiting_time`.
- CV **ZERO_COST** assume the waiting time to be 0.

CK **ean_random_shortest_paths**

CK **ean_use_walking** whether to allow walking transfers in the EAN

7.10 Debug

CK **debug_paths_in_ptn** when set to CV `true`, some routing methods will output the found ptn paths to CK `default_debug_od_link_paths_file` (Fi `Debug/ODLinkPaths.dbg`)

CK **debug_paths_in_ean** when set to CV `true`, some routing methods will output the found ean paths to CK `default_debug_od_activity_paths_file` (Fi `Debug/ODActivityPaths.dbg`).

7.11 Timetabling

CK **tim_mip_gap** the mip optimization gap for the solver, 0.1 equals a gap of 10% (-1=use default value).

CK **tim_model** the timetabling model to use. For an overview of all models, see Section 3.6

CK **tim_pesp_ip_solution_limit** limit the number of feasible solutions found. Only implemented in Gurobi. Set to 0 to deactivate.

CK **tim_pesp_ip_best_bound_stop** a best bound stop criterion, only implemented for Gurobi. For details, see Gurobi documentation. Set to 0 to deactivate.

CK **tim_pesp_ip_mip_focus** set the MIPFocus, only implemented for Gurobi. For details, see Gurobi documentation. Set to 0 to deactivate.

CK **tim_solver** the solver to use for timetabling. Which solvers are implemented depends on the chose CK `tim_model`, see the corresponding documentation.

CK **tim_threads** determine the maximal number of threads to use for the solver (-1=use default value, i.e., no restriction). Note that this will only be used for a possible solver integration of the chosen model, not for the rest of the algorithm.

CK **tim_timelimit** the time limit to use for the solver in seconds (-1 = use default value).

CK **tim_use_old_solution** whether to use the current solution as a starting solution, only implemented for Gurobi and the pesp ip.

CK **tim_write_lp_file** whether to write the lp file of the model to solve

7.12 Tariff Planning

CK **taf_model** either CV `flat`, CV `beeline_distance`, CV `network_distance` or CV `zone`. Determining the model used to calculate a new tariff.

CK **taf_objective** either CV `sum_absolute_deviation` or CV `max_absolute_deviation`. Determining whether the sum or maximum of absolute price deviations is minimized.

CK **taf_weights_objective** either CV `od`, CV `unit` or CV `reference-inverse`. Determining whether the price deviations in the objective are weighted by the OD data, by unit weights or by the inverse of the given reference prices.

CV **taf_zone_counting** either CV `single` or CV `multiple`. If CV `single`, then each zone is only counted once when determining the number of traversed zones of a path. If CV `multiple`, a zone is counted each time that it is entered.

CV **taf_zone_n_zones** positive integer number, specifies the maximum number of zones when calculating a new zone tariff.

CK **taf_zone_enforce_all_zones** boolean, determines whether exactly CK `taf_zone_n_zones`-many zones (CV `true`) or at most that many zones (CV `false`) must be determined.

CK **taf_zone_connected** boolean, specifies whether the subgraph of a zone, induced by the nodes assigned to the zone, needs to be connected (in case of a directed graph it is weakly connected).

CK **taf_zone_enforce_no_elongation** boolean, determining whether the no-elongation property must be satisfied. This property ensures, that it is never cheaper for passengers to buy tickets for more zones than they actually want to travel through. Let p_k be the price of a path that uses k zones. The no-elongation property is satisfied if it holds that

$$p_k \leq p_{k+1} \qquad \text{for all } k \in \{1, ..., (\text{CK taf\_zone\_n\_zones}) - 1\}.$$

CK **taf_zone_enforce_no_stopover** boolean, determining whether the no-stopover property must be satisfied. This property ensures that it is never cheaper for passengers to buy two separate tickets for one journey and combine them instead of buying one ticket for the whole journey. Let p_k be the price of a path that uses k zones. The no-stopover property in the case of single counting holds if

$$p_k \leq p_i + p_j \qquad \text{for all } k \in \mathbb{N}_{\geq 1}, i, j \in \{1, ..., k\} \text{ with } i + j \geq k + 1.$$

In the case of multiple counting the property holds if

$$p_k \leq p_i + p_j \qquad \text{for all } k \in \mathbb{N}_{\geq 1}, i, j \in \{1, ..., k\} \text{ with } i + j = k + 1.$$

CK **taf_zone_symmetry_breaking** determines which symmetry breaking model (see below) should be used. Possible values are CV `A`, CV `B` and CV `NONE`.

CK **taf_routing_generation** either CV `fastest-paths`, CV `read-all` or CV `read-partial-fill`. Determines which routing should be used, see Section 3.7.1.

CK **taf_zone_only_zones** boolean, specifies whether only zones based on given prices are computed.

CK **taf_zone_only_prices** boolean, specifies whether only prices based on given zones are computed.

CK **taf_draw_zones** boolean, specifies whether a PTN with nodes allocated to their zones should be drawn. By default CV `false`.

CK **taf_draw_heatmap** boolean, specifies whether a heatmap should be drawn.

CK **taf_heatmap_mode** either CV `old`, CV `new` or CV `compare`. Specifies which prices or price differences should be shown in a heatmap.

CK **taf_heatmap_log_scale** boolean, specifies whether or not the heatmap should use a logarithmic scale. By default CV `false`.

CK **taf_heatmap_use_annotations** boolean, specifies whether or not the heatmap should be annotated. By default CV `false`.

CK **taf_evaluate_old_prices** points towards a first price matrix for the evaluation and for the heatmap. By default it is the reference price matrix CV `basis/Reference-Price-Matrix.giv`.

CK **taf_evaluate_new_prices** points towards a second price matrix for the evaluation and the heatmap. By default it is the tariff price matrix CV `tariff/Price-Matrix.taf`.

CK **taf_solver** determine the solver to be used. Note that currently only Gurobi is supported.

CK **taf_threads** determine the maximum number of threads to use for the solver (-1=use default value, i.e., no restriction). Note that this will only be used for a possible solver integration of the chosen model, not for the rest of the algorithm.

CK **taf_timelimit** the time limit for the solver in seconds (-1=use default value).

CK **taf_write_lp_file** whether to write the lp file of the model to solve.

CK **taf_mip_gap** sets the MIP optimization gap for the solver. The solver will terminate with an optimal solution if the gap between lower and upper objective bound is less than this value times the absolute value of the incumbent objective value.

7.13 Vehicle Scheduling

CK **vs_depot_index** the stop index of the depot. Set to -1 to disable to consideration of a depot.

CK **vs_eval_cost_factor_empty_trips_duration** the weight factor for the duration of empty trips in the cost function for a vehicle schedule

CK **vs_eval_cost_factor_empty_trips_length** the weight factor for the length of empty trips in the cost function for a vehicle schedule

CK **vs_eval_cost_factor_full_trips_duration** the weight factor for the duration of services in the cost function for a vehicle schedule

CK **vs_eval_cost_factor_full_trips_length** the weight factor for the length of services in the cost function for a vehicle schedule

CK **vs_maximum_buffer_time** the maximal buffer time between the service of two trips

CK **vs_mip_gap** the mip optimization gap for the solver, 0.1 equals a gap of 10% (-1=use default value).

CK **vs_model**] the vehicle scheduling model to use. For an overview of all models, see Section 3.8

CK **vs_solver** the solver to use for vehicle scheduling. Which solvers are implemented depends on the chose CK `vs_model`, see the corresponding documentation.

CK **vs_timelimit** the time limit to use for the solver in seconds (-1 = use default value).

CK **vs_threads** determine the maximal number of threads to use for the solver (-1=use default value, i.e., no restriction). Note that this will only be used for a possible solver integration of the chosen model, not for the rest of the algorithm.

CK **vs_turn_over_time** the minimal time between two services, given in time units.

CK **vs_vehicle_costs** the costs of a vehicle

CK **vs_write_lp_file** whether to write the lp file of the model to solve

7.14 Delay Management

CK **DM_best_of_all_write_objectives** whether to write all objectives to a file, when CK `DM_method` CV `best-of-all` is used

CK **DM_debug** enable debug output

CK **DM_earliest_time** the start of the rollout period

CK **DM_enable_consistency_checks** enable consistency checks for the input data, i.e., 28800 is 08:00.

CK **DM_eval_extended** enable the extended evaluation

CK **DM_latest_time** the end of the rollout period, given in seconds after midnight, i.e., 28800 is 08:00.

CK **DM_method** the delay management model to use. For an overview of all models, see Section 3.9.

CK **DM_mip_gap** the mip optimization gap for the solver, 0.1 equals a gap of 10% (-1=use default value).

CK **DM_opt_method_for_heuristic** the optimization method to use for the heuristics.

CK **DM_solver** the solver to use for vehicle scheduling. Which solvers are implemented depends on the chose CK DM_model, see the corresponding documentation.

CK **DM_threads** determine the maximal number of threads to use for the solver (-1=use default value, i.e., no restriction). Note that this will only be used for a possible solver integration of the chosen model, not for the rest of the algorithm.

CK **DM_timelimit** the time limit to use for the solver in seconds (-1 = use default value).

CK **DM_write_lp_file** whether to write the lp file of the model to solve

CK **DM_verbose** enable verbose output

7.15 Dataset Generation

CK dg_model The model that should be used to create the new dataset. For a detailed description of the available algorithms see Section 4.1.

The following parameters are only valid for the Parametrized-City-Model:

CK dg_param_city_number_subcenters Number of subcenters sourrounding the CBD. The PTN has $2n + 1$ nodes.

CK dg_param_city_alpha Trips proportion from periphery that go to the CBD.

CK dg_param_city_beta Trips proportion from periphery to own subcenter.

CK dg_param_city_eta Portion of displacement of the CBD from the center of the city in an axis CBD-subcenter.

CK dg_param_city_Y Total number of trips generated.

CK dg_param_city_L Distance from any subcenter to the geometrical center of the city.

CK dg_param_city_g Distance periphery-subcenter / Distance subcenter-CBD.

CK dg_param_city_a Trips proportion that depart from the periphery.

CK dg_ring_number_of_rings Number of concentric rings that are generated.

CK dg_ring_nodes_per_ring Number of nodes that each ring consists of

CK dg_ring_length_1 If this boolean parameter is set to true, the lengths of all edges are equal to 1.

CK dg_ring_radius specifies the radius of the inner ring, i.e. the lengths of the edges from the center to the nodes of the inner ring.

CK dg_ring_demand_type specifies the moethod for the creation of the OD data.

7.16 Integrated Models

7.16.1 General

CK **int_solver** the solver to use. Which solvers are implemented depends on the chosen model, see the corresponding documentation.

CK **int_threads** determine the maximal number of threads to use for the solver (-1=use default value, i.e., no restriction). Note that this will only be used for a possible solver integration of the chosen model, not for the rest of the algorithm.

7.16.2 LinTimPass

CK **lin_tim_pass_mip_gap** the mip optimization gap for the solver, 0.1 equals a gap of 10% (-1=use default value).

CK **lin_tim_pass_timelimit** the time limit to use for the solver in seconds (-1 = use default value).

CK **lin_tim_pass_write_lp_file** whether to write the lp file of the model to solve.

7.16.3 LinTimPassVeh

CK **lin_tim_pass_veh_mip_gap** the mip optimization gap for the solver, 0.1 equals a gap of 10% (-1=use default value).

CK **lin_tim_pass_veh_timelimit** the time limit to use for the solver in seconds (-1 = use default value).

CK **lin_tim_pass_veh_write_lp_file** whether to write the lp file of the model to solve.

7.16.4 TimPass

CK **tim_pass_mip_gap** the mip optimization gap for the solver, 0.1 equals a gap of 10% (-1=use default value).

CK **tim_pass_timelimit** the time limit to use for the solver in seconds (-1 = use default value).

CK **tim_pass_write_lp_file** whether to write the lp file of the model to solve.

7.16.5 TimVeh

CK **tim_veh_mip_gap** the mip optimization gap for the solver, 0.1 equals a gap of 10% (-1=use default value).

CK **tim_veh_timelimit** the time limit to use for the solver in seconds (-1 = use default value).

CK **tim_veh_write_lp_file** whether to write the lp file of the model to solve.

7.16.6 TimVehToLin

CK **tim_veh_to_lin_mip_gap** the mip optimization gap for the solver, 0.1 equals a gap of 10% (-1=use default value).

CK **tim_veh_to_lin_timelimit** the time limit to use for the solver in seconds (-1 = use default value).

CK **tim_veh_to_lin_write_lp_file** whether to write the lp file of the model to solve.

7.17 TimPassLib

CK **timpasslib_import_timetable** whether timetable is imported.

CK **timpasslib_export_timetable** whether timetable is exported.

Chapter 8

In- and Output Data

This section will describe all files and their contents that are in- or outputs of the LinTim algorithms.

8.1 Config

Config is the short form for *configuration* and an important tool in LinTim. We will now have a look at the general structure of the LinTim config files.
The LinTim config is contained in several CSV files that have the syntax

```
config_key; config_value
```

It organizes those values that are parameters to the calculation. Typical examples are the period length, the vehicle capacity (if there is only one), which algorithm to use for a specific computation step, e.g. for timetabling and filenames as well and could thus look like

```
period_length; 60
gen_passengers_per_vehicle; 100
tim_model; MATCH
```

Besides key-value pairs the configuration may also include other config files with either the CK `include` or CK `include_if_exists` statement. Former states that the file must exists or else an exception is thrown, in latter case, if the file does not exist, it will not be included. This inclusion is recursive, i.e. files included in already included files are included as well.
If a certain config key occurs twice, the latter value overwrites the former, e.g.

```
period_length; 60
period_length; 120
```

sets the CK `period_length` to 120. As a consequence, all values that belong to keys in an included file overwrite those defined before.
All keys demanded by programs are expected to exist, i.e., there are no in-program default values. Programs accessing config are expected to exit with an error message in case a key does not exist.
The meaning of the parameters is explained in the corresponding sections of this documentation.
Config has the following file hierarchy

Fi `/datasets/Global-Config.cnf` offers a default value for all config parameters that are not network specific, like CK `ptn_name` or CK `period_length`.

Fi `basis/Config.cnf` contains all the values specific to the dataset. Together with the global config this offers a value for all parameters. It includes the global config at the beginning, i.e., every parameter that was already defined in the global config will be overwritten. It roughly looks like

```
include; "../../Global-Config.cnf"
ptn_name; "DATASET"
...
include_if_exists; "State-Config.cnf"
include_if_exists; "Private-Config.cnf"
include_if_exists; "After-Config.cnf"
```

- CK `filename_state_config` (Fi `basis/State-Config.cnf`) is intended to allow programs to not only generate networks, but also to save and modify state information about them, e.g. whether the event activity network is modeled with `frequency_as_attribute` or `frequency_as_multiplicity`which is once set on construction and may be modified by a Periodic Rollout. The network specific state is not part of the version control system, although there are state defaults in the global config.
- Fi `basis/Private-Config.cnf` is used for user specific settings, e.g. for choosing a specific algorithm for solving or manipulating its parameters and is not part of the version control system. Note that if a value is defined in the config or state config as well as in the private config, the one given in the private config is used.
- Fi `basis/After-Config.cnf` can be used for automation and is intended to be *thrown away* upon usage, unlike all other configurations. A script that automatically evaluates a wide range of configurations thus may overwrite the after config in every step. Make sure that at the end of the script, the after config is deleted again or else it still influences manual runs as it overwrites all other configs.

8.2 Statistic

The statistic file CK `default_statistic_file` (Fi `statistic/statistic.sta`) contains the outcome of the evaluation routines described in 5. The content is formatted as follows

`statistic_key; statistik_value`

where the statistic key described what is evaluated and the statistic value gives the corresponding value. Statistic files are intended to be modified, i.e., new entries are added but old entries are not deleted, although the statistic file itself may be deleted any time. Make sure that the entries are up to date, e.g. R `make tim-timetable-evaluate` is run after calculating a new timetable and before evaluating the statistic.

8.3 Basis

Files in the folder Fo `basis` describe the structure of the Public Transportation Network, the demand and the line pool with its corresponding costs.

8.3.1 Additional load

The file CK `filename_additional_load_file` (Fi `basis/Additional-Load.giv`) contains additional load on single PTN links. When CK `load_generator_add_additional_load` is set to CV `true`, these loads will be added to the corresponding links during load generation. For an undirected network, a link may be given in both directions, allowing for different additional load values for the different directions. Unmentioned links will be assumed to have no additional load. The columns of the csv file correspond to:

`edge-id` id of the PTN edge

`left-stop-id` the id of the left stop, i.e., the origin of the edge

`right-stop-id` the id of the right stop, i.e., the destination of the edge

`additional-load` the value of the additional load

8.3.2 Change station

The file CK `filename_change_station_file` (Fi `basis/Change-Stations.giv`) contains a list of change stations, i.e., a list of stops where passengers can transfer. The columns of the csv file correspond to:

`stop-id` id of the stop

8.3.3 Demand

The file CK `default_demand_file` (Fi `basis/Demand.giv`) contains the demand at specified locations. The columns of the csv file correspond to:

`demand-id` id of the demand point

`short-name` short name of the demand point

`long-name` log name of the demand point

`x-coordinate` x-coordinate of the demand point

`y-coordinate` y-coordinate of the demand point

`demand` demand at the demand point

Note: the distance between two demand points can be transformed to kilometers by multiplying with CK `gen_conversion_coordinates`.

8.3.4 Demand geo

The file CK `default_demand_coordinates_file` (Fi `basis/Demand.giv.geo`) gives the geographical coordinates (latitude and longitude) of the demand points. The columns of the csv file correspond to:

`demand-id` id of the demand point

`latitude` latitude of the demand point

`longitude` longitude of the demand point

8.3.5 Edge

The file CK `default_edges_file` (Fi `basis/Edge.giv`) contains information about the edges in the PTN. The columns of the csv file correspond to:

`edge-id` id of the edge

`left-stop-id` id of the left stop (source node in directed case)

`right-stop-id` id of the right stop (target node in directed case)

`length` length of the edge

`lower-bound` minimum time to traverse the edge in minutes

`upper-bound` maximum time to traverse the edge in minutes

Note: whether the edges are directed or undirected in defined by CK `ptn_is_undirected`.
Note: the length of an edge can be transformed to kilometers by multiplying with CK `gen_conversion_length`.

8.3.6 Edge forbidden

The file CK filename_forbidden_links_file (Fi basis/Edge-forbidden.giv) contains information about the edges in the PTN that are forbidden, i.e., that may not be used by the public transport mode that is being planned. These edges should be a subset of the edges in CK default_edges_file (Fi basis/Edge.giv). The columns of the csv file correspond to:

edge-id id of the edge

left-stop-id id of the left stop (source node in directed case)

right-stop-id id of the right stop (target node in directed case)

length length of the edge

lower-bound minimum time to traverse the edge in minutes

upper-bound maximum time to traverse the edge in minutes

Note: whether the edges are directed or undirected in defined by CK ptn_is_undirected.
Note: the length of an edge can be transformed to kilometers by multiplying with CK gen_conversion_length.

8.3.7 Edge infrastructure

The file CK filename_infrastructure_edge_file (Fi basis/Edge-Infrastructure.giv) contains information about the infrastructure edges, i.e., edges that connect infrastructure nodes. The columns of the csv file correspond to:

edge-id id of the edge

left-node-id id of the left stop (source node in directed case)

right-node-id id of the right stop (target node in directed case)

length length of the edge

lower-bound minimum time to traverse the edge in minutes

upper-bound maximum time to traverse the edge in minutes

Note: whether the edges are directed or undirected in defined by CK ptn_is_undirected.
Note: the length of an edge can be transformed to kilometers by multiplying with CK gen_conversion_length.

8.3.8 Edge infrastructure forbidden

The file CK filename_forbidden_infrastructure_edges_file (Fi basis/Edge-Infrastructure-forbidden.giv) contains information about the infrastructure edges that are forbidden, i.e., that may not be used by the public transport mode that is being planned. These edges should be a subset of the edges in CK filename_infrastructure_edge_file (Fi basis/Edge-Infrastructure.giv). The columns of the csv file correspond to:

edge-id id of the edge

left-node-id id of the left node (source node in directed case)

right-node-id id of the right node (target node in directed case)

length length of the edge

lower-bound minimum time to traverse the edge in minutes

upper-bound maximum time to traverse the edge in minutes

Note: whether the edges are directed or undirected in defined by CK ptn_is_undirected.
Note: the length of an edge can be transformed to kilometers by multiplying with CK gen_conversion_length.

8.3.9 Edge walking

The file CK filename_walking_edge_file (Fi basis/Edge-Walking.giv) contains information about the possible walking edges, i.e., connections between infrastructure nodes that can directly be used for walking by the passengers. The columns of the csv file correspond to:

edge-id id of the edge

left-node-id id of the left node (source node in directed case)

right-node-id id of the right node (target node in directed case)

length length of the edge, given in seconds

Note: whether the edges are directed or undirected in defined by CK sl_walking_is_directed.
Note: when read by LinTim, CK sl_max_walking_time will be respected, i.e., only edges with a length smaller than the given value will be read. A value of CV -1 will disable this and allow all edges will be read.
Note: it is possible to preprocess the walking edges by using

R make ptn-preprocess-walking.

With this, walking edges will be filtered by CK sl_max_walking_amount, CK sl_max_walking_ratio (both per node with outgoing demand) and CK sl_max_walking_time, possibly reducing the size of the walking graph.

8.3.10 Existing stop

The file CK default_existing_stop_file (Fi basis/Existing-Stop.giv) contains information about already existing stops in the PTN. The columns of the csv file correspond to:

stop-id id of the stop

short-name short name of the stop

long-name log name of the stop

x-coordinate x-coordinate of the stop

y-coordinate y-coordinate of the stop

Note: the distance between two stops can be transformed to kilometers by multiplying with CK gen_conversion_coordinates.

8.3.11 Existing stop geo

The file CK default_existing_stop_coordinates_file (Fi basis/Existing-Stop.giv.geo) gives the geographical coordinates (latitude and longitude) of the already existing stops. The columns of the csv file correspond to:

stop-id id of the stop

latitude latitude of the stop

longitude longitude of the stop

8.3.12 Existing edge

The file CK `default_existing_edge_file` (Fi `basis/Existing-Edge.giv`) contains information about already existing edges in the PTN. The columns of the csv file correspond to:

edge-id id of the edge

left-stop-id id of the left stop (source node in directed case)

right-stop-id id of the right stop (target node in directed case)

length length of the edge

lower-bound minimum time to traverse the edge in minutes

upper-bound maximum time to traverse the edge in minutes

Note: whether the edges are directed or undirected in defined by CK `ptn_is_undirected`.
Note: the length of an edge can be transformed to kilometers by multiplying with CK `gen_conversion_length`.

8.3.13 Headway

The file CK `default_headways_file` (Fi `basis/Headway.giv`) contains information about the headway needed for the edges in the PTN. The columns of the csv file correspond to:

edge-id id of the edge

headway headway on the edge, i.e., the minimum time between two consecutive vehicles on this edge in minutes

8.3.14 Load

The file CK `default_loads_file` (Fi `basis/Load.giv`) contains information about the load and frequency constraints of the edges in the PTN. The columns of the csv file correspond to:

edge-id id of the edge

load load on the edge

lower-frequency minimal frequency all lines in the line concept have to add up to the edge

upper-frequency maximal frequency all lines in the line concept are allowed to add up to for the edge

8.3.15 Node

The file CK `filename_node_file` (Fi `basis/Node.giv`) contains information about infrastructure nodes. Infrastructure nodes are the smalles unit of nodes in LinTim, they may e.g. represent crossings or (potential) stops. The columns of the csv file correspond to:

node-id the id of the node

name the name of the nod

x-coordinate the x coordinate of the node

y-coordinate the y coordinate of the node

stop-possible? whether its possible for this node to be a stop

Note: x- and y-coordinate are assumed to be planar coordinates, i.e., will be directly used the compute the euclidean distance between stops. The distance between two stops can be transformed to kilometers by multiplying with CK `gen_conversion_coordinates`.

8.3.16 OD

The file CK default_od_file (Fi basis/OD.giv) contains information about the passenger demand between all pairs of stops in the PTN. The columns of the csv file correspond to:

left-stop-id id of the stop the passengers start at

right-stop-id id of the stop the passengers travel to

customers number of passengers traveling

8.3.17 OD node

The file CK filename_od_nodes_file (Fi basis/OD-Node.giv) contains information about the passenger demand between pairs of nodes in the infrastructure network. The columns of the csv file correspond to:

left-node-id id of the node the passengers start at

right-node-id id of the node the passengers travel to

customers number of passengers traveling

8.3.18 Pool

The file CK default_pool_file (Fi basis/Pool.giv) contains information about the line pool. The columns of the csv file correspond to:

line-id id of the line

edge-order where the edge is in the line

edge-id id of the edge

8.3.19 Pool cost

The file CK default_pool_cost_file (Fi basis/Pool-Cost.giv) contains information about the cost and length of lines in the line pool. The columns of the csv file correspond to:

line-id id of the line

length length of the line

cost cost of the line

Note: the length of a line can be transformed to kilometers by multiplying with CK gen_conversion_length.

8.3.20 Reference Price Matrix

The file CK filename_tariff_reference_price_matrix_file (Fi basis/Reference-Price-Matrix.giv) contains the given reference prices for each OD pair. The columns of the csv file correspond to:

origin-id id of the origin of the OD pair,

destination-id id of the destination of the OD pair,

price reference price when travelling from the origin to the destination.

8.3.21 Restricted turns

The file CK `filename_turn_restrictions` (Fi `basis/Restricted-Turns.giv`) contains information about restricted turns, i.e., pairs of link ids of the PTN that are not allowed to be traversed by a line directly after each other. The columns of the csv file correspond to:

`first-edge-id` the first edge id

`second-edge-id` the second edge id

Note: whether the information will be interpreted as directed is dependent on CK `ptn_is_undirected`.

8.3.22 Restricted turns infrastructure

The file CK `filename_turn_restrictions_infrastructure` (Fi `basis/Restricted-Turns-Infrastructure.giv`) contains information about restricted turns, i.e., pairs of edge ids in the infrastructure network that are not allowed to be traversed by a line directly after each other. The columns of the csv file correspond to:

`first-edge-id` the first edge id

`second-edge-id` the second edge id

Note: whether the information will be interpreted as directed is dependent on CK `ptn_is_undirected`.

8.3.23 Ridepool

The file CK `filename_rpool_file` (Fi `basis/Ridepool.giv`) contains a ridepool, i.e. areas in the PTN. The columns of the csv file correspond to:

`area-id` id of the area

`edge-id` id of an edge contained in the area

`demand-factor` edge-specific constants specifying the fraction of time where one vehicle in the area is used on this edge.

8.3.24 Routings

The files CK `filename_routing_ptn_input` (Fi `basis/Routing-ptn.giv`) contains a routing in the PTN, i.e. for each node pair at most one path is specified as a list of nodes. The columns of the csv file correspond to:

`origin-id` id of the first node of the path,

`destination-id` id of the last node of the path,

`node-ids` path specified by the sequence of the stop-ids.

The parameter CK `filename_routing_ptn_output` (Fi `basis/Routing-ptn.giv`) specifies the file, where the computed routing is written to.

8.3.25 Station limits

The file CK `filename_station_limit_file` (Fi `basis/Station-Limits.giv`) contains information about individual station limits on wait or change times. The columns of the csv file correspond to:

`stop-id` the id of the stop

`min-wait-time` the minimal waiting time.

`max-wait-time` the maximal waiting time.

`min-change-time` the minimal change time.

`max-change-time` the maximal change time.

Note: every individual limit may be set to -1 if there is none. Then the corresponding default parameters will be used. The same holds for stops not present in this file.

8.3.26 Stop

The file CK `default_stops_file` (Fi `basis/Stop.giv`) contains information about the stops in the PTN. The columns of the csv file correspond to:

`stop-id` id of the stop

`short-name` short name of the stop

`long-name` log name of the stop

`x-coordinate` x-coordinate of the stop

`y-coordinate` y-coordinate of the stop

Note: x- and y-coordinate are assumed to be planar coordinates, i.e., will be directly used the compute the euclidean distance between stops. The distance between two stops can be transformed to kilometers by multiplying with CK `gen_conversion_coordinates`.

8.3.27 Stop geo

The file CK `default_stops_coordinates_file` (Fi `basis/Stop.giv.geo`) gives the geographical coordinates (latitude and longitude) of the stops. The columns of the csv file correspond to:

`stop-id` id of the stop

`latitude` latitude of the stop

`longitude` longitude of the stop

8.3.28 Terminals

The file CK `filename_terminals_file` (Fi `basis/Terminals.giv`) gives the stop ids of terminals, i.e., stops where lines are allowed to terminate. The columns of the csv file correspond to:

`stop-id` id of the stop

Note: the stop ids should be a subset of the ptn stops, i.e., of CK `default_stops_file` (Fi `basis/Stop.giv`).

8.4 Line Planning

The folder Fo `line-planning` contains information about the line concept.

8.4.1 Line concept

The file CK `default_lines_file` (Fi `line-planning/Line-Concept.lin`) contains information about the line concept. The columns of the csv file correspond to:

line-id id of the line

edge-order where the edge is in the line

edge-id id of the edge

frequency frequency of the line. If this is zero, the line is not used in the line concept.

8.4.2 Fixed lines

The file CK `filename_lc_fixed_lines` (Fi `line-planning/Fixed-Lines.lin`) contains information about the fixed lines that should be in the line concept. It can not be read/respected by all line planning methods, so see Section 3.3 for more information. The columns of the csv file correspond to:

line-id id of the line

edge-order where the edge is in the line

edge-id id of the edge

frequency frequency of the line. If this is zero, the line is not used in the line concept.

8.4.3 Line capacities

The file CK `filename_lc_fixed_line_capacities` (Fi `line-planning/Line-Capacities.lin`) contains information about the capacities of the fixed lines that should be in the line concept. It can not be read/respected by all line planning methods, so see Section 3.3 for more information. The columns of the csv file correspond to:

line-id id of the line

capacity the capacity of the line

8.4.4 Rideconcept

The file CK `filename_rc_file` (Fi `line-planning/Ride-Concept.lin`) contains a ride concept, i.e. a set of ridepooling areas with assigned number of vehicles for each area. The columns of the csv file correspind to:

area-id id of the area

edge-id id of an edge contained in the area

demand-factor edge-specific constants specifying the fraction of time where one vehicle in the area is used on this edge.

number-of-vehicles Number of vehicles operation in the area

8.5 Timetabling

The folder Fo `timetabling` contains information about the periodic event-activity-network and the timetable.

8.5.1 Activities periodic

The file CK `default_activities_periodic_file` (Fi `timetabling/Activities-periodic.giv`) contains information about activities in the periodic EAN. The columns of the csv file correspond to:

`activity-id` id of the activity

`type` type of the activity, can be `drive` for drive activities, `wait` for wait activities, `change` for transfers of passengers, `sync` for synchronization activities between different servings of a line with frequency greater than one or `turnaround` for turnaround activities, i.e., activities of vehicles serving one line after another

`tail-event-id` id of source event, i.e., the start of the activity

`head-event-id` id of target event, i.e., the end of the activity

`lower-bound` the minimal time for this activity, i.e., the minimal time duration needed between the corresponding source and target event to be feasible

`upper-bound` the maximal time for this activity, i.e., the maximal time duration allowed between the corresponding source and target event to be feasible

`passengers` the number of passengers using this activity

8.5.2 Events periodic

The file CK `default_events_periodic_file` (Fi `timetabling/Events-periodic.giv`) contains information about events in the periodic EAN. The columns of the csv file correspond to:

`event-id` id of the event

`type` type of the event, can be `departure` for events which are departures of a line at a stop or `arrival` for events which are arrivals of a line at a stop

`stop-id` id of the corresponding stop

`line-id` id of the corresponding line

`passengers` number of passengers boarding/alighting at the event

`line-direction` direction of the line, > for forward direction (i.e., the direction given in the file Fi `Pool.giv`) or < for the backward direction

`line-freq-repetition` repetition of the line, i.e., how often the line has already been used in the planning period

8.5.3 Fixed times

The file CK `filename_tim_fixed_times` (Fi `timetabling/Fixed-timetable-periodic.tim`) gives restrictions on the allowed times for single events. Not all events need to be included in this file, only the ones with additional restrictions.

`event-id` the periodic event id

`lower-bound` the lower time bound on the event

`upper-bound` the upper time bound on the event

8.5.4 Initial duration assumptions

The file CK filename_initial_duration_assumption (Fi timetabling/Initial-duration-assumption-periodic.giv) may contain a duration for each activity used in the initial passenger distribution of the ean creation. The columns of the csv file correspond to:

activity-id id of the activity

duration the duration to use for the passenger distribution

Note that CK ean_initial_duration_assumption_model needs to be set to CV SEMI_AUTOMATIC for this file to be read. Not all activities need to be present in the file, the duration of activities not present will be computed normally.

8.5.5 Timetable periodic

The file CK default_timetable_periodic_file (Fi timetabling/Timetable-periodic.tim) contains a time for each event in the periodic EAN. The columns of the csv file correspond to:

event-id id of the event

time the periodic time of the event

8.5.6 Timetable for VISUM

The file CK default_timetable_visum_file (Fi timetabling/Timetable-visum-nodes.tim) is an intermediate format for reading a LinTim timetable into VISUM. For more information, see 4.12. The columns of the csv file correspond to:

line-id the line id

line-code the line code, i.e., a short name

direction the direction of the line

stop-order where the stop is in the line

stop-id the id of the stop

frequency the frequency of the line

departure_time the departure time at this stop

arrival_time the arrival time at this stop

line-freq-repetition the repetition of the line

8.6 Tariff Planning

8.6.1 Price Matrix

The file CK filename_tariff_price_matrix_file (Fi tariff/Price-Matrix.taf) contains the (newly calculated) prices with respect to the specified model (CK taf_model) for all OD pairs. The columns of the csv file correspond to:

origin-id id of the origin of the OD pair,

destination-id id of the destination of the OD pair,

price price when travelling from the origin to the destination.

8.6.2 Zones

The file CK `filename_tariff_zone_file` (Fi `tariff/Zones.taf`) contains the assignment of stops to their zones within the zone model CK `taf_model` CV `zone`. The columns of the csv file correspond to:

`zone-id` id of a zone,

`stop-id` id of the stop belonging to that zone.

8.6.3 Zone Prices

The file CK `filename_tariff_zone_price_file` (Fi `tariff/Zone-Prices.taf`) contains the prices in zone model CK `taf_model` CV `zone` for traversing a certain number of zones. The price for traversing more zones than the maximum number of zones specified is just the price for traversing the maximum number of zones specified. The columns of the csv file correspond to:

`n-traversed-zones` number of traversed zones,

`price` price for traversing a given number of zones, i.e the price list.

8.7 Vehicle Scheduling

The folder Fo `vehicle-scheduling` contains information about the vehicle tours in the dataset.

8.7.1 Vehicle schedules

The file CK `default_vehicle_schedule_file` (Fi `vehicle-scheduling/Vehicle_Schedules.vs`) contains information regarding the scheduling of the vehicles. The columns of the csv file correspond to:

`circulation-ID` Id of the corresponding circulation

`vehicle-ID` Id of the vehicle

`trip-number of this vehicle` the trip number of the vehicle

`type` the type of the tour, can be `trip` for a line serving or `empty` for an empty trip

`aperiodic-start-ID` the aperiodic event id of the start event of this serving of the line

`periodic-start-ID` the periodic event id of the start event of this serving of the line

`start-stop-id` the stop id of the start of the line

`start-time` the starting time of this service of the line

`aperiodic-end-ID` the aperiodic event id of the end event of this serving of the line

`periodic-end-ID` the periodic event id of the end event of this serving of the line

`end-stop-id` the stop id of the end of the line

`end-time` the ending time of this service of the line

`line` the line id

8.8 Delay Management

The folder Fo `delay-management` contains information about the aperiodic event-activity-network, timetable and delays with a disposition timetable

8.8.1 Events expanded

The file CK default_events_expanded_file (Fi delay-management/Events-expanded.giv) contains information about events in the aperiodic EAN. The columns of the csv file correspond to:

event-id id of the event

periodic-id the corresponding periodic id

type type of the event, can be departure for events which are departures of a line at a stop or arrival for events which are arrivals of a line at a stop

time the time of the event

passengers number of passengers boarding/alighting at the event

stop-id id of the corresponding stop

8.8.2 Activities expanded

The file CK default_activities_expanded_file (Fi delay-management/Activities-expanded.giv) contains information about activities in the aperiodic EAN. The columns of the csv file correspond to:

activity-id id of the activity

periodic-id the corresponding periodic id

type type of the activity, can be drive for drive activities, wait for wait activities, change for transfers of passengers, sync for synchronization activities between different servings of a line with frequency greater than one or turnaround for turnaround activities, i.e., activities of vehicles serving one line after another

tail-event-id id of source event, i.e., the start of the activity

head-event-id id of target event, i.e., the end of the activity

lower-bound the minimal time for this activity, i.e., the minimal time duration needed between the corresponding source and target event to be feasible

upper-bound the maximal time for this activity, i.e., the maximal time duration allowed between the corresponding source and target event to be feasible

passengers the number of passengers using this activity

8.8.3 Timetable expanded

The file CK default_timetable_expanded_file (Fi delay-management/Timetable-expanded.tim) contains information about the aperiodic timetable, i.e., the time for each aperiodic event. The columns of the csv file correspond to:

event-id id of the event

time the time of the event

8.8.4 Timetable disposition

The file CK `default_disposition_timetable_file` (Fi `delay-management/Timetable-disposition.tim`) contains information about the disposition timetable, i.e., the time for each aperiodic event in the given delay scenario. The columns of the csv file correspond to:

`event-id` id of the event

`time` the time of the event

8.8.5 Delays events

The file CK `default_event_delays_file` (Fi `delay-management/Delays-Events.giv`) contains information about the delay induced at the events. The columns of the csv file correspond to:

`ID` the id of the delayed event

`delay` the delay, given in seconds

8.8.6 Delays activities

The file CK `default_activity_delays_file` (Fi `delay-management/Delays-Activities.giv`) contains information about the delay induced at the activities. The columns of the csv file correspond to:

`ID` the id of the delayed activity

`delay` the delay, given in seconds

8.8.7 Trips

The file CK `default_trips_file` (Fi `delay-management/Trips.giv`) contains information regarding the vehicle trips. A vehicle trips is the serving of a line by a vehicle, i.e., this file contains all line servings in the aperiodic EAN. The columns of the csv file correspond to:

`aperiodic-start-ID` the aperiodic event id of the start event of this serving of the line

`periodic-start-ID` the periodic event id of the start event of this serving of the line

`start-stop-id` the stop id of the start of the line

`start-time` the starting time of this service of the line

`aperiodic-end-ID` the aperiodic event id of the end event of this serving of the line

`periodic-end-ID` the periodic event id of the end event of this serving of the line

`end-stop-id` the stop id of the end of the line

`end-time` the ending time of this service of the line

`line` the line id

8.9 GTFS

Using

R make gtfs

will create all required gtfs files. For this, the stops (CK default_stops_file (Fi basis/Stop.giv)), the line concept (CK default_lines_file (Fi line-planning/Line-Concept.lin)), the aperiodic ean (CK default_events_expanded_file (Fi delay-management/Events-expanded.giv), CK default_activities_expanded_file (Fi delay-management/Activities-expanded.giv)) and the trips (CK default_trips_file (Fi delay-management/Trips.giv)) will be read and the corresponding raw gtfs files will be written to CK gtfs_output_path (Fi gtfs), i.e. the files

- Fi agency.txt,
- Fi stops.txt,
- Fi routes.txt,
- Fi trips.txt,
- Fi stop_times.txt and
- Fi calendar.txt.

Additionally, a zipped file containing the raw data will be created in CK gtfs_output_path (Fi gtfs), named after CK ptn_name.

Chapter 9

Datasets

LinTim provides many datasets to test and evaluate public transport planning algorithms. The following chapter should give an overview over the available datasets and the compatibility with the different planning steps.

9.1 Configuration Parameters for Datasets

There are some configuration parameters used per dataset and not per algorithm. These are set in the file Fi `basis/Config.cnf`.

- CK `gen_conversion_length`: conversion factor used to convert the edge length given in CK `default_edges_file` (Fi `basis/Edge.giv`) to kilometers.
- CK `gen_conversion_coordinates`: conversion factor used to convert the distance between two stations given in CK `default_stops_file` (Fi `basis/Stop.giv`) by the coordinates to kilometers.
- CK `gen_vehicle_speed`: speed of the vehicles in km/h.
- CK `ptn_name`: the name of the network
- CK `ptn_stop_waiting_time`: the time each vehicle has to stop at each stop in average. Used in shortest path computation during OD creation.
- CK `period_length`: the length of a period in time units
- CK `time_units_per_minutes`: the number of time units per minute
- CK `ean_default_minimal_waiting_time`: the lower bound for wait activities in the ean. Used during the creation of the ean.
- CK `ean_default_maximal_waiting_time`: the upper bound for wait activities in the ean. Used during the creation of the ean.
- CK `ean_default_minimal_change_time`: the lower bound for change activities in the ean. Used during the creation of the ean.
- CK `ean_default_maximal_change_time`: the upper bound for change activities in the ean. Used during the creation of the ean.
- CK `ean_change_penalty`: the penalty for using a change activity in the ean. Used for routing passengers in the ean and evaluating the perceived travel time.
- CK `gen_passengers_per_vehicle`: the maximal number of passengers per vehicle. Used in computing lower frequency bounds in preparation of line planning.

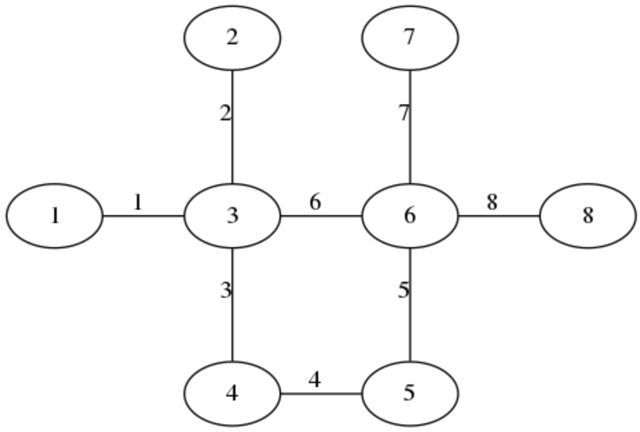


Figure 9.1: The PTN of the toy dataset

9.2 Artificial Datasets

There are two purely artificial datasets in LinTim. These are small examples to test and understand new algorithms.

9.2.1 Toy

The toy dataset is purely designed for testing purposes. It contains 8 nodes, 8 edges and 22 OD pairs, consisting of 2622 passengers in total. An overview of the structure is given in Fig. 9.1.
Since the dataset does not contain the necessary information, stop location is not supported on this dataset out of the box.

9.2.2 Grid

The grid dataset is designed to be overseeable, yet complex enough to contain complex effects. Therefore, the dataset contains a simple PTN structure but a reasonable demand structure designed by transportation planners, see [7]. It is part of the benchmark datasets found at [6].
The dataset contains 25 nodes, 40 edges and 567 OD pairs, consisting of 2546 passengers in total. An overview of the structure is given in Fig. 9.2. Since the dataset does not contain the necessary information, stop location is not supported on this dataset out of the box.

9.2.3 Ring

The ring dataset is a little bit larger than the grid dataset but still maintains a clear structure. It is part of the benchmark datasets found at [6].
The dataset contains 161 nodes, 320 edges and 25760 OD pairs, consisting of 2766.12 passengers in total. An overview of the structure is given in Fig. 9.3. Since the dataset does not contain the necessary information, stop location is not supported on this dataset out of the box.

9.3 Datasets based on real world data

9.3.1 Sioux Falls

The sioux falls dataset is a dataset often used in practical public transport planning. It was first introduced in [19] and is available at [42]. It is a representation of the city of Sioux Falls, South Dakote, USA. It is part of the benchmark datasets found at [6].

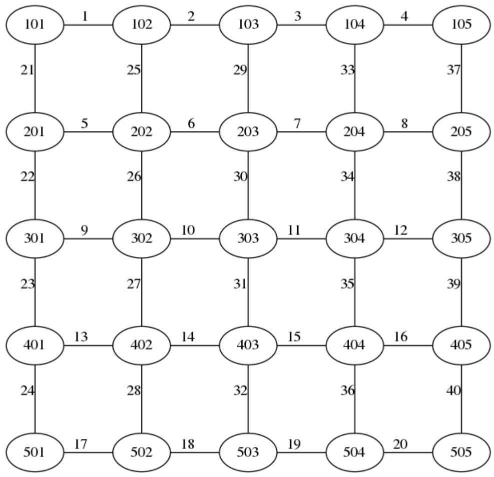


Figure 9.2: The PTN of the grid dataset

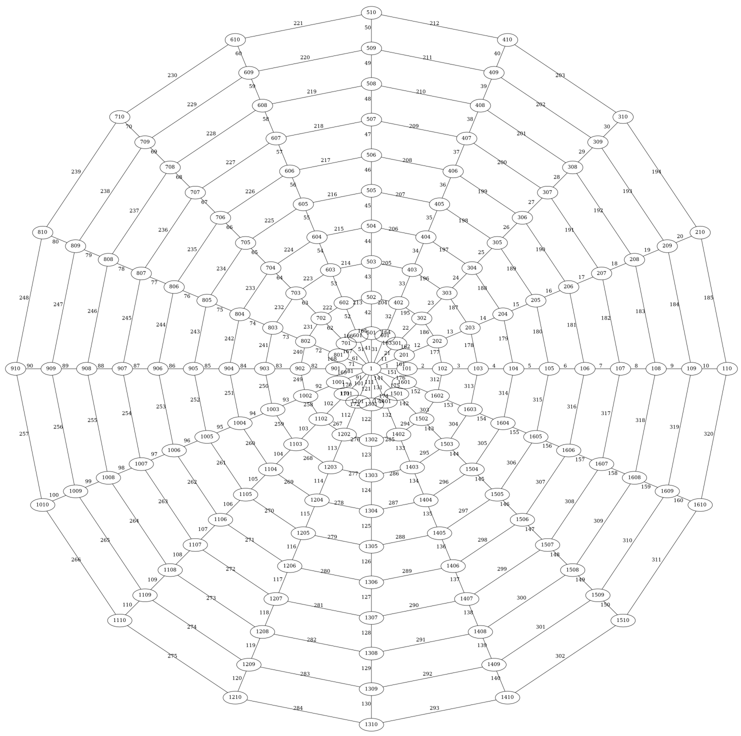


Figure 9.3: The PTN of the ring dataset

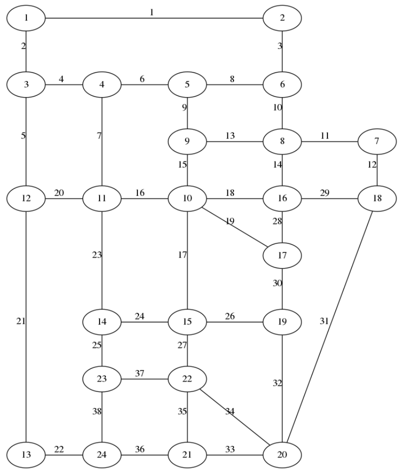


Figure 9.4: Infrastructure of the sioux falls dataset

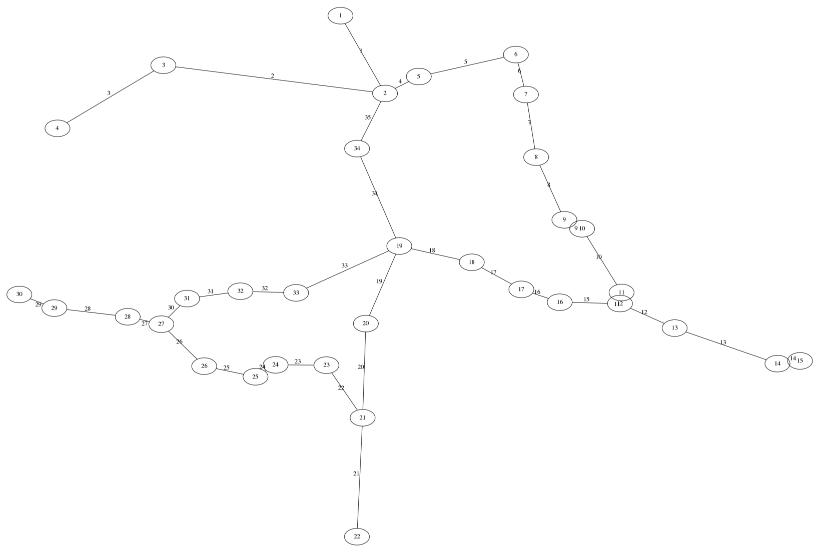


Figure 9.5: Existing infrastructure of the lower saxony dataset

The dataset contains 24 stops, 38 edges and 4114.57 passengers in 552 od pairs. An overview of the structure of the dataset is given in Fig. 9.4.

9.3.2 Lowersaxony

The lower saxony dataset was included to test the effects of stop location and line pool generation. It contains the regional railway data of lower saxony, a region in northern Germany.
The dataset contains 34 existing stops, 35 existing edges and 31 demand points. An overview of the structure given by the existing stops and edges is given in Fig. 9.5. To work with this dataset, you need to start with the stop location step.

9.3.3 Goevb

The goevb dataset represents the bus network in Göttingen, a city in the middle of Germany and home of the LinTim project. It was included as part of a student project in 2011.

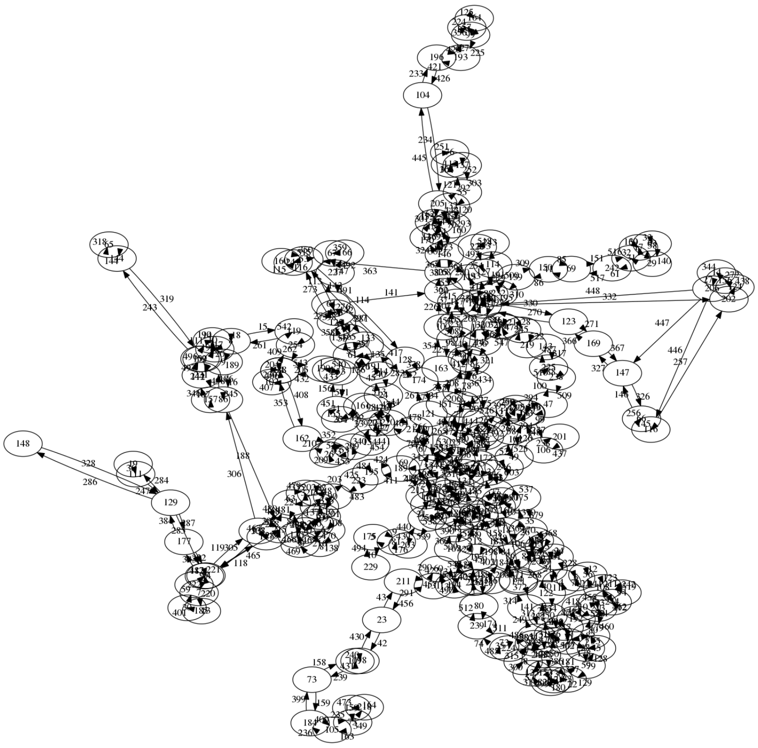


Figure 9.6: The PTN of the goevb dataset

The dataset contains 257 stops, 548 edges and 58226 OD pairs, consisting of 406146 passengers in total. An overview of the structure is given in Fig. 9.6. Since the dataset does not contain the necessary information, stop location is not supported on this dataset out of the box.
Note, that goevb is a directed network!

9.3.4 Athens

The athens dataset represents the metro system in Athens.
The dataset contains 51 stops, 52 edges and 2385 OD pairs, consisting of 63323 passengers in total. An overview of the structure is given in Fig. 9.7. Since the dataset does not contain the necessary information, stop location is not supported on this dataset out of the box.

9.3.5 Bahn-01

Currently not included in the release version of LinTim.
The bahn-01 dataset represents parts of the German railway network, including the long distance network. For larger datasets, see Sec. 9.3.6-9.3.8.
The dataset contains 250 stops, 326 edges and 48842 OD pairs, consisting of 3147382 passengers in total. An overview of the structure is given in Fig. 9.8. Since the dataset does not contain the necessary information, stop location is not supported on this dataset out of the box.

9.3.6 Bahn-02

Currently not included in the release version of LinTim.
The bahn-02 dataset represents parts of the German railway network, including the long distance network. For a smaller dataset see Sec. 9.3.5, for larger datasets, see Sec. 9.3.7 and 9.3.8.

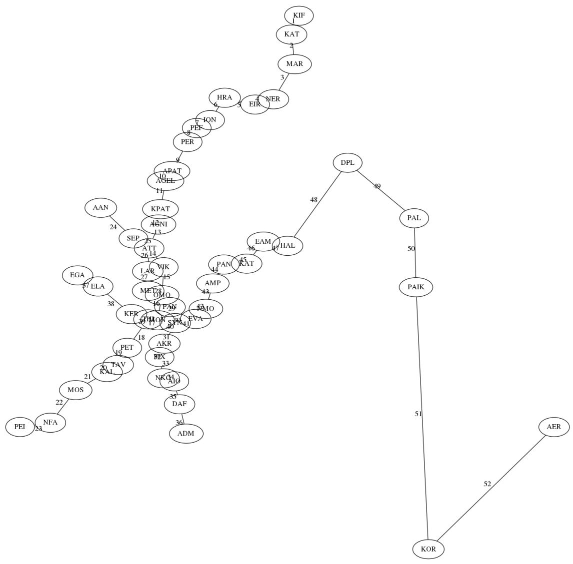


Figure 9.7: The PTN of the athens dataset

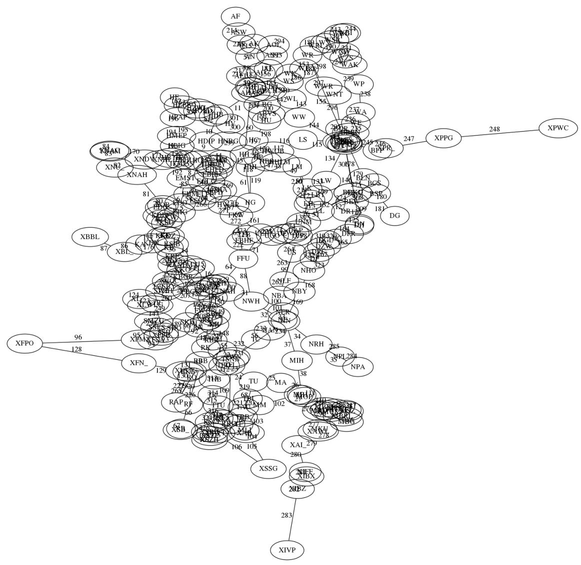


Figure 9.8: The PTN of the bahn-01 dataset

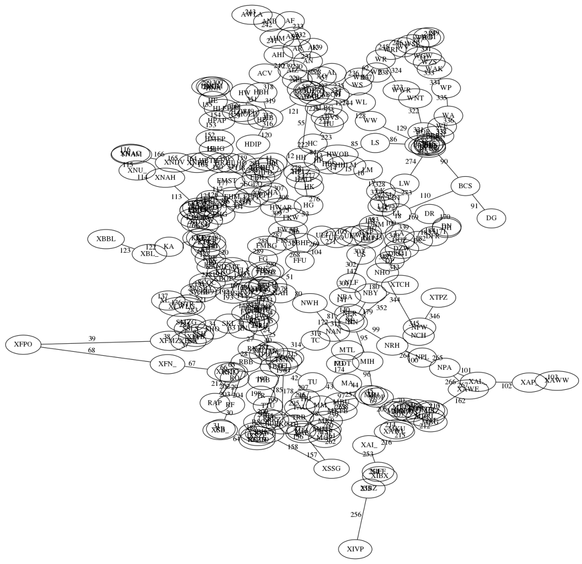


Figure 9.9: The PTN of the bahn-02 dataset

The dataset contains 280 stops, 354 edges and 61110 OD pairs, consisting of 3666720 passengers in total. An overview of the structure is given in Fig. 9.9. Since the dataset does not contain the necessary information, stop location is not supported on this dataset out of the box.

9.3.7 Bahn-03

Currently not included in the release version of LinTim.
The bahn-03 dataset represents parts of the German railway network, including the long distance network. For smaller datasets see Sec. 9.3.5 and 9.3.6, for a larger dataset, see Sec. 9.3.8.
The dataset contains 296 stops, 393 edges and 68284 OD pairs, consisting of 3878392 passengers in total. An overview of the structure is given in Fig. 9.10. Since the dataset does not contain the necessary information, stop location is not supported on this dataset out of the box.

9.3.8 Bahn-04

Currently not included in the release version of LinTim.
The bahn-04 dataset represents parts of the German railway network, including the regional network. For smaller datasets, see Sec. 9.3.5-9.3.7.
The dataset contains 319 stops, 452 edges and 77878 OD pairs, consisting of 4183088 passengers in total. An overview of the structure is given in Fig. 9.11. Since the dataset does not contain the necessary information, stop location is not supported on this dataset out of the box.

9.3.9 Bahn-equal-frequencies

Currently not included in the release version of LinTim.
The bahn-equal-frequencies dataset is based on bahn-01(9.3.5). It is designed, such that running the line planning step with default parameters will result in a line concept with binary frequencies. This is therefore helpful to test algorithms that do not work for frequencies > 1.
The dataset contains 250 stops, 326 edges and 6106 OD pairs, consisting of 385868 passengers in total. An overview of the structure is given in Fig. 9.12. Since the dataset does not contain the necessary information, stop location is not supported on this dataset out of the box.

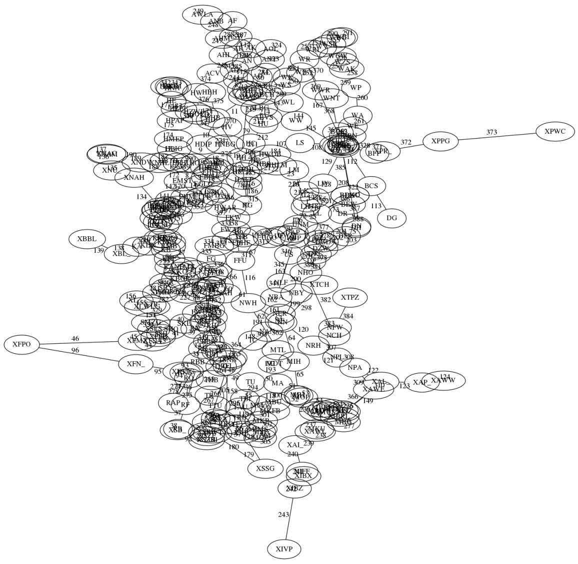


Figure 9.10: The PTN of the bahn-03 dataset

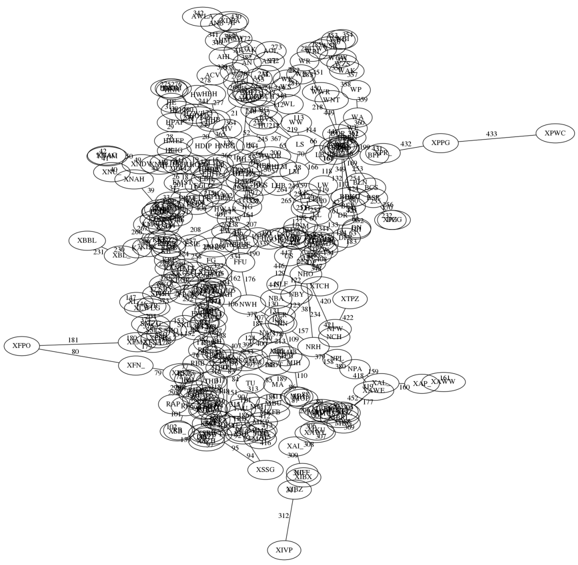


Figure 9.11: The PTN of the bahn-04 dataset

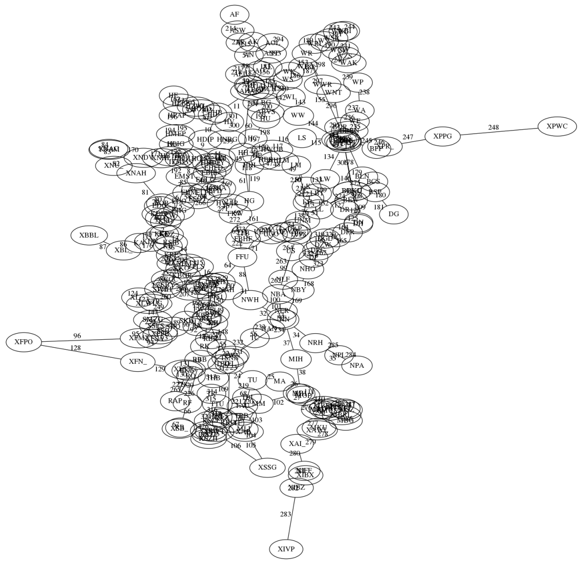


Figure 9.12: The PTN of the bahn-equal-frequencies dataset

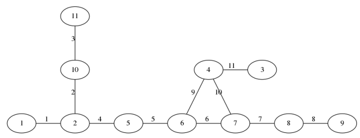


Figure 9.13: The PTN of the BOMHarbour dataset

9.3.10 BOMHarbour

BOMHarbour is based on the metro network in Mumbai, India. Since the metro is quite new, the dataset only consists of a few stations. The main focus investigated in BOMHarbour is to find a feasible timetable for the given line concept.
The dataset contains 11 stops, 11 edges and no passenger information. An overview of the structure is given in Fig. 9.13. Since the dataset does not contain the necessary information, stop location is not supported on this dataset out of the box.

9.3.11 Mandl

Mandl is based on a case study in Switzerland provided by Christoph Mandl, see [20]. The dataset contains 15 stops, 21 edges and 172 OD pairs, consisting of 15570 passengers in total. Since the travel time for each edge was given in minutes, the distances between stations are approximated, considering an average speed of 40 km/h. The OD matrix holds the demand over one day. To obtain reasonable line concepts for a period of one hour, the capacity of the vehicles has to be set to the actual capacity times the number of service hours per day (the default setting in the dataset is 450 corresponding to an actual capacity of 30 and 15 daily service hours). Moreover, since there are no coordinates for the stops in Mandl's work, the coordinates provided by Mumford [23] are used. An overview of the structure is given in Fig. 9.14.

9.4 Adding new datasets

For adding a new dataset, use the content of the `template` dataset as input. Therefore create a new folder in Fi `datasets` and copy the content into a new directory with a name of your choosing. Afterwards, adapt

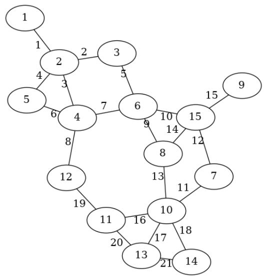


Figure 9.14: The PTN of the Mandl dataset

the local only default parameters in the Fi `basis/Config.cnf` file. For an explanation of the parameters, see Section 9.1.
Before running anything, you need to fill the new dataset with data. To see, which algorithm needs which data, see the respective section in this documentation. For information on the file structure, see Chapter 8.

9.4.1 Adding a dataset from PESPlib

There is a helper method to import a PESPlib dataset. PESPlib ([11]) is a benchmark library for Periodic Event Scheduling Problems, based on timetabling problems in public transport planning.
To import a PESPlib dataset, place the dataset file (e.g. `R1L1.txt`) into Fi `/src/tools/PESPlib_import` and run e.g.

R `python3 pesplib_import.py R1L1`

there. This will create a new dataset folder with the given dataset name and all required files for timetabling in the Fi `/datasets`-directory.

9.4.2 Adding a dataset from TimPassLib

There is a helper method to import a TimPassLib dataset. TimPassLib, see [31, 32], is a library for integrated timetabling and passenger routing problems. To import a dataset, create a new dataset based on the `template` dataset and copy the files `Activities.csv`, `Events.csv`, `OD.csv` and `Config.csv` to CK `filename_timpasslib_activities` (Fi `timpasslib/Activities.csv`), CK `filename_timpasslib_events` (Fi `timpasslib/Events.csv`), CK `filename_timpasslib_od` (Fi `timpasslib/OD.csv`) and CK `filename_timpasslib_config` (Fi `timpasslib/Config.csv`), respectively. Depending on CK `timpasslib_import_timetable`, the timetable is imported, i.e., you might need to copy `Timetable.csv` to CK `filename_timpasslib_timetable` (Fi `timpasslib/Timetable.csv`). Run

R `timpasslib-import`

such that the following files are created:

- CK `default_stops_file` (Fi `basis/Stop.giv`),
- CK `default_edges_file` (Fi `basis/Edge.giv`),
- CK `default_od_file` (Fi `basis/OD.giv`),
- CK `default_pool_file` (Fi `basis/Pool.giv`),
- CK `default_pool_cost_file` (Fi `basis/Pool-Cost.giv`),
- CK `default_lines_file` (Fi `line-planning/Line-Concept.lin`),

- CK default_events_periodic_file (Fi timetabling/Events-periodic.giv),
- CK default_activities_periodic_file (Fi timetabling/Activities-periodic.giv),
- CK default_timetable_periodic_file (Fi timetabling/Timetable-periodic.tim) (if CK timpasslib_import_timetable is set to CV true),

Note that the TimPassLib files contain no information on the coordinates of stops, such that the coordinates (x-coordinate, y-coordinate) in Fi default_stops_file will be set to $(0, 0)$. Similarly, in Fi default_edges_file, the parameter length is set to zero and lower-bound and upper-bound are set according to lower-bound and upper-bound in Fi default_activities_periodic_file.
Additionally, config parameters specified in Fi filename_timpasslib_config will be used to update Fi Config.cnf. Note that parameters in Fi filename_timpasslib_config that are not already present in Fi Config.cnf will not be added. These need to be added manually to the corresponding configuration file.

Exporting a dataset in TimPassLib format

A dataset can be exported in TimPassLib format by running

R timpasslib-import

such that the follwoing files are created according to the specification of [31]:

- CK filename_timpasslib_activities (Fi timpasslib/Activities.csv),
- CK filename_timpasslib_events (Fi timpasslib/Events.csv),
- CK filename_timpasslib_od (Fi timpasslib/OD.csv),
- CK filename_timpasslib_config (Fi timpasslib/Config.csv) and
- CK filename_timpasslib_timetable (Fi timpasslib/Timetable.csv) (depending on CK timpasslib_export_timetable.

Additionally, basic evaluations of the data set are computed and added to CK default_statistic_file (Fi statistic/statistic.sta):

SK **n_stations:** The number of stations.

SK **n_lines:** The number of operated lines. Note that a line can have a frequency higher than one.

SK **od_prop_entries_greater_zero:** The number of OD pairs with $d_{st} > 0$.

SK **od_prop_overall_sum:** The total number of passengers.

SK **n_events:** The number of events in the event-activity network.

SK **n_activities:** The total number of activities in the event-activity network.

SK **n_activities_fixed:** The number of activities in the event-activity network with $\ell_a = u_a$.

SK **n_activities_free:** The number of activities in the event-activity network with $u_a - \ell_a = T - 1$.

SK **n_activities_restricted:** The number of activities in the event-activity network with $\ell_a < u_a < \ell_a + T - 1$.

9.4.3 Dataset generator

There is a make command to create new artificial datasets. To use it, navigate into the Fi /datasets-directory and run

R `make dg-generate-dataset`

This creates a new dataset as new subdirectory with the method specified by CK `dg_model`. For a detailed description of the available models see Section 4.1.

Chapter 10

LinTim Core

For allowing easier extensions of LinTim, its core functionality is provided in two languages, namely Python (3) and Java. There is a version for C++ too, but it is deprecated.
In the following the vocabulary of Java is used, but the versions for Python is structured in the same way. The core is organized into several packages, which are briefly explained in the following sections. Note that for continuity all core libraries follow the naming convention for Java for their public API as far as possible. To create a javadoc version of the documentation run

R `make docs`

in the folder Fo `/src/core/java`. An HTML version of the documentation can then be found in Fo `/src/core/java/docs`.

10.1 Model

The package `model` consists of interfaces which represent basic concepts and classes which represent the basic objects used in public transport planning.

10.1.1 Interfaces

The following interfaces are given.

`Graph` with basic graph functionality

`Node` with basic node functionality

`Edge` with basic edge functionality, can be directed or undirected

`Path` with basic path functionality

`OD` structure to handle OD information

10.1.2 Classes

The following classes are given.

`Stop` representing a stop in a PTN, implementing `Node`

`Link` representing a link in a PTN, implementing `Edge`

`InfrastructureNode` representing a node in an infrastructure network, e.g., a possible stop location or an intersection, implementing `Node`

`InfrastructureEdge` representing an infrastrcture edge between infrastructure nodes, e.g. a street or a track, implementing `Edge`

`WalkingEdge` representing a walking path between infrastructure nodes, implementing `Edge`

`DemandPoint` representing a demand point, i.e., the demand at a certain location

`StationLimit` representing an individual station limit for a stop, containing individual bounds on the transfer or waiting times

`Line` representing a line in the PTN

`LinePool` representing a line pool

`ODPair` representing an origin destination pair

`PeriodicEvent` representing an event in the periodic event activity network

`PeriodicActivity` representing an activity in the periodic event activity network

`PeriodicHeadway` representing a headway activity in the periodic event activity network

`AperiodicEvent` representing an event in the aperiodic event activity network

`AperiodicActivity` representing an activity in the aperiodic event activity network

`AperiodicHeadway` representing a headway activity in the aperiodic event activity network

`Timetable` representation of a timetable

`PeriodicTimetable` representation of a periodic timetable

`Trip` representing an aperiodic trip, e.g., a line serving

`Routing` representing a routing in a PTN or an EAN.

`Zone` representing a zone, i.e. a subset of the nodes of a PTN.

`ZonePrices` stores the prices for travelling through a specified number of zones.

`PriceMatrix` representing a matrix of prices for each node pair in the PTN.

`VehicleTour` collecting multiple trips to represent the tour of a vehicle throughout the day

`Circulation` collecting multiple vehicle tours to represent a circulation

`RidepoolArea` representing a ridepooling area

`Ridepool` representing a ridepool, i.e. a collection of ridepooling areas

10.1.3 Enumerations

The following enumerations are given.

`EventType` possible types of events

`ActivityType` possible types of activities

`LineDirection` possible direction of a line (`FORWARDS, BACKWARDS`)

`TariffModelType` possible types of tariff models.

`TariffObjectiveType` possible objectives in tariff planning.

TariffWeightType possible weight options in the objective for tariff planning.

TariffZoneCountingType possible counting modes in a zone tariff.

TariffZoneSymmetryOption possible options for symmetry breaking in the optimization of a zone tariff.

TariffRoutingType possible options for generating a routing in tariff planning.

10.1.4 Package `model.impl`

The package `model.impl` in the Java core contains different implementations of the interfaces, which might be useful in different scenarios.

SimpleMapGraph graph implementation based on Java Maps. Most of the times faster than an `ArrayListGraph`. May not contain multiple nodes/edges with the same index.

ArrayListGraph graph implementation

LinkedListPath path implementation

MapOD OD implementation used for OD matrices with unknown amount of entries. In most cases the fastest.

FullOD OD implementation used for OD matrices with many entries

SparseOD OD implementation used for OD matrices with few entries

10.2 Input and Output

The package `io` contains reader and writer for all classes in `model` as well as the ones in `util` which need them.

10.3 Algorithm

The package `algorithm` contains implementation of algorithms working on `model` classes, which are needed at several places in LinTim.

Dijkstra shortest path implementation using Dijkstra's algorithm

10.4 Utility

The package `util` contains utility classes and enumerations.

Config a representation of the config

Statistic a representation of the statistic

Pair representation of a tuple consisting of 2 elements

LogLevel wrapper mapping different Java logging levels to the ones we are using

SolverType enumeration of different solver types

10.5 Solver

The package `solver` contains an abstract solver implementation, used to formulate a model once and switch the used solver easily. Currently only a small subset of all possible features is implemented, aimed towards high performance to avoid unneccessary overhead. For more information, see the corresponding Javadoc or documentation in the python code.

10.6 Exceptions

The following error catalog is used. All exceptions inherit from `LinTimException` such that logging is handled only once.

Input

- input file cannot be found: `Error I1: File <filename> cannot be found.`
- format of input files is wrong: `Error I2: File <filename> is not formatted correctly: <x> columns given, <y> needed.`
- inconsistency: `Error I3: Column <x> of file <filename> should be of type <type> but entry in line <line number> is <entry>.`
- inconsistent numbering: `Error I4: Datatype <data-type> is not numbered consistently starting from 1, but <algorithm-name> needs that.`

Output

- output cannot be written: `Error O1: File <filename> cannot be written.`
- no output is produced: `Error O2: Algorithm <algo> did not terminate correctly, no output will be produced.`

Config parameters

- file not found: `Error C1: No config file can be found.`
- existence: `Error C2: Config parameter <configkey> does not exist.`
- type: `Error C3: Config parameter <configkey> should be of type <type> but is <configparameter>.`
- file name not given: `Error C4: No config file name given.`
- invalid value: `Error C5: Value(s) of config parameter(s) <configparameters> are invalid or incompatible in this context.`

Algorithms

- stopping criterion reached: `Error A1: Stopping criterion of algorithm <algo> reached without finding a feasible/optimal solution.`
- infeasible parameter setting: `Error A2: Algorithm <algo> cannot be run with parameter setting <configkey>; <configparameter>.`
- in Dijkstra, distance was queried before computation: `Error A3: Distance to <node> was queried before computation`
- in Dijkstra, path was queried before computation: `Error A4: Path to <node> was queried before computation`
- in Dijkstra, algo was called with node, that was not in the graph, when the class was initialized: `Error A5: Usage of unknown node <node>`. This may happen, when the graph was altered after initialization

- in Dijkstra, there is an edge with negative length: `Error A6: Edge <edge> has negative length <length>`. Dijkstra cannot work reliably with negative edge length.
- in Dijkstra, if the network is not connected: `Error A7: Node <sourceNode> is not connected to node <targetNode>, but a shortest path was queried`. This may happen during computation of a shortest path or when computing all shortest paths starting from a specific node.

Graphs

- multiple nodes with same index: `Error G1: Node with id <node id> already exists.`
- multiple edges with same index: `Error G2: Edge with id <edge id> already exists.`
- left or right node of edge does not exist: `Error G3: Edge <edge id> is incident to node <node id> but node <node id> does not exist.`
- edge between two nodes does not exist: `Error G4: Edge between <node id> and <node id> does not exist.`

Lines

- link cannot be added to line: `Error L1: Link <link id> cannot be added to line <line id>.`
- line contains a circle: `Error L2: Line <line id> contains a circle.`
- line is no path: `Error L3: Line <line id> is no path.`

Routings

- no path specified between two nodes: `Error R1: No path available from <node id> to <node id>`
- path in routing is inconsistent: `Error R2: The given path from <node id> to <node id> is not consistent.`

Data inconsistency

- periodic event to aperiodic event does not exist: `Error D1: Periodic event <event id> to aperiodic event <event id> does not exist.`
- periodic activity to aperiodic activity does not exist: `Error D2: Periodic activity <activity id> to aperiodic activity <activity id> does not exist.`
- index not found: `Error D3: <Element> with index <index> not found.`
- illegal event type: `Error D4: <Event type> of event <event id> is no legal event type.`
- illegal activity type: `Error D5: <Activity type> of activity <activity id> is no legal activity type.`
- illegal line direction: `Error D6: <Line direction> of event <event id> is no legal line direction.`
- number of lines in Pool-Cost-file does not match number of lines in the linepool: `Error D7: Read <number of> entries in the line cost file <filename>, but <number of> lines are in the line pool.`

- multiple paths given for one node pair in a routing: `Error D8: There are multiple paths given from <node id> to <node id>.`
- Path in a routing is inconsistent: `Error D9: The path from <node id> to <node id> is not valid.`
- Routing is incomplete, but complete routing is needed: `Error D10: The given routing is incomplete, but a complete routing is needed.`
- Stop is assigned to no zone: `Error D11: The stop <stop id> is not assigned to any zone.`
- Stop is assigned to multiple zones: `Error D12: The stop <stop id> is assigned to more than one zone.`
- price matrix does not contain a price for all node pairs with distinct origin and destnation node: `Error D13: There is no price specified from <node id> to <node id>.`
- zone price list is inconsistent: `Error D14: Zone price file is inconsistent`
- Ridepooling area is not strongly connected: `Error D15: Ridepool area with id <area id> is not strongly connected`
- Ridepooling area is inconsistent: `Error D16: Ridepool area with id <area id> can not be created`

Solver

- solver not supported: `Error S1: Solver <solver name> not supported for algorithm <algo>.`
- Gurobi Error: `Error S2: Gurobi returned the following error: <exception.toString()>`
- Cplex Error: `Error S3: Cplex returned the following error: <exception.toString()>`
- Cplex Error: `Error S4: The solver <solver name> is not yet implemented in the core solver library.`
- Attribute not implemented: `Error S5: Attribute <attribute name> is not implemented for <solver name> yet.`
- Parameter not implemented: `Error S6: The parameter <parameter name> is not implemented for <solver name> yet.`
- Variable type not implemented: `Error S7: The variable type <variable type> is not implemented for <solver name> yet.`
- Invalid call: There was an invalid call, e.g., reading variables of an infeasible model. Please check the text for further information. `Error S8: <error message>`
- Glpk Error: `Error S9: Glpk returned the following error: <exception.toString()>`
- Solver found no feasible solution: `Error S10: Solver found no feasible solution. Check solver output for further information.`

Statistic

- type mismatch: `Error ST1: Statistic key <key> should have type <type> but has value <value>.`
- key not found: `Error ST2: Statistic parameter <configkey> does not exist.`

Chapter 11

Introduction to extending LinTim

11.1 Logging

The following guidelines govern the output expected from LinTim programs.

11.1.1 Output from LinTim programs

Output from LinTim programs must adhere to the formatting described here.
For software using a LinTim core Library (Java, C++, ...), there are dedicated logging Classes to use for output.
These will default to write to STDOUT, and the Makefile invocations shall do so, but they can also be configured otherwise.
Software not using a LinTim library should use STDOUT or a commonly used facility for its respective programming environment/language that can be configured for writing to STDOUT, so Makefile invocations can do so.

11.1.2 Log levels

The following Levels shall be used:

FATAL for errors that cancel the execution

ERROR for errors that are severe, but do not stop the program

WARN (a.k.a. warning) for messages from the program that need not be a real error, but may be of interest to the user (also hints for probably wrong configuration) because they might want to be cautious about it, as something is probably different from what they might expect

INFO for everything that happens as expected and is of interest to the end user

DEBUG for output that allows to see what's happening under the hood

In the output to STDOUT (be it configurable through a library or not), the loglevel must be written in capital letters, preceded by the current system time formatted as YYYY-MM-DD HH:mm:ss at the beginning of the line, followed by a colon, a space, and the actual message. (Only) DEBUG messages may additionally contain hints to the source code like the classname, source code line, and/or stack traces of Exceptions, etc.. Multi-line messages are allowed for DEBUG messages.

11.1.3 Error messages

The messages outlined in the Error catalog (Section 10.6) shall be used literally for their respective FATAL, ERROR or WARN messages. The level depends upon the severity for the respective program.

11.1.4 Info messages

The following INFO and DEBUG messages should be written at the beginning and end of the respective steps. If a step is not present in a particular program, the respective output can be omitted. Any introductory INFO message(s) (e.g. stating the program name and version) or nothing at all can be output at the beginning.

INFO: Begin reading configuration

DEBUG: Parameter <key> set to <value>

INFO: Finished reading configuration

INFO: Begin reading input data

DEBUG: Reading file <path/and/filename> (done automatically by the respective reader)

INFO: Finished reading input data

INFO: Begin execution

further DEBUG and INFO messages as you see fit

INFO: Finished execution

INFO: Begin writing output data

DEBUG: Writing to file <path/and/filename> or Appending to file <path/and/filename> (done automatically by the respective writer)

INFO: Finished writing output data

Whether the setup of a mathematical program for a solver is done during the reading step (maybe on the fly) or as part of the execution step is up to the author. Solvers may produce their own output to report progress. Whenever possible, the output of a solver shall be configured to go into the filename provided by the configuration key CK `solver_output_file`. (which may contain a relative or absolute path). If the key is the empty string or not set at all, solver output shall be printed to STDOUT, but not through the logging facility (or only at the DEBUG level). (Note: Solver output refers to the usual progress report, not to the results, i.e., values of variables in the solution. Still, intermediate or final results may or may not be part of the solver output.)

11.2 Cleaning

Due to the vast number of algorithms in LinTim, manually cleaning the Fo `src` directory is tedious. Therefore, LinTim provides an automatic capability to do so by running

R `make clean-src`

in a dataset-folder or

R `make clean`

in the Fo `src` directory. There are several file types cleaned automatically from all directories in Fo `src` (see Fi `src/FILES_TO_CLEAN`) but you may add additional files as well. To do so, create a file named Fi `FILES_TO_CLEAN` in the source directory of the algorithm and add all files that should be deleted, one per line. Glob patterns, e.g. Fi `bin/*` are supported.

Chapter 12

Continous Integration

There are some continous integration tests contained in LinTim. They can be found in the folder Fo /ci.

12.1 Running the tests

There are two possibilities, running all test cases and running a specific test.
For running all tests, run the script Fi /ci/run_all_tests.sh. This file will set some basic environment variables for the solvers and run every test separately. Failed tests will output their respective console log and the names of all failed tests will be collected in Fi /ci/failed_tests. Also, you may need to make sure, that the environment variables for running the necessary solvers are set for your system, see Chapter 1.2.
There is also the possibility to run a single test. For this, use Fi /ci/run ussingle_test.sh with the corresponding test name as the first and only parameter.
Additionally, note that the tests are mostly regression tests, designed to find unintended changes on already implemented algorithms. Therefore, the results are based on running specific software versions on specific hardware. They are therefore likely to fail for you. On the other hand, the unit tests should work for every installation of LinTim. You can run them separately with Fi /ci/run_unit_tests.sh

12.2 Adding test cases

There is the possibility to add your own test cases. A test contains of four things, a list of LinTim commands to run, a dataset to run the commands on, a Fi Private-Config.cnf for configuration, and an expected statistic result.
To add your own test, copy the content of Fo /ci/template into a new subdirectory of Fo /ci. In there, the commands to run and the dataset can be changed by setting the corresponding variables in Fi run.sh.
To add your own configuration parameters, adapt Fi basis/Private-Config.cnf in your test directory. This file will be copied in the given dataset before running the test commands.
For the expected results, add data into the file Fi expected-statistic.sta in your test directory. This file will be compared to the statistic file created by the test commands and will determine the success or the failure of the test. For a successful test, all statistic keys in the Fi expected-statistic.sta need to be contained in the produced statistic file and their values need to coincide. Note that the produced statistic file may contain more data, this will not cause the test to fail.
Every test will create a new version of the corresponding dataset, you may therefore not assume the dataset to differ from the currently commited version.

Chapter 13

Changelog

This section contains a brief changelog of the different versions. Note that the changelog is not complete and does only include the most important features. For a complete list of changes, use the version control system. The version numbers of LinTim are based on the date of release and are not semantic.

2024.12

Added

- Added new load generation algorithm based on spanners in the PTN.
- Ridepooling: Added three models to generate ride pool, two models to solve the ridepooling and lineplanning problem simultaneously and corresponding Reader/Writer Classes.

Changed

- Updated code to Gurobi version 12. Gurobi versions older than version 11 are not supported anymore.
- For generated ring datasets, the stop-indices are shifted by 1 and now start at 1.

Fixed

- Line pool and line concept graphics are also created when the value of the configuration keys CK `default_pool_graph_file` and CK `default_line_graph_file` differ from the default setting.
- The configuration file given to Fi `src/line-planning/line-planning.sh` is passed over to the called java programs.
- Fixed line concept evaluation using change model FORMULA_2 or FORMULA_3.

2024.08

Added

- Added functionality for tariff planning including optimizing and evaluating fares.
- New method to generate a line pool based on centers and periphery nodes.
- Added IP model for line planning with passenger routing to minimize the total (estimated) traveling time.
- Added Mandl dataset.

Fixed

- Line pool creators also add lines starting with an edge specified in backwards direction (of the line direction).
- Timetable evaluation parameter SK `tim_overcrowded_time_average` now also considers overcrowded wait activities.
- Correctly compute wait and transfer time in line concept evaluation using change-and-go graph.
- Specified change stations are taken into account when constructing the EAN.
- Upper frequency bounds computed via CK `load_generator_upper_frequency_factor` are rounded up.
- The terminal-to-terminal model for line pool generation uses the standard line cost computation method including the CK `lpool_costs_vehicles`.
- Line reader now uses CK `gen_conversion_length`.

2023.12

Added

- Functionality for creating artificial datasets, see Section 4.1.
- Import and export to TimPassLib format, see Section 9.4.2.
- Multi-commodity flow routing in order to evaluate a timetable, taking into account the capacities, see Section 5.7.1.
- Possibility to draw PTN without stop-coordinates.

Changed

- Allow non-integer activity weights for all activity buffer models.
- If OD pair is not present, it is interpreted as 0 in the python core.

Fixed

- Key feature computation for ml-robustness framework will now correctly respect the chosen routing window.
- Extended evaluation of line concept solve a multi-commodity flow problem instead of independent source problems for all origins.
- Timetable evaluation can now handle non-integer passengers.
- Fixed integer overflow in c++ core when writing large integers.
- Fixed bug in timetable evaluation that allowed routing of passengers on sync and headway activities.
- Periodic timetabling with constraint propagation terminates with an exception if activity file is missing.
- An error when reading a CSV file with the Java core is no longer always indicated by the message "File not found". Message "Wrong encoding" was added.
- Fixed candidate set for stop location; will now correctly handle cases where demand points are exactly CK `sl_radius` distanced from possible stop locations.

- Added CK gen_conversion_length and CK gen_conversion_coordinates to core writers, so that a the values are not changed when a graph is only read and written again.
- Use CK ean_change_penalty under all settings of the configuration parameter CK ean_model_weight_change.

2022.08

Added

- Possibility to read stop geo coordinates in Java and Python core libraries
- Interactive visualization of a PTN, see Section 4.11.1
- Visualization of the OD data via a graph or a heatmap, see Section 4.11.2
- Possibility to visualize the ptn load weights, see Section 4.11.3
- Installation script for installing dependencies automatically, see Section 1.3

Changed

- Remove Station-Distance requirement of

  R `make vs-add-circulations-to-ean`

- Updated JUnit-version from 4.12 to 4.13.2
- Update Java core cplex interface to CPLEX 20.1
- Now most config parameters are case insensitive
- Rewrite line concept evaluation for better performance and more evaluations. Note that the names of some statistic entries changed to be more clear. For all current evaluations, see Section 5.5.

Fixed

- Python Core: Prevent overwriting statistic when trying to append
- Python Core: Correctly parse the entries in stop-possible? for infrastructure nodes
- Python & Java Core: Dijkstra will now return copies of the computed paths to prevent accidental changes by the user.
- Fix solver core interface of the stop location travel time model
- Fix solver dependency of extended line planning cost model
- Line planning direct model can now handle non-zero diagonal entries in the od matrix

2021.12

Added

- Added more integer programming solver support. For an overview which solvers are support by which algorithms, see Section 6. For more information on how to combine solvers with LinTim, see Section 1.2.1.
- Robust integrated planning based on machine learning predictions. For more information, see Section 3.10.5.
- Possibility to run LinTim an ARM-based cpus, e.g. Apple-M1

Fixed

- Add java core dependency installation for terminal-to-terminal line pool generation
- Fix wrong make target for ean passenger reroute
- Fix missing build files for line pool drawing
- Line pool cost computation will now scale the ptn edges acccording to CK `gen_conversion_length`
- Will now read CK `ptn_stop_waiting_time` for the ptn evaluation
- The vehicle-based term of the line costs now accounts for undirected lines as well

Removed

- Possibility to run LinTim on i586 cpus.

2021.10

Added

- Ability to respect additional load per link in load generation, see Section 7.7
- Export to GTFS, see Section 8.9
- Cycle base formulation for periodic timetabling, see Section 3.6.6
- Phase 1 simplex for periodic timetabling, see Section 3.6.7
- Visum-Interface to import datasets from PTV Visum, see Section 4.12. This includes several additions to LinTim:
  - An infrastructure model, more detailed than the current PTN representations, see e.g. Section 3.1.2
  - Possibility of passengers to walk, see e.g. Section 4.5 and Section 4.2.4
  - Respecting transfer stations and line terminals, see e.g. Section 4.5 and Section 3.2.1
  - Forbidding edges in line planning, see e.g. Section 3.3.1

Changed

- Bump used JGraphT version, now JGraphT 1.5 and JHeaps 0.13 are required
- Java 11 is now required
- Maven ($\geq$ 4) is now required
- Rewrite several ip models, using a common naming scheme for solver parameters and align the output of the programs to the rest of LinTim

Fixed

- The rollout step will not read the headways anymore if they are not needed
- Python Core now reads directed ptns correctly
- DM extended evaluation now computes average values correctly
- Rolling out passenger paths now works on aperiodic eans without changes

- PTN load generator will now compute correct variable upper frequency bounds for very small load values
- Rolling out passenger paths does not allow headways in passenger paths anymore
- Fixed Big-M-value for DM1

2020.12

Added

- Additional IP parameters for Gurobi
- Dataset ring

Changed

- Python Core: Replaced usage of DictGraph by SimpleDictGraph to improve performance
- Core: StatisticWriter will default to appending to the file on disc instead of overwriting
- Line planning model direct is now allowed a non-integer budget restriction
- Remove goblin dependency from periodic modulo simplex, use gurobi now instead
- Allow periodic timetable evaluation without an od matrix present

Fixed

- R `make ean-add-simple-vs` will now respect the parameter CK `time_units_per_minute`
- Line Planning method CV `cost_restricting_frequencies` can now be compiled with only one of the supported solvers installed
- Python core will use default statistic for reading if none is given
- Fixed bug in cycle base version of integrated timetabling and passenger routing model
- Adapted ean_change_penalty for time_units_per_minute in dataset athens
- Equals method in periodic and aperiodic ean now working in python core
- Suppress double logging/console output when using the core gurobi solver interface with gurobi 9
- Python core vehicle schedule writer reads correct default config key for the vehicle schedule file
- R `make ean-add-simple-vs` will now throw an error when run on a directed ptn
- CK `time_units_per_minute` are now consistently handled in all vehicle scheduling methods

2020.02

Added

- Sioux Falls dataset
- Models for integrated planning
  - Integrated timetabling and passenger routing
  - Integrated line planning, timetabling and passenger routing
  - Integrated timetabling and vehicle scheduling

  - Integrated line planning, timetabling, passenger routing and vehicle scheduling
  - Computing a new timetable for given line plan and vehicle schedule

- Respect fixed lines in line planning
- Respect fixed lines in timetabling
- Modulo Simplex algorithm for timetabling
- Full support for running under Windows
- Import of VISUM datasets
- New Python core graph implementation
- Automatic cleaning of src folders
- Robustness checks for delay management

Changed

- The export format to visum does now include the line repetition

Deprecated

- the cpp core will not be maintained any more and will be removed in a future version

2018.06

First release version

Bibliography

[1] S. H. Bull, R. M. Lusby, and J. Larsen, *An optimization based method for line planning to minimize travel time*, Proceedings of the 13th conference on advanced systems in public transport (CASPT), 2015.

[2] S. Bunte and N. Kliewer, *An overview on vehicle scheduling models*, Public Transport **1** 2009, no. 4, 299–317.

[3] M. Bussieck, *Optimal lines in public rail transport*, Ph.D. Thesis, 1998.

[4] E. Carrizosa, J. Harbering, and A. Schöbel, *Minimizing the passengers' traveling time in the stop location problem*, Journal of the Operational Research Society **67** 2016, no. 10, 1325–1337.

[5] A. Fielbaum, S. Jara-Diaz, and A. Gschwender, *A parametric description of cities for the normative analysis of transport systems*, Networks and Spatial Economics **17** 2017, no. 2, 343–365.

[6] *Collection of open source public transport networks by DFG Research Unit "FOR 2083: Integrated Planning For Public Transportation"*, 2018. https://github.com/FOR2083/PublicTransportNetworks.

[7] M. Friedrich, M. Hartl, A. Schiewe, and A. Schöbel, *Angebotsplanung im öffentlichen Verkehr - Planerische und algorithmische Lösungen*, Heureka, 2017.

[8] ———, *Integrating passengers' assignment in cost-optimal line planning*, 17th workshop on algorithmic approaches for transportation modelling, optimization, and systems (ATMOS 2017), 2017, pp. 1–16.

[9] ———, *System headways in line planning*, CASPT 2018, 2018.

[10] P. Gattermann, J. Harbering, and A. Schöbel, *Line pool generation*, Public Transport **9** 2017, 7–32.

[11] M. Goerigk, *PESPlib*. https://timpasslib.net/pesplib.html.

[12] ———, *Verallgemeinerte Schnittheuristiken in der periodischen Fahrplangestaltung*, Master's Thesis, 2009.

[13] M. Goerigk and A. Schöbel, *Improving the modulo simplex algorithm for large-scale periodic timetabling*, Computers & Operations Research **40** 2013, no. 5, 1363–1370.

[14] M. Goerigk, A. Schöbel, and F. Spühler, *A phase I simplex method for finding feasible periodic timetables*, 21st symposium on algorithmic approaches for transportation modelling, optimization, and systems (ATMOS 2021), 2021, pp. 6:1–6:13.

[15] J. Harbering, *Delay resistant line planning with a view towards passenger transfers*, TOP **25** 2017, 467–496.

[16] I. Heinrich, O. Herrala, P. Schiewe, and T. Terho, *Using light spanning graphs for passenger assignment in public transport*, 23rd symposium on algorithmic approaches for transportation modelling, optimization, and systems (atmos 2023), 2023, pp. 2:1–2:16.

[17] A. Kaufmann, *Column generation for line planning with minimal traveling time*, Master's Thesis, 2016.

[18] M. Lachmann, *Vehicle scheduling based on a line plan only*, Master's Thesis, 2016.

[19] L. J. LeBlanc, E. K. Morlok, and W. P. Pierskalla, *An efficient approach to solving the road network equilibrium traffic assignment problem*, Transportation Research **9** 1975, no. 5, 309–318.

[20] C. Mandl, *Applied network optimization*, Academic Press, New York, 1979.

[21] M. Müller-Hannemann, R. Rückert, A. Schiewe, and A. Schöbel, *Framework for generating machine learning models for robustness*, 2021. available at https://gitlab.rlp.net/for2083/framework-for-generating-machine-learning-models-for-robustness.

[22] M. Müller-Hannemann, R. Rückert, A. Schiewe, and A. Schöbel, *Towards Improved Robustness of Public Transport by a Machine-Learned Oracle*, 21st symposium on algorithmic approaches for transportation modelling, optimization, and systems (atmos 2021), 2021, pp. 3:1–3:20.

[23] C. L. Mumford, *Research on the urban transit routing problem (bus routing)*, 2016. https://users.cs.cf.ac.uk/C.L.Mumford/Research%20Topics/UTRP/Outline.html#Xmumford2013.

[24] J. Pätzold, A. Schiewe, P. Schiewe, and A. Schöbel, *Look-ahead approaches for integrated planning in public transportation*, 17th workshop on algorithmic approaches for transportation modelling, optimization, and systems (ATMOS 2017), 2017, pp. 17:1–17:16.

[25] J. Pätzold and A. Schöbel, *A matching approach for periodic timetabling*, 16th workshop on algorithmic approaches for transportation modelling, optimization, and systems (ATMOS 2016), 2016, pp. 1:1–1:15.

[26] PTV AG, *Visum 17 user manual*, 2018.

[27] M. Schachtebeck, *Delay management in public transportation: Capacities, robustness, and integration*, Ph.D. Thesis, 2010.

[28] M. Schachtebeck and A. Schöbel, *To wait or not to wait—and who goes first? Delay management with priority decisions*, Transportation Science **44** 2010, no. 3, 307–321.

[29] A. Schiewe and P. Schiewe, *An iterative approach for integrated planning in public transportation*, Georg-August-Universität Göttingen, 2018. Working Paper.

[30] P. Schiewe, *Integrated optimization in public transport planning*, Ph.D. Thesis, 2018.

[31] P. Schiewe, M. Goerigk, and N. Lindner, *TimPassLib*. https://timpasslib.net/.

[32] ______, *Introducing TimPassLib – a library for integrated periodic timetabling and passenger routing*, Operations Research Forum **4** 2023, no. 3.

[33] A. Schöbel, *Optimization models in public transportation*, 2004.

[34] ______, *Optimization in public transportation. stop location, delay management and tariff planning from a customer-oriented point of view*, Optimization and Its Applications, Springer, New York, 2006.

[35] ______, *Integer programming approaches for solving the delay management problem*, Algorithmic methods for railway optimization, 2007, pp. 145–170.

[36] ______, *Line planning in public transportation: models and methods*, OR Spectrum **34** 2012, no. 3, 491–510.

[37] ______, *An eigenmodel for iterative line planning, timetabling and vehicle scheduling in public transportation*, Transportation Research C **74** 2017, 348–365.

[38] A. Schöbel, H. W. Hamacher, A. Liebers, and D. Wagner, *The continuous stop location problem in public transportation*, Asia-Pacific Journal of Operational Research **26** 2009, no. 1, 13–30.

[39] A. Schöbel and S. Scholl, *Line planning with minimal traveling time*, 5th workshop on algorithmic methods and models for optimization of railways, 2006.

[40] A. Schöbel and S. Schwarze, *Finding delay-resistant line concepts using a game-theoretic approach*, Netnomics **14** 2013, no. 3, 95–117.

[41] P. Serafini and W. Ukovich, *A mathematical model for periodic scheduling problems*, SIAM Journal on Discrete Mathematics **2** 1989, no. 4, 550–581.

[42] B. Stabler, *Sioux falls - github*, 2018. available at https://github.com/bstabler/TransportationNetworks/tree/master/SiouxFalls.

[43] A. Uffmann, *Umlaufplanung mit dem Kanalmodell*, Master's Thesis, 2010.