LIN-TIM

An integrated environment for mathematical public transport optimization

Documentation

Developed at
Institute for Numerical and Applied Mathematics
University of Goettingen

Contributors

Head

- Prof. Dr. Anita Schöbel

Technical Lead

- M.Sc. Alexander Schiewe

Research Assistants

- M.Sc. Sebastian Albert
- M.Sc. Julius Pätzold
- M.Sc. Philine Schiewe
- Dr. Jochen Schulz

Student Assistants

- B.Sc. Florentin Hildebrandt
- B.Sc. Felix Spühler

Former Staff

- Dipl.-Math. Rasmus Fuhse
- Dr. Konstantinos Gkoumas
- Dr. Marc Goerigk
- Dr. Jonas Harbering
- Dr. Jonas Ide
- M.Sc. Mridul Roy
- Dr. Michael Schachtebeck
- Dipl.-Math. Michael Siebert
- M.Sc. Vitali Telezki
- Dipl.-Math. Anke Uffmann

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 **Lin**eplanning 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, C and C++ installed on your system. There may be some special algorithms requiring additional programming languages (i.e., python or matlab), but if this is the case this is noted in the respective section of the documentation.
Also, some programs make use of the commercial integer programming solvers Xpress, Cplex and Gurobi, but they are only necessary if all functions of LinTim are desired. Especially, for each of the planning stages line planning and timetabling there are also algorithms working without a solver installed. See the instructions on the respective algorithms for configuring LinTim to use your chosen solver. If you want to use them, make sure that the respective paths are set correctly:
For Xpress, source the `xpvars.sh` script. 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 Cplex, make sure the library paths are set correctly. You might need to export

R
```
export
  CLASSPATH="$CLASSPATH:/usr/local/cplex126/cplex/lib/cplex.jar"
```

Finally, for Gurobi, you might need to set the classpath as well:

R
```
export CLASSPATH="$CLASSPATH:/opt/gurobi/linux64/lib/gurobi.jar"
```

Furthermore we need to add other environment variables. First, we need to set

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

Afterwards, we can use the setup script provided by Gurobi to set the other variables, i.e., run

R
```
source /opt/gurobi/linux64/bin/gurobi.sh
```

Note that this will open a gurobi console, you may close this with `exit()`.

For some of the graphical representations, the installation of the graphviz library is required, see `http://www.graphviz.org/` for instructions.
Some of the python scripts may need packages, e.g., numpy or gurobipy. For reference, see the corresponding scripts. We assume that these libraries are installed on your machine.
Also for using all of LinTim, you will have to fulfill other thirdparty dependencies. For more information, have a look at Fi `/libs/README.md` .

1.3 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 7, 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 6.1.
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 9 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 textfiles structured similarly (# comments a line). The advantage of this concept is that they are very independent of the programming language used.
In the most textfiles, 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 4.7.
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:

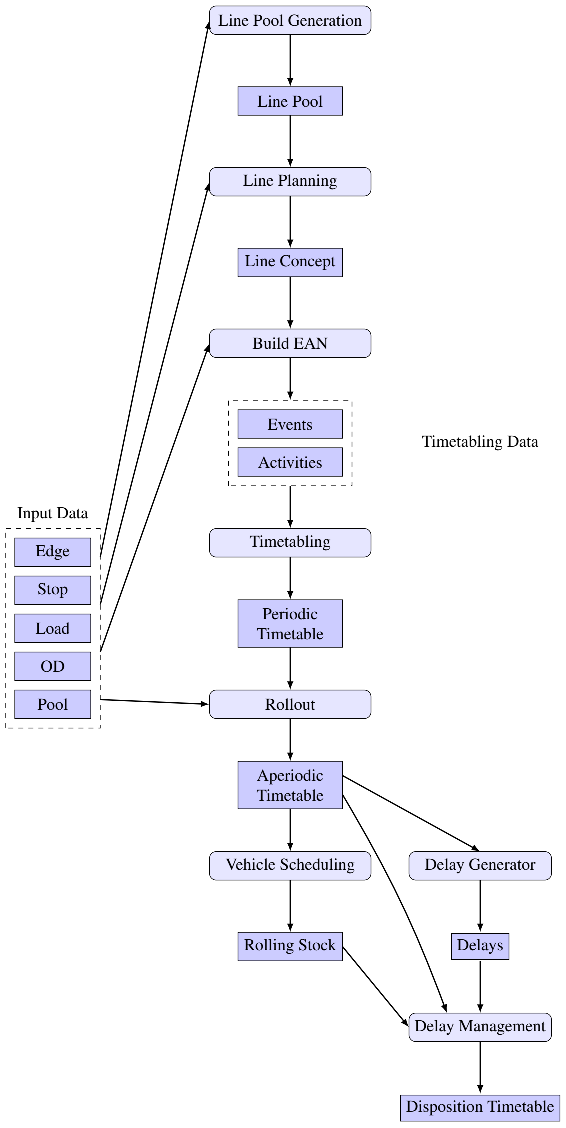

Figure 2.1: Different planning steps considered in LinTim

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:

- 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

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.

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).

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.

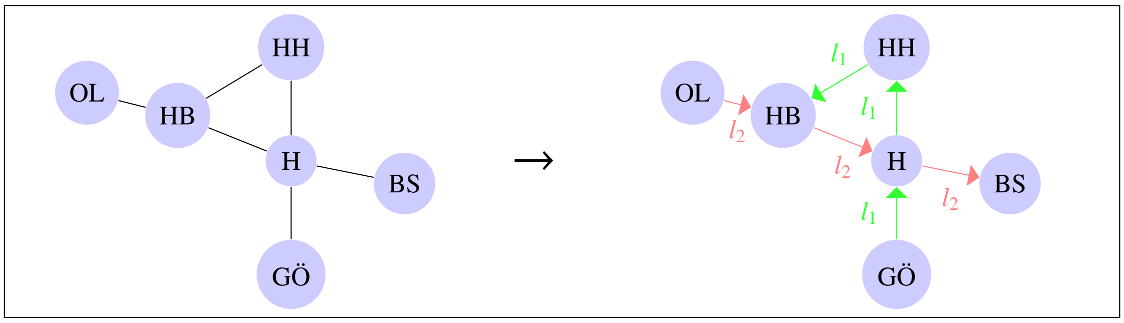

Figure 2.2: Line Planning Step

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 `minchanges_ip` integer program trying to minimize the weighted number of transfers (see 3.3.5)
- CK `lc_model` CV `minchanges_cg` column generation procedure trying to minimize the weighted number of transfers (see 3.3.5)
- 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 Periodic Timetabling

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

2.4.1 Preparation

Run

R `make ean`

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

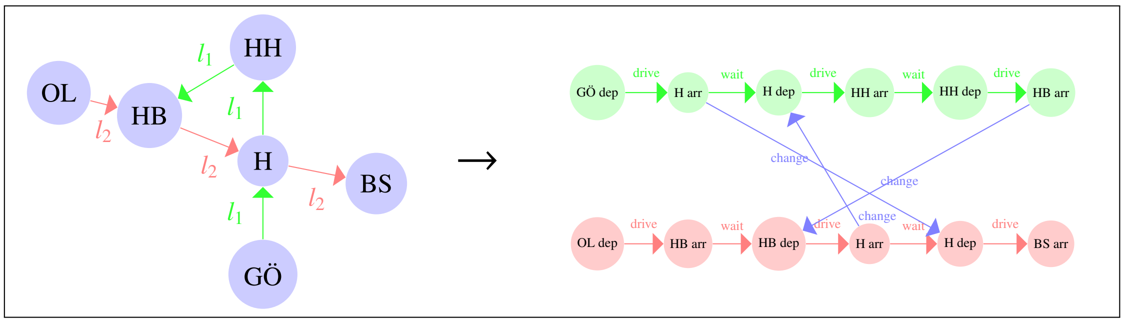

Figure 2.3: Creation of an EAN

2.4.2 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

2.4.3 Output

The following files are produced as output.

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

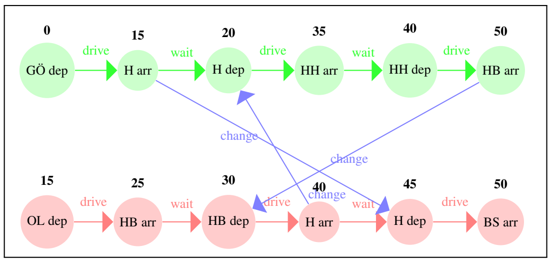

Figure 2.4: Periodic Timetabling Step

2.4.4 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. *Currently not included in the release version of LinTim.*
- 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. *Currently not included in the release version of LinTim.*
- CK `ip` Models the Periodic Timetabling Problem as an IP and solves it.

2.5 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 informations are based on the classic formulations, based on the aperiodic timetable.

2.5.1 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.

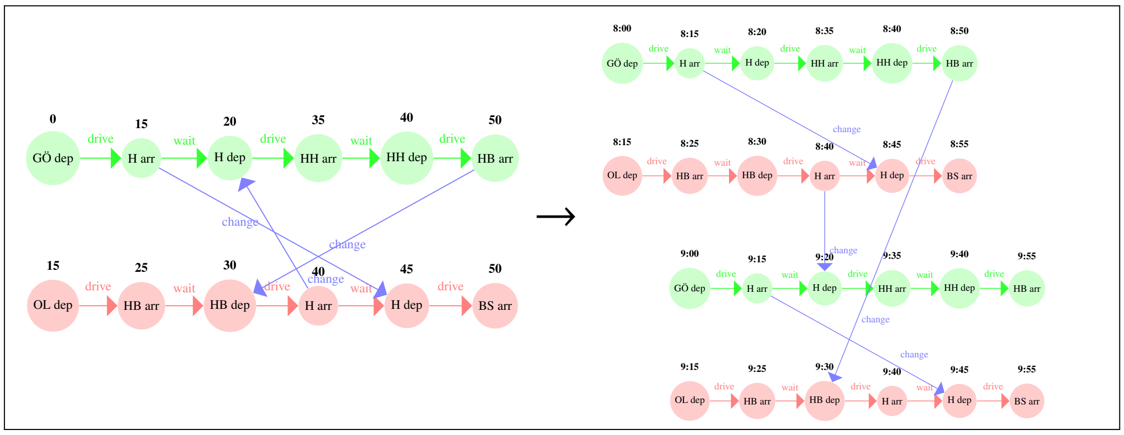

Figure 2.5: Rollout Step

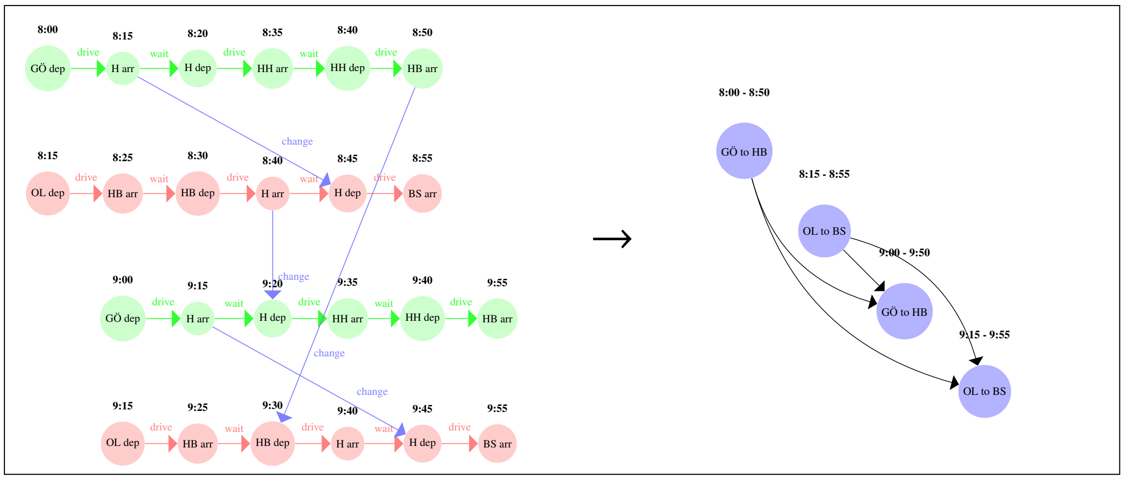

Figure 2.6: Rollout to Trips Step

2.5.2 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

2.5.3 Output

The following files will be produced:

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

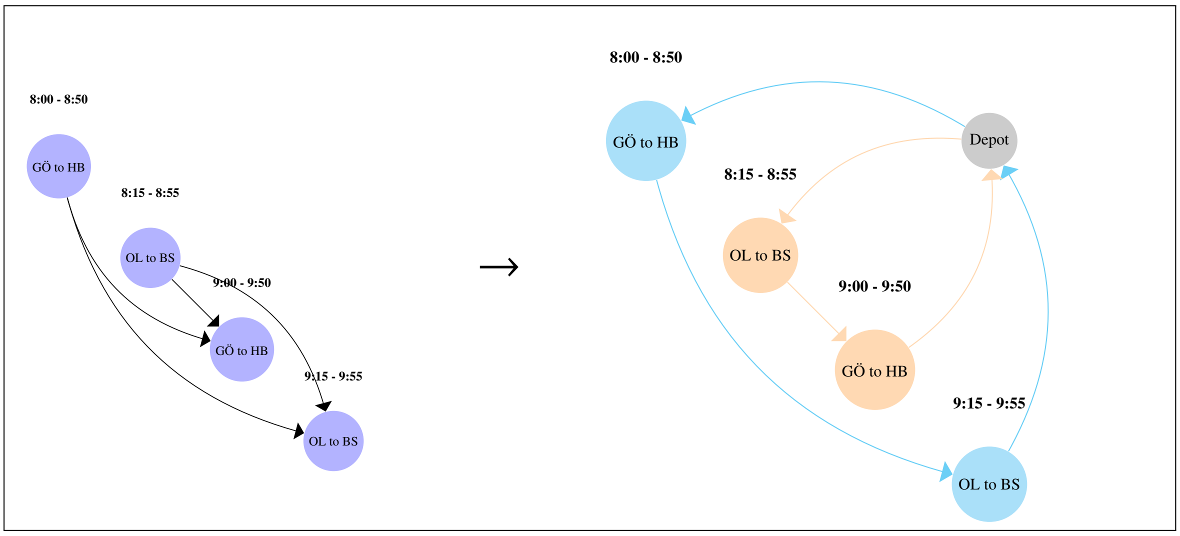

Figure 2.7: Vehicle Scheduling

2.5.4 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 3.5.1)
- CK `vs_model` CV `MDM2` Minimizing the number of vehicles (see 3.5.2)
- CK `vs_model` CV `ASSIGNMENT_MODEL` Minimizing the overall costs (see 3.5.3)
- CK `vs_model` CV `TRANSPORTATION_MODEL` Minimizing the overall costs (see 3.5.4)
- CK `vs_model` CV `NETWORK_FLOW_MODEL` Minimizing the overall costs (see 3.5.5) (see 3.5.1)
- CK `vs_model` CV `CANAL_MODEL` More detailed version of CV `ASSIGNMENT_MODEL` (see 3.5.6)
- CK `vs_model` CV `LINE_BASED` vehicle scheduling only based on line planning (see 3.5.7)
- CK `vs_model` CV `SIMPLE` will create a vehicle schedule driving the lines back and forth (see 3.5.8)
- CK `vs_model` CV `IP` solve a simple ip model (see 3.5.9)

2.6 Delay Management

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

2.6.1 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.

2.6.2 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`

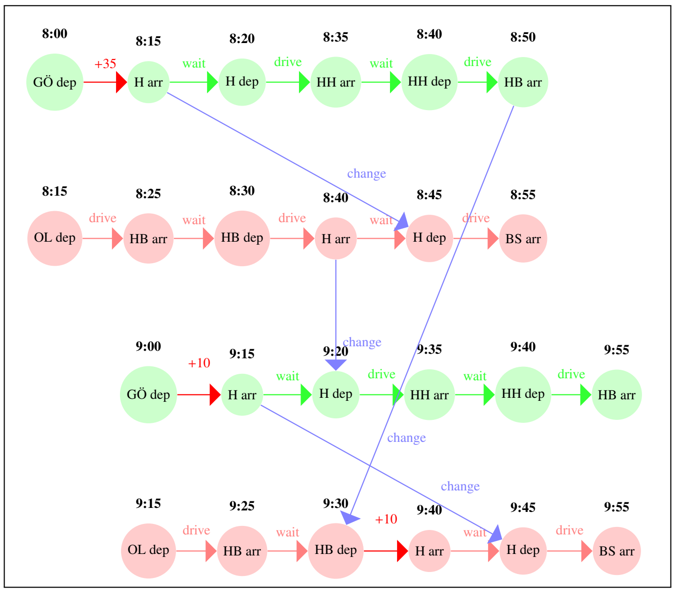

Figure 2.8: Generation of Delays

2.6.3 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`)

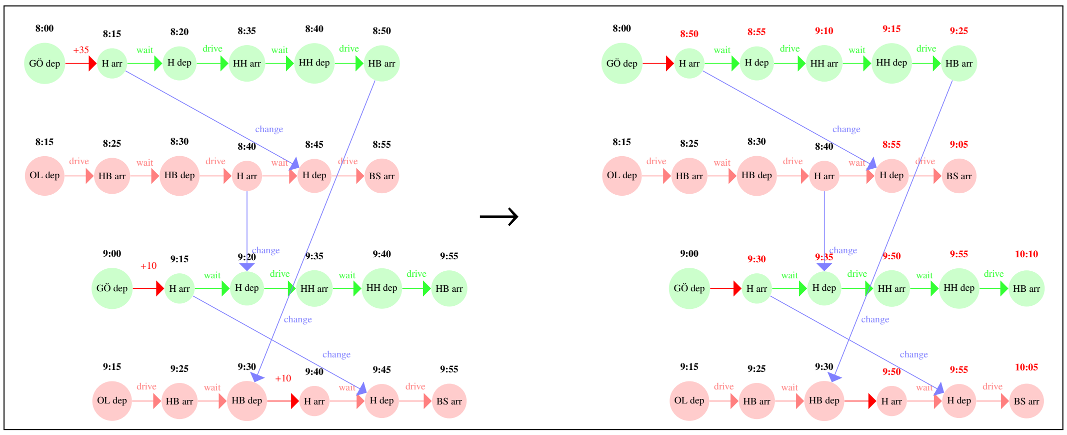

Figure 2.9: Delay Management Step

2.6.4 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).

Chapter 3

Detailed Description of Algorithms

3.1 Stop Location

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.

3.1.1 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 mode

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 [18, 21].

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 [18, 21].

3.1.2 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].

3.2 Line Pool Generation

For creating a new line pool by running

R `make lpool-line-pool`

there are two different possibilities, using CK `lpool_model` CV `tree_heuristic` or CK `lpool_model` CV `k_shortest_paths`.

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`)) and a given CK `default_loads_file` (Fi `basis/Load.giv`) (see Chapter 6) 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 3.2.4.
The algorithm iteratively creates minimal 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 fulfil 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 minimal 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 CK `default_loads_file` (Fi `basis/Load.giv`) 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 weight of an edge which is used to compute the minimal 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 $f_e^{\max} \cdot$ `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`.

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.
For more information, see [8].

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_turnaround_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

$$\begin{aligned}&[\boxed{\text{CK}}\ \texttt{period\_length} - \boxed{\text{CK}}\ \texttt{vs\_turnaround\_time}\\ &-\boxed{\text{CK}}\ \texttt{lpool\_restricted\_maximum\_buffer\_time},\\ &\boxed{\text{CK}}\ \texttt{period\_length} - \boxed{\text{CK}}\ \texttt{vs\_turnaround\_time}].\end{aligned}$$

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 [14].

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 Line costs

The costs of the lines created by

R
```
make lpool-line-pool
```

are of the following form

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

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 [20], 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 [6].

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 [24].

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_max_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 timelimit for the solver (-1=infinity)
- CK `lc_max_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
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., Fi `basis/Pool.giv` and Fi `basis/Pool-Cost.giv` .

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. 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.
The following parameters control the behavior of the algorithm:

- CK `gen_passengers_per_vehicle` capacity of the vehicles
- CK `lc_budget` maximal cost of the resulting line concept

- CK `ean_model_weight_drive` model to compute the shortest paths
- CK `lc_timelimit` the timelimit for the solver in seconds (-1=infinity)
- CK `lc_mip_gap` the mip optimization gap for the solver, 0.1 equals a gap of 10 % (-1=use default value)
- 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.

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_max_frequency`.

- CK `gen_passengers_per_vehicle` capacity of the vehicles
- CK `lc_budget` maximal cost of the resulting line concept
- CK `ean_model_weight_drive` model to compute the shortest paths
- CK `lc_number_of_possible_frequencies` restrict the number of possible frequencies (-1=infinity)
- CK `lc_timelimit` the timelimit for the solver (-1=infinity)
- CK `lc_max_frequency` the maximal allowed frequency

Aggragating the passengers per OD pair

Running

R `make lc-line-concept`

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

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 travelers model and CV `1` to the cost model. The tolerance of feasibility, integrality and optimality can be set by CK `lc_mult_tolerance`.
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 Travelingtime

Executing

R `make lc-line-concept`

with CK `lc_model` CV `traveling_time_cg` solves the traveltime model as stated in [22]. It is a column generation procedure in which the passenger paths are generated throughout the column generation iterations. It is implemented as part of [12]. 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 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 [11].

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 [22]. 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 3.3.5 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 [11].

3.3.6 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 [23].

3.4 Timetabling

3.4.1 Modulo Network Simplex Algorithms

Currently not included in the release version of LinTim.
There are different ways to use the Modulo Network Simplex Algorithm, depending on how to provide a starting solution:

- C `tim_model; "ns_improve"` It is assumed that Timetable-periodic.tim already contains a feasible starting solution; only improvement steps are taken.
- C `tim_model; "csp_ns"` A starting solution is found using Abscon; high reliability, small running times, but the starting solution quality is usually bad – see Section 3.4.2.
- C `tim_model; "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 3.4.3.

There are two search procedures that may be further specified, one for local search and one for fundamental search for cuts, see [10]. 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 [9].

3.4.2 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.

3.4.3 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.

3.4.4 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 [15].

3.4.5 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 [24]. You can set a timelimit and a desired gap by setting

- C `tim_pesp_ip_gap` or
- C `tim_pesp_ip_timelimit`.

The default value of 0 will disable the respective option.
For more information on the model, see [25].

3.5 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 4.6).

3.5.1 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].

3.5.2 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].

3.5.3 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`). For more information on the model, see [2].

3.5.4 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].

3.5.5 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].

3.5.6 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. For more information on the model, see [26].

3.5.7 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 [13].

3.5.8 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.

3.5.9 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 timelimit 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`. For more information on the model, see [2].

3.6 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 4.6), and some primary delays on events and/or activities (see section 4.7). 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)

3.6.1 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`.

3.6.2 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 [16]. 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 [18, 19]. 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 [18, 19]. 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 [16, p. 38] for reducing the size of the event-activity network. For more information, see [18, 19].

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 [16, p. 56]. For more information, see [16, 17]. *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 [16, p. 57]. For more information, see [16, 17]. *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 [16, p. 57]. For more information, see [16, 17]. *Heuristic algorithm – might not find the global optimum. Note that* CV *`FRFS` is always at least as good as this method* [16, 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 [16, p. 57]. For more information, see [16, 17]. *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* [16, 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 [16, p. 68]). For more information, see [16, 17]. *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 [16, Lemma 4.5] and [16, 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 [17]. *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`

Chapter 4

Auxiliary Algorithms

4.1 OD Matrix Creation

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

4.1.1 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

4.1.2 Output

The following file is produced as output:

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

4.1.3 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 proportionally 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 Load distribution

This steps 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 other case, see 4.2.4.

4.2.1 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`)

4.2.2 Output

The following file is produced as output:

- CK `default_loads_file` (Fi `basis/Load.giv`)

4.2.3 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, see 4.3.
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. CV `load_generator_max_iteration` determines the number of iterations allowed before the algorithms terminates, if no convergence is observed. For a more detailed description of the models, see [5].
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`) plus CK `ean_change_penalty`. For waiting at a stop, the behavior of CK `ean_model_weight_wait` is adopted, see 4.3. For a more detailed description of the Change&Go-network see [22]. 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.

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 `lc_passengers_per_vehicle`. The following parameters are used:

CK **`load_generator_lower_frequency_factor`** A factor to multiply the lower frequency after dividing by the capacity. The result is rounded up.

CK **`load_generator_fix_upper_frequency`** Whether a fixed upper frequency should be used. Otherwise the upper frequency depends on the lower frequency.

CK **`load_generator_upper_frequency_factor`** The factor to multiply the lower frequency to get the upper frequency. The result is rounded down.

CK **`load_generator_fixed_upper_frequency`** The fixed upper frequency to use.

4.2.4 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 PTN to EAN

4.3.1 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

4.3.2 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`)

4.3.3 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) 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. The calculation assumes that the times for each activity are given by CK `ean_model_weight_drive` (resp. wait/change/etc.).
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`.

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 ean-headways`

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

4.5 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.

4.6 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.

4.6.1 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`)

4.6.2 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

4.6.3 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 $\boxed{\text{CK}}$ `rollout_whole_trips` *must* be set to $\boxed{\text{CV}}$ `true`. When rolling out for delay management, usually a short time period (e.g. two hours) is considered and $\boxed{\text{CK}}$ `rollout_whole_trips` should be set to $\boxed{\text{CV}}$ `false`. Typically, the combination of vehicle scheduling and delay management could be like this:

1. Set [$\boxed{\text{CK}}$ `DM_earliest_time`, $\boxed{\text{CK}}$ `DM_latest_time`] to a "large" time interval, e.g. one day, and $\boxed{\text{CK}}$ `rollout_whole_trips` to $\boxed{\text{CV}}$ `true`.
2. Run

   $\boxed{\text{R}}$ `make ro-rollout && make ro-trips`
3. Run

   $\boxed{\text{R}}$ `make vs-vehicle-schedules`

   to generate the vehicle schedules.
4. Set [$\boxed{\text{CK}}$ `DM_earliest_time`, $\boxed{\text{CK}}$ `DM_latest_time`] to the time interval needed for delay management, e.g. two hours, and $\boxed{\text{CK}}$ `rollout_whole_trips` to $\boxed{\text{CV}}$ `false`.
5. Run

   $\boxed{\text{R}}$ `make ro-rollout && make 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.

4.6.4 Requirements and Caveats

- If $\boxed{\text{CK}}$ `DM_enable_consistency_checks` is set to $\boxed{\text{CV}}$ `true`, IDs in files are checked to be consecutively numbered beginning from 1.

4.6.5 Generating Trips

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

$\boxed{\text{R}}$ `make ro-rollout`

with $\boxed{\text{CK}}$ `rollout_whole_trips` set to $\boxed{\text{CV}}$ `true`,

$\boxed{\text{R}}$ `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.

4.7 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`] $\subseteq$ [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.

4.8 Visualization

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

4.8.1 PTN

To create an illustration of the PTN run

R `make ptn-draw`

The result for dataset toy is depicted in Figure 4.1.

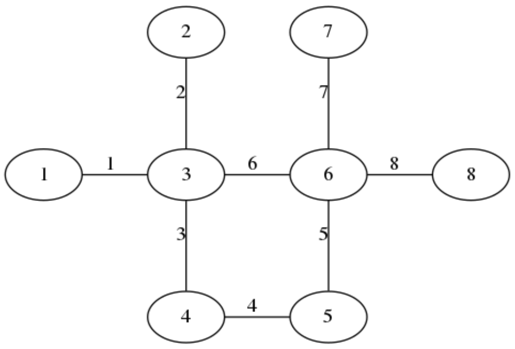

Figure 4.1: The PTN of the toy dataset

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

4.8.2 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 4.2.
The graph can be scaled by adapting CK `lpool_coordinate_factor`.

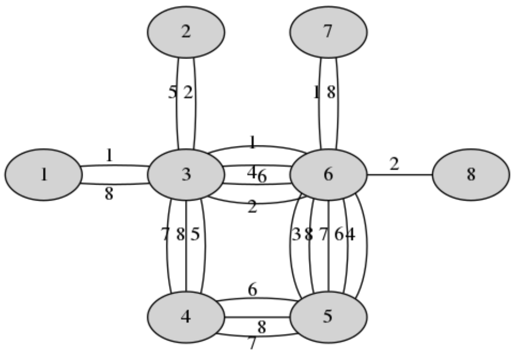

Figure 4.2: The line pool of the toy dataset

4.8.3 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 4.3.

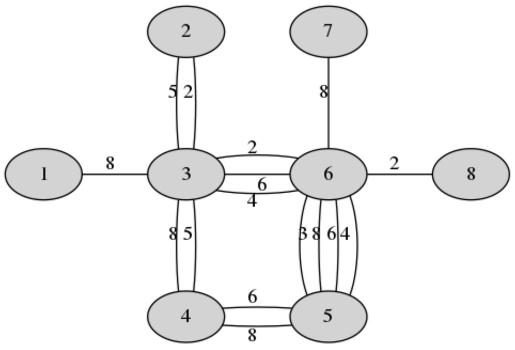

Figure 4.3: One possible line concept of the toy dataset

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

4.8.4 Timetable

To create an illustration of the timetable, run

R make tim-timetable-draw

The result for dataset toy is depicted in Figure 4.4. Note, that this command will draw only the ean, if no timetable is present.

4.8.5 Disposition timetable

To create an illustration of the disposition timetable, run

R make dm-disposition-timetable-draw

The result for dataset toy is depicted in Figure 4.5. 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.

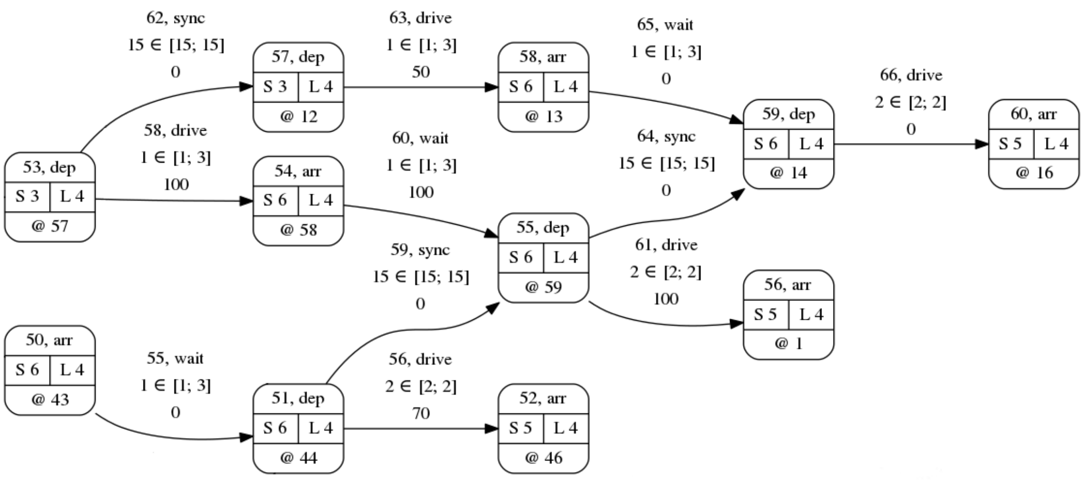

Figure 4.4: Extract of one possible timetable of the toy dataset

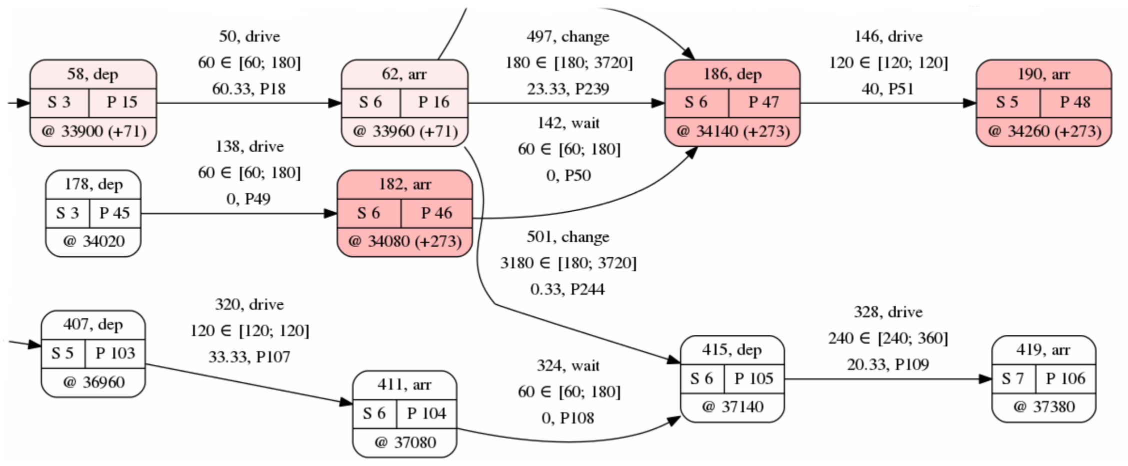

Figure 4.5: Extract of one possible disposition timetable of the toy dataset

4.8.6 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`.

4.9 Interaction with VISUM

During the work on DFG FOR 2083, a basic interface to PTV VISUM ([1]) 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 fixed lines and timetables from VISUM-net-files. The description of the needed format of the VISUM files is out of scope for this documentation, but we want to provide a short idea, how to use the interface.

The interface consists of three parts:

Reading fixed lines: By calling

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

LinTim will read the VISUM-file provided by CK `filename_net_file` (Fi `visum.net`) to find all lines that should be fixed and write them in LinTim-format. For this, we assume that there is a bus-system that should be optimized (noted by "B" in the VISUM files) and other fixed transportation systems. All fixed lines are read and added to the line pool as well as CK `filename_lc_fixed_lines` (Fi `line-planning/Fixed-Lines.lin`) with their respective frequency and the corresponding capacities will be written to
CK `filename_lc_fixed_line_capacities` (Fi `line-planning/Line-Capacities.lin`) . 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.

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.

Transforming timetable to VISUM: By calling

R `make tim-transform-to-visum`

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

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`** $\sum_{l\in\mathcal{L}} cost_l$ - 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`) :

SK **lc_cost** $\sum_{l\in\mathcal{L}} cost_l f_l$ - sum over costs per line times frequency.

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

SK **lc_obj_direct_travelers_sp** Number of direct travelers on all shortest paths.

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

SK **lc_prop_directed_lines** $(2\cdot)|\mathcal{L}|$ - number of directed lines. If a line is undirected, it is counted twice.

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

SK **lc_time_average** Average travel-time of all passengers on shortest path through the PTN where only edges occurring in the line concept with frequency greater than zero are used.

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

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

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

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

SK **lc_prop_changes** Sum over all OD pairs, where the number of passengers is multiplied with the number of necessary changes on the shortest path.

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** $|\mathcal{E}|$ - number of events.

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

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

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

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

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

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

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

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

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

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

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

SK **ean_time_average** $\frac{1}{\sum_{a\in\mathcal{A}} c_a} \sum_{a\in\mathcal{A}} c_a \cdot$ ”duration assumption” - estimated average travel time. For duration assumption see 4.3.

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

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

SK **ean_prop_changes_od_max** $\max\limits_{\substack{a\in\mathcal{A}_{change}\\ c_a>0}}$ ”duration assumption of a” - maximal used change duration.

SK **ean_prop_changes_od_min** $\min\limits_{\substack{a\in\mathcal{A}_{change}\\ c_a>0}}$ ”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`** $L_a \leq ((\pi_j - \pi_i - L_a) \text{mod } T) + L_a \leq U_a$ for all $(i, j) = a \in \mathcal{A}$ - Are lower and upper bounds on travel time on each activity respected.

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

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

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

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

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

SK **`tim_prop_changes_od_max`** $\max_{\substack{(i,j)=a\in\mathcal{A}_{change}\\ c_a>0}} \left(\pi_j - \pi_i\right) \text{mod} T$ - maximal used change duration.

SK **`tim_prop_changes_od_min`** $\min_{\substack{(i,j)=a\in\mathcal{A}_{change}\\ c_a>0}} \left(\pi_j - \pi_i\right) \text{mod} T$ - minimal used change duration.

SK **`tim_number_of_transfers`** Weighted number of transfers.

5.8 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.

5.9 Evaluation of the Delay Management

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.

5.10 Evaluation of the Vehicle Scheduling

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_cost** 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.

Chapter 6

In- and Output Data

This section will describe all files and their contents that are in- or outputs of the LinTim algorithms.

6.1 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; csp
```

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.

6.2 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.

6.3 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.

6.3.1 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`.

6.3.2 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

6.3.3 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`.

6.3.4 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

6.3.5 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`.

6.3.6 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: the distance between two stops can be transformed to kilometers by multiplying with CK `gen_conversion_coordinates`.

6.3.7 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

6.3.8 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`.

6.3.9 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

6.3.10 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

6.3.11 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

6.3.12 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

6.3.13 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`.

6.4 Line Planning

The folder Fo `line-planning` contains information about the line concept.

6.4.1 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.

6.4.2 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 cannot 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.

6.5 Timetabling

The folder Fo `timetabling` contains information about the periodic event-activity-network and the timetable.

6.5.1 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

6.5.2 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

6.5.3 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

6.5.4 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 4.9. 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

6.6 Delay Management

The folder Fo `delay-management` contains information about the aperiodic event-activity-network, timetable and delays with a disposition timetable

6.6.1 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

6.6.2 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

6.6.3 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

6.6.4 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

6.6.5 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

6.6.6 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

6.6.7 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:

`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-station` the stop id of the start of the line

`start-time` the starting time of this service of the line

`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-station` the stop id of the end of the line

`end-time` the ending time of this service of the line

`line` the line id

6.7 Vehicle Scheduling

The folder Fo `vehicle-scheduling` contains information about the vehicle tours in the dataset.

6.7.1 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

`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-station` the stop id of the start of the line

`start-time` the starting time of this service of the line

`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-station` the stop id of the end of the line

`end-time` the ending time of this service of the line

`line` the line id

Chapter 7

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.

7.1 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.

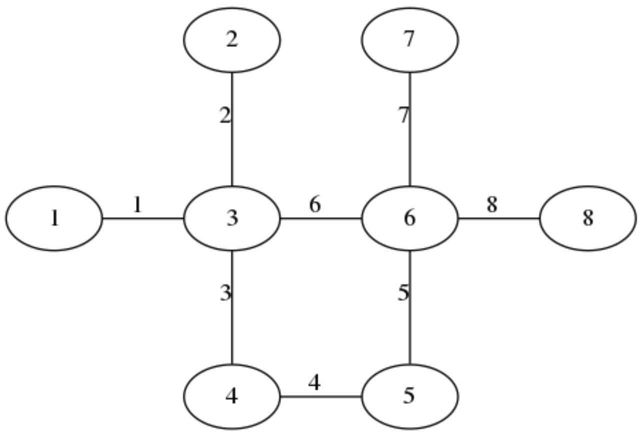

Figure 7.1: The PTN of the toy dataset

7.2 Artificial Datasets

There are two purely artificial datasets in LinTim. These are small examples to test and understand new algorithms.

7.2.1 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. 7.1.
Since the dataset does not contain the necessary informations, stop location is not supported on this dataset out of the box.

7.2.2 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].
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. 7.2. Since the dataset does not contain the necessary informations, stop location is not supported on this dataset out of the box.

7.3 Datasets based on real world data

7.3.1 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. 7.3. To work with this dataset, you need to start with the stop location step.

7.3.2 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.

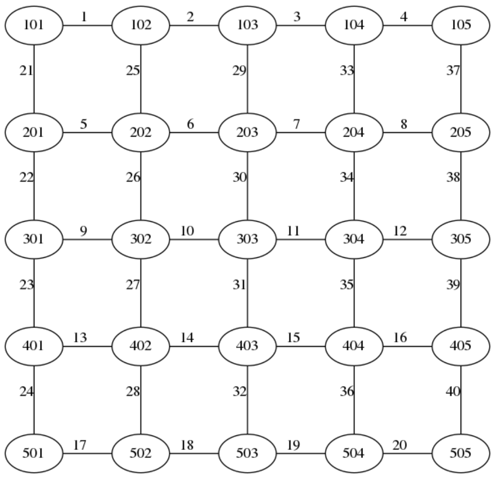
Figure 7.2: The PTN of the grid dataset

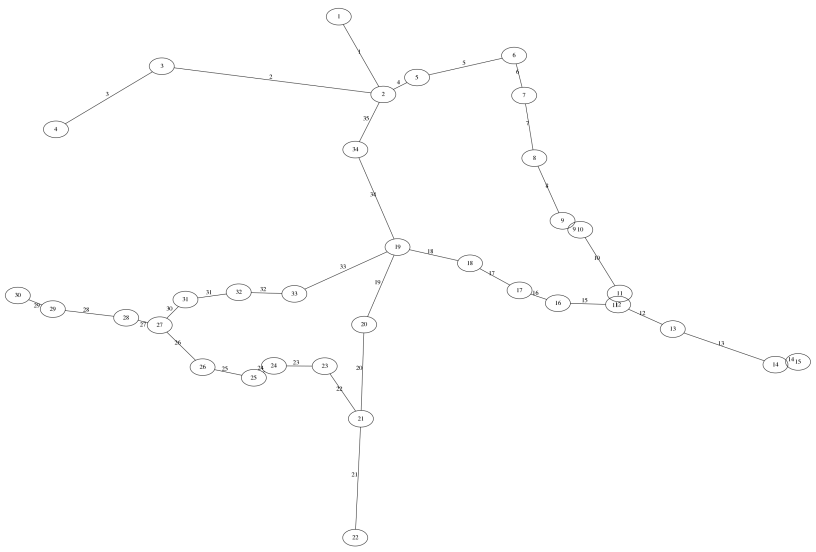
Figure 7.3: Existing infrastructure of the lower saxony dataset

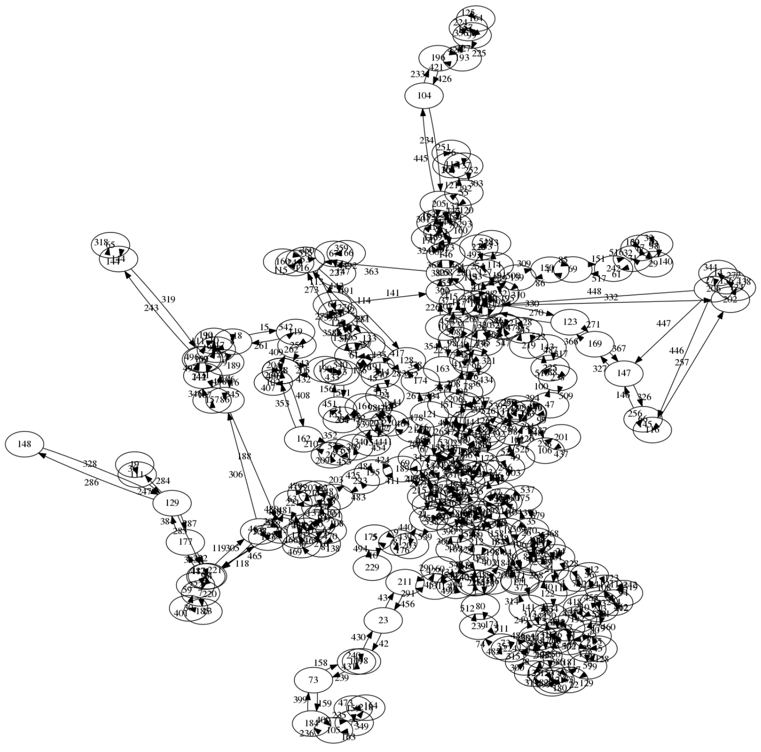

Figure 7.4: The PTN of the goevb dataset

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. 7.4. Since the dataset does not contain the necessary informations, stop location is not supported on this dataset out of the box.
Note, that goevb is a directed network!

7.3.3 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. 7.5. Since the dataset does not contain the necessary informations, stop location is not supported on this dataset out of the box.

7.3.4 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. 7.3.5-7.3.7.
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. 7.6. Since the dataset does not contain the necessary informations, stop location is not supported on this dataset out of the box.

7.3.5 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. 7.3.4, for larger datasets, see Sec. 7.3.6 and 7.3.7.

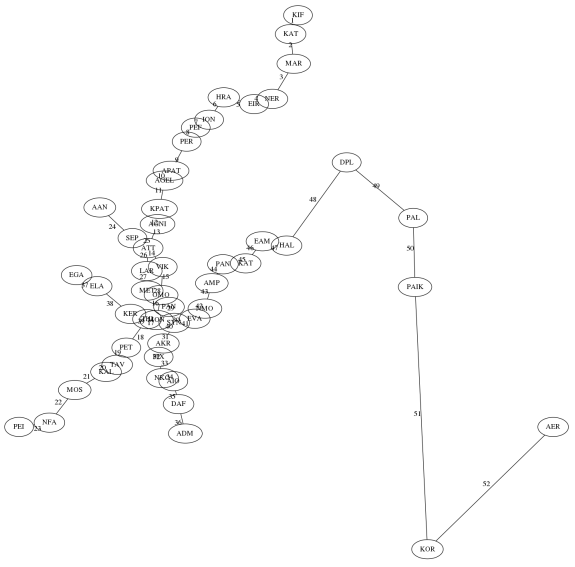
Figure 7.5: The PTN of the athens dataset

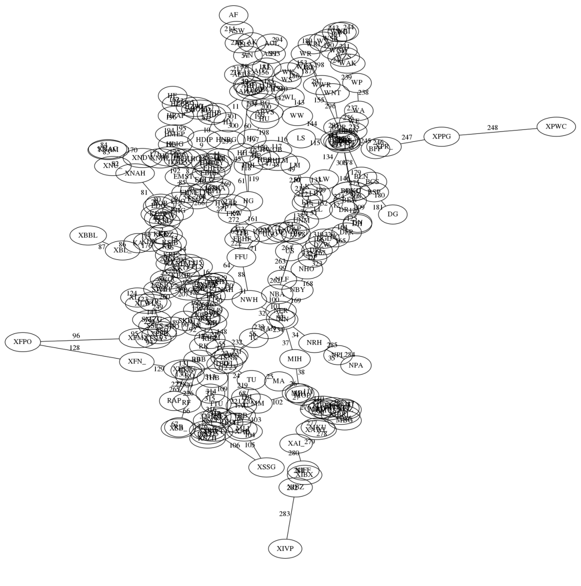
Figure 7.6: The PTN of the bahn-01 dataset

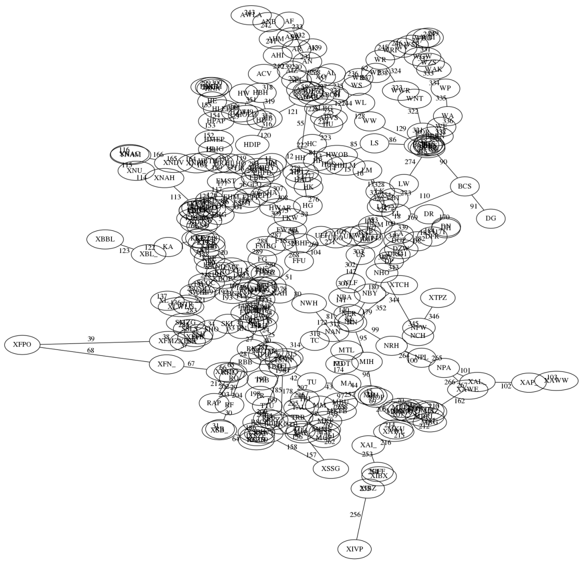

Figure 7.7: The PTN of the bahn-02 dataset

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. 7.7. Since the dataset does not contain the necessary informations, stop location is not supported on this dataset out of the box.

7.3.6 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. 7.3.4 and 7.3.5, for a larger dataset, see Sec. 7.3.7.
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. 7.8. Since the dataset does not contain the necessary informations, stop location is not supported on this dataset out of the box.

7.3.7 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. 7.3.4-7.3.6.
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. 7.9. Since the dataset does not contain the necessary informations, stop location is not supported on this dataset out of the box.

7.3.8 Bahn-equal-frequencies

Currently not included in the release version of LinTim.
The bahn-equal-frequencies dataset is based on bahn-01(7.3.4). 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. 7.10. Since the dataset does not contain the necessary informations, stop location is not supported on this dataset out of the box.

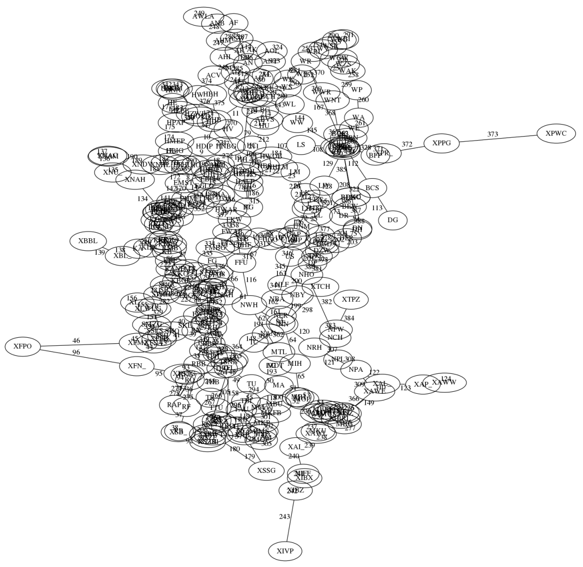
Figure 7.8: The PTN of the bahn-03 dataset

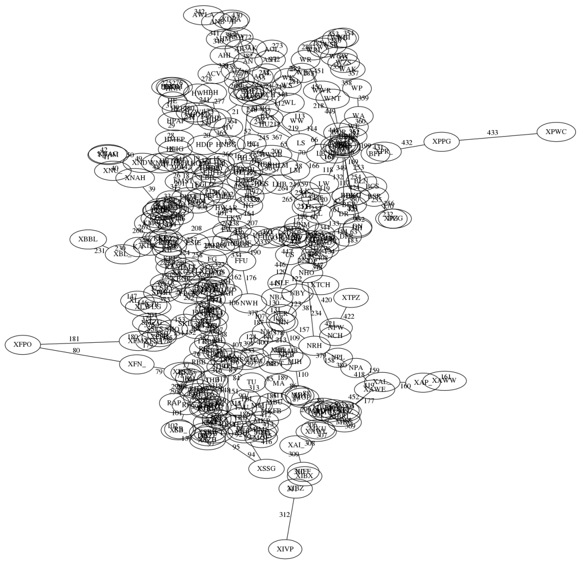
Figure 7.9: The PTN of the bahn-04 dataset

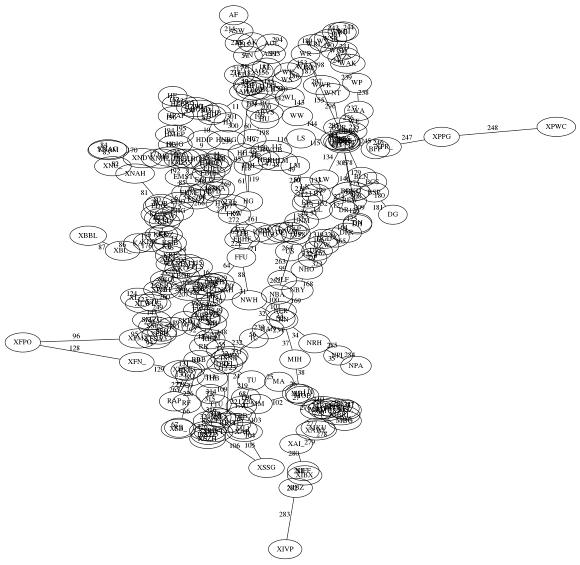
Figure 7.10: The PTN of the bahn-equal-frequencies dataset

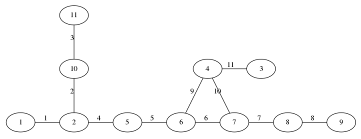

Figure 7.11: The PTN of the BOMHarbour dataset

7.3.9 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. 7.11. Since the dataset does not contain the necessary informations, stop location is not supported on this dataset out of the box.

7.4 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 the local only default parameters in the Fi `basis/Config.cnf` file. For an explanation of the parameters, see Section 7.1.
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 6.

Chapter 8

LinTim Core

For allowing easier extensions of LinTim, its core functionality is provided in two languages, namely C++ and Java. There is a version for Python (3) too, but it is not feature complete yet.
In the following the vocabulary of Java is used, but the versions for C++ 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`.

8.1 Model

The package `model` consists of interfaces which represent basic concepts and classes which represent the basic objects used in public transport planning.

8.1.1 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

8.1.2 Classes

The following classes are given.

`Stop` representing a stop in a PTN, implementing `Node`

`Link` representing a link in a PTN, implementing `Edge`

`DemandPoint` representing a demand point, i.e., the demand at a certain location

`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

8.1.3 Enumerations

The following enumerations are given.

`EventType` possible types of events

`ActivityType` possible types of activities

`LineDirection` possible direction of a line (`FORWARDS, BACKWARDS`)

8.1.4 Package `model.impl`

The package `model.impl` in the Java core contains different implementations of the interfaces, which might be useful in different scenarios.

`ArrayListGraph` graph implementation

`LinkedListPath` path implementation

`FullOD` OD implementation used for OD matrices with many entries

`SparseOD` OD implementation used for OD matrices with few entries

8.2 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.

8.3 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

8.4 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

8.5 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.`

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.`

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.`

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.`

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>`

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 9

Introduction to extending LinTim

9.1 Logging

The following guidelines govern the output expected from LinTim programs.

9.1.1 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.

9.1.2 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.

9.1.3 Error messages

The messages outlined in the Error catalog shall be used literally for their respective FATAL, ERROR or WARN messages. The level depends upon the severity for the respective program.

9.1.4 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) you like (e.g. stating the program name and version) or nothing at all

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>

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 file <path/and/filename> or Appending to file <path/and/filename>

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.)

Chapter 10

Continous Integration

There are some continous integration tests contained in LinTim. They can be found in the folder Fo `/ci`.

10.1 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_tests.sh` . This file will set some basic environment variables for the solvers and run every test separately. Note, that the main script will fail on the first test failure. 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, change into the corresponding subdirectory of Fo `/ci` and run the script Fi `run.sh` . Note, that no environment variables for solvers will be set, therefore this is up to you before starting the test.

10.2 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.

Bibliography

[1] PTV AG, *Visum 17 user manual*, 2018.

[2] Stefan Bunte and Natalia Kliewer, *An overview on vehicle scheduling models*, Public Transport **1** (2009), no. 4, 299–317.

[3] Michael 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] M. Friedrich, M. Hartl, A. Schiewe, and A. Schöbel, *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.

[6] ———, *System Headways in Line Planning*, Caspt 2018, 2018.

[7] M. Friedrich, M. Hartl, A. Schiewe, and A. Schöbel, *Angebotsplanung im öffentlichen Verkehr - Planerische und algorithmische Lösungen*, Heureka, 2017.

[8] P. Gattermann, J. Harbering, and A. Schöbel, *Line pool generation*, Public Transport **9** (2017), no. 1-2, 7–32.

[9] M. Goerigk and A. Schöbel, *Improving the modulo simplex algorithm for large-scale periodic timetabling*, Computers and Operations Research **40** (2013), no. 5, 1363–1370.

[10] Marc Goerigk, *Verallgemeinerte Schnittheuristiken in der periodischen Fahrplangestaltung*, 2009.

[11] J. Harbering, *Delay resistant line planning with a view towards passenger transfers*, TOP (2017). accepted.

[12] A. Kaufmann, *Column generation for line planning with minimal traveling time*, 2016.

[13] Malin Lachmann, *Vehicle scheduling based on a line plan only*, 2016.

[14] 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. 1–16.

[15] 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. 1–15.

[16] M. Schachtebeck, *Delay management in public transportation: Capacities, robustness, and integration*, Ph.D. Thesis, 2010.

[17] 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.

[18] A. Schöbel, *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.

[19] ———, *Integer programming approaches for solving the delay management problem*, Algorithmic methods for railway optimization, 2007, pp. 145–170.

[20] ———, *Line planning in public transportation: models and methods*, OR Spectrum **34** (2012), no. 3, 491–510.

[21] 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.

[22] A. Schöbel and S. Scholl, *Line planning with minimal travel time*, 5th workshop on algorithmic methods and models for optimization of railways, 2006.

[23] A. Schöbel and S. Schwarze, *Finding delay-resistant line concepts using a game-theoretic approach*, Netnomics **14** (2013), no. 3, 95–117.

[24] Anita Schöbel, *Optimization models in public transportation*, 2004.

[25] Paolo Serafini and Walter Ukovich, *A mathematical model for periodic scheduling problems*, SIAM Journal on Discrete Mathematics **2** (1989), no. 4, 550–581.

[26] Anke Uffmann, *Umlaufplanung mit dem Kanalmodell*, 2010.