1 Data Format

1.1 Timetable and Adjustments

The timetable itself is given in a CSV file where each row represents a single leg, i.e. one specific train moving from one station to the next one at a given time. A row consists of the identifier *leg_id* together with additional data on the leg. Namely, the corresponding trains *train_id*, the *start_station_id* and *end_station_id* of the stations the leg departs from and arrives at, the *track_id* of the track being used, the nominally scheduled procedure for this leg as *nominal_departure_configuration*, and finally all alternative *departure_configurations*. Departure configurations are represented as strings of the format *d_t_p* where *d* is the legs departure time in seconds since the beginning of the planning horizon, *t* is the time in seconds required to travel to the next station and *p* is the *profile_id* of the power profile used by the train on this leg. Each leg has exactly one nominal departure configuration, representing the timetable, whereas it may have multiple alternative departure configurations, representing the possible timetable adjustments for this leg. The different departure configurations are separated by a whitespace character and the nominal departure configuration is always included in the departure configuration alternatives. Table 1 summarizes the columns as well as the data types used and gives an example value for each column.

Table 1 Timetable Data Columns

Column Name	Data Type	Example
leg_id	int	1
train_id	int	1
track_id	int	1
start_station_id	int	1
end_station_id	int	2
nominal_departure_configuration	str	"17285_71_4489"
departure_configurations	str	"17285_67_5635 17285_71_4489 ..."

1.2 Power Profiles

Similar to the legs in the timetable, each profile is saved as a row in a CSV file with two columns. A row contains the profiles identifier *profile_id* together with the profiles *power_consumptions*, represented as a string consisting of floats, which are rounded to three decimals and separated by whitespace characters. Each float represents the average power consumption during one second in megawatts. Therefore, a profile corresponding to a travel time of *t* seconds will consist of *t* values. Table 2 states the data types as well as example values.

Table 2 Profile Data Columns

Column Name	Data Type	Example
profile_id	int	5635
power_consumptions	str	"0.225 0.422 0.675 0.939 1.255 ..."

1.3 Constraints

The final building block of an instance is the set of timetabling constraints saved in a JSON file. Table 3 gives an overview of the constraint types together with the data types used to represent them and the remainder of the section consists of paragraphs offering more detail on each constraint type.

Table 3 Constraint Data Dictionaries

Constraint Name	Key	Data Type	Value Example
headway_time_constraints	first_leg_id	int	3131
	second_leg_id	int	3157
	min_headway_time	int	100
single_track_headway_constraints	first_leg_id	int	6069
	second_leg_id	int	458
dwell_time_constraints	first_leg_id	int	1
	second_leg_id	int	2
	min_dwell_time	int	20
terminal_turnaround_constraints	first_leg_id	int	206
	second_leg_id	int	6149
	min_dwell_time	int	420
connection_constraints	first_leg_id	int	8846
	second_leg_id	int	3110
	min_connection_time	int	180
	max_connection_time	int	240
	connection_type	str	"arrival_to_departure"
recuperation_subnets	subnet_id	int	1
	track_ids	List[int]	[1, 2, 3, ...,]

Headway Time Constraints

```
{"first_leg_id": 3131,
 "second_leg_id": 3157,
 "min_headway_time": 100}
```

The field *min_headway_time* is the time in seconds required between the trains represented by the first and second leg, which are consecutively using the same track in the same direction. We assure the minimum headway time between both, the departures as well as the arrivals of the two legs involved. Therefore, a valid timetable must ensure that the constraints

$$d_1 + h \leq d_2$$
$$d_1 + t_1 + h \leq d_2 + t_2$$

holds, where d_1, d_2 are the departure times of the first and second leg, t_1, t_2 the respective travel times and h the minimal headway time. Note, that depending on which train is faster, only the first or only the second constraint must be stated explicitly.

Single Track Headway Constraints

```
{"first_leg_id": 6096,
 "second_leg_id": 458}
```

Between some stations there exists only one physical track which is then used for both directions. In this case, the legs represent two trains using the same track in opposite directions.A valid timetable then has to assert that the first train arrives at its destination before the second leg departs, i.e.

$$d_1 + t_1 \leq d_2,$$

where d_1, d_2 are the departure times of the first and second leg and t_1 the first legs travel time, must hold.

Dwell Time Constraints

```
{"first_leg_id": 1,
 "second_leg_id": 2,
 "min_dwell_time": 20}
```

The second constraint type uses the field *min_dwell_time* to state the required amount of seconds between arrival of a train at a station and its departure towards the next station. Hence, a valid timetable must ensure that

$$d_1 + t_1 + w \leq d_2$$

holds, where d_1, d_2 are the departure times of the first and second leg, t_1 the first legs travel time and w the minimal dwell time.

Terminal Turnaround Constraints

```
{"first_leg_id": 206,
 "second_leg_id": 6149,
 "min_turnaround_time": 420}
```

These constraints are structurally identical to the dwell time constraints, such that a valid timetable needs to ensure that

$$d_1 + t_1 + w \leq d_2$$

holds, where d_1, d_2 are the departure times of the first and second leg, t_1 the first legs travel time and w the minimal turnaround time.

Connection Constraints

```
{"first_leg_id": 8846,
 "second_leg_id": 3110,
 "min_connection_time": 180,
 "max_connection_time": 240,
 "connection_type": "arrival_to_departure"}
```

The connection constrains use the fields *min_connection_time* and *max_connection_time* to state the minimal and maximal connection time in seconds as well as the field *connection_type* to state the constraint type. In particular, "arrival_to_departure" states that the connection time is measured between arrival of the first leg and departure of the second leg, whereas "departure_to_departure" means the time between departures. Therefore, a valid timetable must then ensure that

$$\begin{cases} d_1 + t_1 + \underline{c} \leq d_2 \wedge d_1 + t_1 + \overline{c} \geq d_2, & \text{if constraint type is "arrival\_to\_departure"} \\ d_1 + \underline{c} \leq d_2 \wedge d_1 + \overline{c} \geq d_2, & \text{if constraint type is "departure\_to\_departure"} \end{cases}$$

holds, where d_1, d_2 are the departure times of the first and second leg, t_1 the first legs travel time and $\underline{c}, \overline{c}$. are the minimal and maximal connection time respectively.

Recuperation Subnet Constraints

```
{"subnet_id": 1,
 "track_ids": [1, 2, 3, ...]}
```

In our underground application the power network is subdivided into several power subnets and recuperation is only possible between trains in the same subnet. This is not a regular constraint in the sense that it renders some timetables invalid, but rather limits recuperation possibilities and therefore changes what an energy efficient timetable looks like. Full details can be found in the objective description in Section XXX.

1.4 Parsing Routines

The file formats were designed to be easy to read using Python 3 [VRD09] and the pandas data analysis framework [pdt20, WM10]. The following routines can be used to transform the CSV files into easy-to-use dataframes and the JSON file into a constraint dictionary. Note, that these routines already parse the input to integers and floats respectively. In particular, they parse the string representations of departure configurations to a list containing the three integer values for departure time, travel time and power profile ID.

Parsing timetable.csv

```
import pandas

def parse_dc(s):
    m = map(lambda x: [int(i) for i in x.split('_')],
            s.split(' '))
    return list(m)

dtypes = {'leg_id': 'Int64',
          'train_id': 'Int64',
          'track_id': 'Int64',
          'start_station_id': 'Int64',
          'end_station_id': 'Int64'}

converters = {
    'nominal_departure_configuration': lambda x: parse_dc(x)[0],
    'departure_configurations': parse_dc}

df = pandas.read_csv('timetable.csv',
                     dtype=dtypes,
                     converters=converters)
```

Parsing profiles.csv

```
import pandas

dtypes = {'profile_id': 'Int64'}
converters = {
    'power_consumptions': lambda x: [float(f) for f in x.split(' ')]}
df = pandas.read_csv('profiles.csv',
                     dtype=dtypes,
                     converters=converters)
```

Parsing constraints.json

```
import codecs
import json

with codecs.open('constraints.json', 'r') as infile:
    constraint_dict = json.load(infile, encoding='utf-8', parse_int=int)
```

2 Instances

The rail and power network topologies are based on the underground network of Nürnberg, Germany, which consists of three underground lines supplied by a power network consisting of four subnets. Driverless operation of the lines U2 and U3 has been in place since 2008.

The Timetables used for the underground instances are four real timetables from the year 2020. More precisely, two workday timetables, one timetable for Saturdays and finally a timetable for Sundays and holidays. The workday timetables are both used Monday through Friday, but there is one version for regular workdays and another for school holidays, where transportation in the morning and early afternoon are planned differently. For the underground timetables, all constraint types described in Section 1.3 are relevant.

The Power Profiles used in the underground instances were derived from control signals measured during everyday operation as well as typical train characteristics like weight, electrical power of the engines, current flow, voltage and recuperation efficiency. The measurements include current speed, distance traveled and the acceleration control signal.

For a small subset of trains it was possible to directly measure the power consumptions (and recuperations) at the engines. A comparison to our approximated measurements showed that in most cases, our approximation is close to reality. An example comparison is shown in Fig. 1. We can clearly see that the curves representing the calculated (AZG) and the measured (DL350) power consumptions coincide for the most part with small deviations around the 8, 18 and 50 second marks. The cause of the first two deviations is unknown. However, we know that the third and largest deviation, is caused by no other train being available to use the recuperated energy. In this case the energy is not fed back into the power network (and thus measured by the DL350 device), but instead discharged via heat dissipation.

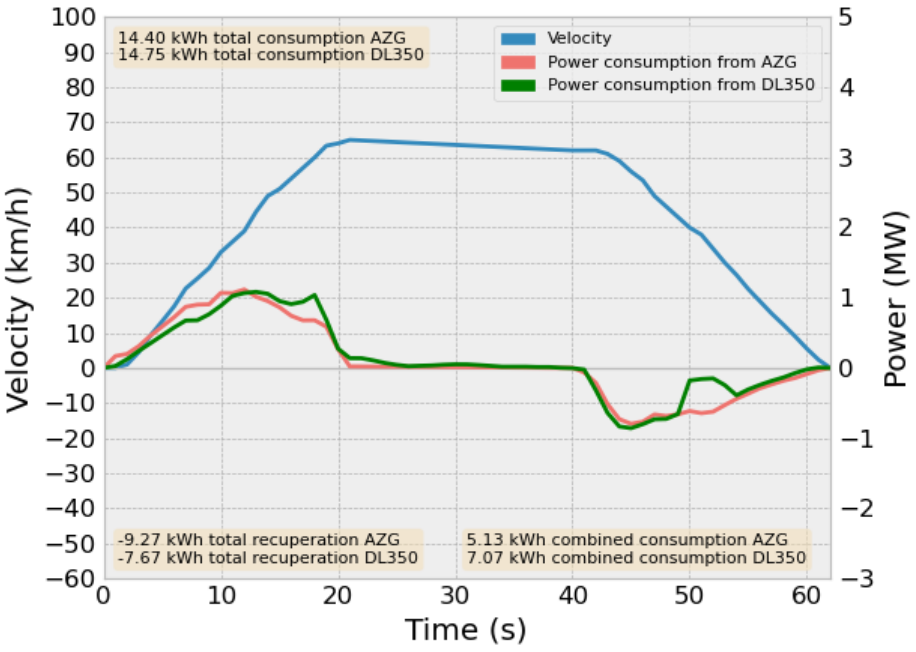

Figure 1: Example comparison for an underground profile measured in 2019 between the stations 'Opernhaus' and 'Plärrer'

Timetable Adjustments For each scheduled departure we allow the departure time to be shifted by up to ± 15 seconds in 5-second intervals. Further, each leg may choose from three or four different travel times, each with their own power profile. The alternate choices represent the three most frequent travel times measured on the respective track, as well as the originally scheduled travel time if it was not among the most frequent ones. For each combination of train type, track and travel time used in the timetable we looked at the set of power profiles for this combination, computed the total power consumption for each profile assuming full usage of the recuperation energy available and chose the one with the median value for the total power consumption.

Instance Characteristics An overview of the underground instances and their characteristics is given in Table 4.

Table 4 Underground instances

			Number of leg dependencies			
Instance	Trains	Legs	Headway	Dwell	Turnaround	Connection
school, line U1	513	11580	11660	11067	487	0
school, lines U2U3	992	12963	12919	11971	954	1
school, all lines	1505	24543	24579	23038	1442	152
school free, line U1	402	10200	10275	9798	381	0
school free, lines U2U3	987	12932	12888	11945	952	1
school free, all lines	1389	23132	23163	21743	1334	152
saturday, line U1	309	7917	7922	7608	295	0
saturday, lines U2U3	771	10188	10144	9417	749	1
saturday, all lines	1080	18105	18066	17025	1044	339
sunday, line U1	240	6240	6303	6000	231	0
sunday, lines U2U3	601	8122	8078	7521	583	0
sunday, all lines	841	14362	14381	13521	814	772

3 Problem

On these instances, the problem to reduce the total power consumption via small timetable adjustments is being solved. We refer the reader to [BGMS18] for full details on preliminary work in a railway setting as well as [BGM20] for full details on work related to the underground network in Nürnberg.

References

[BGM20] Andreas Bärmann, Patrick Gemander, and Maximilian Merkert. The clique problem with multiple-choice constraints under a cycle-free dependency graph. *Discrete Applied Mathematics*, 283:59–77, 2020.

[BGMS18] Andreas Bärmann, Thorsten Gellermann, Maximilian Merkert, and Oskar Schneider. Staircase compatibility and its applications in scheduling and piecewise linearization. *Discrete Optimization*, 29:111–132, 2018.

[pdt20] The pandas development team. Pandas 1.1.3. `https://doi.org/10.5281/zenodo.3509134`, October 2020.

[VRD09] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[WM10] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.