SOFTWARE DOCUMENTATION

1 SOFTWARE DOCUMENTATION OF THE PLANNING TOOL

2 GENERAL

Project: Het COVID-19 Openbaar Vervoer Capaciteitsmodel: een beleidsondersteunend instrument voor de optimalisatie van de coronacapaciteit van het openbaar vervoer

Grant number: 10430042010018 (50-56300-98-1606)

Address: University of Twente Faculty of Engineering Technology Horst Complex (building no. 20), room Z222 De Horst 2 7522LW Enschede The Netherlands

Contact: k.gkiotsalitis@utwente.nl

Version: v1

3 PROJECT SCOPE

Public transport has a vital function within the Dutch society. To minimize the spread of the COVID-19 in public transport, a protocol has been drawn up, including rules for the maximum occupancy of buses and trains. According to the protocol, from 1 June 2020, the capacity will be about 40% for the train and 30-40% for bus, tram and metro, depending on the type of vehicle. This has recently changed because of the second COVID-19 wave.

This project devises **public transport planning tools** to comply with the **pandemic-imposed capacity limit** (which we call "corona-capacity"). The imposed restrictions in public transport have an extreme influence on passenger behavior, turnover and service performance of carriers. The extent to which public transport companies succeed in offering safe and efficient public transport during the COVID-19 crisis has a major influence on the possible spread of the COVID-19 virus, the functioning of society (facilitating, among other things, commuting and business transport) and the costs of public transport. The current public transport planning models focus on optimizing network efficiency: that is, balancing the possible operational costs for the carrier against the quality of service for the traveler. These public transport planning models are not applicable during the COVID-19 crisis, in which we should also ensure sufficient social distancing in public transport.

In this project we develop a **new public transport model that can map the consequences of the corona capacity** and offer **service planning advice**, based on OV-chipkaart data and scenario analysis. The application is aimed at the Keolis public transport network in Regio Twente. Regio Twente has a complex public transport network of bus and train connections (Zutphen-Oldenzaal; Enschede-Zwolle) that serve both urban and non-urban areas.

In this documentation we document the software code for our public transport planning and demand management model that considers the regulations regarding the pandemic-imposed capacity. The model is programmed in an abstract form to increase interoperability (e.g., it can be used by train services or bus services). At the tactical planning level, this model considers the available public transport resources (vehicles) in the study area, the pandemic-imposed capacity limit (which varies over time due to the ever-shifting regulations), the passenger demand per different passenger group, and the demand management possilities per user group. Its aim is to: 1. **Plan the optimal service frequencies** while considering the pandemic-imposed capacity and the associated revenue losses 2. **Incorporate demand management strategies** for specific passenger groups 3. **Calculate the revenue losses** for the public transport operator due to the pandemic-imposed capacity that results in lower vehicle occupancies

4 DOCUMENTATION

4.1 MODEL ASSUMPTIONS

1. The passenger arrival rate is stable within each 1-hour period of the day. That is, passenger arrivals at stations are uniformly distributed within the 1-hour period.
2. The passenger demand in the COVID-19 era is inelastic to the changes made in service frequencies because the considerable demand drop is mainly an effect of travelers preferring private modes of transportation.
3. The bus lines in the network do not share the same corridors.
4. Passengers act rationally and maintain the longest distance possible with each other when they are inside a public transport vehicle.

4.2 MODEL NOTATION

The notation of the model is provided below.

Nomenclature

Sets

$L = \langle 1, ..., l, ... \lvert L \rvert \rangle$	set of public transport lines
$R = \langle 1, ..., r, ... \lvert R \rvert \rangle$	set of user groups that use the bus services (e.g., students, workers)
$S_l = \langle 1, 2, ..., \lvert S_l \rvert \rangle$	ordered set of bus stops served by line $l \in L$ (note that a bi-directional line is considered as a single line that continues its service in the opposite direction)

Indices

l	bus line
r	user group (i.e., students, adults, elderly)
s	bus stop

Parameters

$B_{l,r,sy}$	expected hourly passenger arrival rate of user group $r \in R$ at stop s for passengers whose destination is y and are willing to use line l
T_l	round-trip travel time of line $l \in L$ considering both directions in case of a bi-directional line
k_l	pandemic-imposed capacity limit indicating the maximum allowed passenger load inside each bus operating in line l
W	cost of deploying an extra bus
M_r	fare price per km traveled for user group $r \in R$
F_r	fixed minimum fare when entering the bus for user group $r \in R$
$d_{l,sy}$	traveling distance between bus stops s and y of line $l \in L$
N	number of available buses that can be distributed among all lines
h_l^{min}	minimum headway of line l, where $h_l^{min} > 0$ to ensure that we do not deploy excessively many buses to line l resulting in bus bunching
h_l^{max}	maximum possible headway of line l to ensure that the assigned buses to line l offer a minimum level of service to passengers

Variables

x_l	number of buses assigned to line $l \in L$
h_l	time headway among successive buses of line $l \in L$
$\gamma_{l,s}$	in-vehicle passenger load of each bus serving line l when it departs from stop s
$b_{l,r,sy}$	hourly passenger demand of user group r between stops s and y of line l that can be served by the buses of line l while conforming to the pandemic-imposed capacity limits
$b_{l,sy}$	aggregated hourly passenger demand of all user groups between stops s and y of line l that can be served by the buses of line l while conforming to the pandemic-imposed capacity limits
$\tilde{b}_{l,r,sy}$	hourly passenger demand of user group r between stops s and y of line l that cannot be accommodated by the bus service of line l due to the pandemic-imposed capacity limit k_l

4.3 MODEL FORMULATION

The formulated mathematical model is casted as follows:

$$\min \; W \sum_{l \in L} x_l + \sum_{l \in L} \sum_{r \in R} \sum_{\substack{s \in S_l \\ s \neq |S_l|}} \sum_{\substack{y \in S_l \\ y > s}} (F_r + M_r d_{l,sy}) \tilde{b}_{l,r,sy} \tag{1}$$

$$\text{s.t.} \sum_{l \in L} x_l \leq N \tag{2}$$

$$h_l x_l \geq T_l, \;\; \forall l \in L \tag{3}$$

$$h_l^{min} \leq h_l \leq h_l^{max}, \;\; \forall l \in L \tag{4}$$

$$\gamma_{l,s} \leq k_l, \;\; \forall l \in L, \; \forall s \in S_l \tag{5}$$

$$\gamma_{l,1} = \sum_{y \in S_l} b_{l,1y} h_l, \;\; \forall l \in L \tag{6}$$

$$\gamma_{l,s} = \gamma_{l,s-1} - \sum_{\substack{y \in S_l \\ y < s}} b_{l,ys} h_l + \sum_{\substack{y \in S_l \\ y > s}} b_{l,sy} h_l, \;\; \forall l \in L, \; \forall s \in S_l \setminus \{1\} \tag{7}$$

$$b_{l,r,sy} = B_{l,r,sy} - \tilde{b}_{l,r,sy}, \;\; \forall l \in L, \; \forall r \in R, \; \forall s \in S_l, \; \forall y \in S_l \mid y \geq s \tag{8}$$

$$b_{l,sy} = \sum_{r \in R} b_{l,r,sy}, \;\; \forall l \in L, \; \forall s \in S_l, \; \forall y \in S_l \mid y \geq s \tag{9}$$

$$x_l \in \mathbb{Z}_{\geq 0}, \;\; \forall l \in L \tag{10}$$

$$h_l \in \mathbb{R}_{\geq 0}, \;\; \forall l \in L \tag{11}$$

4.4 SOFTWARE CODE

The code of the software is provided below. Note that the code is written in an abstract form to allow other public transport operators to use it. The data values are read from external files (either .xlsx or .txt). We offer these files for a specific scenario, but public transport operators can exchange the data files when applying this model to their own data.

The software code is developed in *Python 3*. Python is an interpreted, high-level and general-purpose programming language. Python is open source. The optimization part is performed with the optimization solver Gurobi using its Python library *gurobipy*. Gurobi supports linear and quadratic optimization in continuous and integer variables.

4.4.1 Import Python libraries

First, we import the library of the gurobi solver (`gurobipy`) and libraries for reading data from external files (`pandas`, `numpy`).

[1123]:
```
#import libraries
import os, sys
print(sys.executable) # works this time
print(sys.version)
print(sys.version_info)
import gurobipy as gp #import gurobipy library in Python as gp
from gurobipy import GRB
import pandas as pd #import pandas library as pd. It offers data structures and␣
 ↪operations for manipulating numerical tables and time series
```

```
import numpy as np #import numpy library. It adds support for large,␣
 ↪multi-dimensional arrays and matrices
import os #provides functions for interacting with the operating system
import ast #library that processes trees of the Python abstract syntax grammar
```

```
C:\Users\gkiotsalitisk\AppData\Local\Continuum\anaconda3\python.exe
3.6.8 |Anaconda, Inc.| (default, Feb 21 2019, 18:30:04) [MSC v.1916 64 bit
(AMD64)]
sys.version_info(major=3, minor=6, micro=8, releaselevel='final', serial=0)
```

We check the version of the gurobi solver and the gurobipy library. In this implementation, **Gurobi 9.03** is used.

[1124]:
```
print(gp.gurobi.version())
```

```
(9, 0, 3)
```

We initialize our mathematical model in Gurobi

[1125]:
```
#Initialize the Gurobi model
model = gp.Model()
model.Params.OutputFlag = 0
model.Params.LogToConsole = 0
model.setParam("OutputFlag", 0)
```

4.4.2 Set the Parameter values by reading data from files

Read the number of network lines:

[1126]:
```
#read the number of the network lines from a .txt file as an integer number␣
 ↪(dtype='int')
L = np.loadtxt('Data/lines.txt',dtype='int')
L = np.arange(1,L+1)
L = tuple(L)
```

Read the vehicle capacities c_l, $\forall l \in L$ for each public transport line:

[1127]:
```
#read the vehicle capacities for different lines from a .txt file as an integer␣
 ↪number (dtype='int')
c = np.loadtxt('Data/capacities.txt',dtype='int')
c = tuple(c)
c = {i:c[i-1] for i in L}
```

Read the cost, W, of deploying an extra public transport vehicle (e.g,. train or bus, depending on the scenario)

[1128]:
```
#read the cost of deploying an extra public transport vehicle from a .txt file
W = np.loadtxt('Data/W.txt',dtype='float')
```

Read the maximum allowed in-vehicle passenger load, k_l, to conform with the pandemic-imposed capacity regulations. This can differ from line to line and can change over time to comply with the new regulations.

[1129]:
```
#read the in-vehicle passenger load for the vehicles of each line to conform␣
 ↪with the pandemic-imposed capacity limit
k = np.loadtxt('Data/lines_passengerload.txt',dtype='int')
k = tuple(k)
k = {i:k[i-1] for i in L}
```

Read the number of passenger group types, R

[1130]:
```
#read the number of different passenger group types from a .txt file
R = np.loadtxt('Data/R.txt',dtype='int')
```

Read the lost revenue, F_r, for a passenger who is refused service

[1131]:
```
# Read the lost revenue for a passenger who is refused service
file = open("Data/Fr.txt", "r")
contents = file.read(); Fr = ast.literal_eval(contents); file.close()
```

Read the lost revenue per km, M_r, for a passenger who is refused service

[1132]:
```
# Read the lost revenue per km for a passenger who is refused service
file = open("Data/Mr.txt", "r")
contents = file.read(); Mr = ast.literal_eval(contents); file.close()
```

Read the number of available public transport vehicles, N

[1133]:
```
# Read the number of available public transport vehicles at the network level
N = np.loadtxt('Data/N.txt',dtype='int')
```

Read the number of stops served by each line.

[1134]:
```
#read the number of stops served by each line from a .txt file as an integer␣
 ↪number (dtype='int')
S_number = np.loadtxt('Data/line_stops.txt',dtype='int')
```

Read the round-trip travel time in minutes, T_l, for every line $l \in L$.

[1135]:
```
#read round-trip travel times in minutes, including the layover times, for each␣
 ↪line from a .txt file
T = np.loadtxt('Data/line_traveltimes.txt')
T = {i:T[i-1] for i in L}
```

Read the minimum and maximul allowed headway per line, h_l^{min}, h_l^{max}, in minutes.

[1136]:
```
#read the minimum allowed headway per line (in minutes) from a .txt file
file = open("Data/h_l_min.txt", "r")
contents = file.read(); h_l_min = ast.literal_eval(contents); file.close()
```

[1137]:
```
#read the maximum allowed headway per line (in minutes) from a .txt file
file = open("Data/h_l_max.txt", "r")
contents = file.read(); h_l_max = ast.literal_eval(contents); file.close()
```

Read the ordered stops of all lines, S_l.

[1138]:
```
#read the stops of each line from a .txt file in a dictionary form {('line ID',␣
 ↪'ordered stop number'): 'ordered stop number'}
file = open("Data/stations_numberIDs.txt", "r")
contents = file.read(); Sl = ast.literal_eval(contents); file.close()
```

```
#for s in Sl:
    #print(s[0])
```

Read the passenger demand per OD-pair of each line, $B_{l,r,sy}$.

[1139]:
```
#read origin-destination demand between OD-pairs from a .txt file
file = open("Data/Blrsy.txt", "r")
contents = file.read(); Blrsy = ast.literal_eval(contents); file.close()
#print(Blrsy)
```

Read the distance per OD-pair of each line, $d_{l,sy}$.

[1140]:
```
#read the distances between the o-d pairs of each line from a .txt file.
file = open("Data/distance_IDs.txt", "r")
contents = file.read(); d = ast.literal_eval(contents); file.close()
```

4.4.3 Introduce the Model Variables

Set the variable of vehicles per line, x_l. Each line should have at least one vehicle to be operational.

[1141]:
```
#Initialize variable x_l denoting the number of vehicles assigned to each line l
x = model.addVars(L,vtype=gp.GRB.INTEGER, lb=1, name='x')
```

Set the variable of line headways, h_l. The time headways of the lines cannot be negative and are expressed in minutes.

[1142]:
```
#initialize variable h_l denoting the time headway of line l in minutes.
h = model.addVars(L,vtype=gp.GRB.CONTINUOUS, lb=0, name='h')
```

Set veriable $\gamma_{l,s}$ denoting the passenger load of each vehicle of a line l when traveling from stop s to $s+1$. The passenger load cannot be negative and, ideally, it should be lower than or equal to the maximum allowed passenger load to conform with the social distancing regulations.

[1143]:
```
#initialize variable gamma_{l,s} denoting the passenger load of each vehicle␣
 ↪serving line l
gamma = model.addVars(L,Sl,vtype=gp.GRB.CONTINUOUS, lb=0, name='gamma')
```

Set variable $b_{l,r,s,y}$ denoting the hourly passenger demand between stations s and y of line $l \in L$ that can be served by line l for passenger type r while conforming to the pandemic-imposed capacity limit.

[1144]:
```
#Initialize variable b_{l,r,s,y} of the hourly passenger demand between stations␣
 ↪s and y of line l for passenger type r that can be served by line l while␣
 ↪conforming to the pandemic-imposed capacity limit
b = model.addVars(L,R,Sl,Sl,vtype=gp.GRB.CONTINUOUS, lb=0, name='blrsy')
```

Set variable $b_{l,s,y}$ denoting the hourly passenger demand between stations s and y of line $l \in L$ that can be served by line l while conforming to the pandemic-imposed capacity limit.

[1145]:
```
#Initialize variable b_{l,s,y} of the hourly passenger demand between stations s␣
 ↪and y of line l that can be served by line l while conforming to the␣
 ↪pandemic-imposed capacity limit
bsy = model.addVars(L,Sl,Sl,vtype=gp.GRB.CONTINUOUS, lb=0, name='bsy')
```

Initialize variable $\tilde{b}_{l,r,s,y}$ of the hourly passenger demand between stations s and y of line l that cannot be accommodated by the public transport network due to the social distancing requirements.

[1146]:
```
#Initialize variable btilde_{l,r,s,y} of the hourly passenger demand between␣
↪stations s and y of line l for passenger type r that cannot be accommodated by␣
↪the public transport network due to the social distancing requirements
btilde = model.addVars(L,R,Sl,Sl,vtype=gp.GRB.CONTINUOUS, lb=0,␣
↪name='btildelrsy')
```

[1147]:
```
btilde_s = model.addVars(L,Sl,vtype=gp.GRB.CONTINUOUS, lb=0, name='btildels')
```

Initialize a variable that it will compute the overall number of unserved passengers.

[1148]:
```
btilde_overall = model.addVar(vtype=gp.GRB.CONTINUOUS, lb=0,␣
↪name='btildel_overall')
```

4.4.4 Add Constraints

Add constraint $\sum_{l \in L} x_l \leq N$ that does not allow to use more vehicles than the maximum number of available vehicles, N.

[1149]:
```
model.addConstr(sum(x[l] for l in L) <= N)
```

[1149]:
```
<gurobi.Constr *Awaiting Model Update*>
```

Add constraints to ensure that the operating headway is within the limits of the minimum and the maximum possible headway.

[1150]:
```
model.addConstrs(h[l]<=h_l_max[l] for l in L)
model.addConstrs(h[l]>=h_l_min[l] for l in L)
```

[1150]:
```
{1: <gurobi.Constr *Awaiting Model Update*>,
 2: <gurobi.Constr *Awaiting Model Update*>}
```

Add constraints $h_l x_l \geq T_l, \ \forall l \in L$.

[1151]:
```
model.addConstrs( h[l]*x[l] >= T[l] for l in L )
```

[1151]:
```
{1: <gurobi.QConstr Not Yet Added>, 2: <gurobi.QConstr Not Yet Added>}
```

Add constraints $\gamma_{l,s} \leq k_l, \ \forall l \in L, \forall s \in S_l$

[1152]:
```
model.addConstrs(gamma[l,s[0],s[1]] <= k[l] for l in L for s in Sl if s[0]==l)
```

Add constraints $\gamma_{l,1} = \sum_{y \in S_l} b_{l,1,y} h_l, \ \forall l \in L$

[1153]:
```
model.addConstrs(gamma[l,l,1] == 0.01667*h[l]*sum(sum(b[l,r,l,1,y[0],y[1]] for y␣
↪in Sl if y[0]==l and y[1]>1) for r in R) for l in L)
```

[1153]:
```
{1: <gurobi.QConstr Not Yet Added>, 2: <gurobi.QConstr Not Yet Added>}
```

Add constraints $\gamma_{l,s} = \gamma_{l,s-1} - \sum_{y \in S_l \mid y<s} b_{l,ys} h_l + \sum_{y \in S_l \mid y>s} b_{l,sy} h_l \ \ (\forall l \in L, \forall s \in S_l \setminus \{1\})$

[1154]:
```
model.addConstrs(gamma[l,s[0],s[1]] == gamma[l,s[0],s[1]-1] -
                 sum(bsy[l,y[0],y[1],s[0],s[1]]*(1/60)*h[l] for y in Sl if␣
↪y[0]==l and y[1]<s[1]) +
```

```
                sum(bsy[l,s[0],s[1],y[0],y[1]]*(1/60)*h[l] for y in Sl if␣
↪y[0]==l and y[1]>s[1]) for l in L for s in Sl if s[0]==l and s[1]!=1)
```

Add constraint $b_{l,sy} = \sum_{r \in R} b_{l,r,sy}, \forall l \in L,\ \forall s \in S_l,\ \forall y \in S_l \mid y \geq s$.

[1155]:
```
model.addConstrs(bsy[l,y[0],y[1],s[0],s[1]]==sum(b[l,r,y[0],y[1],s[0],s[1]] for␣
↪r in R) for l in L for y in Sl if y[0]==l for s in Sl if s[0]==l and s[1]>y[1])
```

[1156]:
```
model.addConstrs(btilde_s[l,y[0],y[1]]==sum(sum(btilde[l,r,y[0],y[1],s[0],s[1]]␣
↪for s in Sl if s[0]==l and s[1]>y[1])for r in R) for l in L for y in Sl if␣
↪y[0]==l)
```

[1157]:
```
model.addConstr(btilde_overall==sum(sum(btilde_s[l,y[0],y[1]] for y in Sl if␣
↪y[0]==l) for l in L))
```

[1157]:
```
<gurobi.Constr *Awaiting Model Update*>
```

Add constraint $b_{l,r,sy} = B_{l,r,sy} - \tilde{b}_{l,r,sy}\ \ (\forall l \in L, r \in R, s \in S_l, y \in S_l \mid y > s\ )$

[1158]:
```
model.addConstrs(b[l,r,s[0],s[1],y[0],y[1]] ==␣
↪Blrsy[l,r,s[1],y[1]]-btilde[l,r,s[0],s[1],y[0],y[1]]
                for l in L for r in R for s in Sl if s[0]==l for y in Sl if␣
↪y[0]==l and y[1]>s[1])
```

4.4.5 Set the Objective Function

$$z(x,h) := W \sum_{l \in L} x_l + \sum_{l \in L} \sum_{r \in R} \sum_{s \in S_l \setminus \{|S_l|\}} \sum_{y \in S_l \mid y > s} \left(F_r + M_r d_{l,sy}\right) \tilde{b}_{l,r,sy}$$

[1159]:
```
#Declare objective function
obj = W*sum(x[l] for l in L) + sum(sum(sum(sum(␣
↪(Fr[r]+Mr[r]*d[l,s[1],y[1]])*btilde[l,r,s[0],s[1],y[0],y[1]] for y in Sl if␣
↪y[0]==l and y[1]>s[1]) for s in Sl if s[0]==l) for r in R) for l in L)

#Add objective function to model and declare that we solve a minimization problem
model.setObjective(obj,GRB.MINIMIZE)
```

Solve the model and return results.

[1160]:
```
model.params.NonConvex = 2 #allow to handle quadratic equality constraints -␣
↪which are always non-convex
model.optimize()
if model.status == GRB.OPTIMAL: #check if the solver is capable of finding an␣
↪optimal solution
    model.printAttr('X')
    print(model.status,'optimal')
    print('Obj: %g' % model.objVal)
else:
    print(model.status,'not optimal')

#uncomment the following lines to print full results
#for v in model.getVars():
    #if v.x>0:
```

```
        #print('%s %g' % (v.varName, v.x))
```

```
2 optimal
Obj: 79.9225
```