PUBLIC TRANSPORT OPTIMIZATION

PHASE-3

I. INTRODUCTION:

So far, not much attention has been given to the problem of improving public transportation networks. In many cities these networks have been built sequentially and do not fit to the needs of the users any more. The results are long travel times and an unnecessarily high number of people who have to transfer. Compared to other investments for improving the service level of public transportation systems, the costs of rerouting the public vehicles are low and can, yet, highly improve the performance of the system.

II. PROGRAMMING:

```
import
random
class
Net:
""" Net as the graph model """
def
__init__(self):
# network model time
self.time =0
# duration of the network simulation [min]
self.duration =0
# network geography
self.nodes = []self.links = []self.lines = []
# transport demand
self.demand = []
# resulting characteristics
self.total_wait_time =0self.sum_vehicles_time
=0self.num_serviced_passengers =0
def
contains_node(self, node_code):
""" Determines if the network contains a node with the specified
code """
for
n
in
self.nodes:
if
n.code == node_code:
```

```
return
True
return
False
def
get_node(self, code):
""" Returns the first found node with the specified code """
for
n
in
self.nodes:
if
n.code == code:
return
n
return
None
def
contains_link(self, out_node, in_node):
""" Checks if the net contains a link """
for
l
in
self.links:
if
l.out_node
is
out_node
and
l.in_node
is
in_node:
return
True
return
False
def
get_link(self, out_node, in_node):
""" Returns the first found link with the specified out and in nodes
"""
for
l
in
self.links:
if
l.out_node
is
```

```
out_node
and
l.in_node
is
in_node:
return
l
return
None
def
add_link(self, out_code, in_code, weight=0, directed=False):
""" Adds a link with the specified characteristics """
if
self.contains_node(out_code):
# out-node is already in the net
out_node =self.get_node(out_code)
if
self.contains_node(in_code):
# in-node is already in the net
in_node =self.get_node(in_code)
if
self.contains_link(out_node, in_node):
# out-node and in-node are already linked: change the link weight
self.get_link(out_node, in_node).weight = weight
else
:
# there is no such a link in the net: add a new one
new_link = link.Link(out_node, in_node,
weight)out_node.out_links.append(new_link)in_node.in_links.append(new
_link)self.links.append(new_link)
else
:
# the net contains the specified out-node, but there is no in-node#
with the specified code
in_node = node.Node(in_code)new_link = link.Link(out_node, in_node,
weight)out_node.out_links.append(new_link)in_node.in_links.append(new
_link)self.nodes.append(in_node)self.links.append(new_link)



else
:
# the net does not contain the specified out-node
out_node = node.Node(out_code)
if
self.contains_node(in_code):
# in-node is already in the net
```

```
in_node =self.get_node(in_code)
else
:
# there are no in-node and out-node with the specified codes
in_node = node.Node(in_code)
# create new link
new_link = link.Link(out_node, in_node,
weight)out_node.out_links.append(new_link)in_node.in_links.append(new
_link)self.nodes.append(in_node)self.nodes.append(out_node)self.links
.append(new_link)
# add the reverse link
if not
directed:self.add_link(in_code, out_code, weight,True)
# sort the nodes (is useful for calculating the short distances
matrix)# self.nodes.sort()
def
generate(self, nodes_num, links_num, s_weight):
"""nodes_num - number of nodes in the netlinks_num - number of links
in the nets_weight - stochastic variable of the links weight"""#
limit lower bound for the number of nodes
if
nodes_num <2:nodes_num =2
# limit lower bound for the number of links
if
links_num <1:links_num =1
# limit upper bound for the number of links
if
links_num > nodes_num*(nodes_num -1):links_num = nodes_num *
(nodes_num -1)
# define a set of the network nodes
for
i
in
range(1, nodes_num +1):self.nodes.append(node.Node(i))
# generate random set of the network links# ! some nodes in the
network could not be linked
l_num =0
# counter for the links number
 while
l_num < links_num:out_node = random.choice(self.nodes)in_node =
random.choice(self.nodes)
 while
out_node
is
in_node:in_node = random.choice(self.nodes)
if not
```

```
self.contains_link(out_node, in_node):self.add_link(out_node.code,
in_node.code, s_weight.get_value(),True)l_num +=1
def
gen_lines(self, lines_num, s_stops_num):
"""Generates specified number of lines which contain the random
number of stopslines_num - number of lines, s_stop_num - stochastic
variable of the stops number"""# line could contain more than 1 stop
in the same node
for
idx_line
in
range(lines_num):stops_num =int(s_stops_num.get_value())
if
stops_num <2:stops_num =2stops = []stop = random.choice(self.nodes)
# begin stop
 while
len(stop.out_links) ==0:stop =
random.choice(self.nodes)stops.append(stop.code)
for
idx_stop
in
range(stops_num -1):next_stop =
(random.choice(stop.out_links)).in_node
 while
len(next_stop.out_links) ==0
or
next_stop.code
in
stops:next_stop = (random.choice(stop.out_links)).in_nodestop =
next_stopstops.append(stop.code)self.lines.append(line.Line(self,
stops))



def
gen_demand(self, duration):
"""Generates demand for trips in the networkduration - duration of
the simulation period, hrs"""
self.demand = []
for
nd
in
self.nodes:time =0
 while
time <= duration:interval =round(nd.s_interval.get_value(),1)time +=
interval
# generating a new passenger
```

```
new_passenger = passenger.Passenger()new_passenger.m_appearance =
timenew_passenger.origin_node = nd
# defining the destination node - random choice rule# (can't be the
same as origin node)
destination_node = random.choice(self.nodes)
 while
destination_node == nd:destination_node =
random.choice(self.nodes)new_passenger.destination_node =
destination_node
# adding the passenger to the origin node collection
nd.pass_out.append(new_passenger)self.demand.append(new_passenger)
def
simulate(self, duration=8*60, time_step=1):
""" Simulation of the transport network """
self.duration = duration
# demand generation
self.gen_demand(self.duration)
for
ln
in
self.lines:
# define schedules
ln.define_schedule()
for
v
in
ln.vehicles:
# put zero values to the vehicle characteristics
v.servicing = {}v.passengers = []v.serviced_passengers = []
# correct the simulation duration
if
self.duration < v.schedule[-1][0]:self.duration = v.schedule[-1][0]
# run the lines simulation
self.time =0
 while
self.time <=self.duration:
for
ln
in
self.lines:ln.run()self.time += time_step
# printing out simulation results
self.total_wait_time =0self.num_serviced_passengers
=0self.sum_vehicles_time =0
for
ln
in
self.lines:
```

```
for
v
in
ln.vehicles:self.sum_vehicles_time += v.schedule[-1][0] -
v.schedule[0][0]self.num_serviced_passengers
+=len(v.serviced_passengers)
# calculate sum of waiting time
for
ps
in
v.serviced_passengers:self.total_wait_time += ps.wait_time
# estimate total wait time of unserved passengers# (under condition
that they wait till the end of simulation)
up_wait_time =0upn =0
for
ps
in
self.demand:
if
ps.used_vehicle
is
None:up_wait_time +=self.time - ps.m_appearanceupn
+=1self.total_wait_time += up_wait_time
```

III. CONCLUSION:

Public transportation is more than just a means of getting from one place to another. It is a tool for urban development, social equity, and environmental sustainability. Despite the challenges it faces, with proper planning, sufficient funding, and the integration of advanced technologies, public transportation can continue to serve as a vital component of urban life, shaping our cities for the better.