PUBLIC TRANSPORT OPTIMIZATION

Problem Definition and Design Thinking

Problem Definition

TRANSPORTATION OPTIMIZATION is the process of analyzing shipments, rates and. constraints to produce realistic load plans that reduce overall freight spend and gain efficiencies across entire transportation networks. Once actual freight rates, approved carriers, specified capacities, transit times and other available data are entered into the optimization platform, the software generates a realistic, executable load plan for the shipment. With this information in hand, shippers, 3PLs, and consultants can make better decisions, save valuable resources, improve revenues and cut costs.

Design Thinking

1. Project Objectives:

a) **Real time transit information:** Real-time information, broadly defined, means any information available to transit providers or customers about the current status of vehicles, including approximate locations and predictive arrival times.

b) **Arrival time prediction:** a tool that estimates the time it will take for a person or object to arrive at a particular destination. It uses data such as current location, traffic conditions, and historical travel patterns to make this prediction.

c) **Ridership monitoring:** Real-time public transit ridership flow and origin– destination (O–D) information is essential for improving transit service quality and optimizing transit networks in smart cities. The effectiveness and accuracy of the traditional surveybased methods and smart card data-driven methods for O–D information inference have multiple disadvantages in terms of biased results, high latency, insufficient sample size, and the high cost of time and energy.

d) **Enhanced public transportation services:** Such solutions hold great potential to improve the efficiency of public transit systems through initiatives such as smart-ticketing, one nation-one card, security surveillance, fleet management, traffic management and real-time passenger information among others.

2. IOT Sensor Design:

a) **Identify Locations:**Acess the public places (public roads)where public transportation monitoring is necessary and prioritize them based on their usage .

b) : **Choose appropriate sensors:** Select IoT sensors capable of measuring public transportation accurately and reliably in the identified locations. Consider factors such as sensor efficiency, durability, cost-effectiveness, and ease of maintenance.

c) **Determine deployment strategy:** Plan the installation and placement of IoT sensors in the identified locations to ensure optimal data collection and coverage. Consider factors such as proximity to ease of access for
maintenance, and minimizing tampering and theft.

3. Real Time Transmit Information Platforms:

a) **Design a user interface**: Develop a user-friendly mobile app interface or website that displays real-time water consumption data in an easily understandable format.

b) **Intractive Features:** Include interactive features such as charts, graphs, and notifications to engage users and raise awareness about water conservation.

c) **Data privacy and security:** Implement robust security measures to protect the collected data and ensure user privacy.

4. Integration Approach:

a) **Communication protocols:** Determine the communication protocols to be used for transmitting data from the IoT sensors to the data-sharing platform. Consider factors such as reliability, data transfer rate, and compatibility with existing infrastructure.

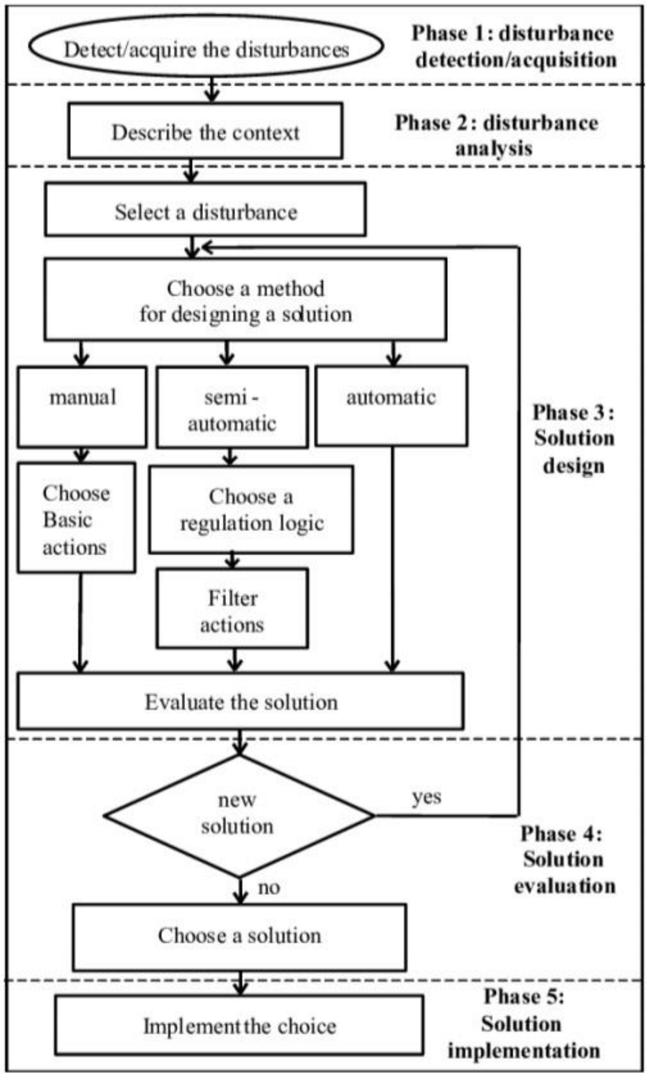
b) **Data-sharing platform:** Develop a centralized data-sharing platform where the collected water consumption data can be stored and accessed by the mobile app or website.

c) **Integration withIoT technology and Python:** Use IoT technology and Python programming to integrate the IoT sensors, data-sharing platform, and user interface. This will involve developing APIs, setting up data transfer protocols, and implementing data analytics and visualization tools.

PRINCIPLE:

This document is based on a set of lecture notes prepared in 2007-2010 for a University of California, Berkeley graduate course, Public Transportation Systems, a course targeted to first year graduate students with diverse academic backgrounds. Systems are examined in order of increased complexity so that generic insights evident in simple systems can be put to use as knowledge building blocks for the study of more complex systems. The document is organized in eight modules: five on planning (general, shuttle systems, corridors, two dimensional systems, and unconventional transit); two on management (vehicles and employees); and one on operations (how to stay on schedule).

BLOCK DIAGRAM:


Detect/acquire the disturbances
Phase 1: disturbance detection/acquisition
Describe the context
Phase 2: disturbance analysis
Select a disturbance
Choose a method for designing a solution
manual
semi - automatic
automatic
Phase 3: Solution design
Choose Basic actions
Choose a regulation logic
Filter actions
Evaluate the solution
new solution
yes
no
Phase 4: Solution evaluation
Choose a solution
Implement the choice
Phase 5: Solution implementation

COMPONENT SELECTION:

1) Freight data

2) land use information
3) economic development
4) environmental data
5) historic preservation

6) recreation
7) tourism data
8) natural resources

PROCEDURE(WORKING PRINCIPLE):

➢ ensure convenient pedestrian and bicycle access to
railway stations and public transport interchanges

1. Provide continuous, direct pedestrian and bicycle access routes from the surrounding neighbourhood to railway stations and public transport interchanges.
2. Provide a continuous active frontage along pedestrian approach paths to railway stations and public transport interchanges.
3. Arrange pedestrian approach paths with clear sightlines to and from railway
station buildings, and to and from public transport interchanges.
4. Where a bus or tram interchange is co-located with a railway station, connect
them with a direct, sheltered pedestrian path.

➢ ensure safety and amenity around railway stations and
public transport interchanges

1. Locate active public spaces and secondary uses adjacent to railway stations
and public transport interchanges.
2. Locate public transport waiting areas, particularly pick-up and drop-off areas,
and taxi ranks where they are clearly visible from the pedestrian approach paths and nearby buildings.
3. Where railway stations are co-located with a bus interchange, arrange waiting
areas with clear views to approaching buses.

➢ ensure comfortable and serviceable railway stations and
public transport interchanges

1. Provide weather protection, comfortable seating and public amenities, such as
waste bins and drinking fountains.
2. Locate way finding signage at logical and visible points along approach paths
to and within the railway station or public transport interchange.
3. Locate real–time travel information where it can be seen by waiting
passengers in all light conditions.
4. Provide both casual and secure bicycle storage near the railway station or
public transport interchange.

.

➢ ensure the railway station or public transport
interchange contributes to a sense of place and local
character
1. Develop a palette of materials, furnishings and plantings for public space
within the railway station precinct or public transport interchange that is
consistent with the preferred palette of the surrounding area.
➢ effectively maintain public transport environs
1. Establish a place management agreement that identifies management and
maintenance responsibilities and processes.
METHODOLOGY

The system is placed in every bus the system built around Arduino nano. When system turns on, the LCD and
GSM Module is initialized. The GSM module is used to accessed wireless internet in the bus. GPS is used to get
the latitude and longitude that is exact position of bus. Arduino reads the position continuously. Ignition is
checked. If ignition is on then the GPS co-ordinates that is latitude and longitude are read by Arduino nano.

Arduino nano sends GPS co-ordinates, passenger count to webpage. IR sensors are placed at the entry and exit doors of bus to count the number of passengers in bus. If ignition is off then output of IR sensors at the entry and at exit is sensed by Arduino nano. If the IR sensor at the entry, senses passenger then Arduino nano increments passenger count by 1. If the IR sensor at the exit, senses passenger then Arduino nano decrements passenger count by 1. Arduino sends all these details of bus continuously to a webpage through internet connection the web page also contains the details of bus like bus number, bus route, bus timing which is manually uploaded by authority. Public access all these details of bus on visiting to that particular web page.

For simplicity to access webpage, the QR code of link of webpage is attached to every bus stop.

III. LITERATURE SURVEY

In past works given in SeokJuLee[1], they have actualized transport vehicle tracking for UCSI University, kuala Lumpur, Malaysia. It is developed for settled course, giving the candidates with status of bus after determined time period utilizing LED panel smart phone application. Technique used is Ardunio microcontroller Atmega328 based Arduino UNOR3 microcontroller. GPS, GSM or GPRS module a similar controller is used.

Program to control them is composed in C programming language, compiled and saved in microcontroller's flash memory. The testing results in this paper give; testing in-vehicle module, testing web server and database, testing smart phone app.

In PengfeiZhou[2], foreseeing transport entry time with cell phones is given. Innovation utilized is participatory detecting of users. This model framework with various sorts of Android based cell phones and thoroughly

explores different avenues regarding the NTU grounds carry transports and in addition Singapore transports
over a 7-week time span, then taken after by London in 4-weeks. The proposed framework is arrangement is all
the more for the most part accessible and is vitality agreeable. The assessment comes about recommend that
the proposed framework accomplishes extraordinary expectation exactness contrasted and those operator
initiated and GPS based solutions. The model framework predicts transport entry time with average tolerance
of 80 sec.
In Maman Abdurohman[3], versatile tracking framework is utilized to monitor vehicles position and in
uncommon cases there are much helpful data can be studied, for example, speed, cabin temperature and no. of
passenger. This monitoring procedure is done utilizing GPS module, and sending the information to a server
through GSM modem. It is proposed machine to machine communication from which Open Machine Type
Communication as correspondence platform for collecting and preparing area information. The google outline
shows the area. The Open MTC platform that is produced by Fraunhofer FOKUS in view of ETSI M2M Rel.1
specification.

IV. MODELING AND DESIGING

Waterfall model control the system. Schedule is set for deadlines all stages of development and projects can
process by using the development process model stage one by one.
Requirement Gathering and Analysis: -
The passible requirement of system is developed in phase. In our project there are two types of requirements
namely hardware and software requirements.

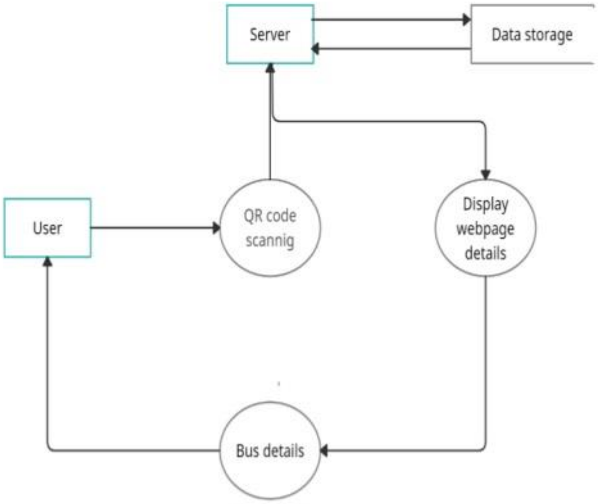
 Firstly, the connection between GSM module and webpage should be established. This connectivity will take place through internet. To establish this connectivity, required commands are sent to GSM module by the controller. Hence, TCP connection with UBIDOTS server is linked.

 Once the proposed required of our system TCP connection is linked, it is analyzed and used to send the data to webpage.

System Design: -

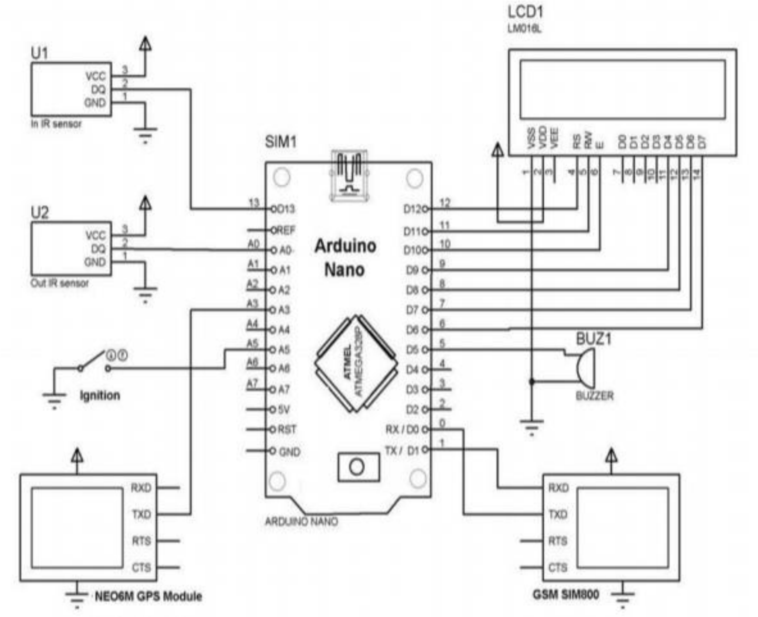
 The specifications from the first phase are studied in this phase and the system design is prepared. Specifically, DFD (Data Flow Diagram), architectural design, block design and use case design of our system is prepared which helps to specify hardware and software requirement and helps in defining the overall system infrastructure.

 DFD (Data Flow Diagram): - Signifies the exact flow of data in the system.



This is the Data flow diagram of our project, first user scans the QR code after that request sends to the server access the data from cloud then the server displays the whole details of bus, that is route of the bus, real time location of bus, bus number and exact count the number of passenger in bus.

 Architecture design: - It makes the architecture of system understandable.

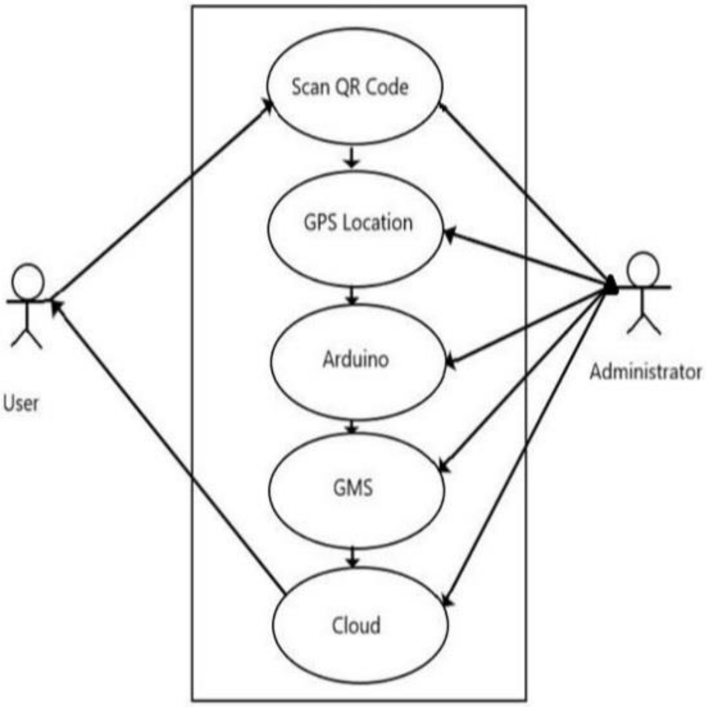


This system is placed on every bus. The system built around Arduino nano. GPS module, GSM module IR sensor and LCD are connected to the Arduino nano. The GSM is used to accesses wireless internet in the bus. GPS is used to get the latitude and longitude that is position of bus. Arduino read the position continuously. IR sensors are placed at the entry and exit doors of bus to count the total number of passengers in the bus.

The ignition of the bus is connected to Arduino to get the status of bus that is whether bus is on or off. Arduino sends all these details of bus continuously to a web page though internet connection. The web page also contains the details of bus like bus number, bus routes, bus timings which is manually uploaded by authority.

Public access all these details of bus on visiting to that particular web page. For simplicity to access web page, the QR code of link of web page is attached to every bus stop.

 Use case design: -User end functionalities is shown in our system.



We proposing QR code . User can scan QR code, open the website, after scan the QR code. User get the bus details like bus route, Changing count of passenger present in bus.

Implementation:

 Inputs from the system design, the system is first develop in small program called units such as reading GPS data, Establishment of connectivity using GSM module, sending data to webpage are integrated in next phase. the unit is develop as per its significant functionality.

Testing:

 All the units developed in the implementation phase are integrated into a system after confirming functionalities of each unit is in a proper manner. After integration the entire system is tested before it is deployed. The reading of GPS data that is latitude and longitude readings, connectivity to internet through GSM module, IR sensors at the in and out doors are tested.

Deployment:

 Once the system has been tested, the full functional system is deployed in the customer environment. User will use it to make sure either the system is correctly functioning or not.

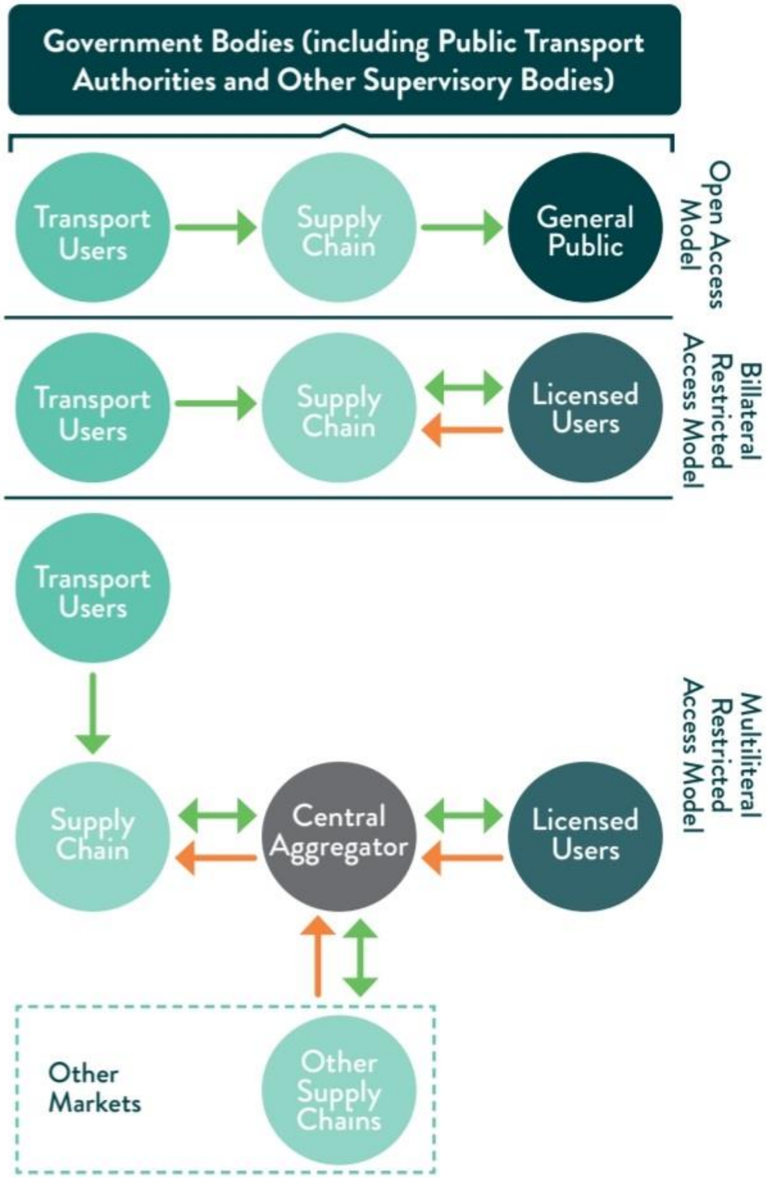
Maintenance:
 There are some issues which come up in the user environment such as in connectivity. Also, to enhance the
product some better versions of the system will be released.

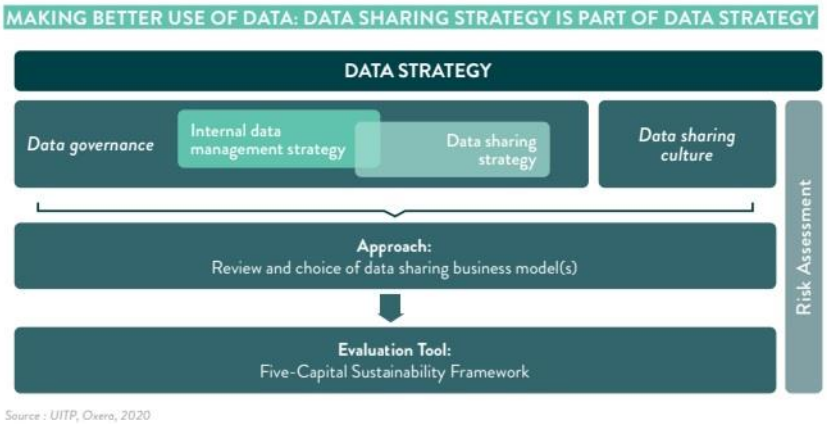
DATA SHARING BUSINESS MODELS

Having understood the why question, the next questions are with who the data should be shared, and how. The data sharing ecosystem requires sustainable business models to enable long-term data exchange. This study has reviewed three frequently used data sharing models – the Open Access, the Bilateral Restricted Access, and the Multilateral Restricted Access.

The three business models can be selected or combined in different ways, depending on the data and the stakeholders involved. The study has highlighted the application of these models across several cities, including London, Taipei and Singapore.

Data Sharing Business Models and their Stakeholders' Relationships





MAKING BETTER USE OF DATA: DATA SHARING STRATEGY IS PART OF DATA STRATEGY

DATA STRATEGY
Data governance
Internal data management strategy
Data sharing strategy
Data sharing culture
Approach:
Review and choice of data sharing business model(s)
Evaluation Tool:
Five-Capital Sustainability Framework
Risk Assessment

Source : UITP, Oxera, 2020

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
```

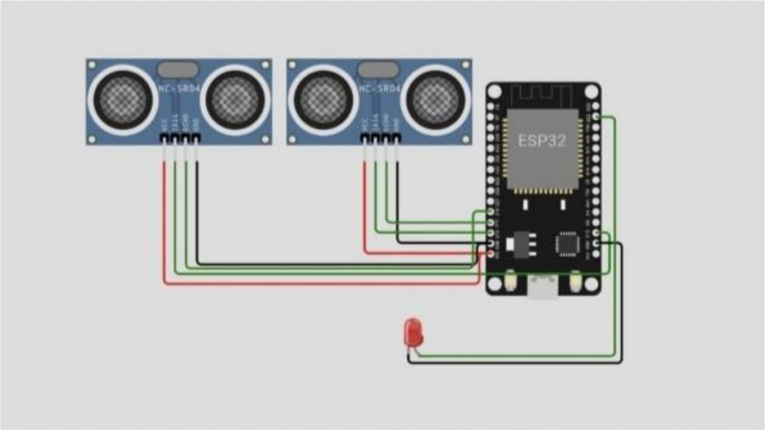
```
# adding the passenger to the origin node collection
nd.pass_out.append(new_passenger)self.demand.append(new_passenger)
def
simulate(self, duration=8*60, time_step=1):
""" Simulation of the transport network """
self.duration = duration
# demand generation
self.gen_demand(self.duration)
for
ln
in
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
```

IOT BASED PUBLIC TRANSPORT OPTIMIZATION USING ARDUINO:

IoT(Internet of Things) can assist in integration of communication, control and information processing across various transportation systems. In public transportation, there is lack of real time information. The public transit usage can be improved if real time information of the vehicle such as the seating availability, current location and time taken to reach the destination are provided with easier access. It would also be helpful for the passengers to find alternate choices depending on their circumstances. As excessive long waiting often discourage the travelers and makes them reluctant to take buses. A smart information system has been proposed where the travelers get prior information about current location, next location of bus and crowd level inside the bus. This system is designed using ARDUINO UNO, IR Sensor and GPS Module. An Intelligent Transport System (ITS) removes the barriers for public transport usage and creates the positive impact about the bus journey.

Circuit diagram:



COMPONENTS

- HC SR04
- ESP32

- Connecting wires
- LED

CODE:

```
{
"cells": [
{
"cell_type": "code",
"execution_count": 15,
"metadata": {
"id": "R4ag3SOp2uAM"
},
"outputs": [],
"source": [
"import numpy as np, random, operator, pandas as pd, matplotlib.pyplot as plt"
]
},
{
"cell_type": "code",
"source": [
"city_name_data = pd.read_csv('sgb128_name.csv',header=None)\n",
"city_dist_data = pd.read_csv('sgb128_dist.csv',header=None)\n",
"city_weight_data = pd.read_csv('sgb128_weight.csv',header=None)\n",
"# city_length = 25"
],
"metadata": {
"id": "YYzhm4cjUVk4"

},
"execution_count": 16,
"outputs": []
},
{
```

```
"cell_type": "code",
"execution_count": 17,
"metadata": {
"id": "f40wgYcI2yTr"
},
"outputs": [],
"source": [
"class City:\n",
" def __init__(self,name,population):\n",
" self.name=name\n",
" self.population=population\n",
" \n",
" def distance(self, city):\n",
" distance=city_dist_data.iloc[self.name,city.name]\n",
" return distance\n",
" \n",
" def __repr__(self):\n",
" return \"\\\"\"+str(city_name_data.iloc[self.name,0])
+\"[\"+str(self.population)+\"]\"+\"\\\"\""
]
},
{
```

```
"cell_type": "code",


"execution_count": 18,
"metadata": {
"id": "OizIyOa0265B"
},
"outputs": [],
"source": [
"class Fitness:\n",
" def __init__(self, route):\n",
" self.route = route\n",
" self.distance = 0\n",
" self.fitness= 0.0\n",
" self.total_population= 0\n",
" \n",
" def routeDistance(self):\n",
" pathDistance = 0\n",
" for i in range(0, len(self.route)):\n",
" fromCity = self.route[i]\n",
" toCity = None\n",
" if i + 1 < len(self.route):\n",
" toCity = self.route[i + 1]\n",
" else:\n",
" break\n",
" pathDistance += fromCity.distance(toCity)\n",
" self.distance = pathDistance\n",
" return self.distance\n",
" \n",
" def routePopulation(self):\n",

" path_population = 0\n",
" for i in range(0, len(self.route)):\n",
" City = self.route[i]\n",
" path_population += City.population\n",
" self.total_population = path_population\n",
" return self.total_population\n",
" \n",
" def routeFitness(self):\n",
" if self.fitness == 0:\n",
" self.fitness = self.routePopulation() / float(self.routeDistance()) \n",
" return self.fitness"
]
},
{
```

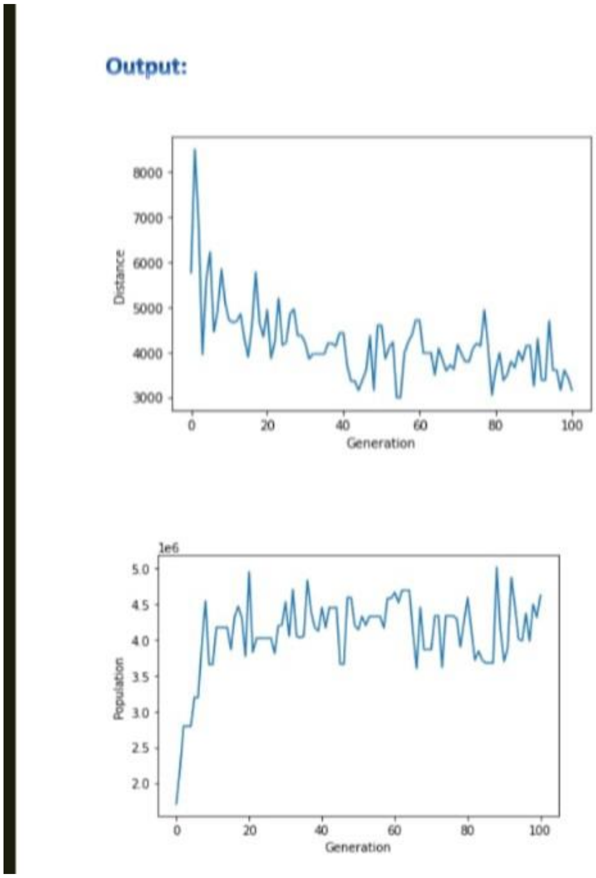
```
"cell_type": "code",
"execution_count": 19,
"metadata": {
"id": "K4uLoioc3FYE"
},
"outputs": [],
"source": [
"def createRoute(cityList):\n",
" route = random.sample(cityList, 10)\n",
" return route\n",
" \n",
"def initialPopulation(popSize, cityList):\n",
" population = []\n",

" for i in range(0, popSize):\n",
" population.append(createRoute(cityList))\n",
" return population"
]
},
{
"cell_type": "code",
"execution_count": 20,
"metadata": {
"id": "N2F1Ei5O3KCR"
},
"outputs": [],
"source": [
"def rankRoutes(population):\n",
" fitnessResults = {}\n",
" for i in range(0,len(population)):\n",
" fitnessResults[i] = Fitness(population[i]).routeFitness()\n",
" \n",
" return sorted(fitnessResults.items(), key = operator.itemgetter(1), reverse = True)"
]
},
{
"cell_type": "code",
"execution_count": 21,
"metadata": {
"id": "GvHkFsLx3Nzo"
},

"outputs": [],
"source": [
"def selection(popRanked, eliteSize):\n",
" selectionResults = []\n",
" df = pd.DataFrame(np.array(popRanked), columns=[\"Index\",\"Fitness\"])\n",
" df['cum_sum'] = df.Fitness.cumsum()\n",
" df['cum_perc'] = 100*df.cum_sum/df.Fitness.sum()\n",
" \n",
" for i in range(0, eliteSize):\n",
" selectionResults.append(popRanked[i][0])\n",
" for i in range(0, len(popRanked) - eliteSize):\n",
" pick = 100*random.random()\n",
" for i in range(0, len(popRanked)):\n",
" if pick <= df.iat[i,3]:\n",
" selectionResults.append(popRanked[i][0])\n",
" break\n",
" return selectionResults"
]
},
{
"cell_type": "code",
"execution_count": 22,
"metadata": {
"id": "DIPlzfSc3RKe"
```

```
},
"outputs": [],
"source": [

"def matingPool(population, selectionResults):\n",
" matingpool = []\n",
" for i in range(0, len(selectionResults)):\n",
" index = selectionResults[i]\n",
" matingpool.append(population[index])\n",
" return matingpool"
]
},
{
"cell_type": "code",
"execution_count": 23,
"metadata": {
"id": "JMVLhepw3bHn"
},
"outputs": [],
"source": [
"def breed(parent1, parent2):\n",
" child = []\n",
" childP1 = []\n",
" childP2 = []\n",
" \n",
" geneA = int(random.random() * len(parent1))\n",
" geneB = int(random.random() * len(parent1))\n",
" \n",
" startGene = min(geneA, geneB)\n",
" endGene = max(geneA, geneB)\n",
"\n",

" for i in range(startGene, endGene):\n",
" childP1.append(parent1[i])\n",
" for item in parent2:\n",
" if item not in childP1 and len(childP1)<10:\n",
" childP1.append(item)\n",
" child = childP1\n",
" return child\n",
"\n",
"def breedPopulation(matingpool, eliteSize):\n",
" children = []\n",
" length = len(matingpool) - eliteSize\n",
" pool = random.sample(matingpool, len(matingpool))\n",
"\n",
" for i in range(0,eliteSize):\n",
" children.append(matingpool[i])\n",
" \n",
" for i in range(0, length):\n",
" child = breed(pool[i], pool[len(matingpool)-i-1])\n",
" children.append(child)\n",
" return children"
]
},
{
"cell_type": "code",
"execution_count": 24,
"metadata": {
"id": "Wq1SM1Hg3c7c"

},
"outputs": [],
"source": [
"def mutate(individual, mutationRate):\n",
" for swapped in range(len(individual)):\n",
" if(random.random() < mutationRate):\n",
" swapWith = int(random.random() * len(individual))\n",

" \n",

" city1 = individual[swapped]\n",

" city2 = individual[swapWith]\n",

" \n",

" individual[swapped] = city2\n",

" individual[swapWith] = city1\n",

" return individual\n",

" \n",

"def mutatePopulation(population, mutationRate):\n",

" mutatedPop = []\n",

" \n",

" for ind in range(0, len(population)):\n",

" mutatedInd = mutate(population[ind], mutationRate)\n",

" mutatedPop.append(mutatedInd)\n",

" return mutatedPop"

]

},

{

"cell_type": "code",

"execution_count": 25,
"metadata": {
"id": "OzUaK9kK3kdo"
},
"outputs": [],
"source": [
"def nextGeneration(currentGen, eliteSize, mutationRate):\n",
" popRanked = rankRoutes(currentGen)\n",
" selectionResults = selection(popRanked, eliteSize)\n",
" matingpool = matingPool(currentGen, selectionResults)\n",
" children = breedPopulation(matingpool, eliteSize)\n",
" nextGeneration = mutatePopulation(children, mutationRate)\n",
" return nextGeneration"
]
},
{
"cell_type": "code",
"execution_count": 26,
"metadata": {
"id": "Zg77N1J13nn2"
},
"outputs": [],
"source": [
"def geneticAlgorithm(population, popSize, eliteSize, mutationRate,
generations):\n",
" pop = initialPopulation(popSize, population)\n",
" print(\"Initial distance: \" +
str(Fitness(pop[rankRoutes(pop)[0][0]]).routeDistance()))\n",
" print(\"Initial population: \" +
str(Fitness(pop[rankRoutes(pop)[0][0]]).routePopulation()))\n",
" \n",

" for i in range(0, generations):\n",
" pop = nextGeneration(pop, eliteSize, mutationRate)\n",
"\n",
" total_distance=0\n",
" total_population=0\n",
" for i in range (0,10):\n",
" bestRouteIndex = rankRoutes(pop)[i][0]\n",
" bestRoute = pop[bestRouteIndex]\n",
" fitness = Fitness(bestRoute)\n",
" fitness.routeFitness()\n",
" print((i+1),\"route:\", bestRoute)\n",
" total_distance += fitness.distance\n",
" print(\"distance: \" + str(fitness.distance))\n",
" total_population += fitness.total_population\n",
" print(\"Population: \" + str(fitness.total_population))\n",
"\n",
" print(\"Total distance= \"+ str(total_distance))\n",
" print(\"Total population= \" + str(total_population))\n",
" return bestRoute"
]
},
{
"cell_type": "code",
"execution_count": 27,
"metadata": {
"id": "xGtP1FF-3qYh"
},

"outputs": [],
"source": [
"# cityList = []\n",
"# len(cities.City)\n",
"# for i in range(0,len(cities.City)):\n",
"cityList = []\n",
"for i in range(0,len(city_name_data)):\n",
" cityList.append(City(name=i,population=city_weight_data.iloc[i,0]))\n"
```

```
]
},
{
"cell_type": "code",
"execution_count": 28,
"metadata": {
"colab": {
"base_uri": "https://localhost:8080/"
},
"id": "ycdq0fIW3sqN",
"outputId": "e6d7d2da-5aed-4d5f-f1c7-eb0ae9e82d88"
},
"outputs": [
{
"output_type": "stream",
"name": "stdout",
"text": [
"Initial distance: 8587\n",
"Initial population: 3114379\n",


"1 route: [\"Saint Louis, MO[453085]\", \"San Jose, CA[629546]\",
\"Worcester,
MA[161799]\", \"San Diego, CA[875538]\", \"Ravenna, OH[11987]\",
\"Winnipeg, MB[564473]\",

\"Rochester, NY[241741]\", \"Toledo, OH[354635]\", \"Rockford,
IL[139712]\", \"San Francisco,

CA[678974]\"]\n",

"distance: 3693\n",

"Population: 4111490\n",

"2 route: [\"Washington, DC[638432]\", \"Winnipeg, MB[564473]\",
\"Rochester,

NY[241741]\", \"Saint Joseph, MO[76691]\", \"San Antonio, TX[786023]\",
\"Rockford, IL[139712]\",

\"Toledo, OH[354635]\", \"San Jose, CA[629546]\", \"Worcester,
MA[161799]\", \"San Diego,

CA[875538]\"]\n",

"distance: 4098\n",

"Population: 4468590\n",

"3 route: [\"Toronto, ON[599217]\", \"Rockford, IL[139712]\", \"San
Antonio, TX[786023]\",

\"Saint Louis, MO[453085]\", \"Worcester, MA[161799]\", \"San Jose,
CA[629546]\", \"Toledo,

OH[354635]\", \"Rochester, NY[241741]\", \"Seattle, WA[493846]\", \"San
Diego, CA[875538]\"]\n",

"distance: 4527\n",

"Population: 4735142\n",
"4 route: [\"Seattle, WA[493846]\", \"Saint Louis, MO[453085]\", \"San Jose,
CA[629546]\",

\"Worcester, MA[161799]\", \"San Diego, CA[875538]\", \"Rockford,
IL[139712]\", \"Ravenna,

OH[11987]\", \"San Antonio, TX[786023]\", \"Rochester, NY[241741]\",
\"Toledo, OH[354635]\"]\n",

"distance: 4006\n",

"Population: 4147912\n",

"5 route: [\"Saint Louis, MO[453085]\", \"Worcester, MA[161799]\", \"San
Diego,

CA[875538]\", \"Winnipeg, MB[564473]\", \"Rochester, NY[241741]\", \"San
Antonio, TX[786023]\",

\"Toronto, ON[599217]\", \"Rockford, IL[139712]\", \"Ravenna,
OH[11987]\", \"San Jose,

CA[629546]\"]\n",

"distance: 4342\n",

"Population: 4463121\n",

"6 route: [\"San Jose, CA[629546]\", \"Rochester, NY[241741]\", \"San Diego,
CA[875538]\",

\"San Antonio, TX[786023]\", \"Rockford, IL[139712]\", \"Toronto,
ON[599217]\", \"Ravenna,

OH[11987]\", \"Saint Louis, MO[453085]\", \"Winnipeg, MB[564473]\",
\"Washington,

DC[638432]\"]\n",

"distance: 4846\n",
"Population: 4939754\n",


"7 route: [\"San Jose, CA[629546]\", \"Toledo, OH[354635]\", \"Rochester,
NY[241741]\",

\"San Antonio, TX[786023]\", \"Washington, DC[638432]\", \"Toronto,
ON[599217]\", \"Ravenna,

OH[11987]\", \"Winnipeg, MB[564473]\", \"San Diego, CA[875538]\",
\"Worcester,

MA[161799]\"]\n",

"distance: 4788\n",

"Population: 4863391\n",

"8 route: [\"Washington, DC[638432]\", \"Winnipeg, MB[564473]\",
\"Rochester,

NY[241741]\", \"San Antonio, TX[786023]\", \"Toledo, OH[354635]\",
\"Rockford, IL[139712]\",

\"Saint Joseph, MO[76691]\", \"San Jose, CA[629546]\", \"Worcester,
MA[161799]\", \"San Diego,

CA[875538]\"]\n",

"distance: 4540\n",

"Population: 4468590\n",

"9 route: [\"Worcester, MA[161799]\", \"San Jose, CA[629546]\",
\"Rochester, NY[241741]\",

\"San Diego, CA[875538]\", \"Toronto, ON[599217]\", \"Ravenna,
OH[11987]\", \"Saint Louis,
MO[453085]\", \"Saint Joseph, MO[76691]\", \"San Francisco, CA[678974]\", \"Rockford,

IL[139712]\"]\n",

"distance: 4031\n",

"Population: 3868290\n",

"10 route: [\"San Diego, CA[875538]\", \"Worcester, MA[161799]\", \"San Jose,

CA[629546]\", \"Seattle, WA[493846]\", \"Saint Louis, MO[453085]\", \"Toledo, OH[354635]\",

\"Rochester, NY[241741]\", \"San Antonio, TX[786023]\", \"Rockford, IL[139712]\", \"Ravenna,

OH[11987]\"]\n",

"distance: 4424\n",

"Population: 4147912\n",

"Total distance= 43295\n",

"Total population= 44214192\n"

]

},

{

"output_type": "execute_result",

"data": {

"text/plain": [


"[\"San Diego, CA[875538]\",\n",
" \"Worcester, MA[161799]\",\n",
" \"San Jose, CA[629546]\",\n",
" \"Seattle, WA[493846]\",\n",
" \"Saint Louis, MO[453085]\",\n",
" \"Toledo, OH[354635]\",\n",
" \"Rochester, NY[241741]\",\n",
" \"San Antonio, TX[786023]\",\n",
" \"Rockford, IL[139712]\",\n",
" \"Ravenna, OH[11987]\"]"
]
},
"metadata": {},
"execution_count": 28
}
],
"source": [
"\n",
"geneticAlgorithm(population=cityList, popSize=600, eliteSize=20,
mutationRate=0.15,
generations=100)\n"
]
},
{
```

```
"cell_type": "code",
"execution_count": 29,
"metadata": {
"id": "R8XUydKH3uXX"
},

"outputs": [],
"source": [
"def geneticAlgorithmDistancePlot(population, popSize, eliteSize, mutationRate,
generations):\n",
" pop = initialPopulation(popSize, population)\n",
" progress = []\n",
" progress.append(Fitness(pop[rankRoutes(pop)[0][0]]).routeDistance())\n",
" \n",
" for i in range(0, generations):\n",
" pop = nextGeneration(pop, eliteSize, mutationRate)\n",
" progress.append(Fitness(pop[rankRoutes(pop)[0][0]]).routeDistance())\n",
" \n",
" plt.plot(progress)\n",
" plt.ylabel('Distance')\n",
" plt.xlabel('Generation')\n",
" plt.show()\n",
"def geneticAlgorithmPopulationPlot(population, popSize, eliteSize,
mutationRate,
generations):\n",
" pop = initialPopulation(popSize, population)\n",
" progress = []\n",
"
progress.append(Fitness(pop[rankRoutes(pop)[0][0]]).routePopulation())\n"
,
" \n",
" for i in range(0, generations):\n",
" pop = nextGeneration(pop, eliteSize, mutationRate)\n",
"
progress.append(Fitness(pop[rankRoutes(pop)[0][0]]).routePopulation())\n"
,
" \n",
" plt.plot(progress)\n",
" plt.ylabel('Population')\n",


" plt.xlabel('Generation')\n",
" plt.show()"
]
},
{
"cell_type": "code",
"execution_count": 31,
"metadata": {
"id": "jD_0FY4bh9bF",
"colab": {
"base_uri": "https://localhost:8080/",
"height": 552
},
"outputId": "33c134bc-72eb-4ea0-b557-aa60f48255e5"
},
"outputs": [
{
"output_type": "display_data",
"data": {
"text/plain": [
"<Figure size 432x288 with 1 Axes>"
],
},
"metadata": {
"needs_background": "light"
}
}
],
"source": [
"\n",
"geneticAlgorithmDistancePlot(population=cityList, popSize=600,
eliteSize=80,
mutationRate=0.15, generations=100)\n",
"geneticAlgorithmPopulationPlot(population=cityList, popSize=600,
eliteSize=80,
mutationRate=0.15, generations=100)\n"
]
}
],
"metadata": {
"colab": {
"name": "Route optimization with genetic approach.ipynb",
"provenance": [],
"collapsed_sections": []
},
"kernelspec": {
"display_name": "Python 3",
"name": "python3"
},
"language_info": {
```

```
"name": "python"
}
},
"nbformat": 4,
"nbformat_minor": 0
}
```



Output:

Conclusion :

In this paper a smart information system is presented for the bus passengers that have the ability to interconnect passengers with real-world public bus. The Smart information system based on

distributed IoT System consists of IR Sensors, GPS Module, Arduino UNO, and IoT Module to count

the number of passengers and to provide information about current and next location of the bus.

Since the users are provided with real time information about the vacant seats, the passengers

will able to take better decisions in terms of bus.

Thankyou!