WIA1002 DATA STRUCTURES

Technical Report

Group Name: ERROR 404

Group Members:

1. HONGWEI LI (Leader) - S2018420 - OCC 3

2. MOHAMMAD IRFANUL HAQUE - S2012063 - OCC 7

3. LEE KAI YANG - U2005398 - OCC 2

4. TIANYI LIU - S2018504 - OCC 3

Table of contents

Task assigned

Project 2: ‘Always On Time’ Delivery

Introduction & Problem Statement:

Our friend’s delivery company ‘Never On Time Sdn Bhd’ is receiving tons of complaints from customers as they feel that the delivery process is far too slow. Delivery men in our friend’s company are always lost in the middle of their delivery route, don’t know where to deliver the parcel and which road they should take to shorten the delivery time. Sometimes they feel angry and exhausted when they lose their direction and they eventually take it out on the parcels which causes more complaints from customers. Our friend tried out many ways to solve the problem but to no avail. Hence, we are the last hope to save his company.

In this assignment, we as professional engineers were assigned to write a program to simulate the delivery process and planning in a country to help our friend shorten their delivery time.

Project requirement

Basic Requirements:

There are a total of 3 outputs you have to generate, we define each output as a simulation:

1. **Basic Simulation**
   In the first simulation, we are requested to find the best tour for a given case with small N using Breadth-First Search / Depth-First Search traversal implementation which we will learn in the class.

2. **Greedy Simulation**
   Secondly, we are requested to implement a Greedy Search that is able to find a good solution given a case with small or large N. Greedy Search means we simply look for the best option in the next move when we travel through the whole graph.

3. **MCTS Simulation**

We want to search for the best tour that we could search for in a limited computation time. This means that if we are given enough time, then we will provide the best tour, else we will just provide the best tour we could find if we are given a limited time with large N.

For every simulation, we need to show the tour's cost, followed by every used vehicle's information which includes the route taken, vehicle's used capacity and cost of the route taken.

Extra Features

Graphic User Interface (GUI) - Build your simulator with a nice looking GUI. You can either simulate it in a graph or in a graph with a graphic map background. Your program should simulate the process of delivery (movement of vehicles between locations with respect to time).

Random Parcel Pick Up Spawning - During the parcel delivery process, there might be customers requesting parcel pick up from their home to be delivered to other places. Thus, in order to minimize cost, we have to update the path used by couriers (vehicles) whenever there is a new request for parcel pickup from a new location.

Pickup and Delivery - In this case, parcels are not initially located at the depot, instead parcels are on the customer's site. For every customer, you need to send a vehicle to their location to pick up the parcel (demand) and send it to another location specified by the customer (demand is released at the destination).

There are few things you might need to be careful of: first, a delivery point cannot come before its respective pickup point when you find the best route for your couriers. Secondly, all vehicles departing from the depot have 0 used capacity (as parcels are not inside the depot anymore). When a vehicle reaches a pickup point, it decrements the available capacity in the vehicle as it picks up the parcel. When a vehicle reaches a delivery point with its respective parcel, it releases the parcel and thus increments back the available capacity of the vehicle.

Heterogeneous Vehicle Capacity - In basic requirements, we assume that every vehicle shares the same capacity, C. In fact, we might have different types of vehicles that have different capacity (e.g. a lorry can deliver more loads than a van). In order to produce a simulation closer to real life, you might need to consider adding this feature.

Time constraint from customer - In real life, we not only have to minimize the time and fuel (represented by distance) used by couriers, we also have to consider the expected arrival time of every parcel to their owners. We should not deliver the parcel later than its expected arrival time as it would result in bad customer reviews. In your simulation, you might want to add this feature as well.

For every customer, you might need to add a time window [t1, t2] to specify the time range we can deliver a parcel to a customer. If we arrive at the customer location before t1, then we have to wait for it and do nothing else, and if we deliver the parcel after t2, then we will receive a penalty which is undesired. Thus, your tour should strictly follow the time window for every customer and at the same time minimize the cost.

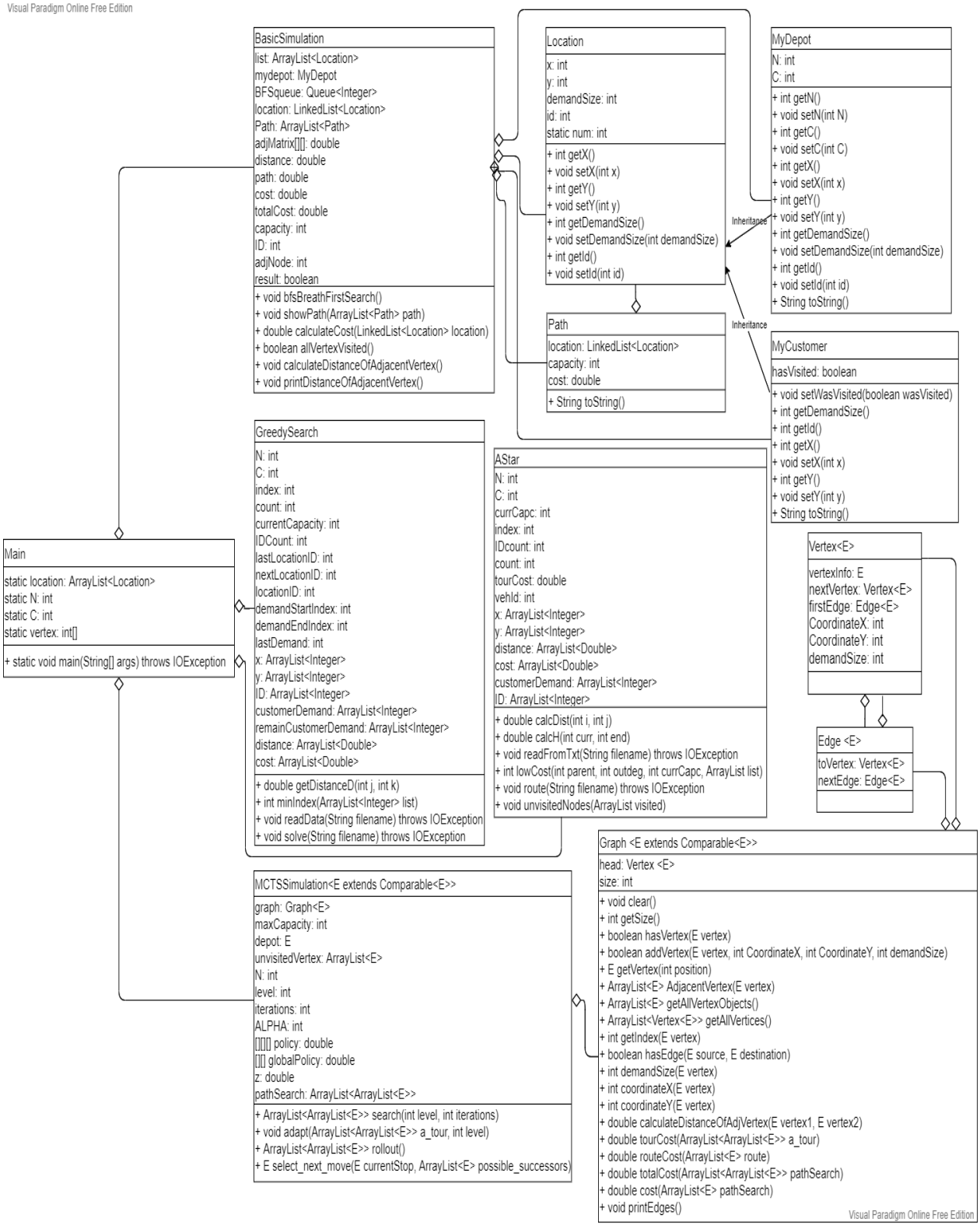
After considering all the extra features provided, our team finally have chosen to implement an extra searching algorithm, **A star searching method** and also a **Graphical User Interface (GUI)** to simulate and display the delivery process.

Project approach

We start by defining all the terms we are going to use in this problem context. A customer is an entity that has a certain demand and therefore requires the presence of a vehicle, a unit that can move between customers and the depot, a unit that initially possesses the demands of the customers. All vehicles are capacitated that they can only contain goods (the customer's demands) up to a certain maximum capacity. Moving a vehicle between the depot and the customers comes with a certain cost. A route is a sequence of visited customers by a certain vehicle, starting and ending at a depot while a tour is a list of routes of all vehicles to fulfil all customers' demands. We can imagine the underlying structure of this problem as a complete undirected graph G(V, E).

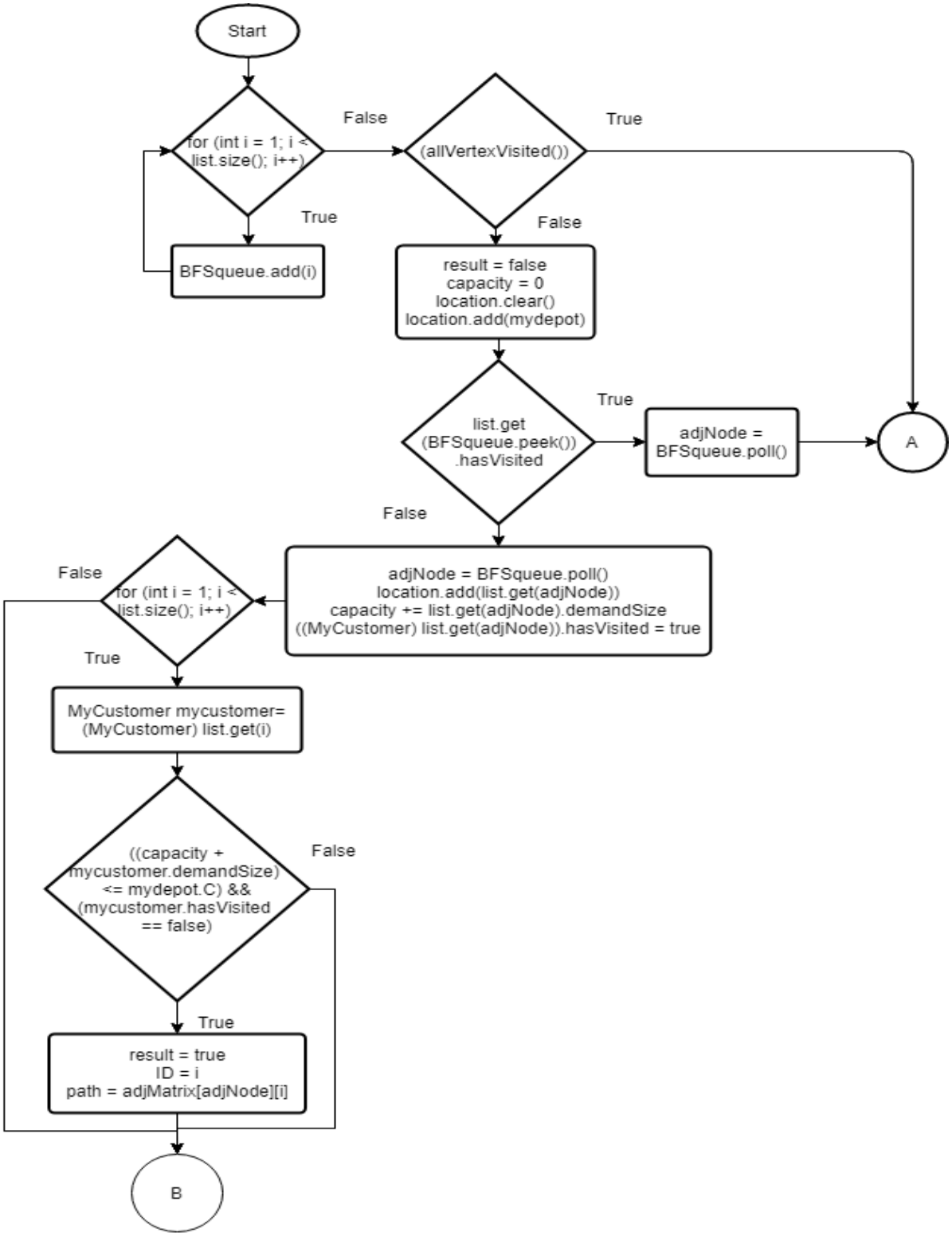
Given a text file, where the first row indicates the number of customers (including depot), N and maximum capacity of all vehicles, C. After this, starting from the second row onwards are the N rows of information. In particular, every row of information contains 3 data, which are x-coordinate, y-coordinate and lastly demand size of a customer. The second row represents the depot and its demand size is always 0 while the rest of the rows show customer information. Three searching simulation outputs are generated, which include **basic simulation**, **greedy simulation** and **MCTS simulation**.

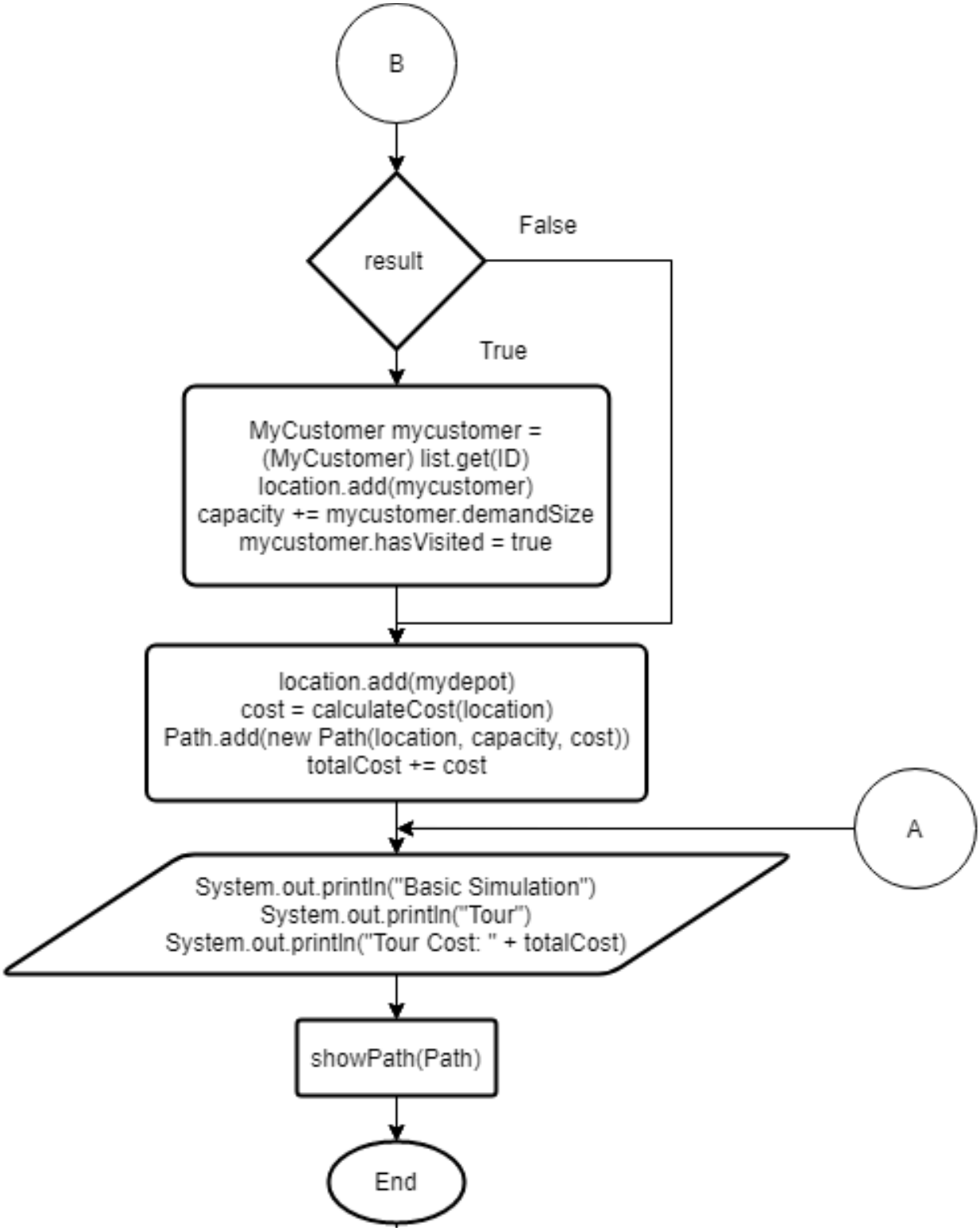
UML Diagram


Visual Paradigm Online Free Edition
BasicSimulation
list: ArrayList<Location>
mydepot: MyDepot
BFSqueue: Queue<Integer>
location: LinkedList<Location>
Path: ArrayList<Path>
adjMatrix[][]: double
distance: double
path: double
cost: double
totalCost: double
capacity: int
ID: int
adjNode: int
result: boolean
+ void bfsBreathFirstSearch()
+ void showPath(ArrayList<Path> path)
+ double calculateCost(LinkedList<Location> location)
+ boolean allVertexVisited()
+ void calculateDistanceOfAdjacentVertex()
+ void printDistanceOfAdjacentVertex()
Location
x: int
y: int
demandSize: int
id: int
static num: int
+ int getX()
+ void setX(int x)
+ int getY()
+ void setY(int y)
+ int getDemandSize()
+ void setDemandSize(int demandSize)
+ int getId()
+ void setId(int id)
MyDepot
N: int
C: int
+ int getN()
+ void setN(int N)
+ int getC()
+ void setC(int C)
+ int getX()
+ void setX(int x)
+ int getY()
+ void setY(int y)
+ int getDemandSize()
+ void setDemandSize(int demandSize)
+ int getId()
+ void setId(int id)
+ String toString()
Inheritance
Path
location: LinkedList<Location>
capacity: int
cost: double
+ String toString()
Inheritance
MyCustomer
hasVisited: boolean
+ void setWasVisited(boolean wasVisited)
+ int getDemandSize()
+ int getId()
+ int getX()
+ void setX(int x)
+ int getY()
+ void setY(int y)
+ String toString()
Main
static location: ArrayList<Location>
static N: int
static C: int
static vertex: int[]
+ static void main(String[] args) throws IOException
GreedySearch
N: int
C: int
index: int
count: int
currentCapacity: int
IDCount: int
lastLocationID: int
nextLocationID: int
locationID: int
demandStartIndex: int
demandEndIndex: int
lastDemand: int
x: ArrayList<Integer>
y: ArrayList<Integer>
ID: ArrayList<Integer>
customerDemand: ArrayList<Integer>
remainCustomerDemand: ArrayList<Integer>
distance: ArrayList<Double>
cost: ArrayList<Double>
+ double getDistanceD(int j, int k)
+ int minIndex(ArrayList<Integer> list)
+ void readData(String filename) throws IOException
+ void solve(String filename) throws IOException
AStar
N: int
C: int
currCapc: int
index: int
IDcount: int
count: int
tourCost: double
vehId: int
x: ArrayList<Integer>
y: ArrayList<Integer>
distance: ArrayList<Double>
cost: ArrayList<Double>
customerDemand: ArrayList<Integer>
ID: ArrayList<Integer>
+ double calcDist(int i, int j)
+ double calcH(int curr, int end)
+ void readFromTxt(String filename) throws IOException
+ int lowCost(int parent, int outdeg, int currCapc, ArrayList list)
+ void route(String filename) throws IOException
+ void unvisitedNodes(ArrayList visited)
Vertex<E>
vertexInfo: E
nextVertex: Vertex<E>
firstEdge: Edge<E>
CoordinateX: int
CoordinateY: int
demandSize: int
Edge <E>
toVertex: Vertex<E>
nextEdge: Edge<E>
MCTSSimulation<E extends Comparable<E>>
graph: Graph<E>
maxCapacity: int
depot: E
unvisitedVertex: ArrayList<E>
N: int
level: int
iterations: int
ALPHA: int
[][][] policy: double
[][] globalPolicy: double
z: double
pathSearch: ArrayList<ArrayList<E>>
+ ArrayList<ArrayList<E>> search(int level, int iterations)
+ void adapt(ArrayList<ArrayList<E>> a_tour, int level)
+ ArrayList<ArrayList<E>> rollout()
+ E select_next_move(E currentStop, ArrayList<E> possible_successors)
Graph <E extends Comparable<E>>
head: Vertex <E>
size: int
+ void clear()
+ int getSize()
+ boolean hasVertex(E vertex)
+ boolean addVertex(E vertex, int CoordinateX, int CoordinateY, int demandSize)
+ E getVertex(int position)
+ ArrayList<E> AdjacentVertex(E vertex)
+ ArrayList<E> getAllVertexObjects()
+ ArrayList<Vertex<E>> getAllVertices()
+ int getIndex(E vertex)
+ boolean hasEdge(E source, E destination)
+ int demandSize(E vertex)
+ int coordinateX(E vertex)
+ int coordinateY(E vertex)
+ double calculateDistanceOfAdjVertex(E vertex1, E vertex2)
+ double tourCost(ArrayList<ArrayList<E>> a_tour)
+ double routeCost(ArrayList<E> route)
+ double totalCost(ArrayList<ArrayList<E>> pathSearch)
+ double cost(ArrayList<E> pathSearch)
+ void printEdges()
Visual Paradigm Online Free Edition

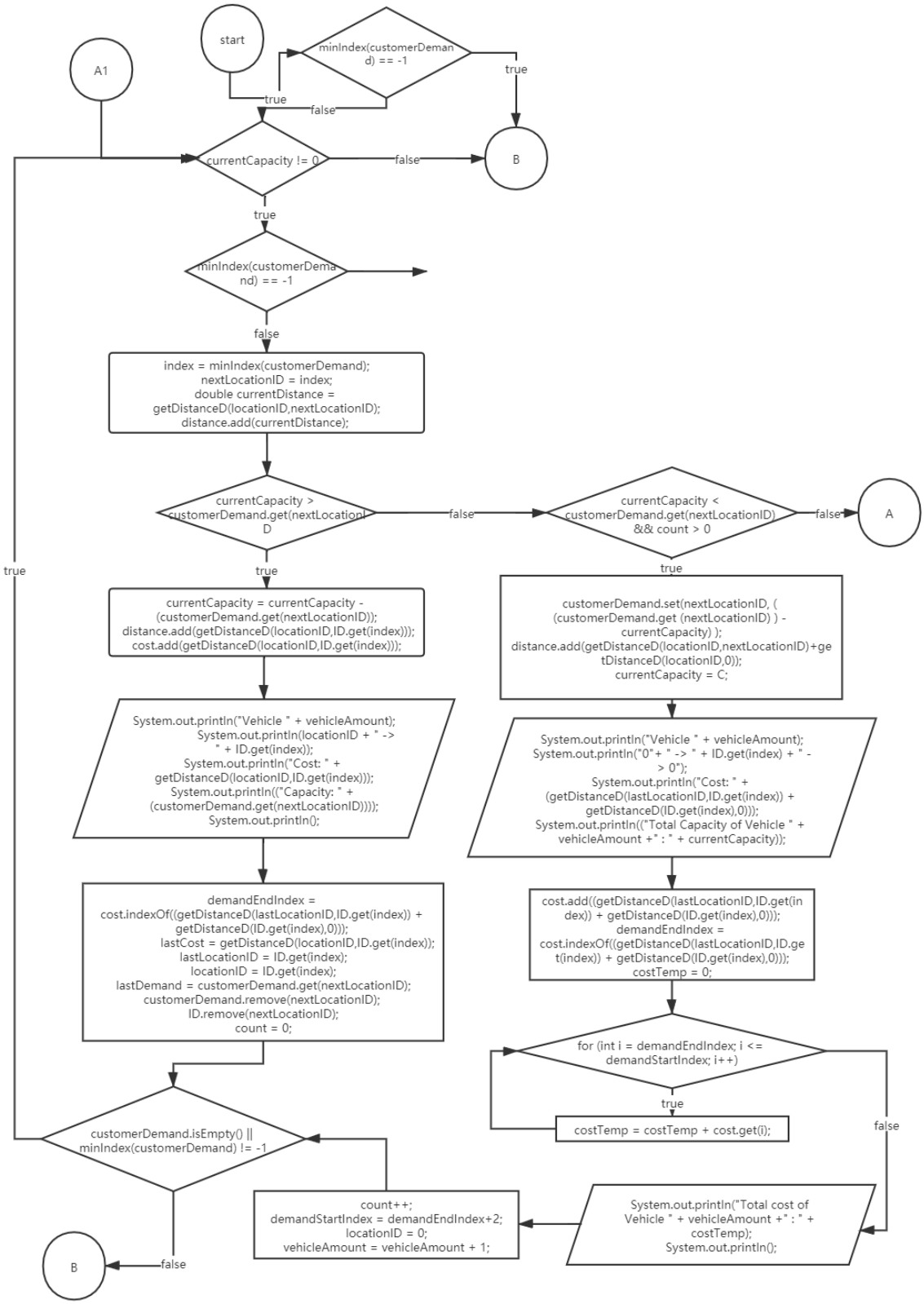
Flowchart

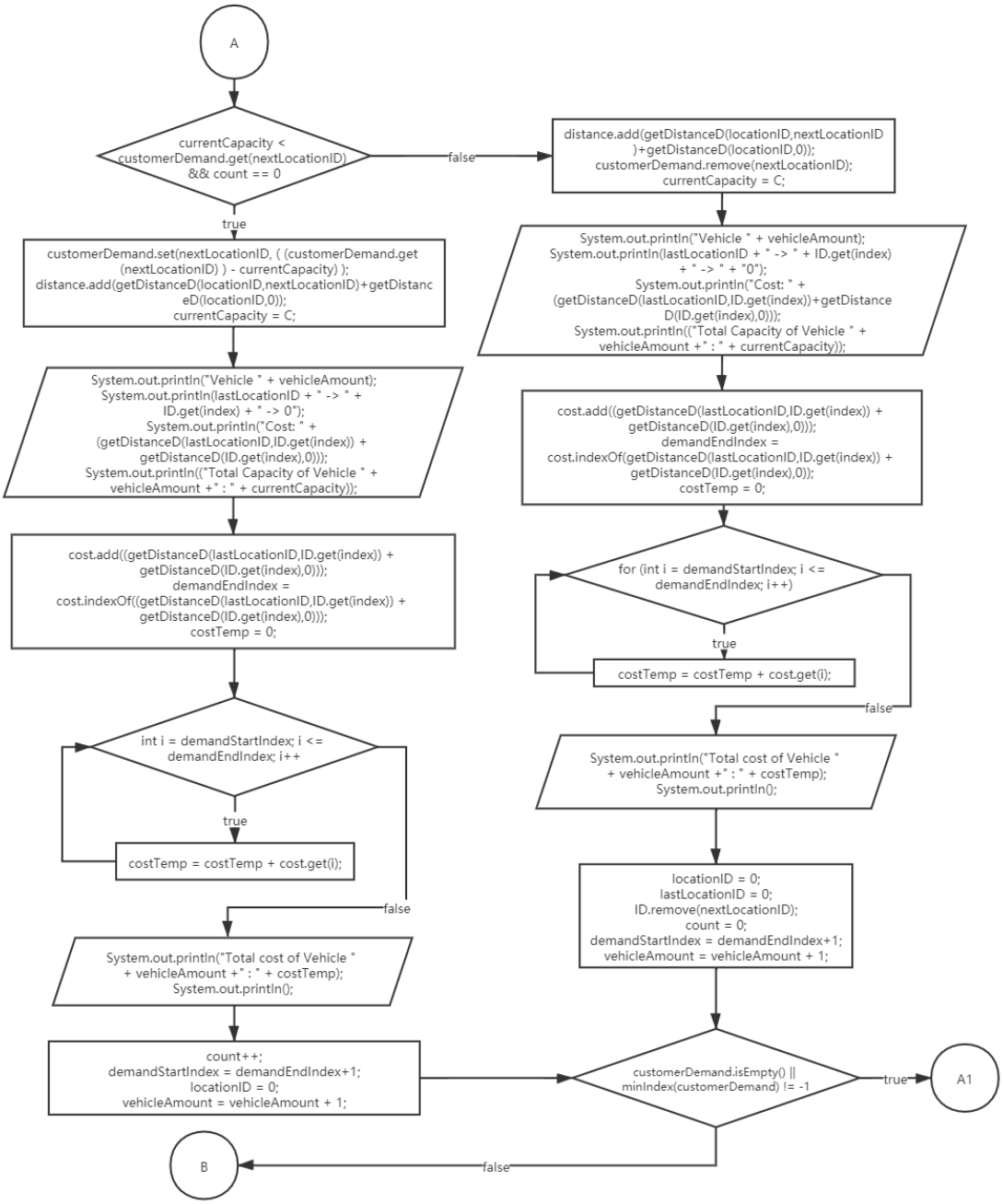
Basic Simulation

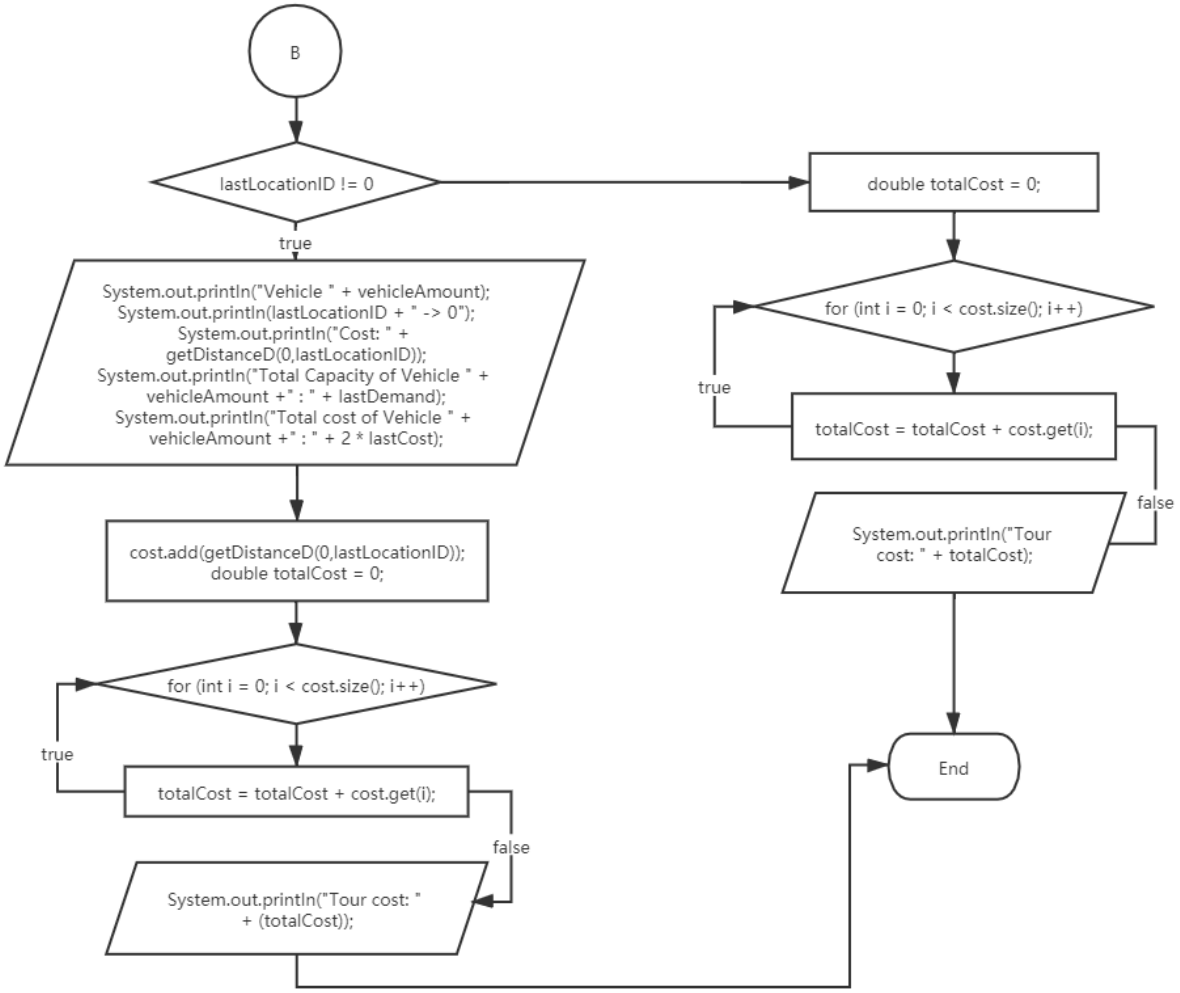



B
result
False
True
MyCustomer mycustomer =
(MyCustomer) list.get(ID)
location.add(mycustomer)
capacity += mycustomer.demandSize
mycustomer.hasVisited = true
location.add(mydepot)
cost = calculateCost(location)
Path.add(new Path(location, capacity, cost))
totalCost += cost
A
System.out.println("Basic Simulation")
System.out.println("Tour")
System.out.println("Tour Cost: " + totalCost)
showPath(Path)
End

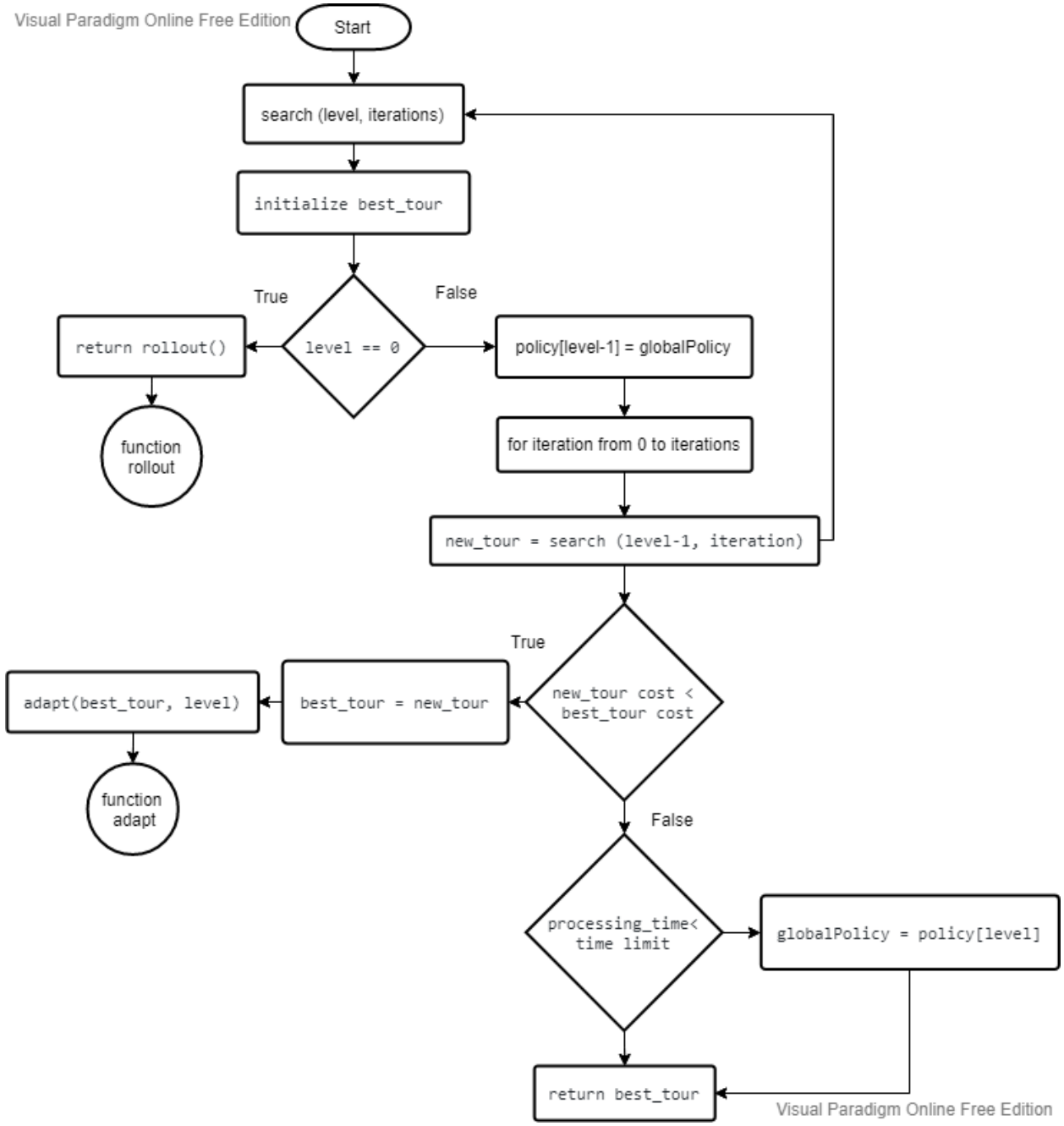
Greedy Simulation

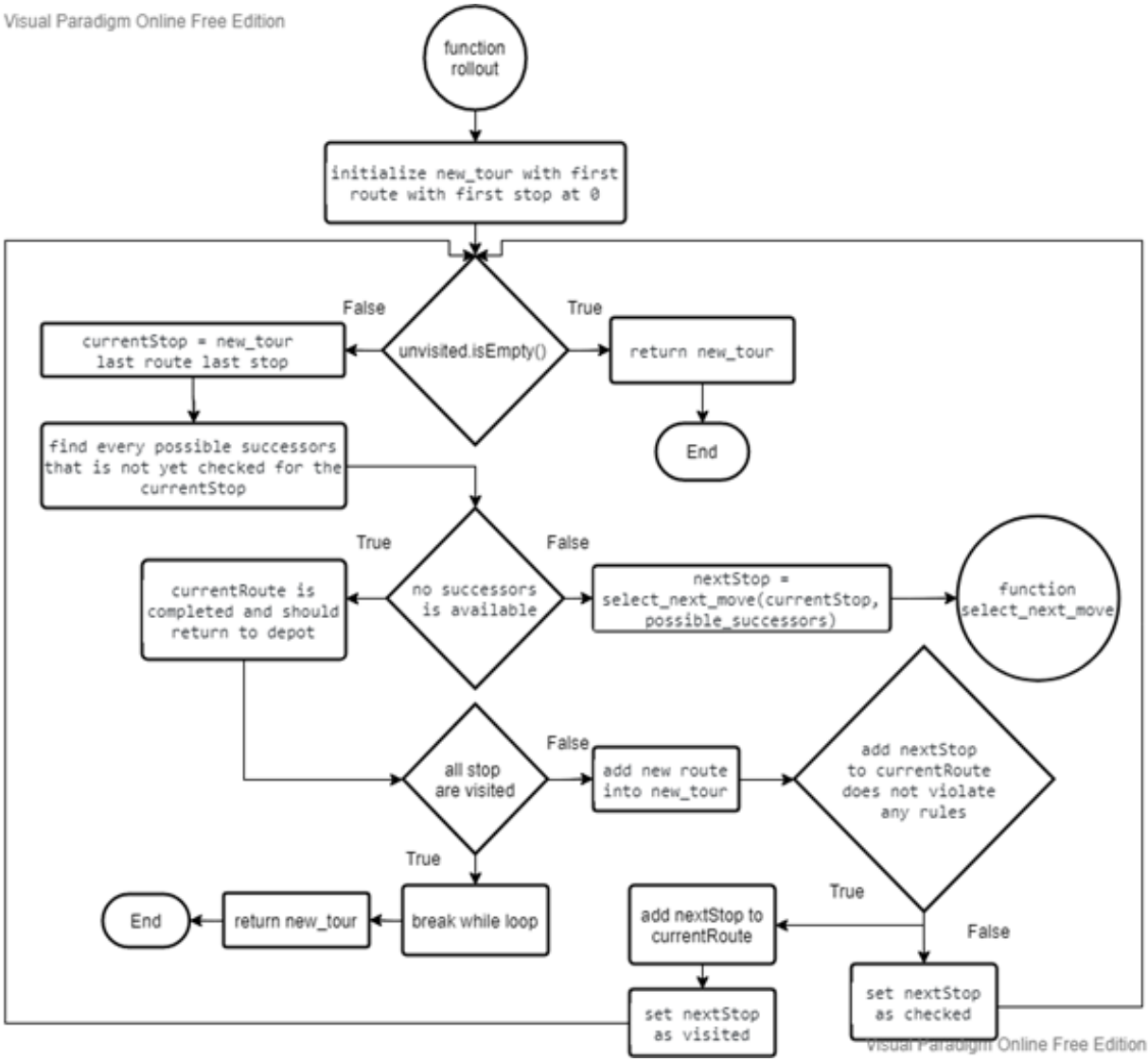


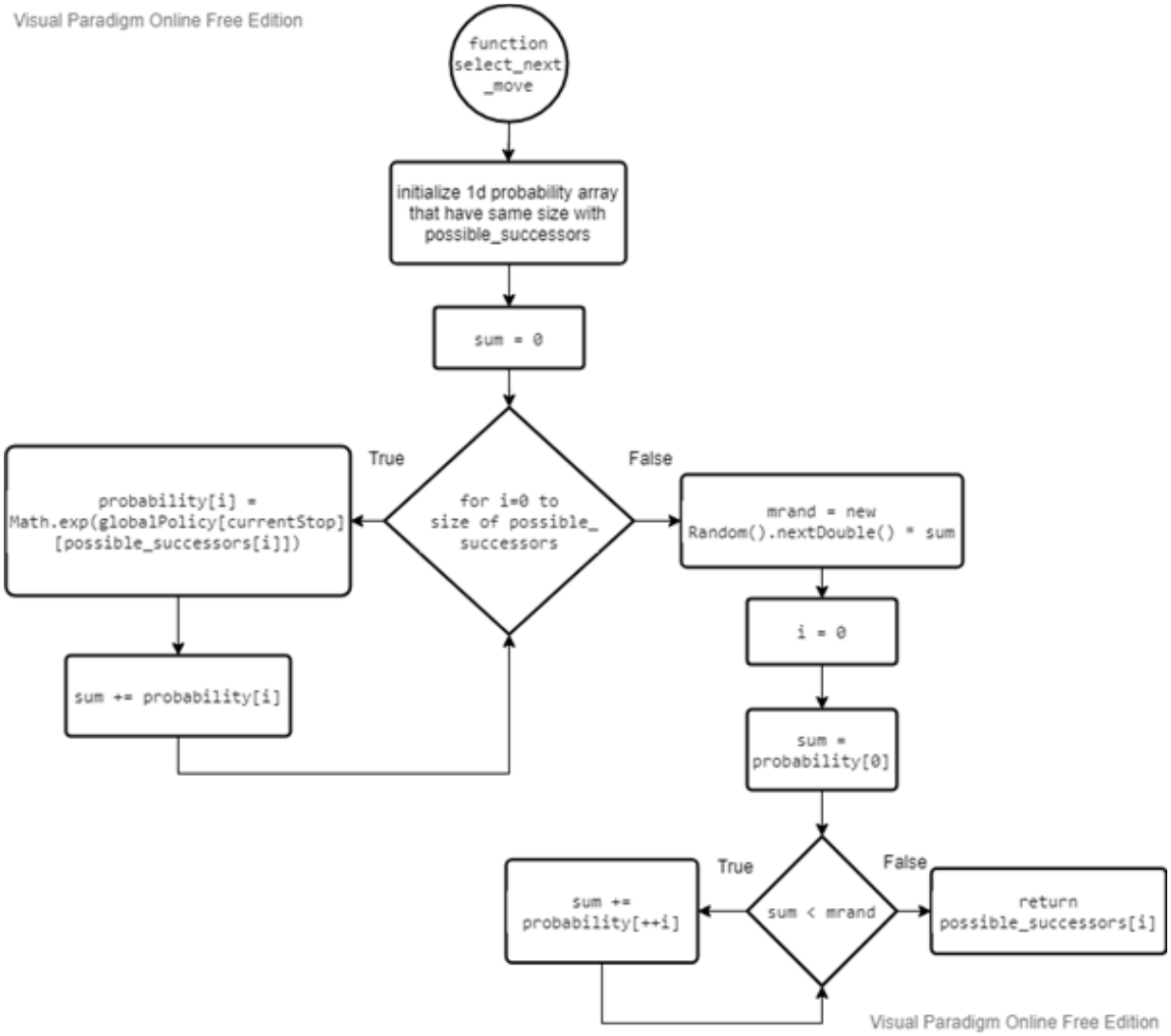

A
currentCapacity < customerDemand.get(nextLocationID) && count == 0
false
true
distance.add(getDistanceD(locationID,nextLocationID)+getDistanceD(locationID,0));
customerDemand.remove(nextLocationID);
currentCapacity = C;
customerDemand.set(nextLocationID, ((customerDemand.get(nextLocationID)) - currentCapacity));
distance.add(getDistanceD(locationID,nextLocationID)+getDistanceD(locationID,0));
currentCapacity = C;
System.out.println("Vehicle " + vehicleAmount);
System.out.println(lastLocationID + " -> " + ID.get(index) + " -> " + "0");
System.out.println("Cost: " + (getDistanceD(lastLocationID,ID.get(index))+getDistanceD(ID.get(index),0)));
System.out.println(("Total Capacity of Vehicle " + vehicleAmount +" : " + currentCapacity));
System.out.println("Vehicle " + vehicleAmount);
System.out.println(lastLocationID + " -> " + ID.get(index) + " -> 0");
System.out.println("Cost: " + (getDistanceD(lastLocationID,ID.get(index)) + getDistanceD(ID.get(index),0)));
System.out.println(("Total Capacity of Vehicle " + vehicleAmount +" : " + currentCapacity));
cost.add((getDistanceD(lastLocationID,ID.get(index)) + getDistanceD(ID.get(index),0)));
demandEndIndex = cost.indexOf((getDistanceD(lastLocationID,ID.get(index)) + getDistanceD(ID.get(index),0)));
costTemp = 0;
cost.add((getDistanceD(lastLocationID,ID.get(index)) + getDistanceD(ID.get(index),0)));
demandEndIndex = cost.indexOf(getDistanceD(lastLocationID,ID.get(index)) + getDistanceD(ID.get(index),0));
costTemp = 0;
for (int i = demandStartIndex; i <= demandEndIndex; i++)
true
costTemp = costTemp + cost.get(i);
false
System.out.println("Total cost of Vehicle " + vehicleAmount +" : " + costTemp);
System.out.println();
int i = demandStartIndex; i <= demandEndIndex; i++
true
costTemp = costTemp + cost.get(i);
false
System.out.println("Total cost of Vehicle " + vehicleAmount +" : " + costTemp);
System.out.println();
locationID = 0;
lastLocationID = 0;
ID.remove(nextLocationID);
count = 0;
demandStartIndex = demandEndIndex+1;
vehicleAmount = vehicleAmount + 1;
count++;
demandStartIndex = demandEndIndex+1;
locationID = 0;
vehicleAmount = vehicleAmount + 1;
customerDemand.isEmpty() || minIndex(customerDemand) != -1
true
A1
B
false

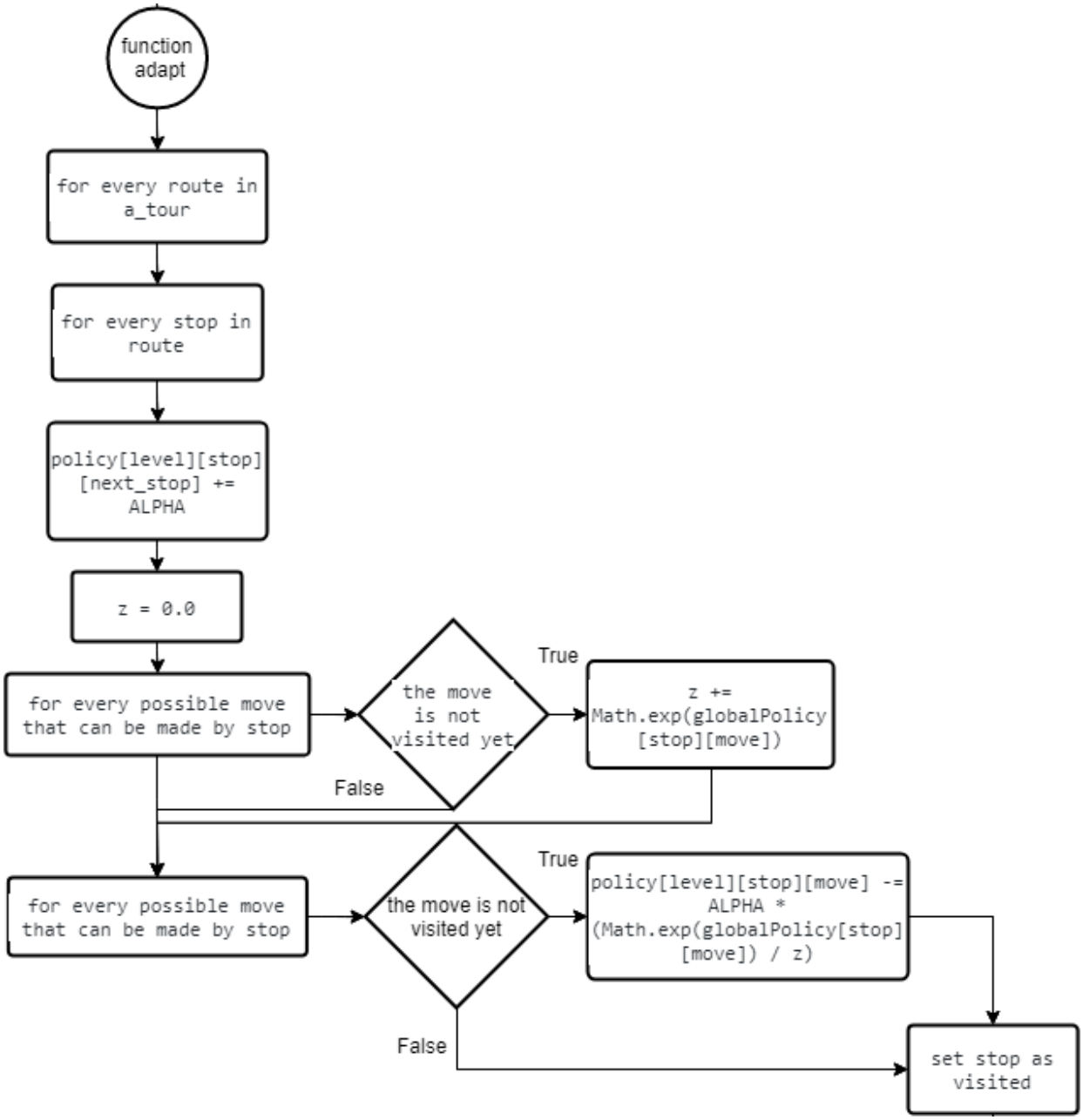

B
lastLocationID != 0
true
System.out.println("Vehicle " + vehicleAmount);
System.out.println(lastLocationID + " -> 0");
System.out.println("Cost: " +
getDistanceD(0,lastLocationID));
System.out.println("Total Capacity of Vehicle " +
vehicleAmount +" : " + lastDemand);
System.out.println("Total cost of Vehicle " +
vehicleAmount +" : " + 2 * lastCost);
cost.add(getDistanceD(0,lastLocationID));
double totalCost = 0;
for (int i = 0; i < cost.size(); i++)
true
totalCost = totalCost + cost.get(i);
false
System.out.println("Tour cost: "
+ (totalCost));
double totalCost = 0;
for (int i = 0; i < cost.size(); i++)
true
totalCost = totalCost + cost.get(i);
false
System.out.println("Tour
cost: " + totalCost);
End

MCTS Simulation

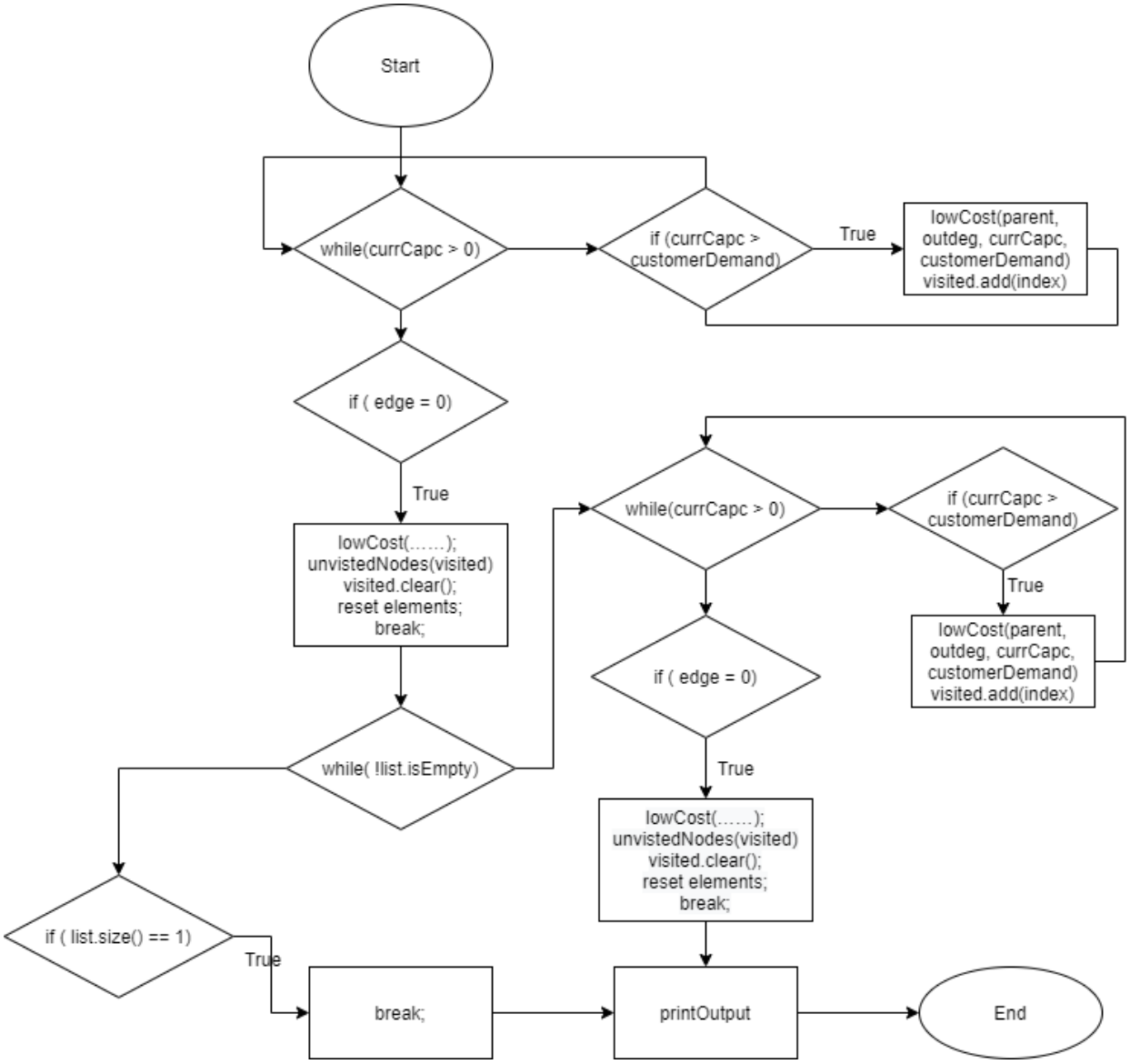



Visual Paradigm Online Free Edition
function rollout
initialize new_tour with first route with first stop at 0
unvisited.isEmpty()
False
True
currentStop = new_tour last route last stop
return new_tour
End
find every possible successors that is not yet checked for the currentStop
no successors is available
True
False
currentRoute is completed and should return to depot
nextStop = select_next_move(currentStop, possible_successors)
function select_next_move
all stop are visited
False
add new route into new_tour
add nextStop to currentRoute does not violate any rules
True
End
return new_tour
break while loop
True
add nextStop to currentRoute
False
set nextStop as visited
set nextStop as checked
Visual Paradigm Online Free Edition


Visual Paradigm Online Free Edition
function select_next _move
initialize 1d probability array that have same size with possible_successors
sum = 0
for i=0 to size of possible_ successors
True
probability[i] = Math.exp(globalPolicy[currentStop] [possible_successors[i]])
sum += probability[i]
False
mrand = new Random().nextDouble() * sum
i = 0
sum = probability[0]
sum < mrand
True
sum += probability[++i]
False
return possible_successors[i]
Visual Paradigm Online Free Edition

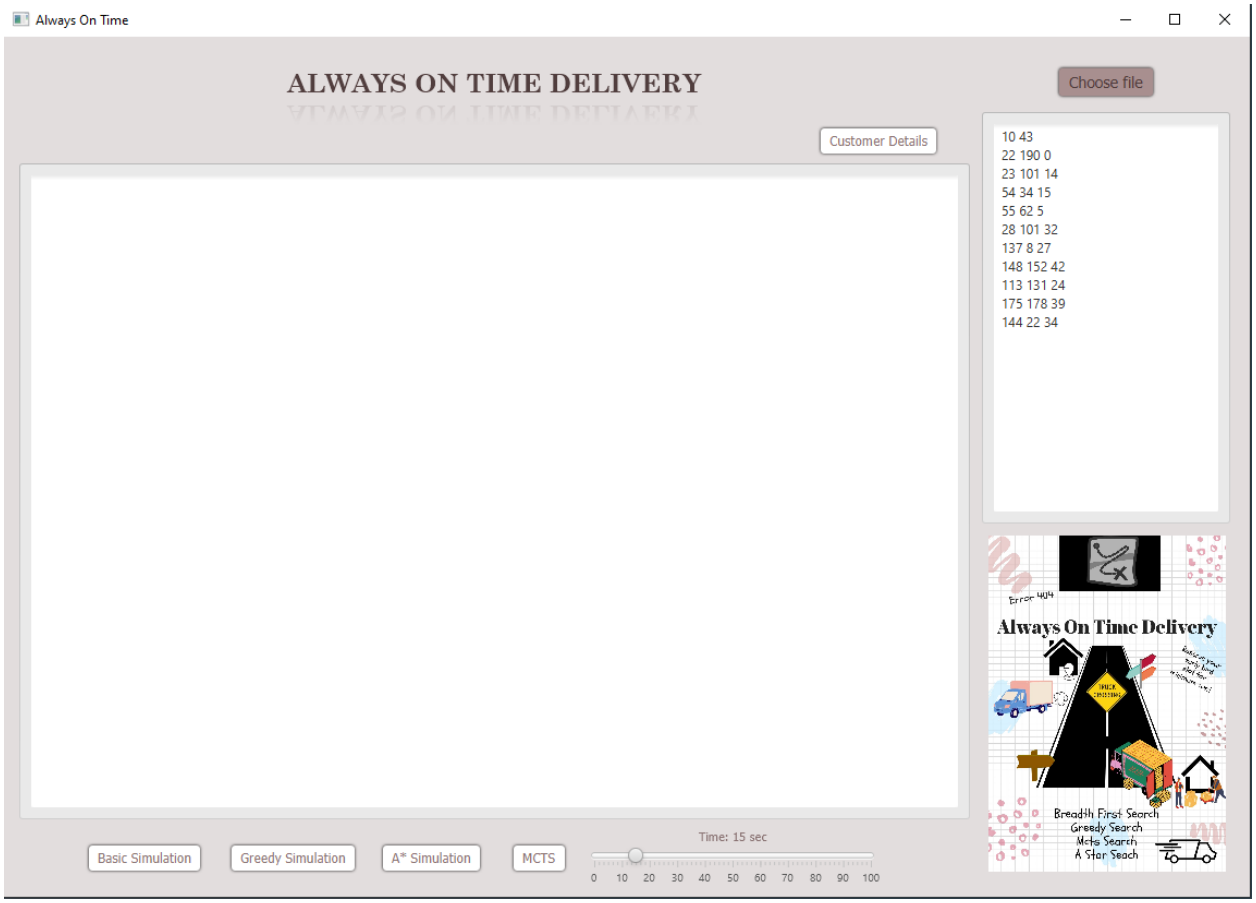

function adapt
for every route in a_tour
for every stop in route
policy[level][stop] [next_stop] += ALPHA
z = 0.0
for every possible move that can be made by stop
the move is not visited yet
True
z += Math.exp(globalPolicy [stop][move])
False
for every possible move that can be made by stop
the move is not visited yet
True
policy[level][stop][move] -= ALPHA * (Math.exp(globalPolicy[stop] [move]) / z)
False
set stop as visited

A* Algorithm

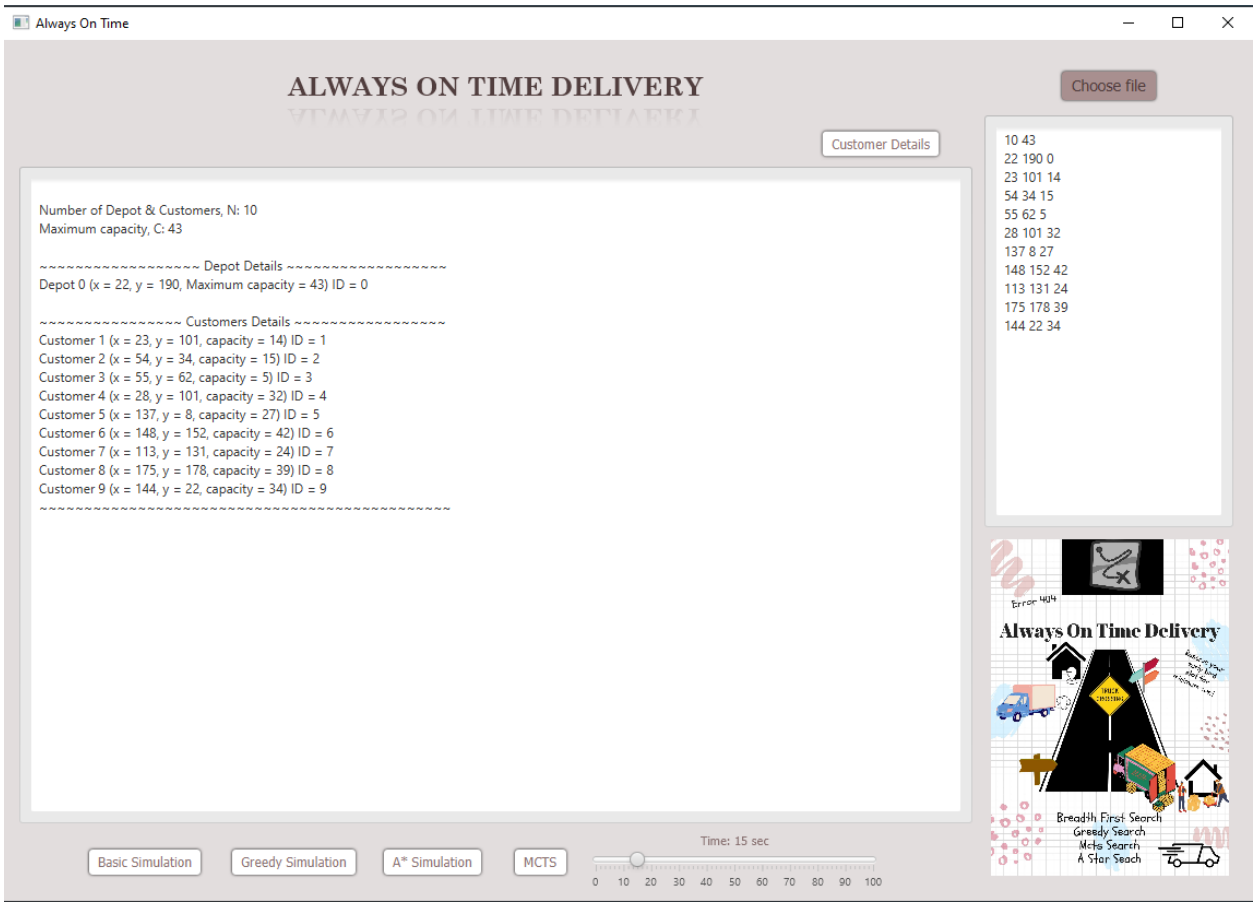

Start
while(currCapc > 0)
if (currCapc > customerDemand)
True
lowCost(parent, outdeg, currCapc, customerDemand) visited.add(index)
if (edge = 0)
True
lowCost(……); unvistedNodes(visited) visited.clear(); reset elements; break;
while(!list.isEmpty)
while(currCapc > 0)
if (currCapc > customerDemand)
True
lowCost(parent, outdeg, currCapc, customerDemand) visited.add(index)
if (edge = 0)
True
lowCost(……); unvistedNodes(visited) visited.clear(); reset elements; break;
if (list.size() == 1)
True
break;
printOutput
End

Sample output

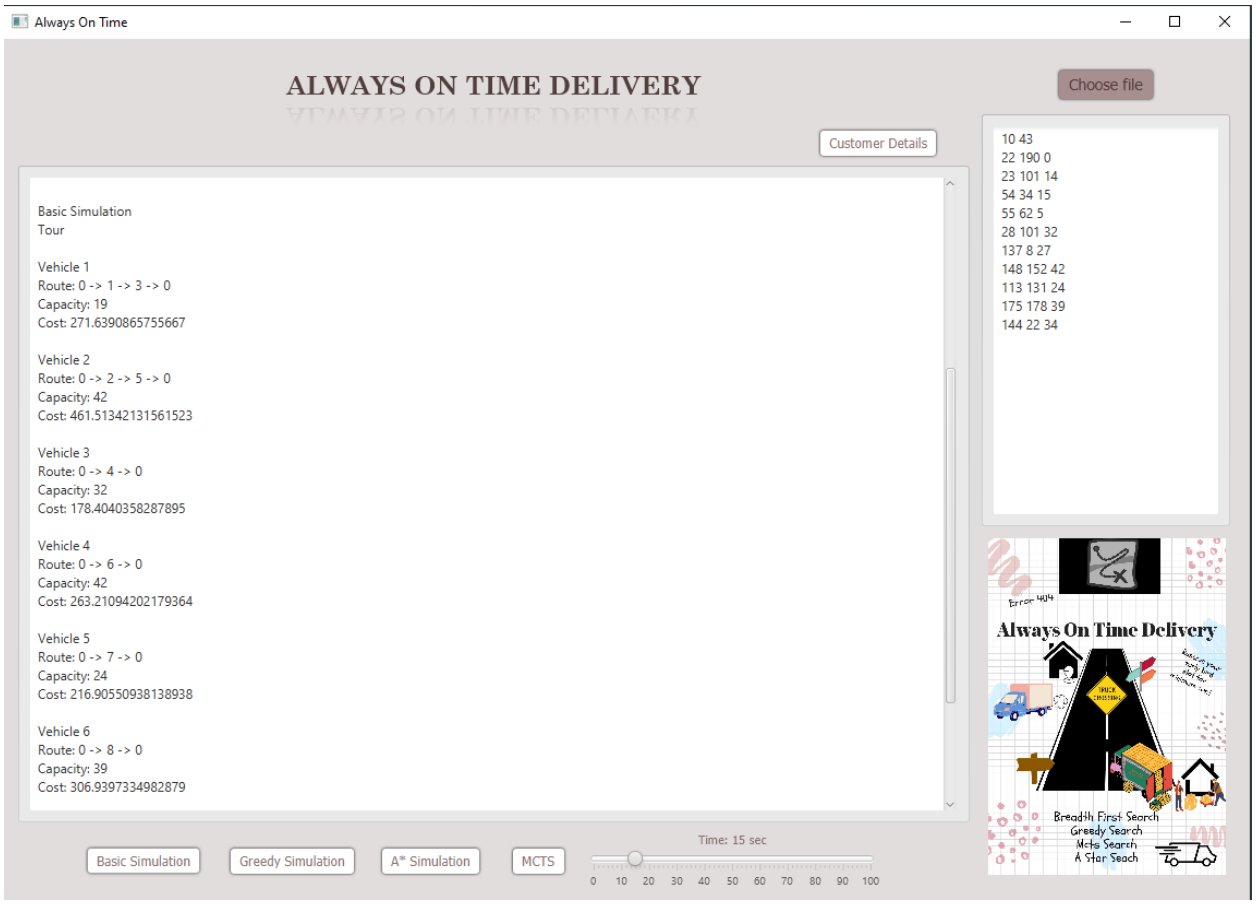
File Input:

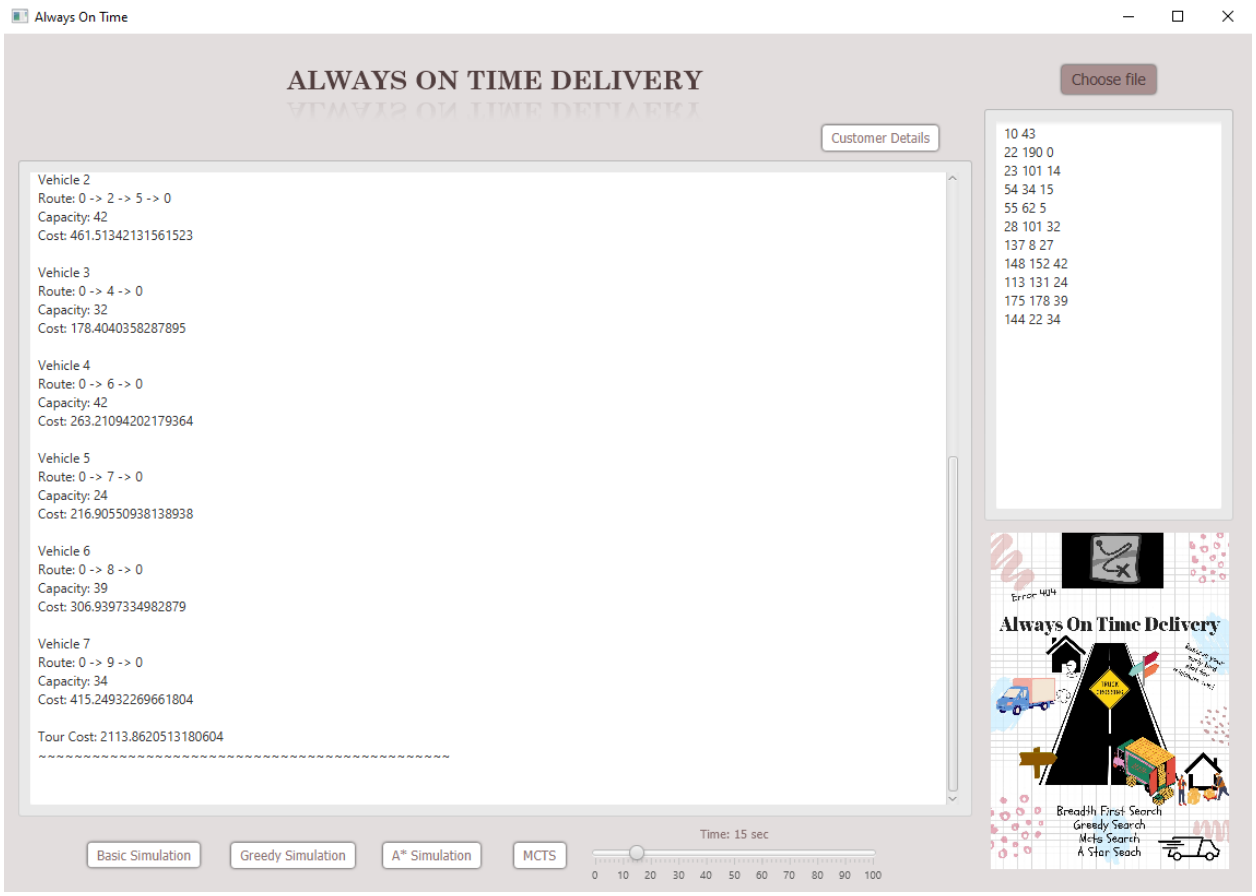


Customer Details:

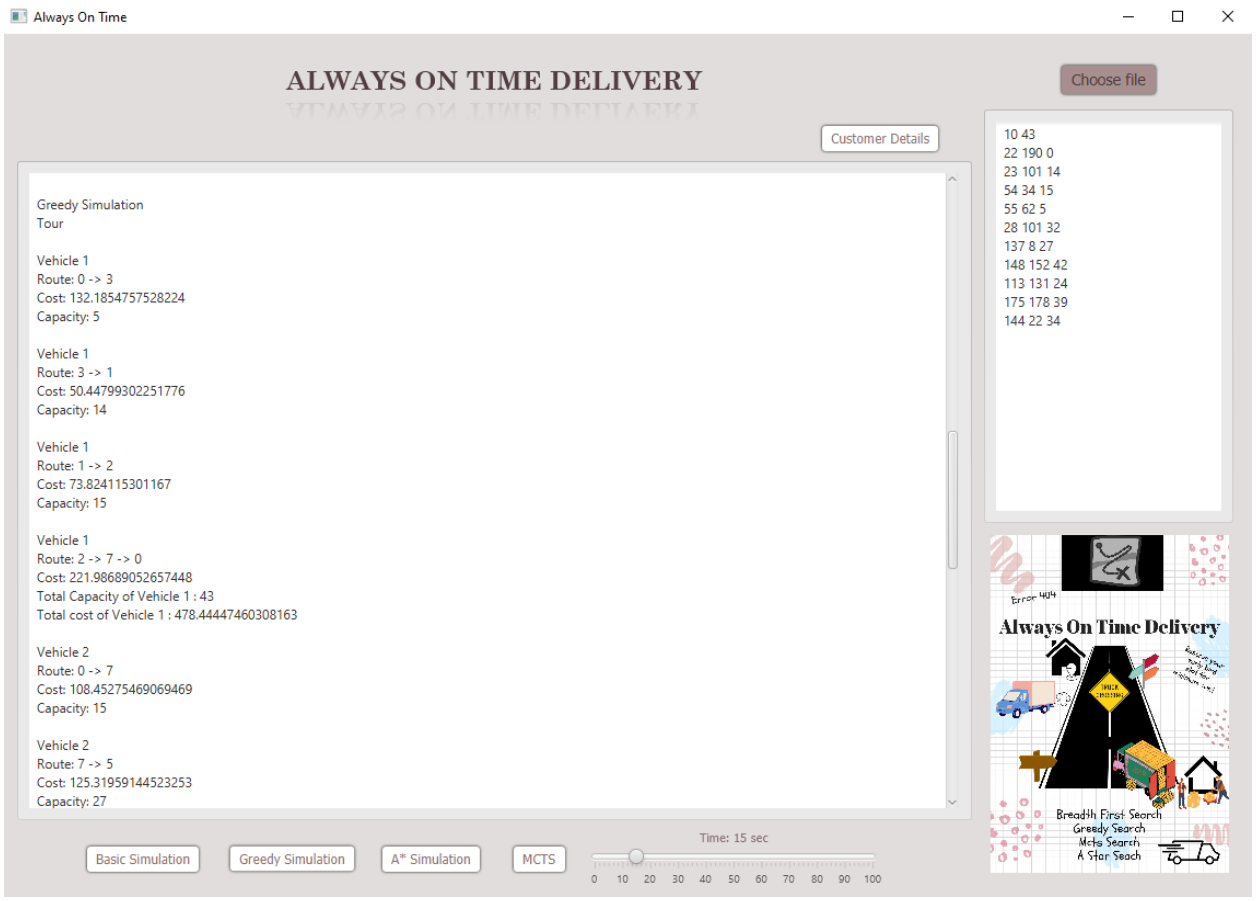


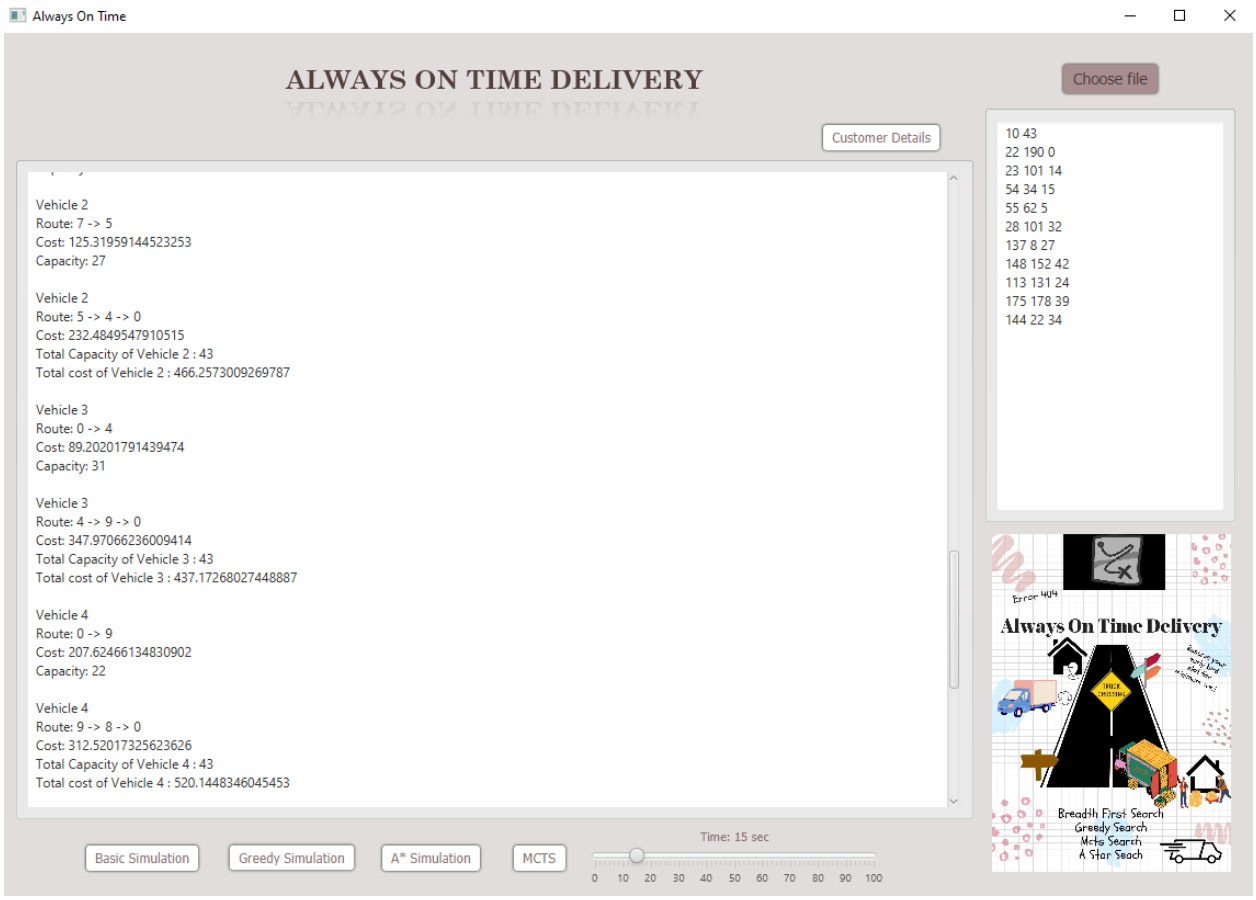
Basic Simulation:

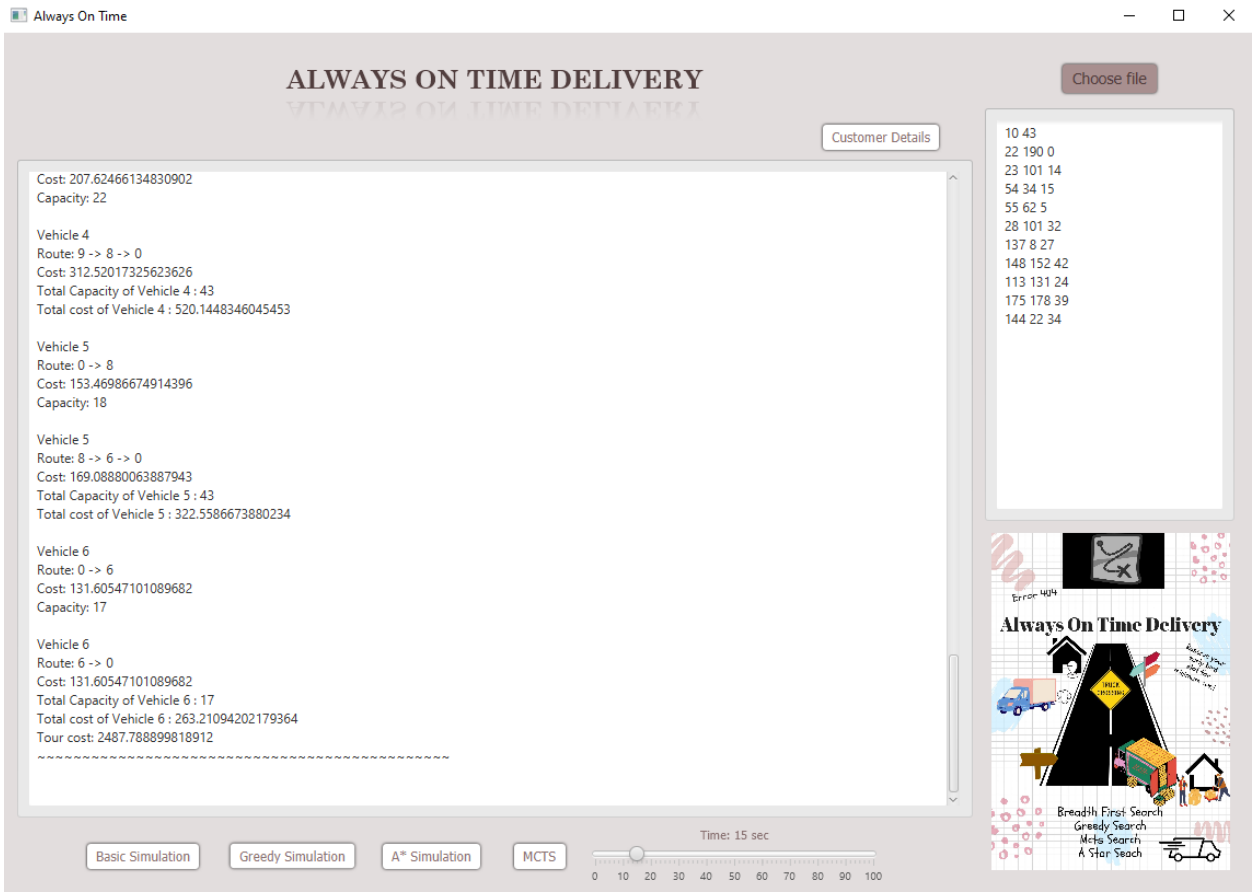



Always On Time
ALWAYS ON TIME DELIVERY
Choose file
Customer Details
Vehicle 2
Route: 0 -> 2 -> 5 -> 0
Capacity: 42
Cost: 461.51342131561523
Vehicle 3
Route: 0 -> 4 -> 0
Capacity: 32
Cost: 178.4040358287895
Vehicle 4
Route: 0 -> 6 -> 0
Capacity: 42
Cost: 263.21094202179364
Vehicle 5
Route: 0 -> 7 -> 0
Capacity: 24
Cost: 216.90550938138938
Vehicle 6
Route: 0 -> 8 -> 0
Capacity: 39
Cost: 306.9397334982879
Vehicle 7
Route: 0 -> 9 -> 0
Capacity: 34
Cost: 415.24932269661804
Tour Cost: 2113.8620513180604
10 43
22 190 0
23 101 14
54 34 15
55 62 5
28 101 32
137 8 27
148 152 42
113 131 24
175 178 39
144 22 34
Always On Time Delivery
Breadth First Search
Greedy Search
Mcts Search
A Star Seach
Basic Simulation
Greedy Simulation
A* Simulation
MCTS
Time: 15 sec
0 10 20 30 40 50 60 70 80 90 100

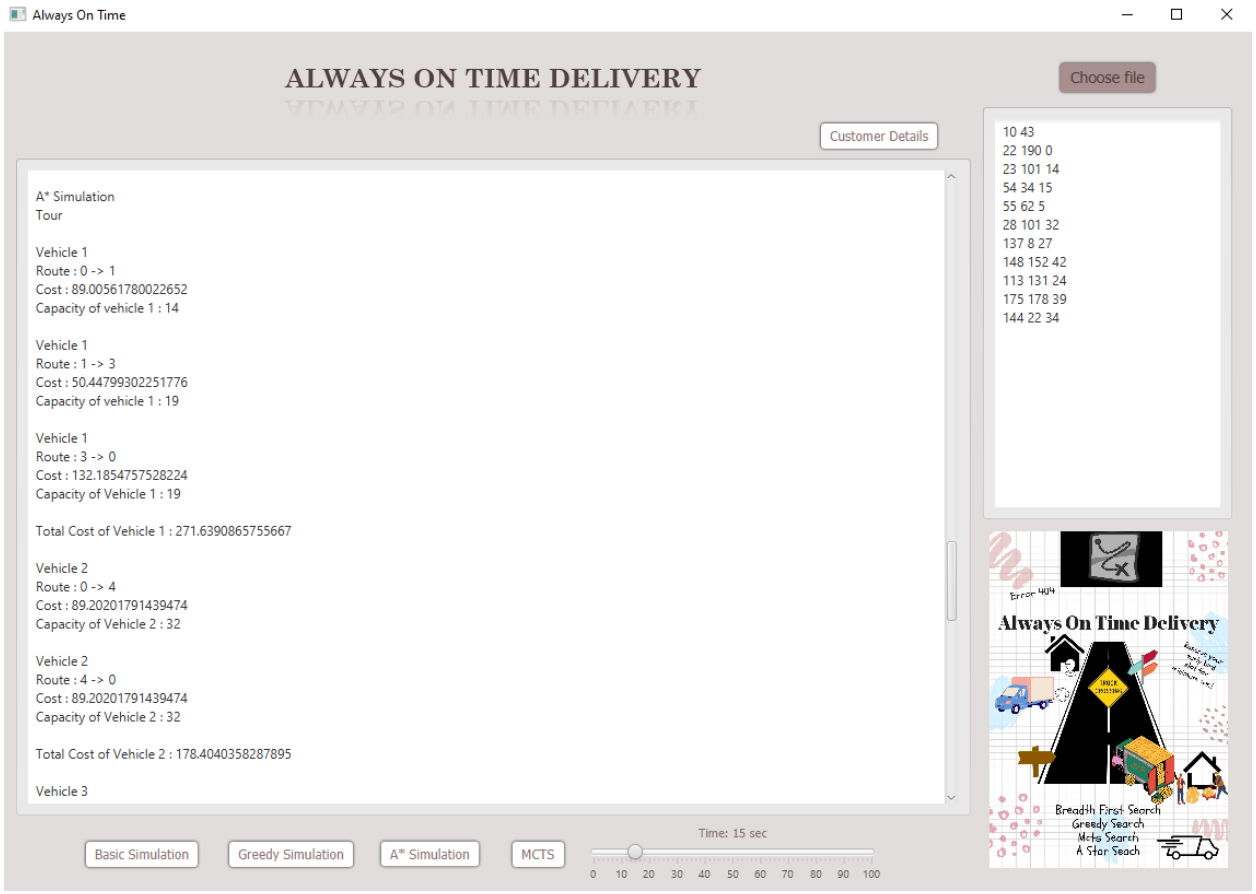
Greedy Simulation:

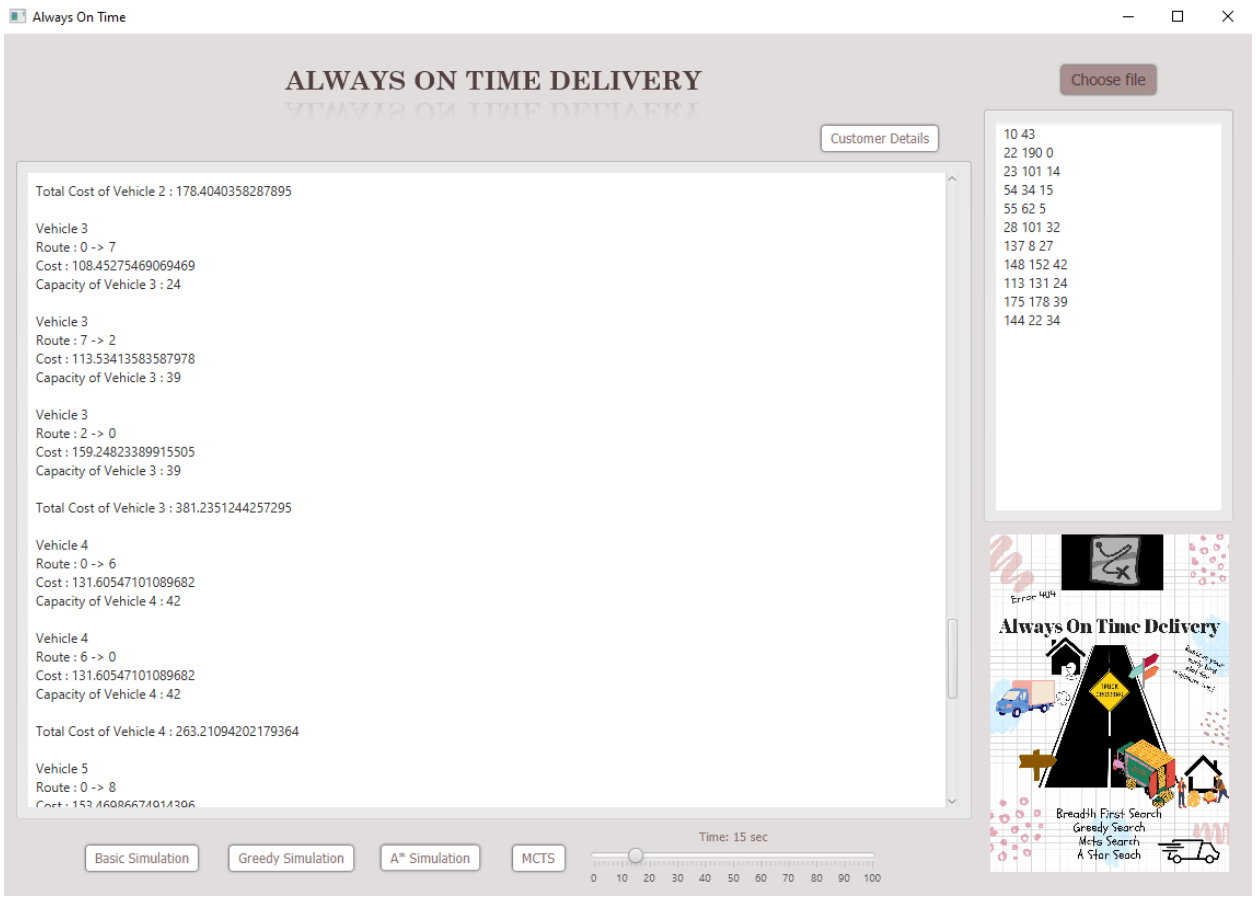


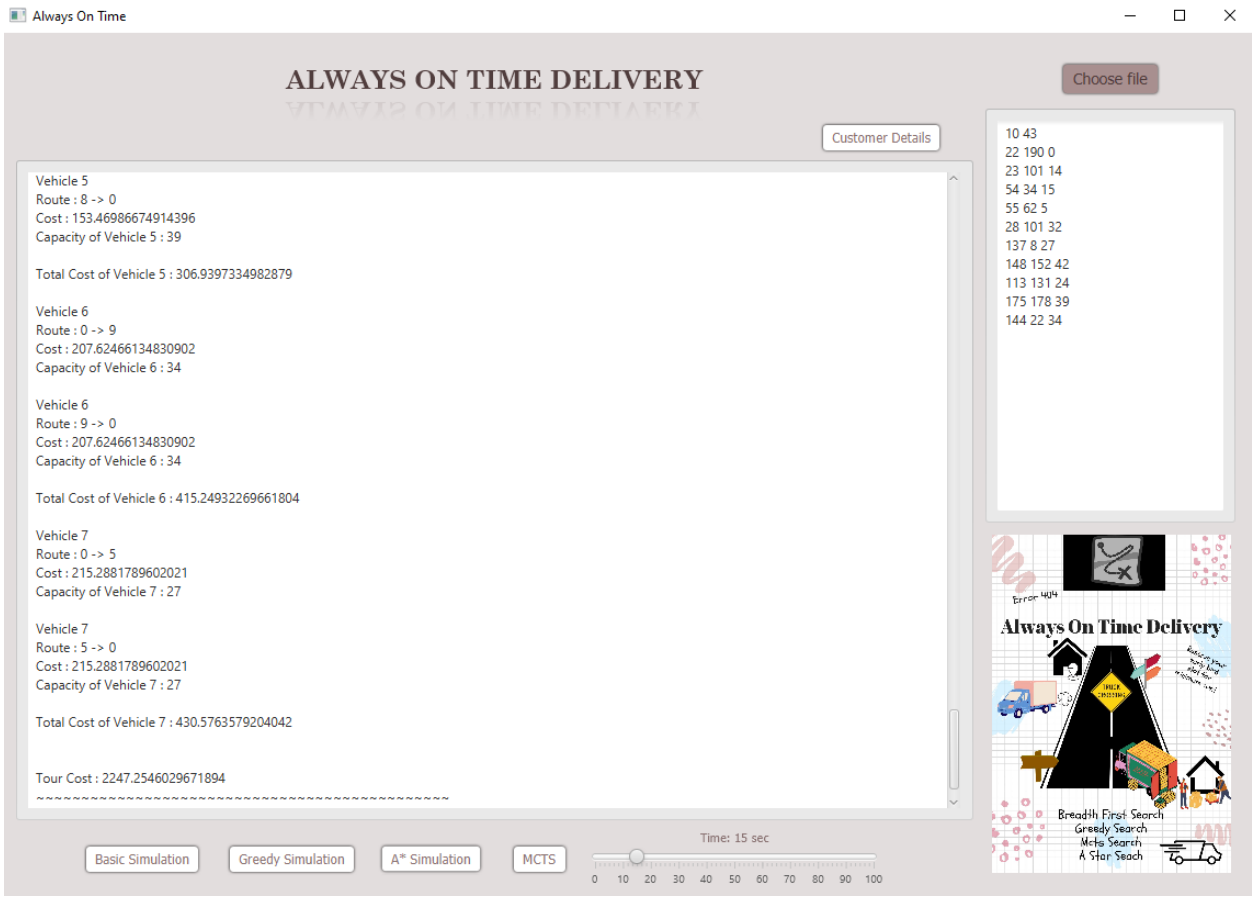

Always On Time
ALWAYS ON TIME DELIVERY
Choose file
Customer Details
Vehicle 2
Route: 7 -> 5
Cost: 125.31959144523253
Capacity: 27
Vehicle 2
Route: 5 -> 4 -> 0
Cost: 232.4849547910515
Total Capacity of Vehicle 2 : 43
Total cost of Vehicle 2 : 466.2573009269787
Vehicle 3
Route: 0 -> 4
Cost: 89.20201791439474
Capacity: 31
Vehicle 3
Route: 4 -> 9 -> 0
Cost: 347.97066236009414
Total Capacity of Vehicle 3 : 43
Total cost of Vehicle 3 : 437.17268027448887
Vehicle 4
Route: 0 -> 9
Cost: 207.62466134830902
Capacity: 22
Vehicle 4
Route: 9 -> 8 -> 0
Cost: 312.52017325623626
Total Capacity of Vehicle 4 : 43
Total cost of Vehicle 4 : 520.1448346045453
10 43
22 190 0
23 101 14
54 34 15
55 62 5
28 101 32
137 8 27
148 152 42
113 131 24
175 178 39
144 22 34
Always On Time Delivery
Breadth First Search
Greedy Search
Mcts Search
A Star Seach
Basic Simulation
Greedy Simulation
A* Simulation
MCTS
Time: 15 sec
0 10 20 30 40 50 60 70 80 90 100


Always On Time
ALWAYS ON TIME DELIVERY
Choose file
Customer Details
Cost: 207.62466134830902
Capacity: 22
Vehicle 4
Route: 9 -> 8 -> 0
Cost: 312.52017325623626
Total Capacity of Vehicle 4 : 43
Total cost of Vehicle 4 : 520.1448346045453
Vehicle 5
Route: 0 -> 8
Cost: 153.46986674914396
Capacity: 18
Vehicle 5
Route: 8 -> 6 -> 0
Cost: 169.08880063887943
Total Capacity of Vehicle 5 : 43
Total cost of Vehicle 5 : 322.5586673880234
Vehicle 6
Route: 0 -> 6
Cost: 131.60547101089682
Capacity: 17
Vehicle 6
Route: 6 -> 0
Cost: 131.60547101089682
Total Capacity of Vehicle 6 : 17
Total cost of Vehicle 6 : 263.21094202179364
Tour cost: 2487.788899818912
~~~~~~
10 43
22 190 0
23 101 14
54 34 15
55 62 5
28 101 32
137 8 27
148 152 42
113 131 24
175 178 39
144 22 34
Always On Time Delivery
Breadth First Search
Greedy Search
Mcts Search
A Star Seach
Basic Simulation
Greedy Simulation
A* Simulation
MCTS
Time: 15 sec
0 10 20 30 40 50 60 70 80 90 100
~~~~~~

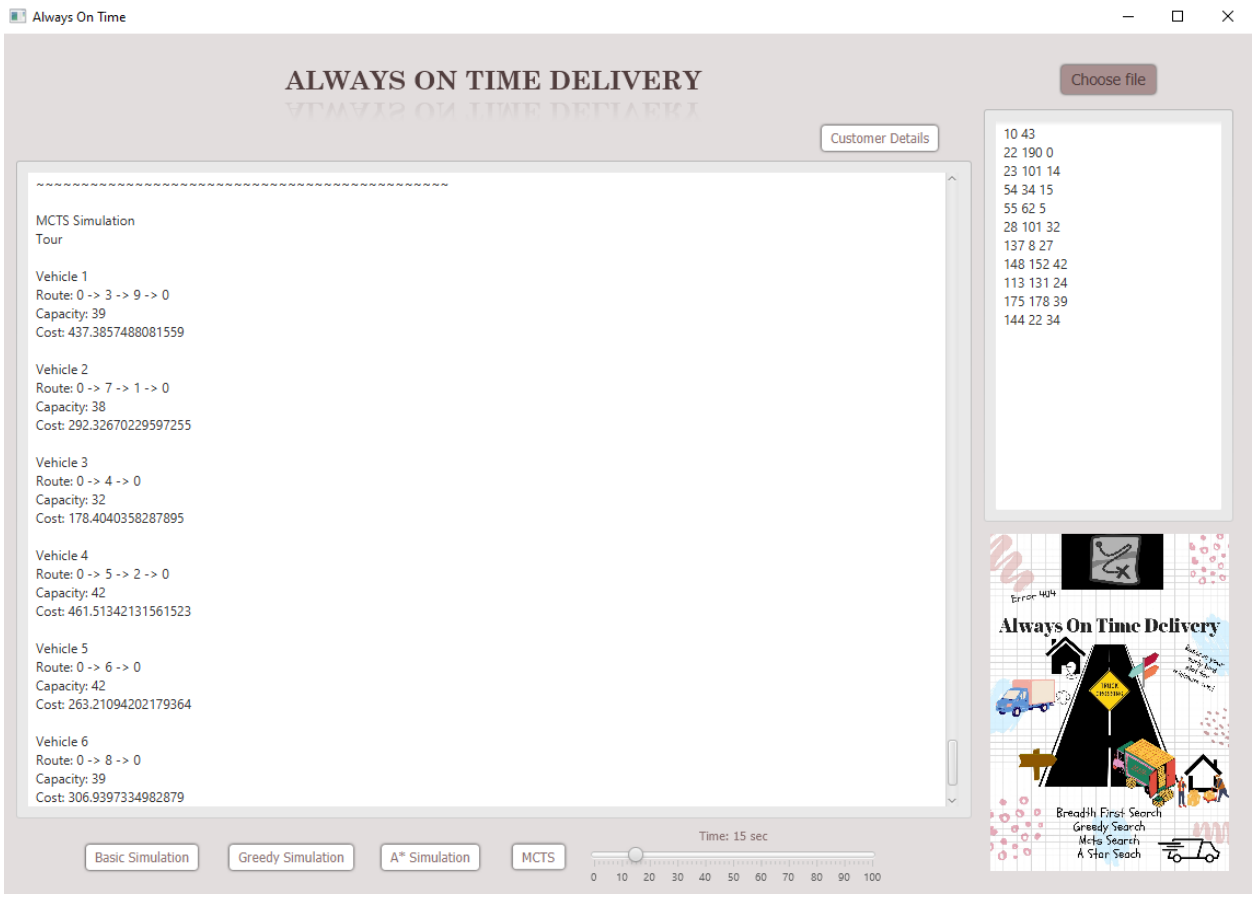
A* Simulation:

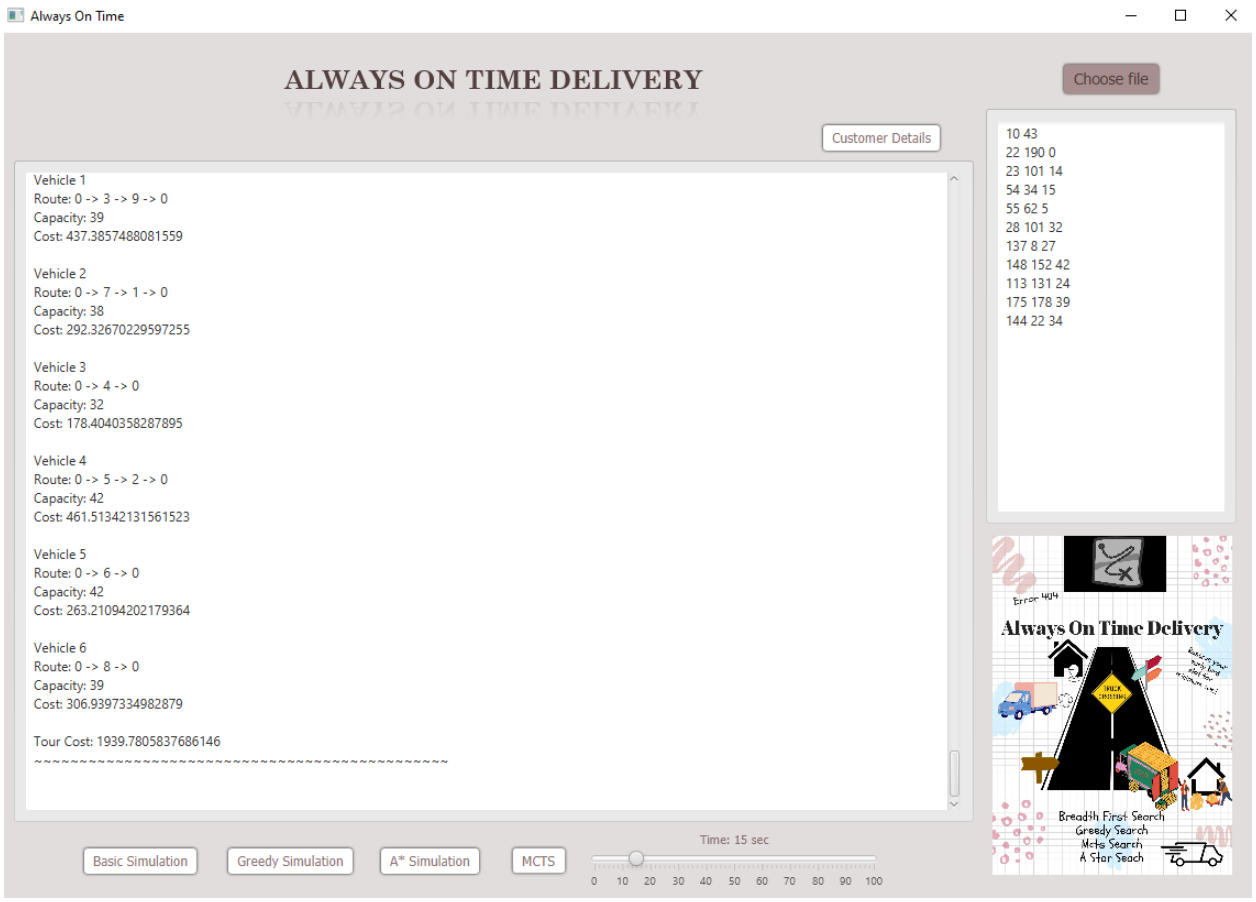



Always On Time
ALWAYS ON TIME DELIVERY
Choose file
Customer Details
Total Cost of Vehicle 2 : 178.4040358287895
Vehicle 3
Route : 0 -> 7
Cost : 108.45275469069469
Capacity of Vehicle 3 : 24
Vehicle 3
Route : 7 -> 2
Cost : 113.53413583587978
Capacity of Vehicle 3 : 39
Vehicle 3
Route : 2 -> 0
Cost : 159.24823389915505
Capacity of Vehicle 3 : 39
Total Cost of Vehicle 3 : 381.2351244257295
Vehicle 4
Route : 0 -> 6
Cost : 131.60547101089682
Capacity of Vehicle 4 : 42
Vehicle 4
Route : 6 -> 0
Cost : 131.60547101089682
Capacity of Vehicle 4 : 42
Total Cost of Vehicle 4 : 263.21094202179364
Vehicle 5
Route : 0 -> 8
10 43
22 190 0
23 101 14
54 34 15
55 62 5
28 101 32
137 8 27
148 152 42
113 131 24
175 178 39
144 22 34
Always On Time Delivery
Breadth First Search
Greedy Search
Mcts Search
A Star Seach
Basic Simulation
Greedy Simulation
A* Simulation
MCTS
Time: 15 sec
0 10 20 30 40 50 60 70 80 90 100


Always On Time
ALWAYS ON TIME DELIVERY
Choose file
Customer Details
Vehicle 5
Route : 8 -> 0
Cost : 153.46986674914396
Capacity of Vehicle 5 : 39
Total Cost of Vehicle 5 : 306.9397334982879
Vehicle 6
Route : 0 -> 9
Cost : 207.62466134830902
Capacity of Vehicle 6 : 34
Vehicle 6
Route : 9 -> 0
Cost : 207.62466134830902
Capacity of Vehicle 6 : 34
Total Cost of Vehicle 6 : 415.24932269661804
Vehicle 7
Route : 0 -> 5
Cost : 215.2881789602021
Capacity of Vehicle 7 : 27
Vehicle 7
Route : 5 -> 0
Cost : 215.2881789602021
Capacity of Vehicle 7 : 27
Total Cost of Vehicle 7 : 430.5763579204042
Tour Cost : 2247.2546029671894
10 43
22 190 0
23 101 14
54 34 15
55 62 5
28 101 32
137 8 27
148 152 42
113 131 24
175 178 39
144 22 34
Always On Time Delivery
Breadth First Search
Greedy Search
Mcts Search
A Star Seach
Basic Simulation
Greedy Simulation
A* Simulation
MCTS
Time: 15 sec
0 10 20 30 40 50 60 70 80 90 100

MCTS Simulation:




Always On Time
ALWAYS ON TIME DELIVERY
Choose file
Customer Details
Vehicle 1
Route: 0 -> 3 -> 9 -> 0
Capacity: 39
Cost: 437.3857488081559
Vehicle 2
Route: 0 -> 7 -> 1 -> 0
Capacity: 38
Cost: 292.32670229597255
Vehicle 3
Route: 0 -> 4 -> 0
Capacity: 32
Cost: 178.4040358287895
Vehicle 4
Route: 0 -> 5 -> 2 -> 0
Capacity: 42
Cost: 461.51342131561523
Vehicle 5
Route: 0 -> 6 -> 0
Capacity: 42
Cost: 263.21094202179364
Vehicle 6
Route: 0 -> 8 -> 0
Capacity: 39
Cost: 306.9397334982879
Tour Cost: 1939.7805837686146
10 43
22 190 0
23 101 14
54 34 15
55 62 5
28 101 32
137 8 27
148 152 42
113 131 24
175 178 39
144 22 34
Always On Time Delivery
Breadth First Search
Greedy Search
Mcts Search
A Star Seach
Basic Simulation
Greedy Simulation
A* Simulation
MCTS
Time: 15 sec
0 10 20 30 40 50 60 70 80 90 100