UNIVERSITY OF MALAYA

WIA1002 DATA STRUCTURE

PROJECT 2 "ALWAYS ON TIME DELIVERY"

TECHNICAL REPORT

LECTURER NAME : MUHAMMAD SHAHREEZA SAFIRUZ BIN KASSIM

COURSE OCCURRENCE : 7

GROUP NAME. : JAVA SLAYER

STUDENT NAME	MATRIC NO.
FRANKIE LIM QI QUAN (Leader)	U2005263/1
GOH CHEE LAM	U2005382/1
HO ZHI YI	U2005261/1
SYED FAIQ ALI FAISAL	S2013598/1

Table of Contents:

1.0 Explanation of Assigned Task

Your 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 your friend's company are always lost in the middle of their delivery route and 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. Your friend tried out many ways to solve the problem but to no avail. Hence, you are your friend's last hope to save his company.

As a professional engineer, you are requested to simulate the delivery process and planning in a country to help your friend shorten their delivery time.

2.0 Task Requirements

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 so 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.

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

2.1 Basic Simulation

You are requested to find the best tour for a given case with small N using Breadth-First Search / Depth-First Search traversal implementation which you will learn in the class.

2.2 Greedy Simulation

You are requested to implement a Greedy Search that can find a good solution given a case with a 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. For illustration, in this context, the vehicle will always look for the shortest distance between the current location and all the next possible locations to select the next location to go.

2.3 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 a large N. Thus, we are going to implement another searching algorithm, which is known as Monte Carlo Tree Search.

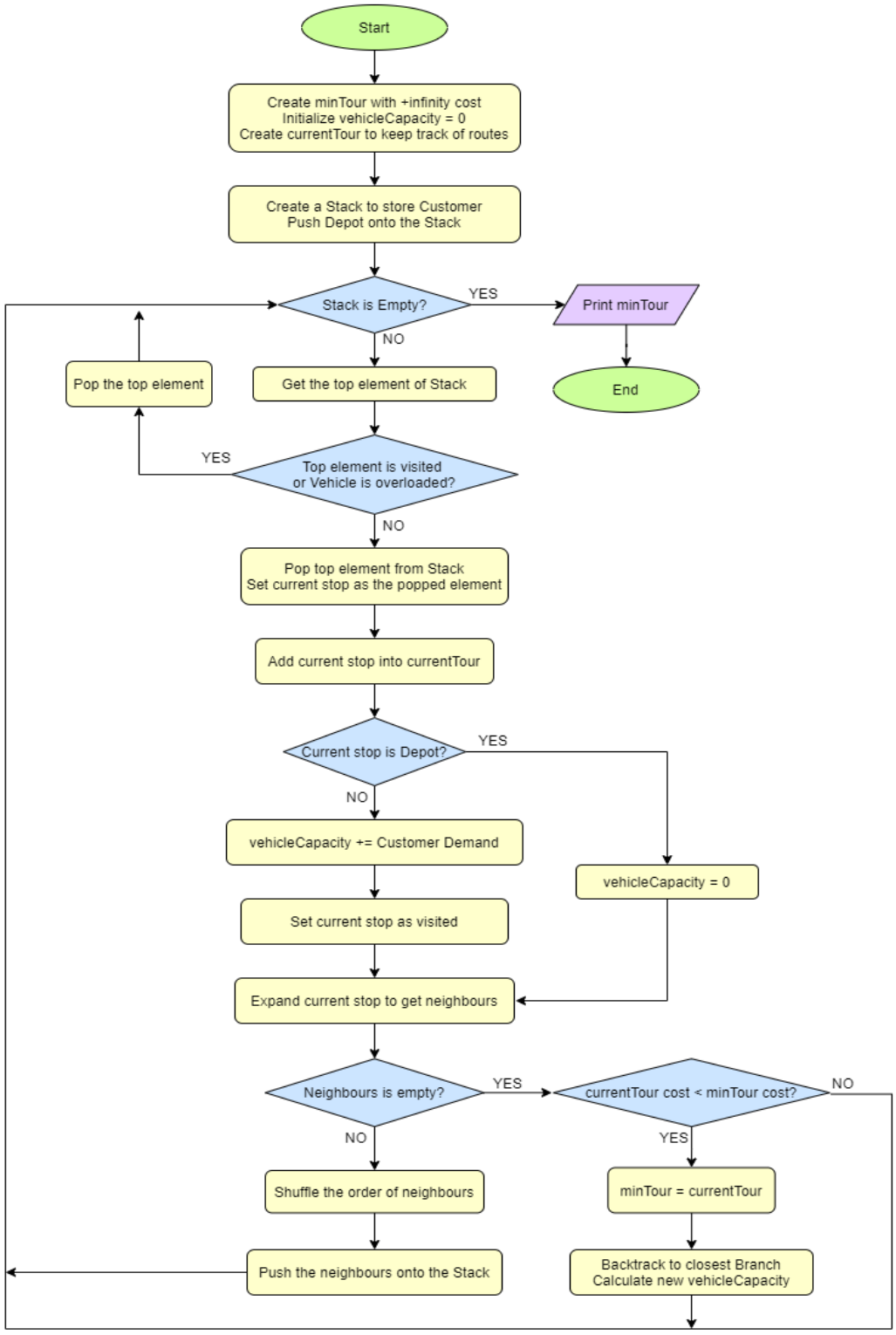
For every simulation, you need to show the tour's cost, followed by every used vehicle's information, including the route taken, the vehicle's used capacity, and the cost of the route taken.

3.0 Approaches Taken to Solve The Task

1) YouTube video
    a) To understanding how searching and other features of the graph work we watched Videos on Youtube to further enhance our understanding of the topic
2) GitHub issues (ask the NeverOnTime Sdn Bhd)
    a) Whenever we came across a problem we would often ask NeverOnTime Sdn Bhd, often he would answer and include additional info to help us further solve the problem. Other people's Issues on GitHub also helped us a lot with troubleshooting
3) Discussion among group members
    a) Whenever we would get stuck on a problem, we would send a message on the group, which the whole group would look at and then solve together
4) WIA1002 Lecture slides
    a) The basic building blocks of the program such as nodes and edges were made with the aid of the notes.
5) Programming websites (Stack Overflow, Javatpoint, GeeksforGeeks)
    a) Websites such as Javatpoint and GeeksforGeeks allowed us a better understanding of the topic at hand, by providing additional information about graphs and better and more advanced examples.
    b) Stack Overflow enables us to troubleshoot our program and often includes best practices.
6) Draw the simulation on a piece of paper to find out the logic and patterns
    a) Some problems can be better solved by breaking them down into their smaller parts. Drawing enabled us to do so, and also helped when we would explain the problem to each other.

4.0 Solution

4.1 Basic Simulation



For ***Basic Simulation***, we first initialize a ***tour*** with a positive infinity cost and a capacity for ***vehicle*** (namely ***vehicleCapacity***) as zero. Besides, a ***tour*** object named ***currentTour*** is created to store the valid ***routes*** taken and a ***stack*** to store ***Customer*** objects. Firstly, the ***Depot*** will be pushed into the ***Customer*** stack. A loop has started to check if the ***stack*** is empty. While it is not, it will carry out the following actions:

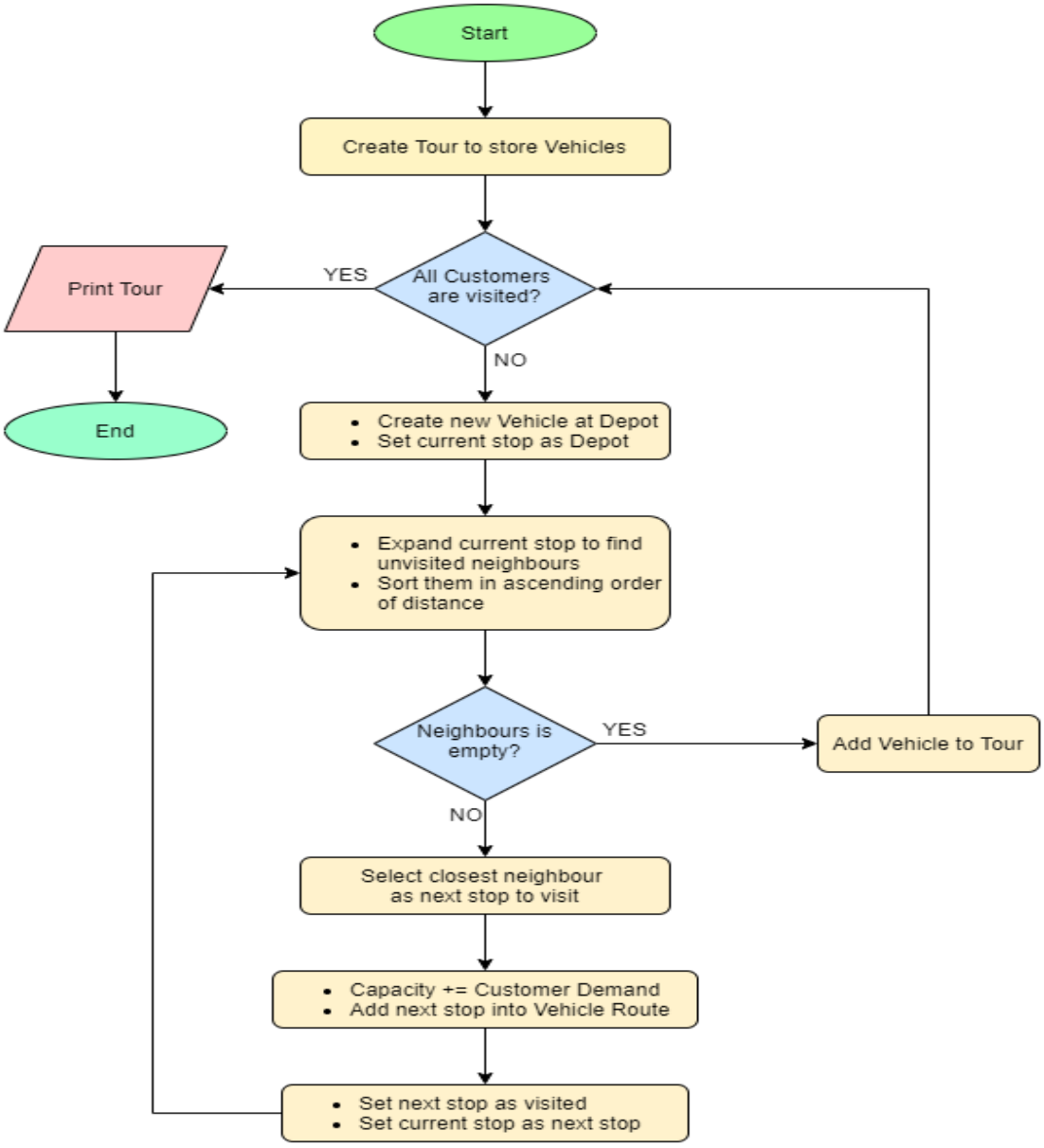
Obtain the top element of the ***Stack*** and check if it is a valid ***customer***. If it is visited or the ***vehicle*** will be overloaded by carrying it, it is not a valid ***customer*** and will be popped out of the ***stack***, returning to the state of checking if the ***stack*** is empty.

If the top element is a valid ***customer***, it is popped and set as the ***currentStop***. The ***currentStop*** will be added into ***currentTour***. If the ***currentStop*** is ***Depot***, then the ***vehicleCapacity*** will be set as zero and the program will continue with expanding the ***currentStop's*** neighbours. Otherwise, the ***vehicleCapacity*** will be incremented with the current ***Customer*** demand. The ***customer*** will be set as visited, only then the program proceeds with expanding its neighbours.

If the ***currentStop*** does not have any neighbours and its cost is less than the ***minTour's*** cost, we will update the ***minTour*** as the ***currentTour***. Then, backtracking to the closest branch is performed to calculate new ***vehicleCapacity***. However, if the ***currentStop*** does not have any neighbours and if the ***currentTour's*** cost is greater than the ***minTour's*** cost, then the program will head back to the very initial point where it checks if the stack is empty. On the other hand, if the ***currentStop*** has neighbours, the ***neighbouring*** customers will be pushed into ***Stack***, and the program returns to the initial checking - is the ***Stack*** empty?

Lastly, the while loop of checking if the ***stack*** is empty ends when the ***currentStop*** has no more possible unvisited ***customers***. The program will print out the ***minTour***.

4.2 Greedy Simulation



For ***Greedy Simulation***, a ***Tour*** object is created for the purpose of storing all valid ***Vehicle*** objects. Then, a while loop started to check if all ***Customer*** objects have been visited. If there are still unvisited ***Customers***, the program initializes a ***Vehicle*** object with its first stop set as the ***Depot***. Then, a ***Customer*** object is initialized to

temporarily hold the current stop the ***Vehicle*** is at. In this case, the ***currentStop*** will be ***Depot***.

Then, the ***currentStop*** will be expanded to obtain its unvisited neighbours. All the unvisited neighbours are stored in a list, and will be sorted in ascending order based on their route costs.

With the list of unvisited neighbours created, the program checks if the list is empty. if it is not empty, the program selects the closest neighbour as the next stop to visit. The capacity of the vehicle will be incremented with the demand of the current ***Customer***. The current stop will be updated to become the next stop.

However, if the list of unvisited neighbours is empty, the ***Vehicle*** will be added to ***Tour***, returning the program to the initial checking to determine if all ***customers*** are visited. If all are visited, the ***Tour*** will be printed and the program ends.

4.3 MCTS Simulation

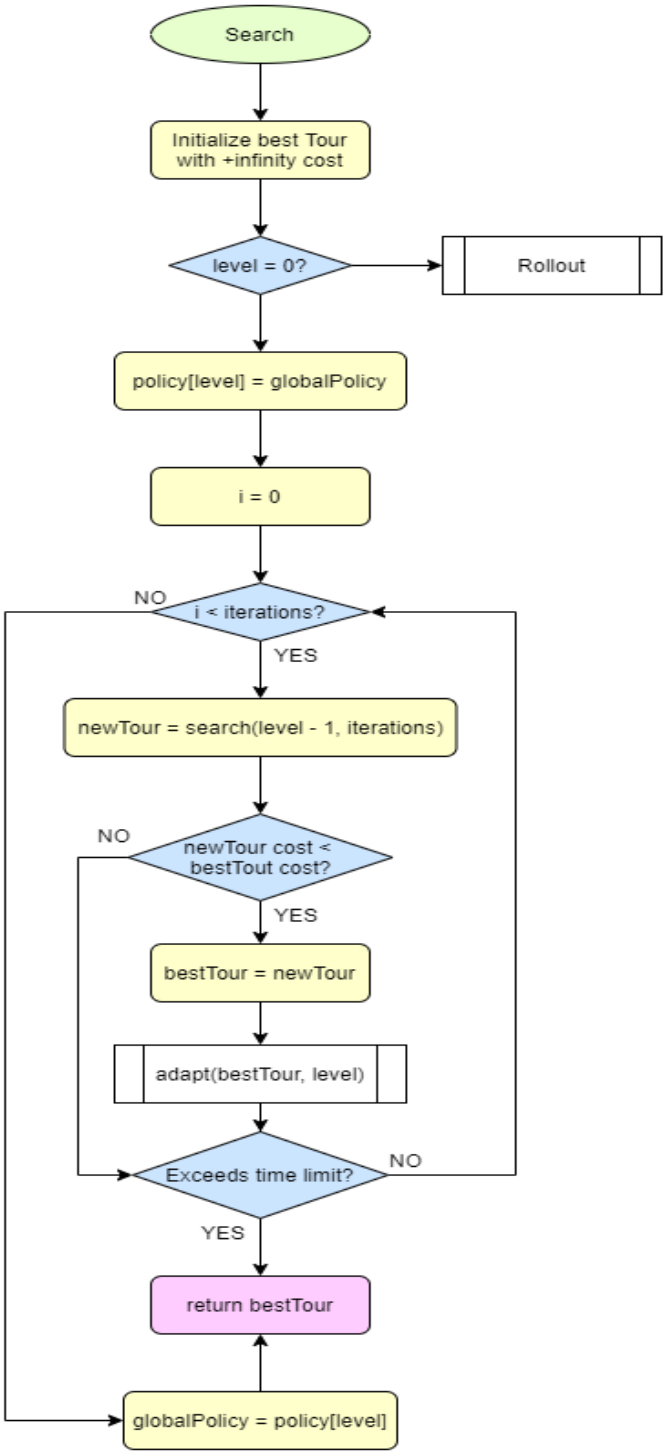
The ***Greedy Simulation*** has four methods – ***Search***, ***Rollout***, ***Select Next Move***, and ***Adapt***.

- **Search Flowchart**

Search method begins with initializing the best ***Tour*** with a positive infinity cost. Then, a while loop starts by checking if the level has reached zero or not. While not, the ***policy*** for the current level will be updated with the ***globalPolicy***. Then, an iterative loop starts. A ***newTour*** is initialized by calling the search method. If the cost of the ***newTour*** is less than the ***bestTour***, the ***bestTour*** will be updated with the ***newTour***. The ***adapt*** method will be called.

If the time limit has been reached, the ***bestTour*** will be returned and printed out. This ends the MCTS search.

Search Flowchart

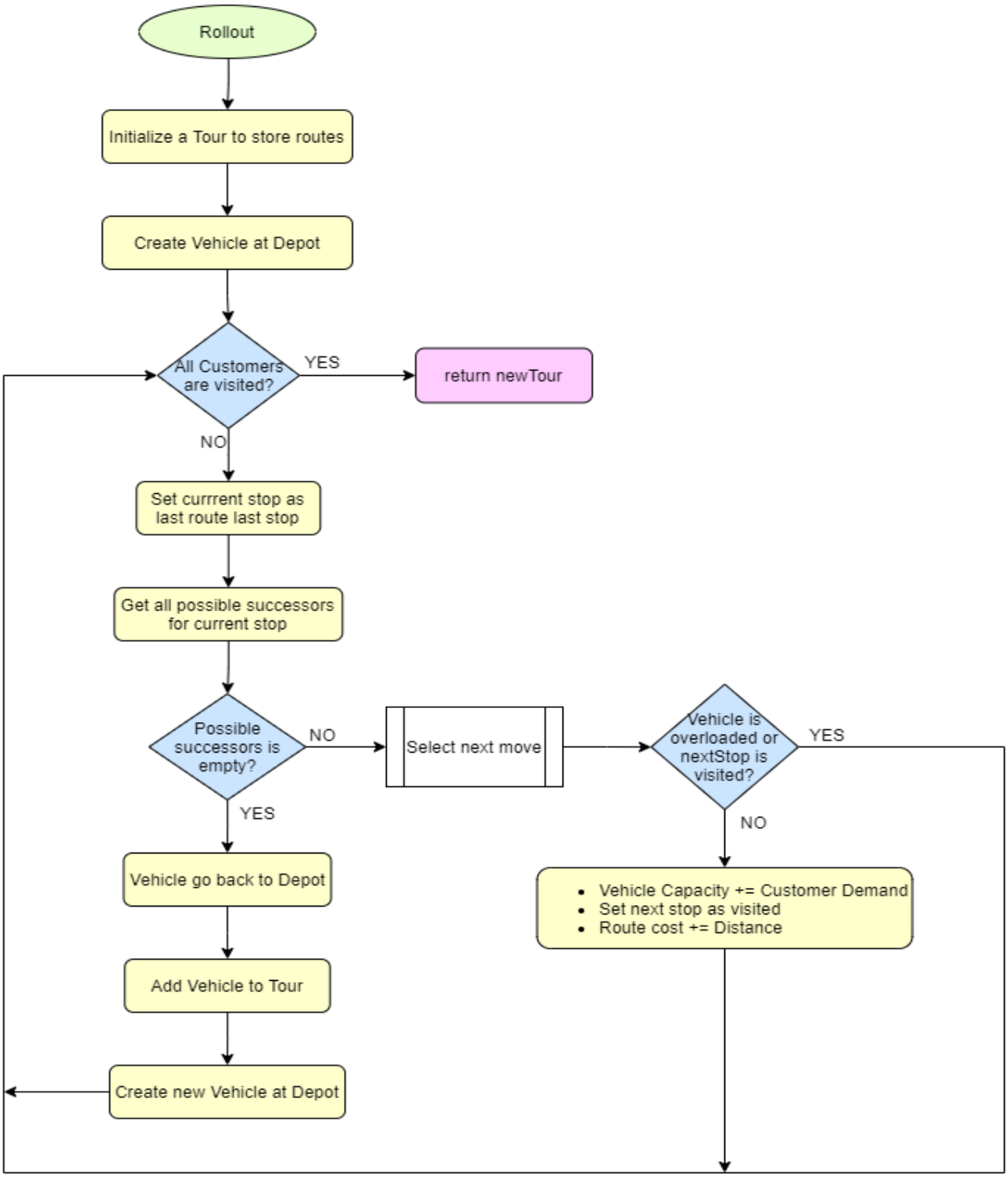


- **Rollout Flowchart**

For the ***Rollout*** method, a ***Tour*** object is initialized to store the routes taken. A ***Vehicle*** object is created with its first stop as the ***Depot***. A while loop starts to check if all customers have been visited. While not, the program initializes the current stop as the last stop of the last route. All possible successors of the current stop will be obtained and stored in a list. If the list is empty, then there are no more successors and the ***Vehicle*** will head back to ***Depot***. The ***Vehicle*** is added into the ***Tour***, and the program returns to the initial checking to see if all customers are visited.

However, if there are still possible successors, the ***select_next_move*** method will be called to determine the next successors to choose. Then, the program checks if the next stop is a valid customer by checking the vehicle's available capacity and if the next stop has been visited or not. If the next stop is a valid customer, then the capacity of the Vehicle will be updated by incrementing it with the ***Customer's*** demand. This stop will be set as visited. The ***Route*** cost will be incremented with the cost to reach the stop too.

Rollout Flowchart



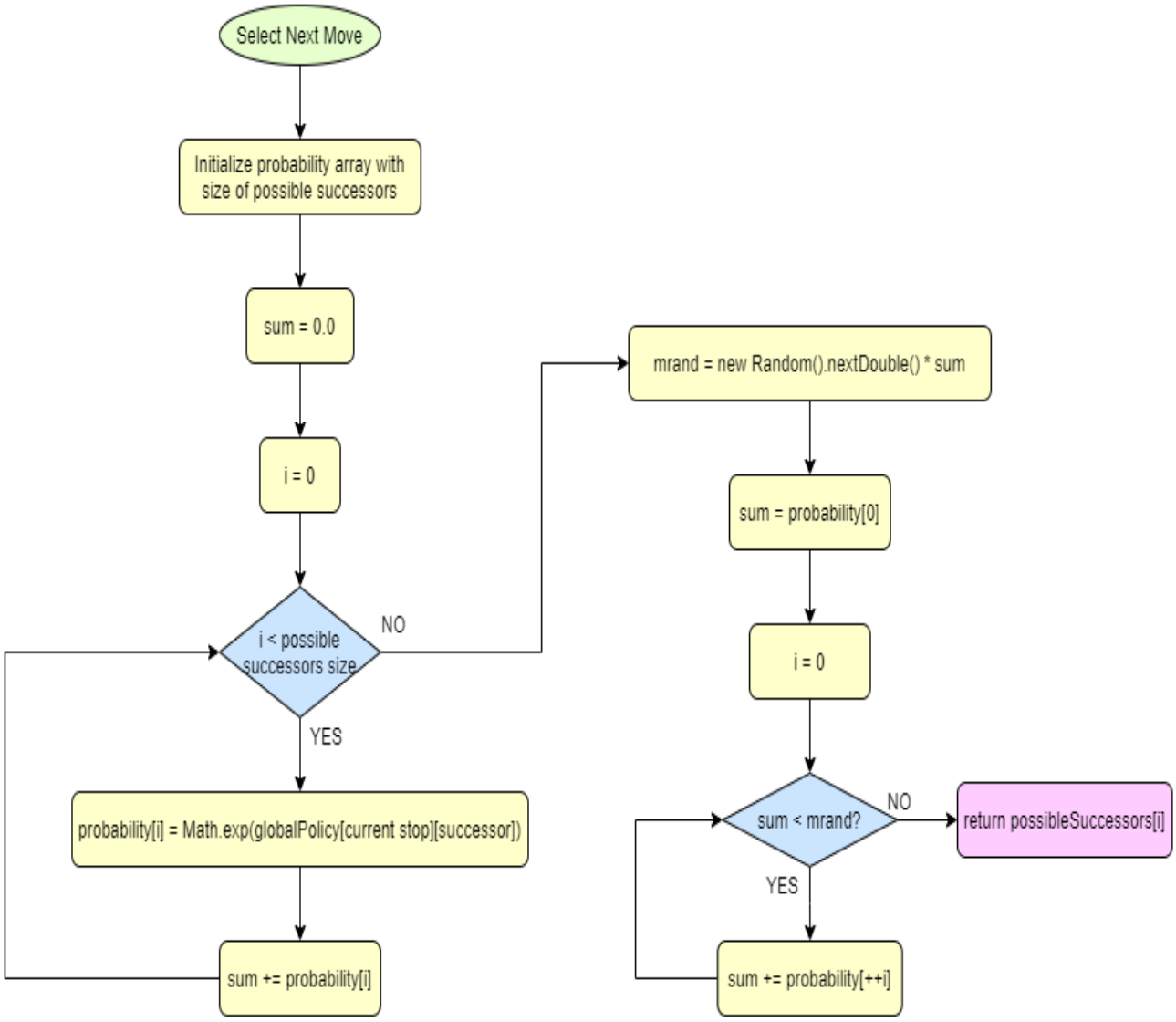
- **SelectNextMove Flowchart**

The ***Select Next Move*** method starts with initializing the ***probability*** array with the size of the number of possible successors. A double ***sum*** is initialized as 0.0. Then, an iterative loop starts with integer ***i*** as zero.

While ***i*** is less than the number of possible successors, the ***probability*** array will be updated with the exponent of the ***globalPolicy*** of the current stop's successor. The ***sum*** will then be incremented with the ***probability*** of ***i***.

If ***i*** is finally equal to the number of possible successors, a double value named ***mrand*** will be initialized as a random number of double type multiplied with the sum value. Then, ***i*** will be set to zero. The ***sum*** will be updated as the first value of the ***probability*** array. Finally, while ***sum*** is still less than ***mrand***, the value of ***sum*** will be incremented with the value of the element of the probability array. Afterwards, the element with index ***i*** in the list of possible successors will be returned.

SelectNextMove Flowchart



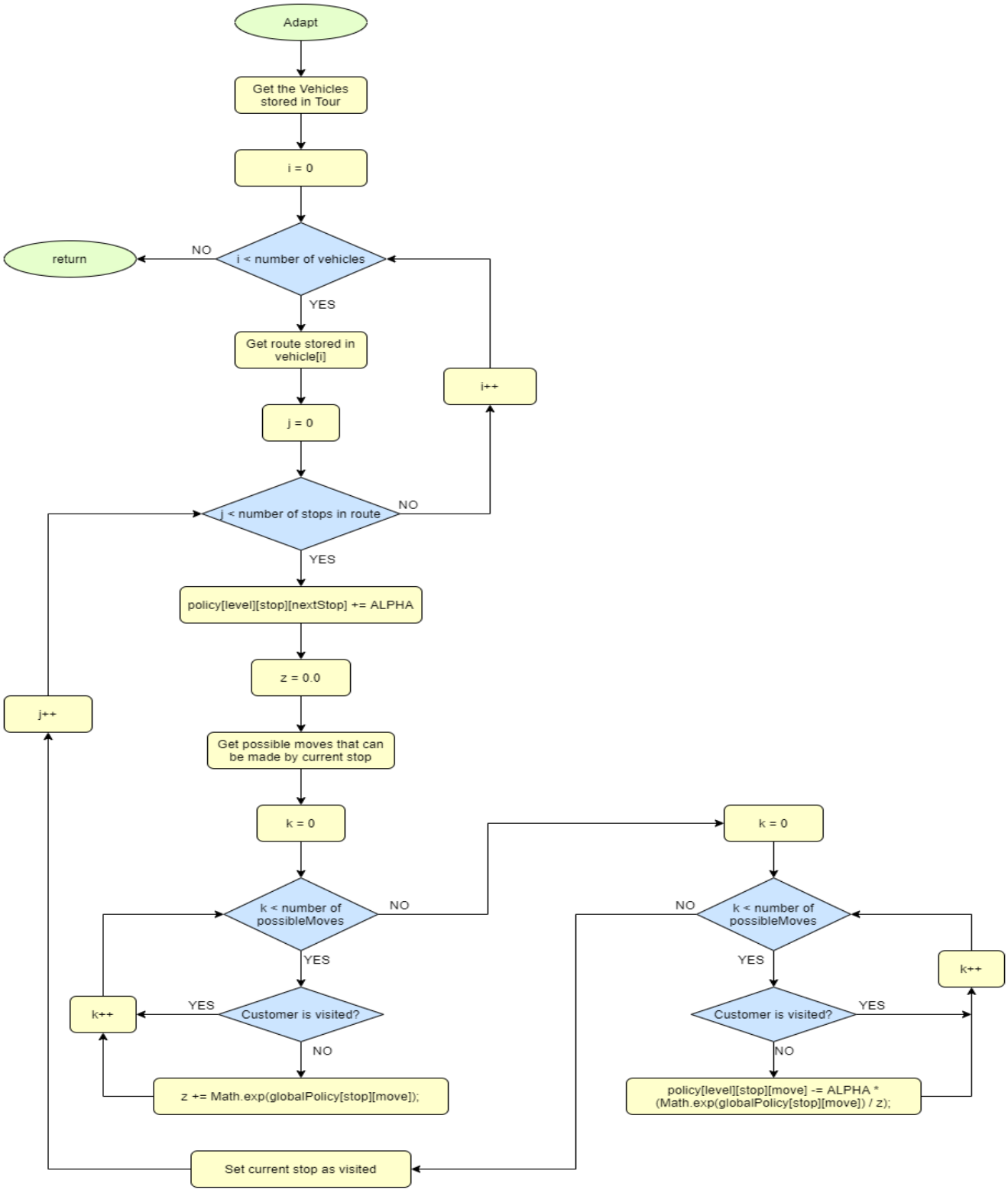
- **Adapt Flowchart**

The ***Adapt*** method will retrieve the ***Vehicle*** objects stored the ***Tour*** and then start an iterative loop with integer ***i*** as zero. While ***i*** is less than the number of ***Vehicles*** available, the ***route*** of the ***Vehicle*** will be retrieved. Another iterative loop with integer ***j*** as zero starts to check every stop in the ***route***.

For every ***stop*** in the ***route***, its ***policy*** value will be incremented with ***ALPHA***. A double value ***z*** will be initialized as zero. For every possible ***moves*** that could be made from the current ***stop***, if it is not yet visited, ***z*** will be incremented with the exponent of the ***globalPolicy*** of that unvisited ***stop***.

Again for every possible ***move*** that could be made from the current ***stop***, if it is not yet visited, the ***policy*** value of the move will be decremented with the value of ***ALPHA*** multiplied with the exponent of the value of the ***globalPolicy*** of that move divided with ***z***. Then, the ***stop*** will be set as visited.

Adapt Flowchart



5.0 Extra Features Implemented

5.1 Graphical User Interface (GUI)

We chose Swing which is a GUI widget toolkit for Java because it is platform-independent and easy to learn. The documentation is very useful for exploring the features in Swing. There are a large number of resources available on the internet to learn Swing. Since we are not familiar with GUI development, Swing will be a good framework for us to pick up.

We also found an external library for Java on GitHub which is **JFreeChart**. It provides all the necessary tools for creating an attractive chart. A scatterplot is used to visualize the Customer's house and depot while the line plot is used to visualize the route taken by the Vehicle to collect the demand from the Customers. The chart is pannable by pressing the Ctrl key followed by the mouse drag. The user also can zoom in and out of the chart by using the mouse wheel or pressing the Zoom buttons. Besides, the user can save the chart as a PNG image for future reference.

The user has the opportunity to configure the parameters before running the simulation such as choosing the input file containing the customer data, selecting the desired simulations (Basic, Greedy and MCTS), setting the time limit in seconds and changing the configuration parameters for MCTS simulation such as level, iterations and alpha (learning rate).

When running multiple simulations, the user can switch between the tabs to take a look at the searching progress for each simulation. The searching progress is shown as a progress bar where the time limit of searching is set by the user. The best tour found is illustrated as a combination of scatterplot and line plot.

5.2 Parallelism (Threading)

Parallelism is implemented where the user can select and run multiple simulations at the same time thus saving a lot of time. Each of the simulations is run as a Thread and stored inside an ArrayList to keep track of the searching progress.

As soon as a simulation is completed, its tour result will be shown as a chart. If the searching time exceeds the time limit set by the user, the searching is forced to stop and the best tour found is shown.

By implementing parallelism, it can save the user a lot of time because the user does not need to wait for the simulation to be run sequentially.

5.3 Music

When running the simulation, relaxing music is played so that the user can listen to the music. Hence, the user will not feel bored when waiting for each of the simulation to complete if the time limit set by the user is long.

5.4 Color

We use ANSI escape codes to change the font color of the text displayed in the terminal. Each simulation will have different colors to make it more easily to be observed by the user. Therefore, the user will not find it difficult to read the tour in the terminal.

5.5 Heterogeneous Vehicle Capacity

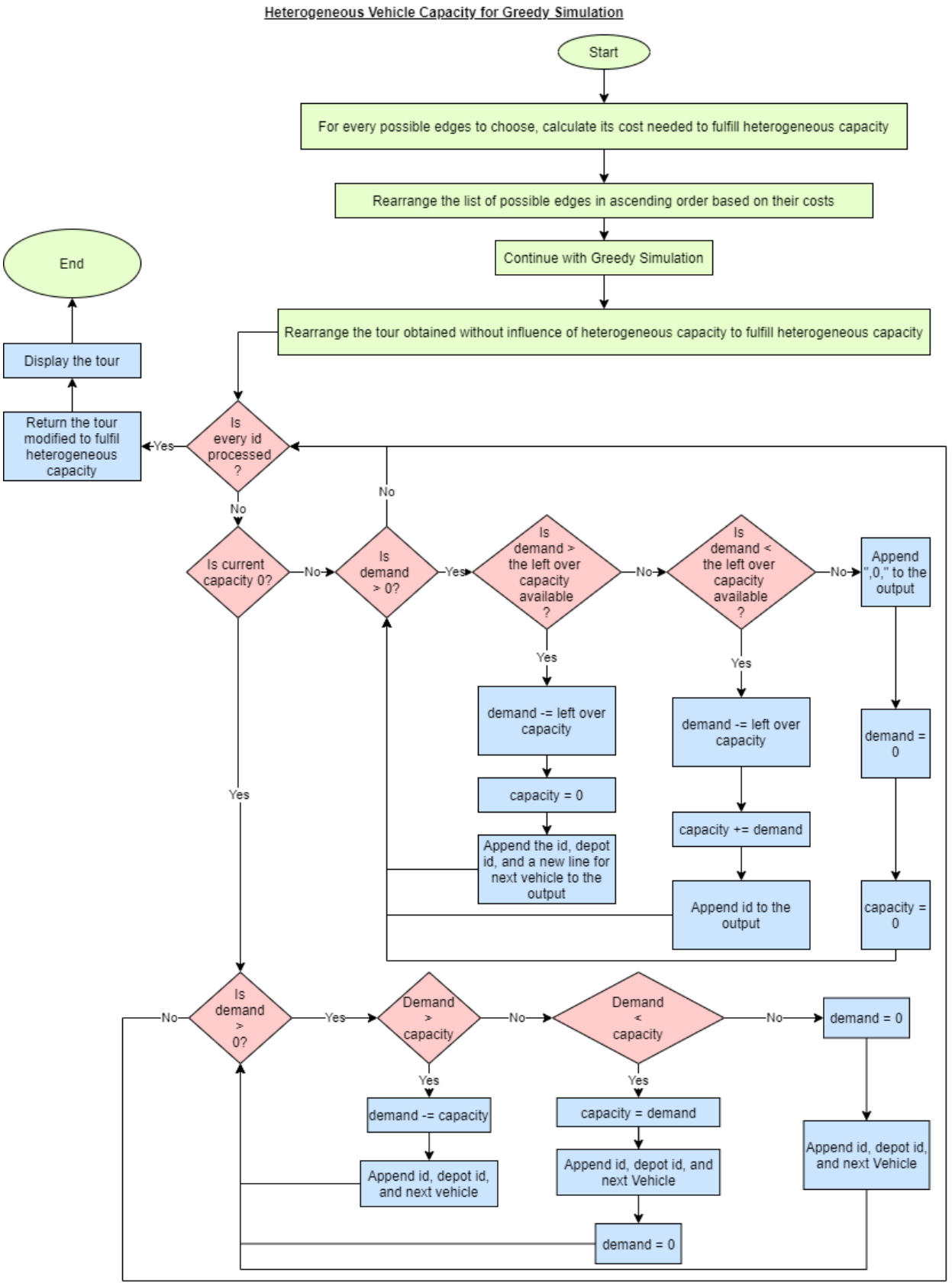
There are three types of vehicles - truck, van, and motorcycle, with the original capacity read from the input file, half of the original capacity read, and one third of the original capacity read respectively.

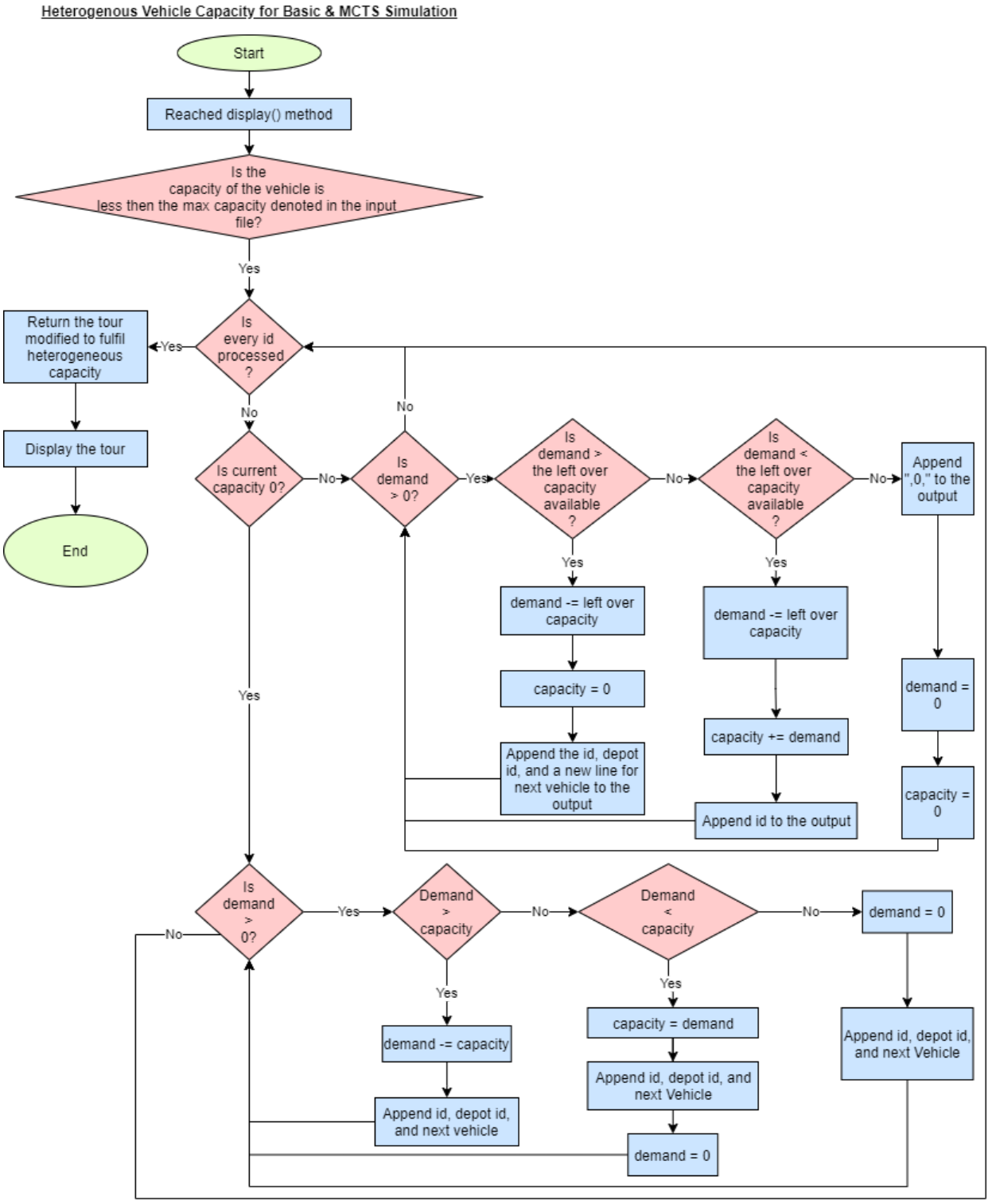
After implementing the ***heterogeneous vehicle capacity***, most of the demands of the customers will exceed the capacity of the vehicle itself. Therefore, the vehicle will have to come back and forth from the depot to the customer to pick up all parcels.

After finishing picking up all demands from the current customer, if there is still leftover capacity on the vehicle, it will continue to pick up the demand from the next customer.

To implement the ***heterogeneous vehicle capacity*** in ***Basic*** and ***MCTS*** simulation, the simulation will run as usual without the influence of the heterogeneous vehicle capacity. Before printing the result, only then the heterogeneous vehicle capacity kicks in. It will rearrange the Tour to fulfil the capacity which is less than the original capacity.

For ***Greedy*** Simulation, it will run as usual but when it retrieves the list of possible edges to choose, the ***heterogeneous vehicle capacity*** will influence the list. The list is sorted ascendingly based on the route cost, but the route cost differs if ***heterogeneous vehicle capacity*** occurs. So, the route cost for heterogeneous capacity will be calculated and the list will be sorted ascendingly with the modified route cost.


Heterogeneous Vehicle Capacity for Greedy Simulation
Start
For every possible edges to choose, calculate its cost needed to fulfill heterogeneous capacity
Rearrange the list of possible edges in ascending order based on their costs
Continue with Greedy Simulation
Rearrange the tour obtained without influence of heterogeneous capacity to fulfill heterogeneous capacity
Is every id processed ?
Yes
No
Return the tour modified to fulfil heterogeneous capacity
Display the tour
End
Is current capacity 0?
No
Yes
Is demand > 0?
No
Yes
Is demand > the left over capacity available ?
No
Yes
Is demand < the left over capacity available ?
No
Yes
Append ",0," to the output
demand -= left over capacity
capacity = 0
Append the id, depot id, and a new line for next vehicle to the output
demand -= left over capacity
capacity += demand
Append id to the output
demand = 0
capacity = 0
Is demand > 0?
No
Yes
Demand > capacity
No
Yes
Demand < capacity
No
Yes
demand = 0
demand -= capacity
Append id, depot id, and next vehicle
capacity = demand
Append id, depot id, and next Vehicle
Append id, depot id, and next Vehicle
demand = 0

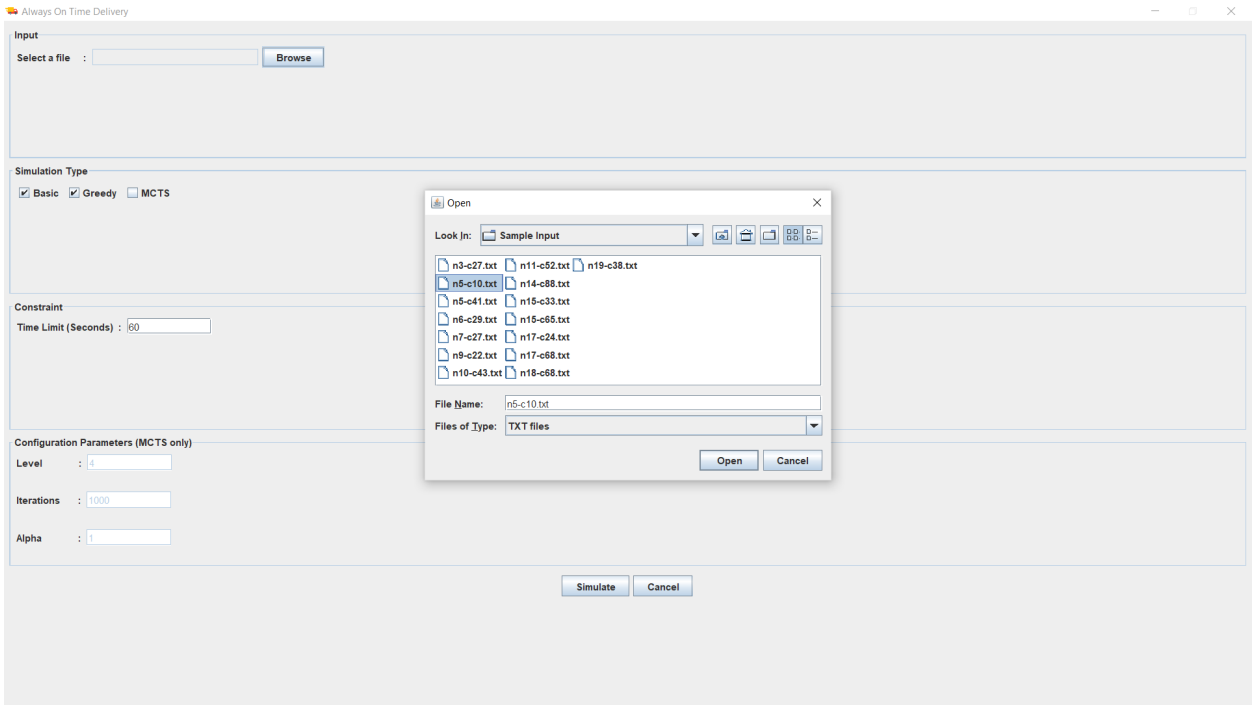

Heterogenous Vehicle Capacity for Basic & MCTS Simulation
Start
Reached display() method
Is the capacity of the vehicle is less then the max capacity denoted in the input file?
Yes
Is every id processed ?
Yes
Return the tour modified to fulfil heterogeneous capacity
Display the tour
End
No
Is current capacity 0?
No
Is demand > 0?
No
Yes
Is demand > the left over capacity available ?
No
Is demand < the left over capacity available ?
No
Append ",0," to the output
Yes
demand -= left over capacity
capacity = 0
Append the id, depot id, and a new line for next vehicle to the output
Yes
demand -= left over capacity
capacity += demand
Append id to the output
demand = 0
capacity = 0
Yes
Is demand > 0?
No
Yes
Demand > capacity
No
Demand < capacity
No
demand = 0
Yes
demand -= capacity
Append id, depot id, and next vehicle
Yes
capacity = demand
Append id, depot id, and next Vehicle
demand = 0
Append id, depot id, and next Vehicle

6.0 Sample Snapshots of Program Output

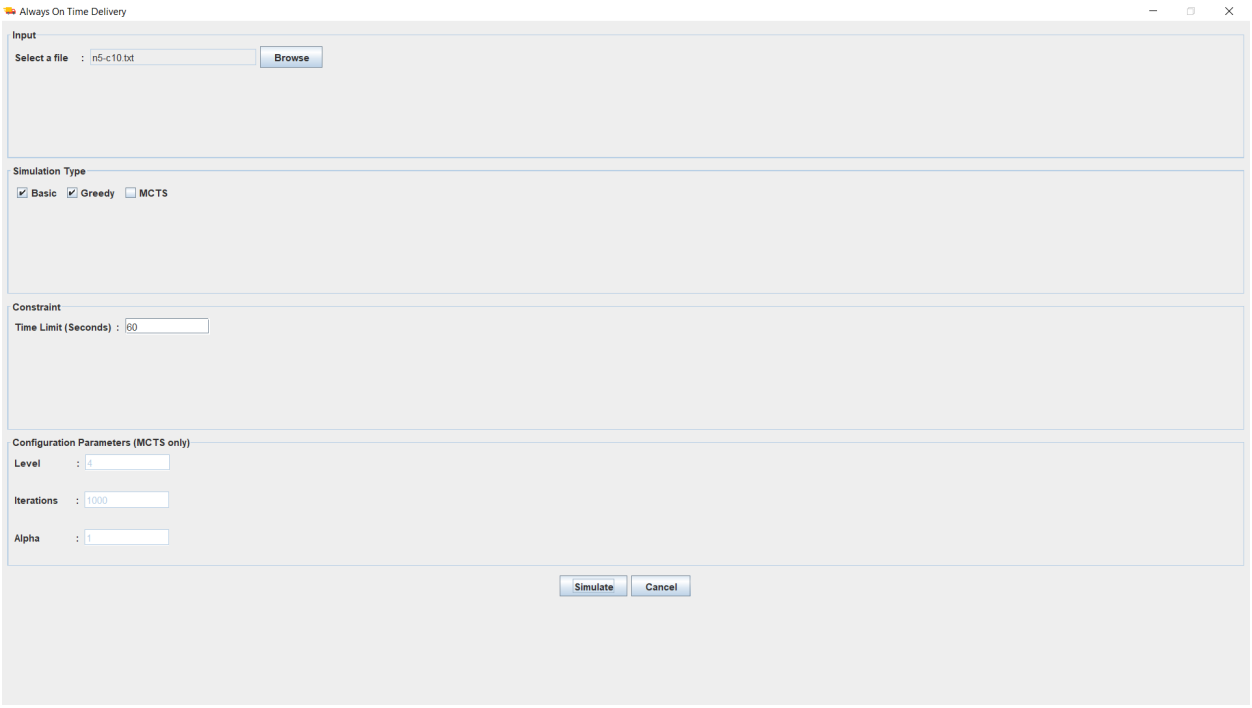
GUI

- **Main Menu with Start and Exit buttons**

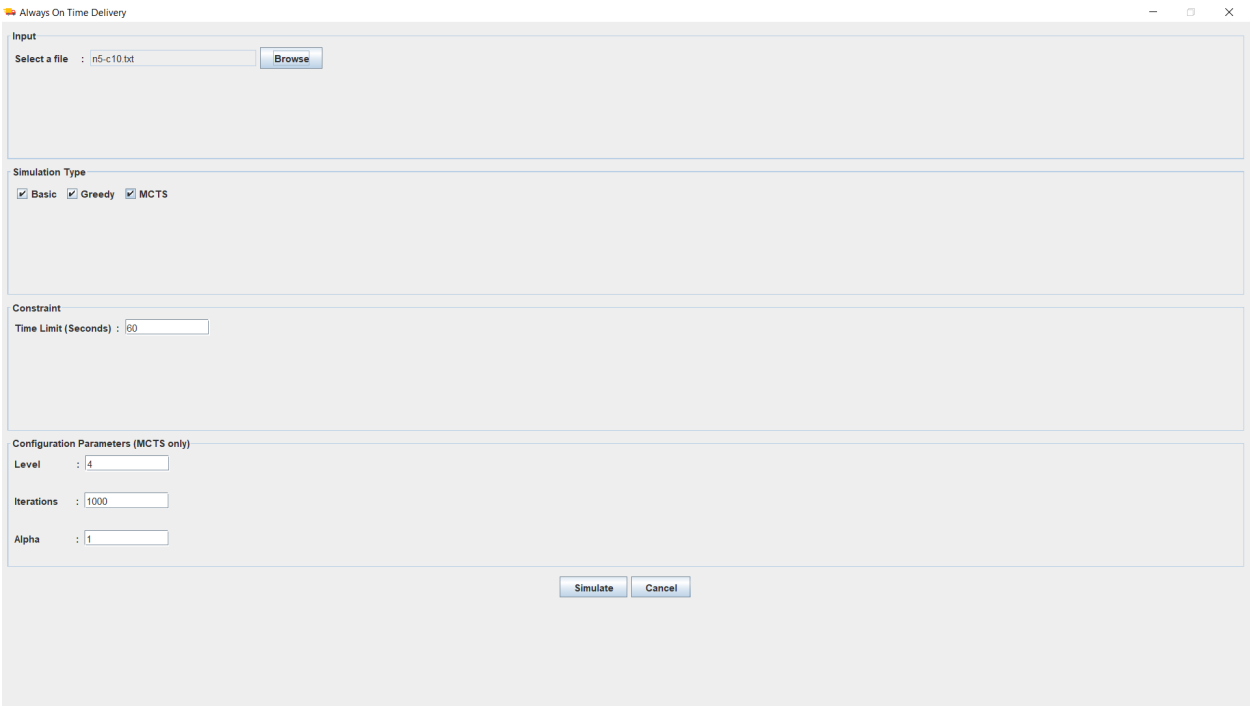
- **File Chooser to pick input file**



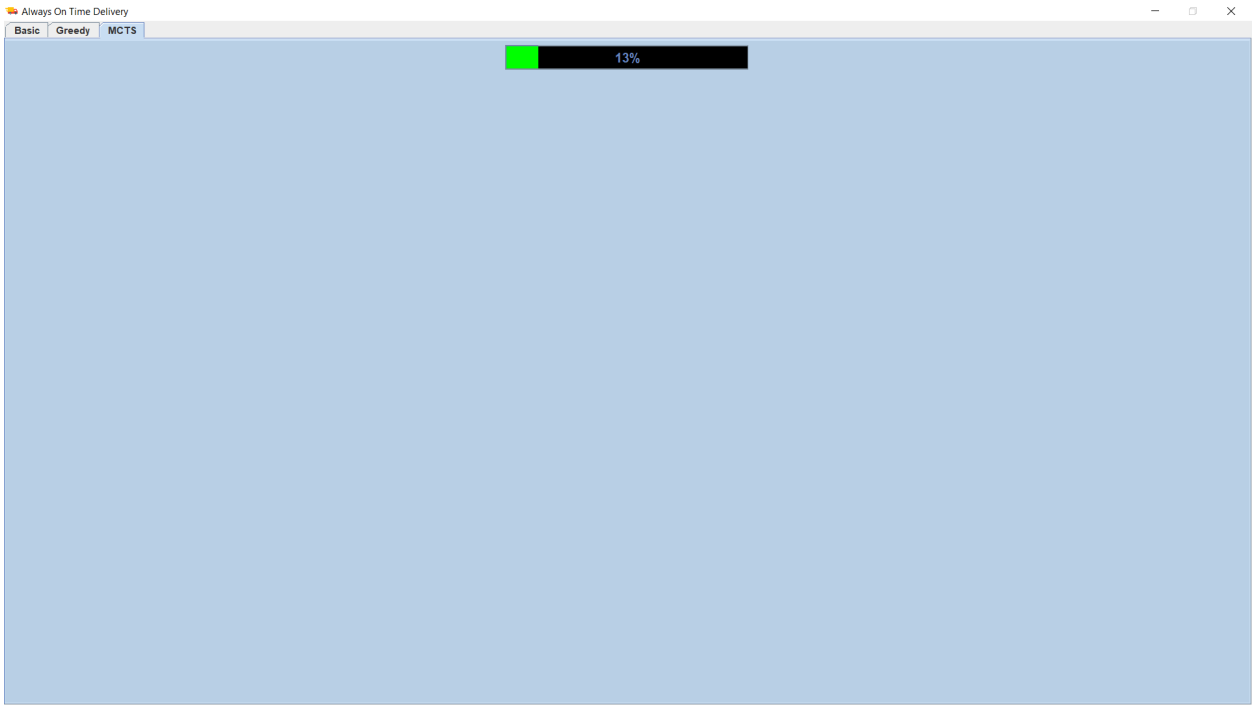
- **Searching Configurations (without MCTS simulation)**



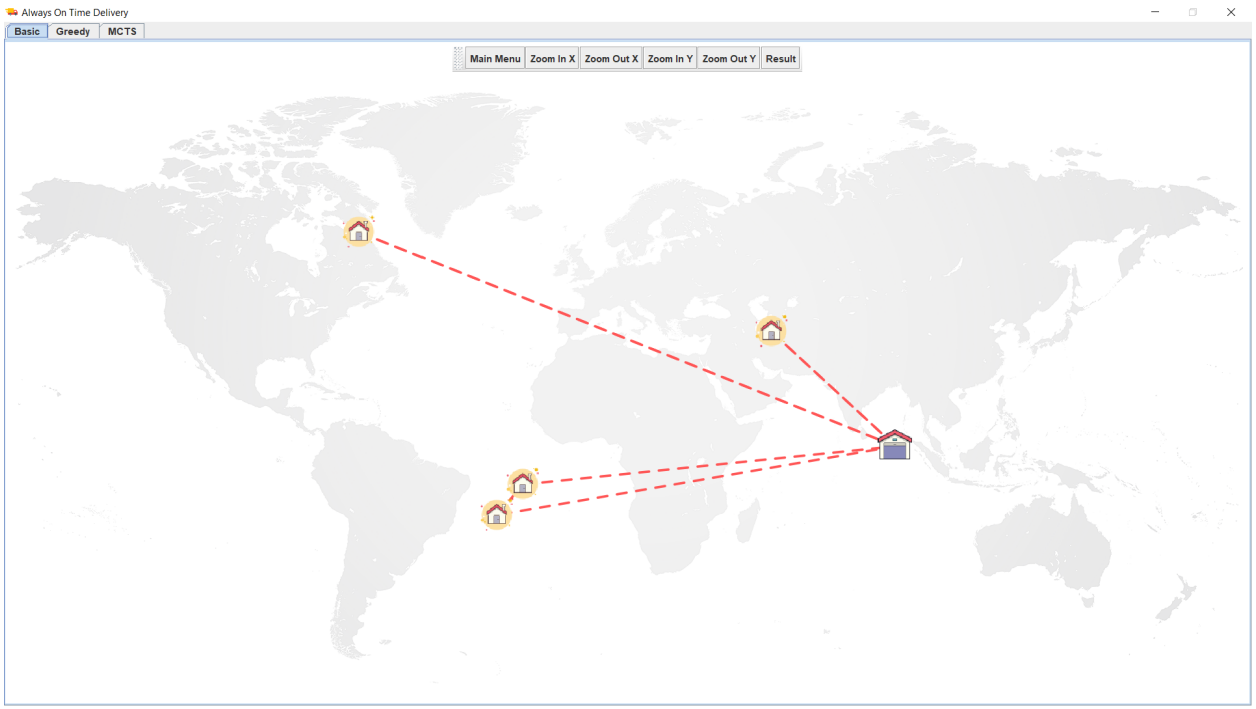
- **Searching Configurations (with MCTS simulation)**



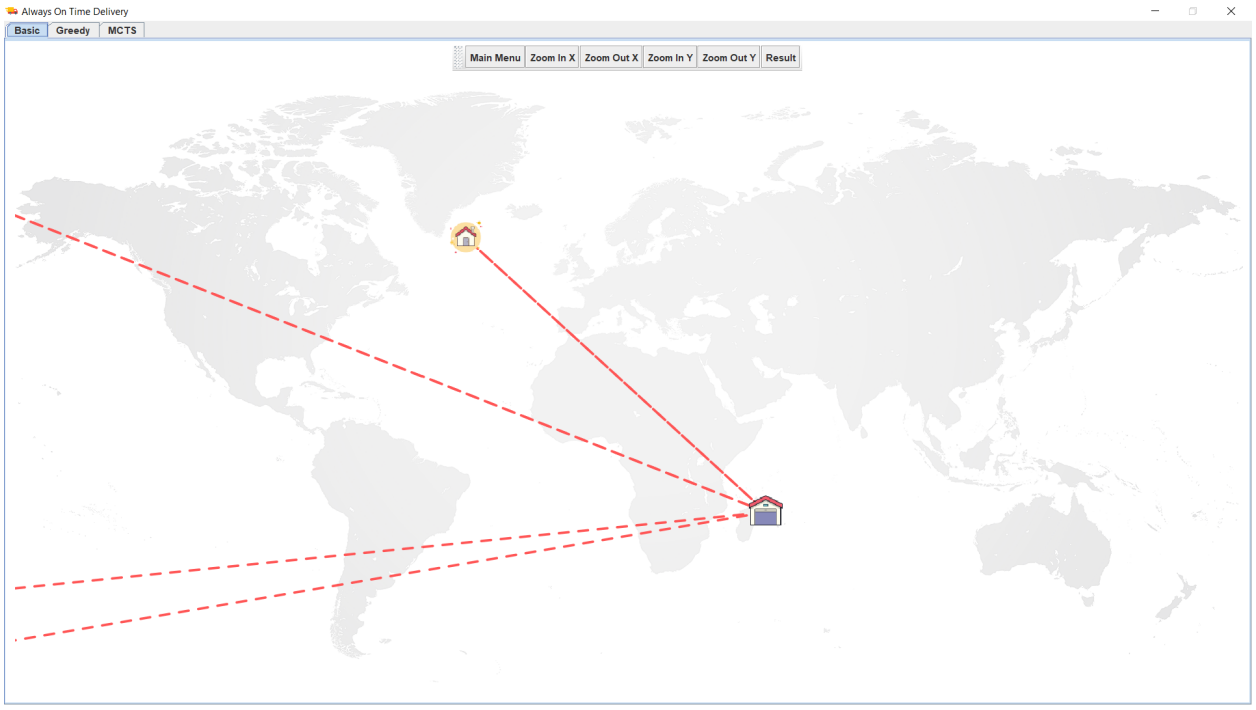
- **Progress bar for time elapsed**



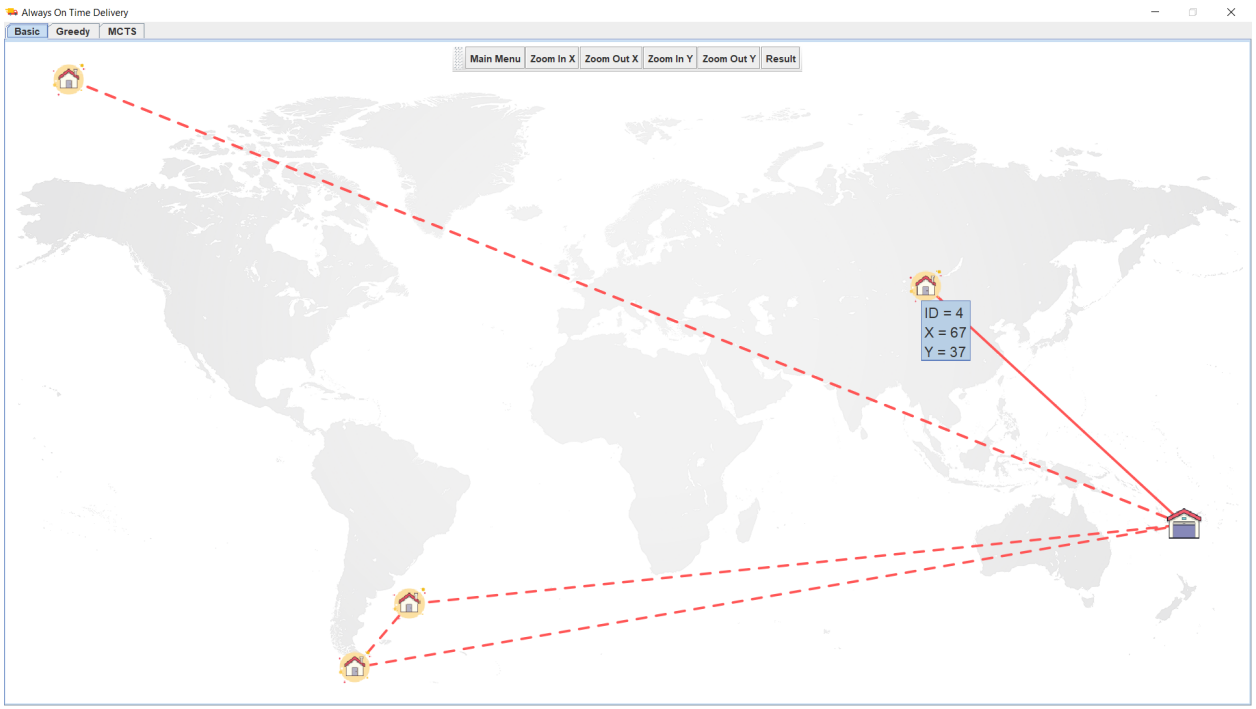
- **Zooming in and out by mouse wheel or Zoom buttons**



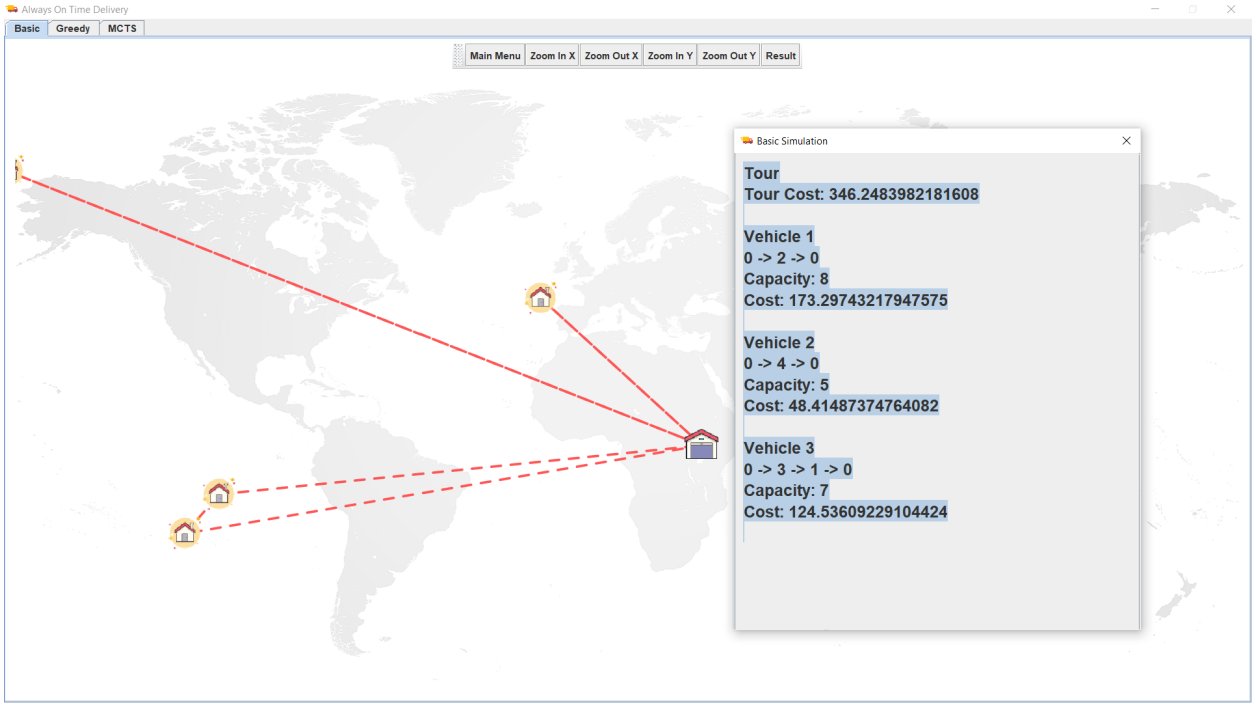
- **Pannable by pressing Ctrl key + mouse dragging**



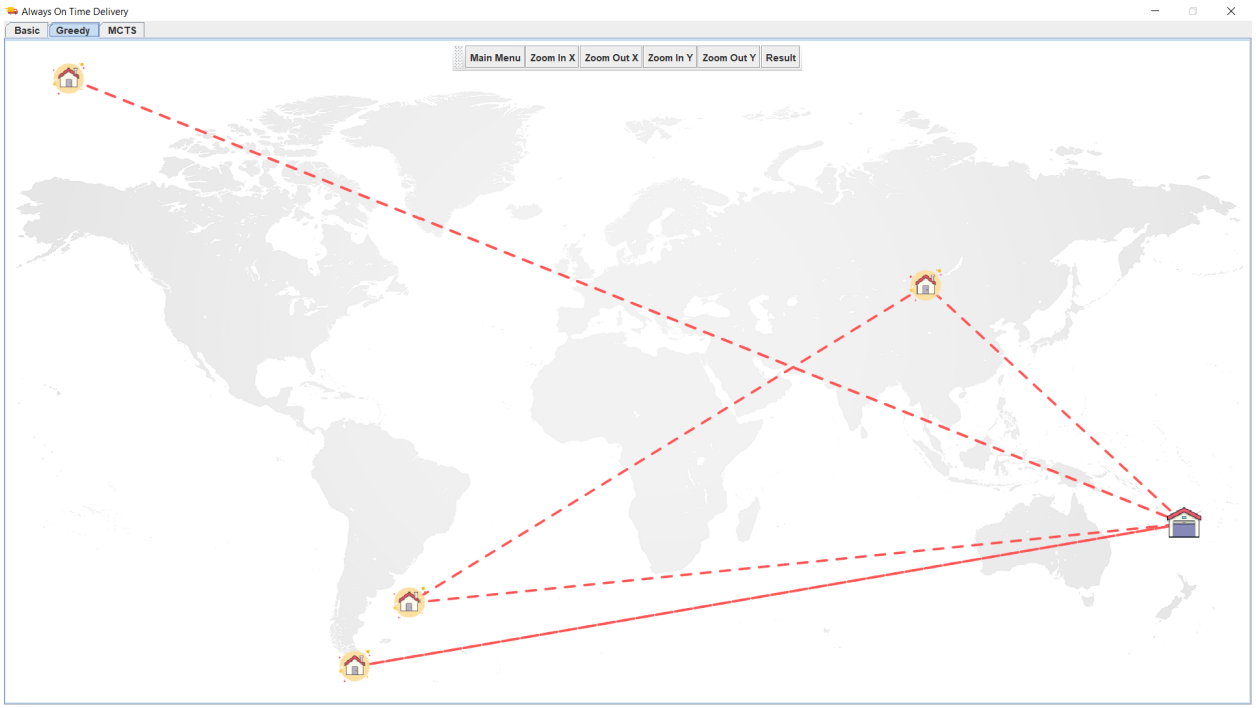
- **Tooltip on mouse hover**



- **Showing copyable tour result after clicking Result button**



- **Switching between tabs for multiple simulations**



- **Exit confirmation**

CLI (Heterogeneous Vehicle Capacity)

- **File n6-c29.txt (Motorcycle)**

```
[1] n10-c43.txt
[2] n11-c52.txt
[3] n14-c88.txt
[4] n15-c33.txt
[5] n15-c65.txt
[6] n17-c24.txt
[7] n17-c68.txt
[8] n18-c68.txt
[9] n19-c38.txt
[10] n3-c27.txt
[11] n5-c10.txt
[12] n5-c41.txt
[13] n6-c29.txt
[14] n7-c27.txt
[15] n9-c22.txt
Enter the file to read in (1 to 15): 13
```

```
Oh no... There's only motorcycles left in the hub.
I can only carry 9 items.

                       =======
            ______    //@@@@@||
           ______    //@@@@@@||        ==[###]
           ______    ||@@@@@@||          \\
                      ========   ~~~~     \\
           ______     ___________||      //\\
           ______     ___________\\     //  \\
          ______     //  *******     =====  **\\***
         ______     // ***********  //_//  ****\\****
         ______       *****________//     ************
                       ***********         **********
                         *********           ********
Basic Simulation
Tour
Tour Cost: 4984.64827753592
```

```
Vehicle 1
0 -> 2 -> 0
Capacity: 9
Cost: 279.29196193231195
Vehicle 2
0 -> 2 -> 0
Capacity: 9
Cost: 279.29196193231195
Vehicle 3
0 -> 2 -> 3 -> 4 -> 1 -> 0
Capacity: 9
Cost: 543.8711769653457
```

```
Vehicle 4
0 -> 1 -> 0
Capacity: 9
Cost: 331.78908963376114
Vehicle 5
0 -> 1 -> 5 -> 0
Capacity: 9
Cost: 373.9483820101427
Vehicle 6
0 -> 5 -> 0
Capacity: 9
Cost: 228.04385543136215
Vehicle 7
0 -> 5 -> 0
Capacity: 9
Cost: 228.04385543136215
Vehicle 8
0 -> 5 -> 0
Capacity: 3
Cost: 228.04385543136215

Greedy Simulation
Tour
Tour Cost: 5042.769969204218
Vehicle 1
0 -> 5 -> 0
Capacity: 9
Cost: 228.04385543136215
```

```
Vehicle 2
0 -> 5 -> 0
Capacity: 9
Cost: 228.04385543136215
Vehicle 3
0 -> 5 -> 4 -> 0
Capacity: 9
Cost: 254.46351094030695
Vehicle 4
0 -> 4 -> 3 -> 2 -> 0
Capacity: 9
Cost: 408.17005694824763
Vehicle 5
0 -> 2 -> 0
Capacity: 9
Cost: 279.29196193231195
Vehicle 6
0 -> 2 -> 1 -> 0
Capacity: 9
Cost: 459.7935646509953
Vehicle 7
0 -> 1 -> 0
Capacity: 9
Cost: 331.78908963376114
Vehicle 8
0 -> 1 -> 0
Capacity: 3
Cost: 331.78908963376114
```

```
Vehicle 2
0 -> 5 -> 0
Capacity: 9
Cost: 228.04385543136215
Vehicle 3
0 -> 5 -> 4 -> 0
Capacity: 9
Cost: 254.46351094030695
Vehicle 4
0 -> 4 -> 3 -> 2 -> 0
Capacity: 9
Cost: 408.17005694824763
Vehicle 5
0 -> 2 -> 0
Capacity: 9
Cost: 279.29196193231195
Vehicle 6
0 -> 2 -> 1 -> 0
Capacity: 9
Cost: 459.7935646509953
```

```
Vehicle 7
0 -> 1 -> 0
Capacity: 9
Cost: 331.78908963376114
Vehicle 8
0 -> 1 -> 0
Capacity: 3
Cost: 331.78908963376114
```

```
Time elapsed: |████████████████████████████████████████████████████████████| 10 s (Searching is forced to stop!)
MCTS Simulation
Tour
Tour Cost: 5018.050826737055
Vehicle 1
0 -> 4 -> 5 -> 0
Capacity: 9
Cost: 254.46351094030695
Vehicle 2
0 -> 5 -> 0
Capacity: 9
Cost: 228.04385543136215
Vehicle 3
0 -> 5 -> 0
Capacity: 9
Cost: 228.04385543136215
Vehicle 4
0 -> 5 -> 2 -> 0
Capacity: 9
Cost: 317.4801332160123
Vehicle 5
0 -> 2 -> 0
Capacity: 9
Cost: 279.29196193231195
Vehicle 6
0 -> 2 -> 3 -> 1 -> 0
Capacity: 9
Cost: 538.1239171496491
```


```
Vehicle 7
0 -> 1 -> 0
Capacity: 9
Cost: 331.78908963376114
Vehicle 8
0 -> 1 -> 0
Capacity: 3
Cost: 331.78908963376114
```

- **File n6-c29.txt (Van)**

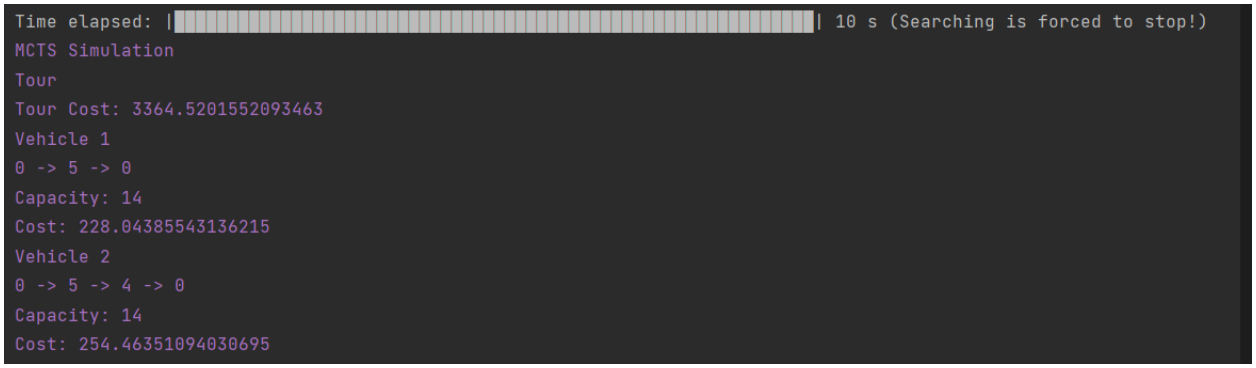
```
[1] n10-c43.txt
[2] n11-c52.txt
[3] n14-c88.txt
[4] n15-c33.txt
[5] n15-c65.txt
[6] n17-c24.txt
[7] n17-c68.txt
[8] n18-c68.txt
[9] n19-c38.txt
[10] n3-c27.txt
[11] n5-c10.txt
[12] n5-c41.txt
[13] n6-c29.txt
[14] n7-c27.txt
[15] n9-c22.txt
Enter the file to read in (1 to 15): 13
```

```
Phew, Although there's no truck, a van should suffice!
I can carry 14 items!

            ------------------------------------
            ||                                  \\
      ------ ||                                   \\
     ------  ||                                    \\
    ------   ||                                     \\
   ------    ||                                      \\
  ------     ||                                       ||
 ------      ||    *******                *******     ||
            [|  ***********             ***********    |]
            ==========================================
                ***********             ***********
                 ********                ********
Basic Simulation
Tour
Tour Cost: 3298.4562213556874
```

```
Vehicle 1
0 -> 1 -> 0
Capacity: 14
Cost: 331.78908963376114
Vehicle 2
0 -> 1 -> 4 -> 3 -> 2 -> 0
Capacity: 14
Cost: 543.8711769653459
Vehicle 3
0 -> 2 -> 5 -> 0
Capacity: 14
Cost: 317.4801332160123
Vehicle 4
0 -> 5 -> 0
Capacity: 14
Cost: 228.04385543136215
Vehicle 5
0 -> 5 -> 0
Capacity: 10
Cost: 228.04385543136215
```

```
Greedy Simulation
Tour
Tour Cost: 3364.5201552093463
Vehicle 1
0 -> 5 -> 0
Capacity: 14
Cost: 228.04385543136215
Vehicle 2
0 -> 5 -> 4 -> 0
Capacity: 14
Cost: 254.46351094030695
Vehicle 3
0 -> 4 -> 3 -> 2 -> 0
Capacity: 14
Cost: 408.17005694824763
Vehicle 4
0 -> 2 -> 1 -> 0
Capacity: 14
Cost: 459.7935646509953
Vehicle 5
0 -> 1 -> 0
Capacity: 10
Cost: 331.78908963376114
```

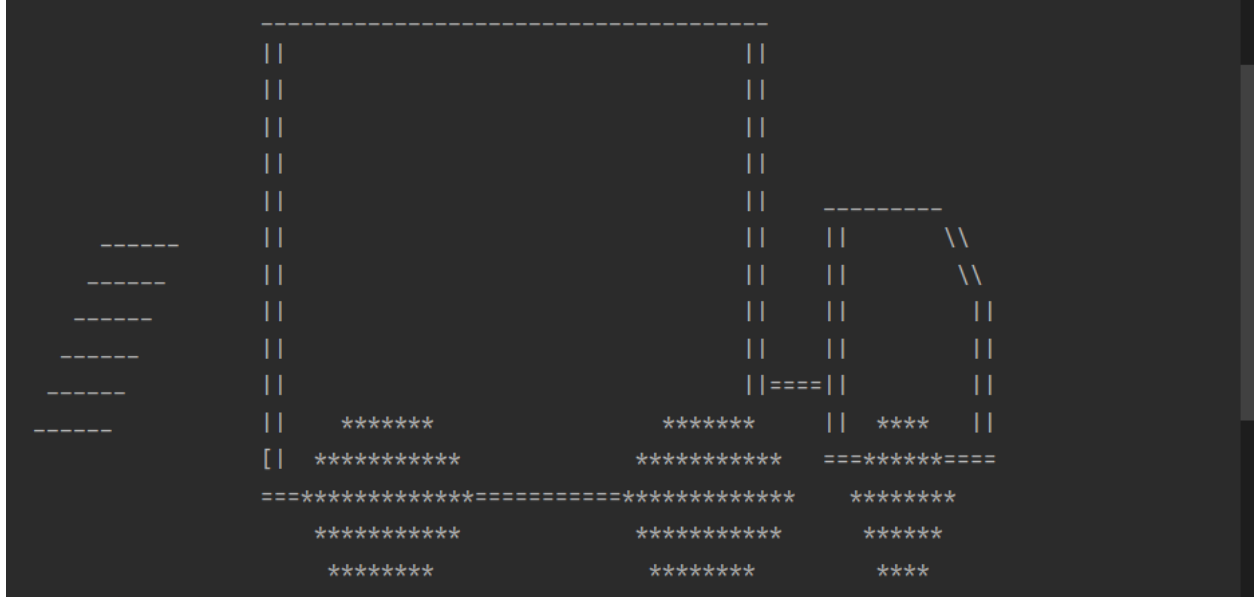

Time elapsed: | 10 s (Searching is forced to stop!)
MCTS Simulation
Tour
Tour Cost: 3364.5201552093463
Vehicle 1
0 -> 5 -> 0
Capacity: 14
Cost: 228.04385543136215
Vehicle 2
0 -> 5 -> 4 -> 0
Capacity: 14
Cost: 254.46351094030695


Vehicle 3
0 -> 4 -> 3 -> 2 -> 0
Capacity: 14
Cost: 408.17005694824763
Vehicle 4
0 -> 2 -> 1 -> 0
Capacity: 14
Cost: 459.7935646509953
Vehicle 5
0 -> 1 -> 0
Capacity: 10
Cost: 331.78908963376114

- **File n6-c29.txt (Truck)**

```
[1] n10-c43.txt
[2] n11-c52.txt
[3] n14-c88.txt
[4] n15-c33.txt
[5] n15-c65.txt
[6] n17-c24.txt
[7] n17-c68.txt
[8] n18-c68.txt
[9] n19-c38.txt
[10] n3-c27.txt
[11] n5-c10.txt
[12] n5-c41.txt
[13] n6-c29.txt
[14] n7-c27.txt
[15] n9-c22.txt
Enter the file to read in (1 to 15): 13
```

```
Hooray, a truck will be used! I can finish my work earlier today.
I can carry 29 items!
```



```
Basic Simulation
Tour
Tour Cost: 869.2223963996562
Vehicle 1
0 -> 3 -> 2 -> 0
Capacity: 22
Cost: 302.6967512610107
```

```
Vehicle 2
0 -> 4 -> 1 -> 0
Capacity: 19
Cost: 338.48178970728344
Vehicle 3
0 -> 5 -> 0
Capacity: 25
Cost: 228.04385543136215

Greedy Simulation
Tour
Tour Cost: 888.9493518350788
Vehicle 1
0 -> 4 -> 5 -> 0
Capacity: 29
Cost: 254.46351094030695
Vehicle 2
0 -> 3 -> 2 -> 0
Capacity: 22
Cost: 302.6967512610107
Vehicle 3
0 -> 1 -> 0
Capacity: 15
Cost: 331.78908963376114
```

```
Time elapsed: |████████████████████████████████████████████████████████| 10 s (Searching is forced to stop!)
MCTS Simulation
Tour
Tour Cost: 888.9493518350788
Vehicle 1
0 -> 5 -> 4 -> 0
Capacity: 29
Cost: 254.46351094030695
Vehicle 2
0 -> 3 -> 2 -> 0
Capacity: 22
Cost: 302.6967512610107
Vehicle 3
0 -> 1 -> 0
Capacity: 15
Cost: 331.78908963376114
```