Simulation of "We-Doo" Delivery Service

Jia Lin
Postgraduate Diploma in Science in Data Analytics
National College of Ireland
x22117644@student.ncirl.ie

***Abstract*—This document presents a simulation process of creating a particular delivery service on off-duty time. And according to the simulation results, the corresponding optimisation suggestions are given to help the company to improve their service efficiency and to expand their business to the whole country.**

***Index Terms*—Optimisation, Modelling, Simulation, A^* Algorithm, Statistics, Visualisation**

I. Introduction

The aim of this project is to create a model to simulate a particular delivery service in a town based on the prerequisites provided by a company, called "We-Doo". By running this simulation system under some preconditions, the whole delivery process is illustrated more intuitively. In this simulation process, the driver's daily working hours, the driver's daily riding distance, the number of undelivered parcels every day and the delay time of each parcel in the distribution centre are counted and visualised. Finally, by properly adjusting some relevant parameters in this system, the delivery service quality and efficiency are improved.

A. Background

As a start-up company, "We-Doo" provides after-hours delivery service for customers who cannot receive packages during working hours on week days. The company has set up a distribution warehouse in a town, where all their customers' parcels will be centrally stored. Until the off-duty time, from 18:00pm to 21:00pm in the evening, local couriers will deliver these packages door-to-door to these customers on an electric cargo bicycle. The driver is working from Monday to Friday.

B. Initial Generation

In order to achieve the purpose of this project, a town map, M, with their distribution warehouse location, W, and customer locations, C, are initially generated[1], see Fig. 1. In terms of the generated data, the planned average delivery capacity, N, per day, is determined by the sum of the expected parcel, P, of each customer in 22 working days.

Ideally, the average number of packages deposited to the distribution warehouse location per day is the same as the mean of packages planned to be delivered each day. In this way, the package delivery business can be completed on time during the experimental period (22 working days). However, there are some constraints have to be considered:

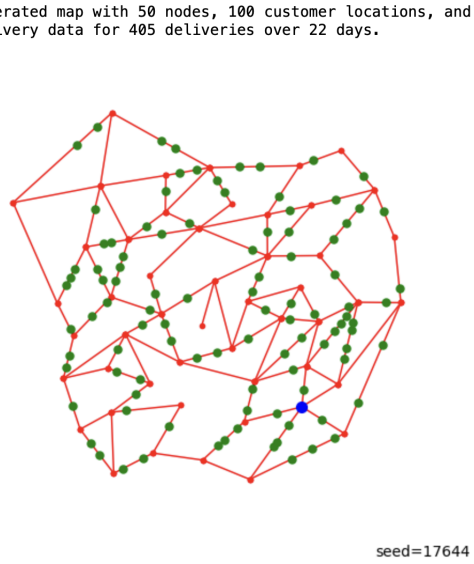


Fig. 1: The town map with distribution warehouse (blue point) and customer locations (green points)

- The standard cargo bike range $\mathsf{R} = 30$ km.
- The planned working time per day is $\mathsf{T} = 3$ hours.
- Every working day, the accumulative preparation time, including packing parcels, sorting them and planning routs according the customers' locations, is suppose to be 50 sec per parcel. In this case, the average preparation time is $50 \times \mathsf{N}/60 = 15$ mins/day.
- On the road, the average speed is 15 km/h.
- When arriving one of target customer location, the average waiting time of handing over the parcel is composed of the time of contacting the customer on call and the customer coming to the door and open it. Such waiting time is approximate 40 seconds. The total time for a day depends on the number of the customers on that day.
- For each face to face customer, the handing over time is estimated 10 seconds per parcel. The total time for each day is determined by the number of parcels are delivered.
- The process of day-end work, consisting of reporting and configuring the cargo-bike, consumes 10 mins.

Hence, it is possible that after one working day, some parcels that should have been delivered are left over to the

[1]This initial map is generated by the provided python source code file in Jupyter Notebook, named "CA Starter File.ipynb", in Moddle. In this file, by calling the `generateData()` method with a parameter `17644`, the initial map was generated.

distribution centre. In order to avoid these parcels being held up for too long time, these parcels have delivery priority on the next working day. And they have to be sorted with the next working day's parcel, and so on.

In order to save time and avoid exceeding the delivery range of the cargo bike, different parcels of the same customer should be delivered together in one working day. In addition, the delivery route should be carefully planned, and strive to find the shortest route starting from the distribution warehouse, W, delivering the parcels to the corresponding customers in that working day, and returning to the distribution warehouse location, W. These factors are all considered and implemented in the simulation process of this project.

II. Literature Review

Another vital aim of this project is to learn and understand how to employ modelling and simulation technology to solve the real world problem. In other words, modelling and simulation are treat as a problem-solving tool [1]. Different from the current data analytics techniques, simulation is a dynamic system, which is used to imitate some characteristics of a system, to obtain some acceptable solution to the problem [2].

Through this project, the relationship between modelling, simulation and validation will be clearly presented and understood. Modelling is used to remove the complexities from the real world problem and to build a mathematical model to represent the core part of such system using several parameters. Simulation is a process of playing the running process of a real system under some controllable premises. These controllable assumptions are the parameters can be changed to improve the behaviour of the original system. And by adjusting one or more parameters, the simulation system is used to forecast the behaviour of the system. The process of tuning the parameter of the model and improving the model behaviour and optimising the system in different conditions is called validation.

Ingalls provides a tutorial that introduce a discrete-event simulation [3], which explain the components of a simulation system, such as entity, attributes (variable, parameter), state, event and activity in detail. A previous study has been done by creating a simulation system to present a real world traffic scenario for the city of Bologna [4].

III. Methodology

A graph (the map M) with 50 nodes has been generated. The distribution warehouse is one of the node, and 100 customers' locations are all presented as points on the edge of this graph.

Firstly, based on this map, the A^* algorithm is taken to identify the shortest path between two points in this graph. In addition, the approach to resolve the travelling salesman problem (TSP) is used to plan a tour that finished up with the starting point, the distribution centre, with the minimum distance. And in a specific daily delivery procedure, each customer is only visited once.

Secondly, the model will be built by defining entities, such as Driver, Customer, Parcel and Distribution Centre, with attributes of them. Events related to each entity has been created to simulate the actions of these entities in a delivery process. And a recorder has been constructed to record relative events and to capture the data required for the final statistics.

Finally, by setting up all parameters of prerequisites, after creating a load generator function, named $\mathsf{generateInputData(D)}$, a simulation routine, called $\mathsf{simulation(M, W, C, D, log = False, plot = False)}$, is defined to run the simulation of this delivery service. After running the simulation, the required statistical results are generated.

A. Optimisation: Finding The Shortest Delivery Route

Since locations of customers are all at the edge of the graph, from one customer location to another, only has two paths, forward or backward. This feature greatly reduces the complexity of the algorithm and It allows us to choose the A^* algorithm (see Fig. 2) to identify the shortest path between two points. This algorithm is implemented by defining the

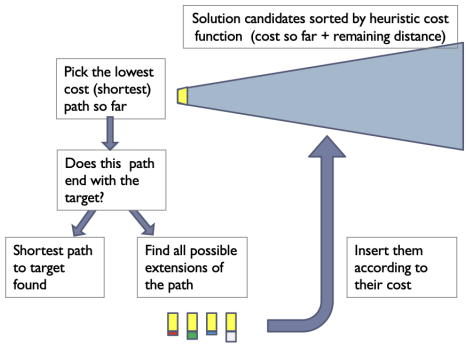


Fig. 2: Using A^* Algorithm to Identify the Shortest Path[a.]

[a.] This figure is got from the lecture's slides of Greedy and Heuristic Algorithms

function $\mathsf{shortestPath(M, A, B)}$, see Fig. 3, in which parameter M represents the map or graph, A and B are starter point and target point respectively in such graph. And a heuristic function, $\mathsf{h(p)}$ with p indicates the path so far, is defined to calculate the sum of the length of p, $\mathsf{pathLength(p)}$, and the remaining distance, $\mathsf{dist(p[-1], B)}$, between the last point in p and the target point B. The dist function is used to compute the Euclidean distance between two points.

```
def shortestPath(M, A, B):

    def h(p):
        return pathLength(p)+dist(p[-1],B)
```

Fig. 3: The Heuristic Function Defined in the Function shortestPath.[b.]

[b.] This function is got from the lecture's code file "Finding Shortest Path.ipynb"

A solution candidate is defined as a pair of path p and its heuristic function $\mathsf{h(p)}$. Whenever a new path is explored, the position of the corresponding candidate in the candidate list is determined by the value of $\mathsf{h(p)}$. The candidate with the smallest value of heuristic cost function is always be the first

element in the solution candidate list. When the path p ends with the target point B. The shortest path found.

The TSP is a well-known optimisation problem, the solution of such problem is used to help the driver plan his daily delivery route. The driver departures from the distribution centre, delivery parcels to the target customers, then back to the distribution centre. In order to deliver as many parcels as possible within the specified time and cargo bicycle range limits, the driver needs to plan the shortest route. Since the distance from point A to B equals to the distance from B to A, this is a symmetric TSP.

A function createLoop(M, T), see Fig. 4, has been defined to identify the shortest loop route, in which the parameter M indicates the graph, and T is a set of points in this graph. By defining the createTables(M, T) function, the shortest path distance between every two point in T has been stored in a matrix. Then by using the pulp package in Python to implement an Integer Linear Programming (ILP) to resolve this TSP. In the source code segment of function createLoop, three for loop have been used to adding the constraints that make sure every point is visited only once and prevent the subtour. The specific role of each for loop has been marked by the corresponding comments. By given the set of points, the path P has been derived.

```
def createLoop(M, T):
    D, P = createTables(M, T)
    # Matrix of the shortest path between every two points.
    n = len(T)
    # create variables
    x = pulp.LpVariable.dicts("x", ( range(n), range(n) ),
                              lowBound=0, upBound=1, cat=pulp.LpInteger)
    # create problem
    prob = pulp.LpProblem("Loop",pulp.LpMinimize)
    # add objective function
    prob += pulp.lpSum([ D[i][j]*x[i][j]
                         for i in range(n) for j in range(n) ])
    # add constraints
    constraints=0
    for j in range(n):# The location of the customer is entered only once.
        prob += pulp.lpSum([ x[i][j] for i in range(n) if i!=j ]) ==1
    constraints += n
    for i in range(n):# The location of the customer is left only once.
        prob += pulp.lpSum([ x[i][j] for j in range(n) if i!=j ]) ==1
    constraints += n
    for i in range(n):# prevent subtour
        for j in range(n):
            if i!=j:
                prob += x[i][j]+x[j][i] <= 1
                constraints += 1
```

Fig. 4: Solving TSP Problem by Using The Function createLoop.[c.]

[c.] This function is in the lecture's code file "Finding Shortest Delivery Route.ipynb"

B. Modelling: Entities, Attributes and Events

A simulation system contains following components:

- Entity: An entity is an object in the system. In this project, there are four entities, Driver, Customer, Distribution Centre and Parcel.
- Attribute: A property of an entity. If an attribute is changeable, it is called a variable, otherwise it is called a parameter. In this study, take the customer entity as an example, variables of a customer are atHome, answersDoor, parcelsReceived. A parameter of a customer is location.
- State: A state is a set of required variables which is used to illustrate the system change. A state of a customer is determined by the value of all its variables.
- Event: An event leads to a change to the state of system. For example, in this system, the driver arrive for work, the driver return to delivery centre, the customer accept parcel and so on.
- Activity: An activity is a process needs a period of time. For example, in this case, the driver's activity drive, wait and handover one or more parcels to a customer.

One objective of this study is to visualise the driver's working time per day. The length of working time equals to the time of the driver returns to the distribution centre or delivery centre subtracts the time of the driver arrive at work. See the event graph of the Driver entity, Fig. 5, all events of the driver entity are presented. additionally, the variable tour is marked, which is used to derive the route length per day.

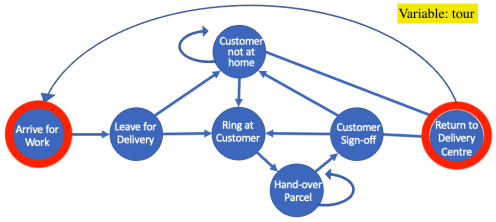


Fig. 5: Event Graph for Driver[d.]

[d.] This figure is modified from the lecture's slides Event Graphs

Another objective of this project is to display the number of leftover parcels every day, this can be achieved by recording the value of variable in the Delivery Centre entity. All events of the Delivery Center entity can be seen in Fig. 6.

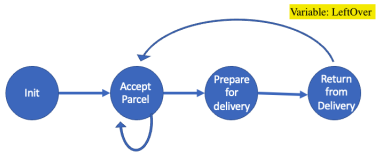


Fig. 6: Event Graph for Distribution Centre[d.]

[d.] This figure is modified from the lecture's slides Event Graphs

The delay time of each parcel in days is determined by the time of the parcel is accepted by the customer, see the corresponding event in Fig. 7, and the time of the parcel arrive in the distribution centre, see relevant event in Fig. 8.

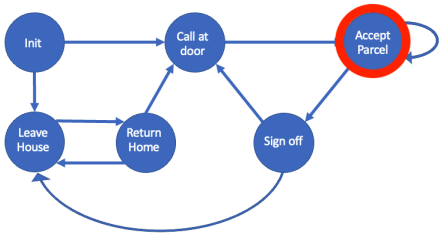


Fig. 7: Event Graph for Customer[d.]

[d.] This figure is modified from the lecture's slides Event Graphs

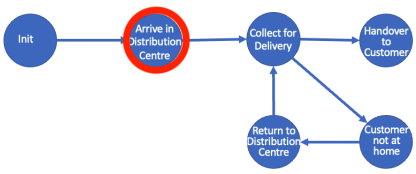


Fig. 8: Event Graph for Parcel[d].

[d] This figure is modified from the lecture's slides Event Graphs

C. Simulation: Recording, Loading and Verification

Follow the convention of building a simulation system, a Recorder class, see Fig. 9, is created firstly to record and trace required events of the system. For each simulation there is only one recorder, each entity has reference to such recorder. In the __init__ function of ClassRecorder, two data frames have been created to store relevant data. A daily data frame is created to store the daily information, such as the time of driver starts work and the ends work, the distance of a day tour and the number of leftover parcels. A perParcel data frame is built to keep the data of parcels, including the time of parcel arrived at the distribution centre and the time of the parcel is accepted by a customer. The data are captured by a list of record functions and some of features are calculated in the finish function (see Fig. 10). For example, the driver's working time per day and the delay time per parcel.

```
class Recorder:

    def __init__(self, env, M, W, C, days,
                 log=False, plot=False):
        self.env = env
        self.M = M
        self.W = W
        self.C = C
        self.days = days
        self.log = log
        self.plot = plot
        Customer.REGISTER = []
        Parcel.REGISTER = []

        # create a data frame for time records per working day
        self.daily = pd.DataFrame()
        self.daily['driver_starts'] = [None]*days # The time of the driver starts to work per day
        self.daily['driver_ends'] = [None]*days # The time of the driver ends work per day
        self.daily['distance'] = [None]*days # The route distance per day
        self.daily['leftover'] = [None]*days # The number of leftover parcels per day

        # create a data frame for time records per parcel
        self.perParcel = pd.DataFrame()
        self.perParcel['arrived'] = [None]*len(Parcel.REGISTER) # The time of the parcel arrived at DC
        self.perParcel['accepted'] = [None]*len(Parcel.REGISTER) # The time of the parcel be accepted
```

Fig. 9: The Constructor of Recorder Class[e].

[e] This class is modified from the lecture's code file "Simulation Step 2 Recording Working Time.ipynb"

Next, the Class of four entities have been created to represent the entities. They carry the reference to the recorder in proper functions. For example, in ClassDriver, in arriveForWork function, by calling the recorder function recordDriverBeginsWork(), the time of the time of starts to work is recorded. And in the process() function, the ends work time and the length of tour are recorded.

Similarly, the recorder reference for tracing the number of leftover parcels is in the sendForDelivery() function of the ClassDeliveryCentre. The time of the customer accept parcels is recorded in acceptParcel function in ClassCustomer. And the time of parcels arrived at the delivery centre is recorded in arrivedAtDeliveryCentre() function in ClassParcel.

```
def recordTourLength(self, length):
    self.daily.at[day(self.env.now), 'distance'] = length/1000

def recordLeftover(self, n):
    self.daily.at[day(self.env.now), 'leftover'] = int(n)

def recordDriverBeginsWork(self):
    #self.trace("Driver arrives for work")
    self.daily.at[day(self.env.now), 'driver_starts'] = int(round(self.env.now))

def recordDriverEndsWork(self):
    #self.trace("Driver goes home")
    self.daily.at[day(self.env.now), 'driver_ends'] = int(round(self.env.now))

def recordParcelArrived(self,index):
    #self.trace("Parcel has been arrived:")
    self.perParcel.at[index, 'arrived'] = day(self.env.now)

def recordCustomerAccepted(self,index):
    #self.trace("Parcel has been accepted by the customer")
    self.perParcel.at[index, 'accepted'] = day(self.env.now)

def finish(self):
    self.daily['time'] = (self.daily['driver_ends']-self.daily['driver_starts'])/60
    self.perParcel['delay'] = self.perParcel['accepted']-self.perParcel['arrived']
```

Fig. 10: Functions in Recorder Class[f].

[f] These functions are built base on the lecture's code file "Simulation Step 2 Recording Working Time.ipynb"

A load generator function, named generateInputData(D), see Fig. 11, has been defined to produce time information for each parcel in this simulation system based on the original planned daily delivery capacity. In this function, the initial arrived time of each parcel, the process time of each parcel, the day information of each parcel and the destination information of each parcel are generated for the simulation system.

```
def generateInputData(D):

    R = [ len(d) for d in D ]
    N = sum(R)

    DAY_LENGTH = 24*3600    # measured in minutes
    DAY_START = 8*3600      # first delivery in the morning
    DAY_END = 17*3600     # last delivery during day time

    IARR = (DAY_END-DAY_START-2*3600)*len(D) / N  # measured in minutes

    x = pd.DataFrame()

    x['iarr'] = [None]*N
    x['time'] = [None]*N
    x['day'] = [None]*N
    x['dest'] = [None]*N

    current_day = 0
    delivered_today = 0
    time = DAY_START
    last_time = 0
```

Fig. 11: The Constructor of Load Generator Function[g].

[g] This function is obtained from the lecture's code file "Simulation Step 2 Recording Working Time.ipynb"

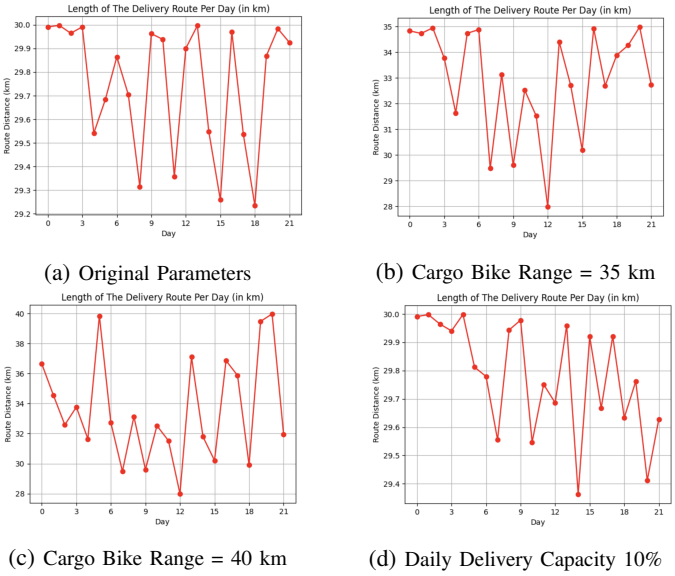
Finally, simulation(M, W, C, D, log = False, plot = False) function, see Fig. 12, is created to run the simulation routine. In this function, the environment has been built by using the Python package simpy which provides the environment for running simulations. An instance of recorder is made to active all recorders for required events. In particular, the function finish(), which has been introduced in the ClassRecorder, is called to calculate required data result. Then by loading the generated data, a simulation process starts to run and return the recorder which contains all information needs for the final statistics. The verification is completed so far[2].

[2] Most of classes and functions introduced in this section are defined by the MSO module lecturer in file "Simulation Step 2 Recording Working Time.ipynb". The author of this paper did some modifications, finally completed in the file "JiaLin_x22117644_MSO_CA1_2_Simulation_Verification(Full Size 17644).ipynb"

```python
def simulation(M, W, C, D, log=False, plot=False):

    env = simpy.Environment()
    rec = Recorder(env, M, W, C, len(D), log=log, plot=plot)

    x = generateInputData(D)

    print(f"Simulating delivery of {len(x):d} parcels "
          f"over {len(D):d} days to {len(C):d} customers")

    def generatorProcess(env):

        for c in C:
            Customer.getCustomer(rec, c)
        DC = DeliveryCentre(rec, M, W)
        D = Driver(rec, DC)

        # generate the parcels based on input data x
        for i in range(len(x)):
            yield env.timeout(x.at[i, 'iarr'])
            custLoc = C[x.at[i, 'dest']]
            cust = Customer.getCustomer(rec, custLoc)
            p = Parcel.getParcel(rec, i, cust)
            DC.acceptParcel(p)

    env.process(generatorProcess(env))
    env.run()
    rec.finish()
    return rec
```

Fig. 12: The Simulation Function[g.]

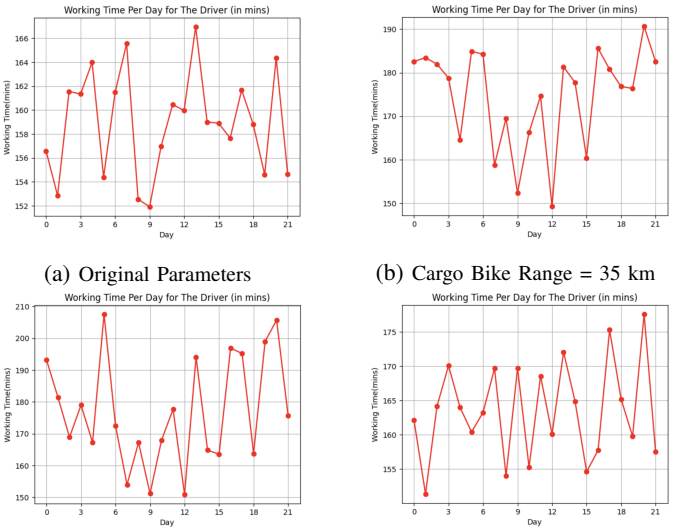
[g.] This function is obtained from the lecture's code file "Simulation Step 2 Recording Working Time.ipynb"



(a) Original Parameters (b) Cargo Bike Range = 35 km

(c) Cargo Bike Range = 40 km (d) Daily Delivery Capacity 10%

Fig. 13: The Distribution of The Delivery Route Distance Per Day (in km)



(a) Original Parameters (b) Cargo Bike Range = 35 km

(c) Cargo Bike Range = 40 km (d) Daily Delivery Capacity 10%

Fig. 14: The Distribution of The Driver Working Time Per Day (in minutes)

D. Simulation Validation, Statistics and Visualisation

With the original parameters of this simulation system, the following results are got and presented in statistical plots, see each subfigure (a). Three validations[3] have been done to adjust values of some parameters to improve the performance of simulation. In the first place, the original cargo bike range is 30 km, by change a cargo bicycle with larger range to 35 km and 40 km, the results are displayed in every subfigures (b) and (c). Finally, by increasing the daily delivery capacity 10% more, see the result changed in each subfigure (d).

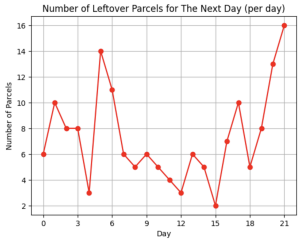
The distribution of the length of delivery route per day (in km) is presented in Fig. 13. By changing a larger range cargo bike to 35 km, see Fig. 13b, the length of tour is increased in most of days. However, by extend the cargo bike range to 40 km, see Fig. 13c, only few days reach a longer journey. Then by planning daily delivery capacity 10% more, the route distance decreased after the 13th day, see Fig. 13d. The better option is change a cargo bike with the range of 35 km.

The distribution of the working time per day for the driver (in minutes), see Fig. 14. The rising up of the distance of the tour leads to the longer working time. From the labour cost perspective, a balance needs to be considered. In Fig. 14b, although the working time of some days are more than 3 hours, but only 10 mins more. However, when adjust the bike range to 40 km, see Fig. 14c, the working hours exceeded too much. Finally, by increasing the planned daily delivery capacity 10% more, see Fig. 14d, the working time of every day is under the limit, better the original data. Hence, changing the bike with range of 35 km and increasing the planned daily delivery capacity both are recommended.

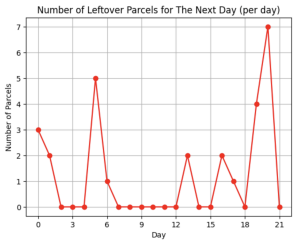
The distribution of the number of leftover parcels for the next day (per day) shows in Fig. 15. The original data has many leftover parcels, see Fig. 15a. By increasing the range of cargo bike range, the number leftover parcel decreased. The bigger of range, the smaller of number of leftover parcels, see Fig. 15b and 15c. However, by increasing the planned daily delivery capacity 10%, the leftover parcels getting more, see Fig. 15d. In this case, increasing the cargo bike range is a suggested choice.

The distribution of the delay time of the parcels (in days) is displayed in Fig. 16. The delay time of the parcels related to the number of leftover parcels. By rising up the range of cargo bike, the number of leftover parcels decreased and the delay time of each parcel getting shorter, see Fig. 16b and

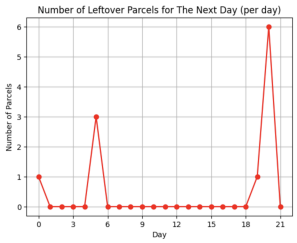
[3]Source code that used to generate different validations are kept separately. Validations can be distinguished by the code file name.



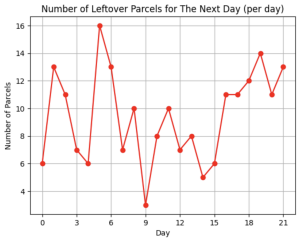
(a) Original Parameters



(b) Cargo Bike Range = 35 km



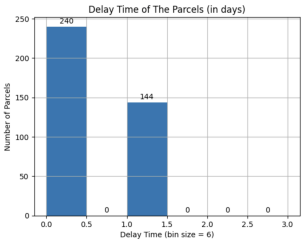
(c) Cargo Bike Range = 40 km



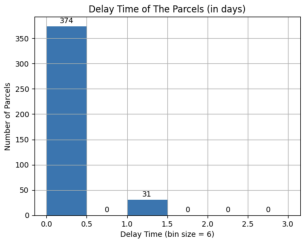
(d) Daily Delivery Capacity 10%

Fig. 15: The Distribution of The Number of Leftover Parcels (per day)

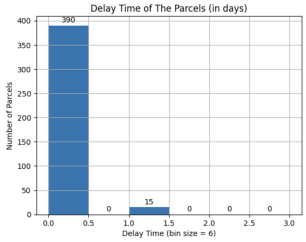
16c. By increasing the daily delivery capacity 10% more, the delay time of parcels is getting longer. In this case, still the increasing the cargo bike range is recommended.



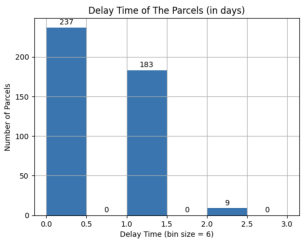
(a) Original Parameters



(b) Cargo Bike Range = 35 km



(c) Cargo Bike Range = 40 km



(d) Daily Delivery Capacity 10%

Fig. 16: The Histogram of The Delay Time of The Parcels (in days)

In summary, the four statistical results shows that the adjustment of the cargo bike range is a greater suggestion, in this case, changing a cargo bike with 35 km range is the optimal choice.

IV. Results and Interpretation

The planned daily delivery capacity follows Poisson distribution. The specific data was generated by the generateData() function in the file "CA Starter File.ipynb". After running simulation, the real delivered parcels are the number of parcels accepted by the customer per day. In Tab. I, the statistics of planned and real number of daily delivered parcels are listed. The max and mean value of such numbers are used to present the planned daily delivery capacity and the maximal sustainable daily delivery capacity.

TABLE I: The Daily Delivery Capacity (DDC)

Daily Delivery Capacity	Max	Mean
Initial Generation Planned DDC	25	18
After Simulation Verification, Real DDC	22	17
Validation1: Cargo Bike Range to 35 km, Real DDC	25	18
Validation2: Cargo Bike Range to 40 km, Real DDC	24	18
Validation3: Increasing DDC 10% Planned DDC	27	20
After Validation3, Real DDC	30	19

```
import collections
# using Counter to find frequency of elements
frequency = collections.Counter(rec17.perParcel['accepted'])
# printing the frequency
realAccepted=list(dict(frequency).values())[:22]
print(realAccepted)
print('The Maximal Daily Delivery Capacity', max(realAccepted))
#print(min(realAccepted))
print('The Mean of Daily Delivery Capacity', int(np.mean(realAccepted)))
```

Fig. 17: The Source Code to Generate The Maximal Sustainable Daily Delivery Capacity.

V. Reflections and Future Work

By adjusting some of parameters, three validations of such simulation system have been complement to help the company to improve delivery efficiency and reduce parcels retention. The final suggestion is to change a cargo bike with range of 35 km. However, each validation in this study is only change one parameter. And the original verification only consider the condition that all customers are stay home during the delivery time. In future work, the author would like to make more complex simulation of such delivery system, for instance, given the random probability of a customer is not at home during off-duty time. And more validations would be made to improve the simulation. Let the simulation system more realistically take into account various real-world situations.

References

[1] L.G. Birta, and G. Arbez, "Modelling and simulation." London: Springer. 2013

[2] Lecture S1- Introduction to Simulation. [online] Available at: https://mymoodle.ncirl.ie/mod/resource/view.php?id=42050.

[3] R. G. Ingalls. "Introduction to simulation". InProceedings of the 2011 winter simulation conference (WSC) 2011 Dec 11 (pp. 1374-1388). IEEE. 2011.

[4] L. Bieker, D. Krajzewicz, A. Morra, C. Michelacci, and F. Cartolano, "Traffic simulation for all: a real world traffic scenario from the city of Bologna". In Modeling Mobility with Open Data: 2nd SUMO Conference 2014 Berlin, Germany, May 15-16, 2014 (pp. 47-60). Springer International Publishing. 2015.