Waste Management Simulation

Executive Summary
Model Description
Model assumptions
Model Specification
    Model Variables
    Update rules
Implementation
    Code for testing
    Simulation
Results and Discussion
    Empirical analysis
        Basic statistics
        Target variables statistics
    Theoretical analysis
Decision recommendation
Critique of the model
References
HCs/LOs
Appendix - code
    Code for testing
    Simulation

Executive Summary

👉 This model includes farms that produce waste, drop-off sites to clean up the waste from trucks, trucks that can travel to different sites, and a headquarter for truck refueling fuel. Our goal is to maximize waste collected with minimal possible fuel to optimize profits. We recommend always going to the closest farm from a truck's current location but with several considerations.

Model Description

In this project, we use Python to implement a model that simulates a waste collection and removal system. The goal is to find the optimal strategy (maximize waste collection, minimize travel distance, and visit randomly) for the waste removal truck driver in order to maximize the performance of the system in terms of the time taken and the total amount of fuel consumed. We compare different strategies and analyze the performance of the system in order to determine the best approach. Based on our experiments, we provide recommendations for the optimal strategy.

Model assumptions

1. Each truck has a finite capacity for carrying waste.
2. Each truck has a finite fuel supply. It needs to return to the company headquarters to refuel when running low on fuel.
3. The waste removal company wants to optimize profit by driving between farms as little as possible and conserving fuel.
4. Time doesn't increase or decrease the cost of trucks, because we don't consider paying the truck drivers.
5. A truck might drive to another farm, wait at the current farm (using time but not fuel), go to a waste drop-off site, or return to the company headquarters, where it can refuel.
6. A truck can go to a farm without collecting all of the waste on a farm. It collects as much as it can till it reaches its full waste capacity.
7. Every site is connected to other sites at different distances. (Not every site is connected to other sites in the reality, but randomly removing edges might result in isolated sites that aren't connected with any other sites. Hence, we vary the distance of the nodes, unconnected sites can be represented by nodes that have very far distance away from each other.)
8. We can have multiple trucks and multiple farms, but we only have one headquarter.
9. After researching, we assume the truck speed is 50 miles per hour on average. Yet, depending on the traffic condition, the speed can vary. It follows this distribution: `sts.norm(50, 5)`
10. After researching, we know that a truck can drive 12 miles per gallon. Therefore, the fuel usage rate per km is 0.08 gallons per mile.
11. After researching, we assume that a truck's full fuel capacity is 150 gallons.
12. After researching, we assume that fuel costs 0.25 USD per mile.
13. For the waste profit per ton, we assume that the earnings are 200 USD per ton. It is based on research on the profit of recycling glass bottles (0.1 cents for 1 bottle). Since one glass bottle is around 500 grams, and 1 ton = 10^6 gram = 2* 10^3 bottles, so the company earns 200 USD per ton of trash.
14. We assume loading waste takes time with small fluctuations because it's a well-established process. It follows this distribution: `sts.norm(5, 0.1)`
15. We assume drop-off waste takes time with small fluctuations because it's a well-established process. It follows this distribution: `sts.norm(3, 0.1)`
16. We assume the amount of waste produced has a bigger fluctuation because it depends on many factors (weather, season, demand). Hence, we choose half cauchy distribution because the values can only be non-zero, and the distribution has a fatter tail that represents the uncertainty. It follows this distribution: `sts.halfcauchy(0, 5)`

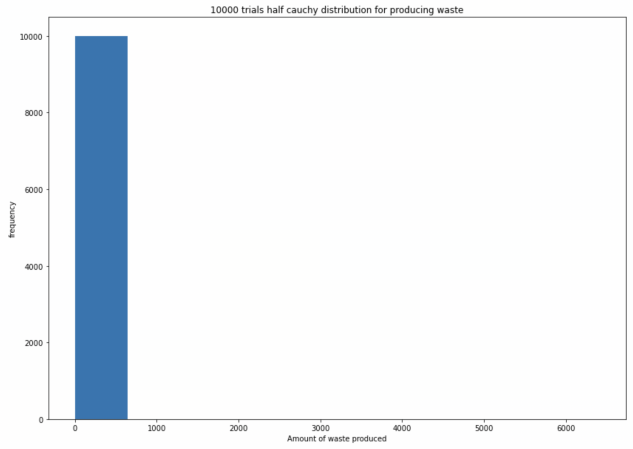

Figure 1. 1000 trials for half cauchy distribution. We use half cauchy rather than cuachy to ensure the value is positive. From the figure, we can see that most waste ranges from 0 to 1000 tons but the value can be as large as 6000 tons.

17. We assume the time interval produces waste follows a normal distribution: `sts.norm(5, 0.1)`. In addition, this model assumes all the farms produce waste at the same time.
18. It takes a truck 5 hours to recharge in the HQ.
19. We assume the distance between the two sites ranges from 100 miles to 1500 miles. We set the upper bound as 1500 miles because a truck with full fuel can run 150 * 12 = 1800 miles, so we can ensure sites are always reachable.
20. We will always go to the closest drop-off sites when a truck's waste capacity is fuel.
21. We will always go through the closest route to headquarters when a truck run out of fuel.

Model Specification

Based on the assumptions, the chosen model, variables, and update rules are shown below:

Model Variables

- For farms
  - Farm id
  - Amount of waste
- For truck
  - Truck id
  - Current carrying waste
  - Max waste carrying capacity
  - Total waste collected (Including waste that has been thrown away in the drop-off sites)
  - Current fuel
  - Max fuel capacity
  - Current location
  - Speed
  - Fuel usage rate
  - All fuels consumed (All the fuels that a truck has consumed since day 1)

Update rules

- For farms
  - The amount of waste the farm accumulates per day follows a half cauchy distribution
- For truck
  - Every truck leaves from headquarters.
  - We check if the truck reaches its waste-carrying capacity.
    - If yes, we consider going to the closest drop-off site.
    - If not, we consider going to a farm.
  - Before going to the closest drop-off site or a farm, we need to calculate if we have enough fuel to come back to HQ if we go to that site.
    - If yes, we go to either the farm or the drop-off site.
    - If not, we go back to HQ to recharge the fuel through the shortest route.
  - We have three strategies to find the next farm for the truck;
    - Max waste: Find the neighbor farms with the max waste.

        - Closest farm: Find the closest farm.
        - Random: Randomly select a farm to go to.
    - Every time the truck reaches the company, the fuel goes back to the maximum capacity
    - Every time the truck reaches a drop-off site, the waste goes to 0.
    - No matter which site a truck goes to (farms, drop-off sites, HQ), it takes time before departing the site. (See model assumptions for details)

Implementation

In this project, we use an object-oriented approach to implement the model. We define classes for trucks, farms, waste drop-off sites, companies, and road systems, which keep track of the properties of each object and handle their update rules. We also implement an event and schedule class to schedule specific events, such as departing from the company, arriving at a farm, and arriving at drop-off sites so we can test our code and check whenever an event happens. We use the NetworkX library to implement the network, and visualize different entities using different colors (HQ: green, farm: light blue, drop off: yellow). We use the size of the node to represent the amount of waste in different farms. Our code can be divided into three blocks, code for testing the model code for simulations, and visualization.

Code for testing

In order to check if our code is correct, we set up a simple case and print it out. Here is our initial setup for the ease of code testing.

- 5 farms
- 1 truck
- fixed waste production of farms: 2 units of waste per 5 hr
- Truck max fuel capacity = 100 gallon
- Truck max waste capacity = 5 tons
- Run the schedule for 35 hr
- fixed speed = 50 miles/hr
- fixed loading time: 5 hr
- fixed drop-off time: 3 hr
- fixed time interval for waste produced: 5 hr
- strategy: max waste

You can see the output with the final statistics.

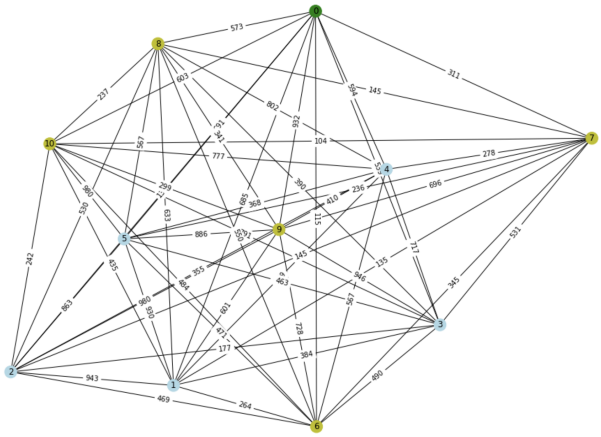

Figure 2. Initial network

```
🐷 Farms add waste now, cur waste(id: waste) [(1, 2), (2, 2), (3, 2), (4, 2), (5, 2)]
🐷 Farms add waste now, cur waste(id: waste) [(1, 4), (2, 4), (3, 4), (4, 4), (5, 4)]
🐷 Farms add waste now, cur waste(id: waste) [(1, 6), (2, 6), (3, 6), (4, 6), (5, 6)]
🚚 Truck id 0 go to farm id 5 with waste 6 cur truck waste 0
    🔴 Truck 0 finishs collecting the waste and reach full waste capacity
🚚 Truck id 0 Truck loc 5 truck updated fuel 58.88 truck waste 5
🐷 Farms add waste now, cur waste(id: waste) [(1, 8), (2, 8), (3, 8), (4, 8), (5, 3)]
🗑️ Truck 0 go to dropoff
🐷 Farms add waste now, cur waste(id: waste) [(1, 10), (2, 10), (3, 10), (4, 10), (5, 5)]
Truck id 0 go back to HQ to refuel ⛽ (not farm), cur truck is at loc 7 , cur fuel 40.0 , go to
next farm need fuel  22.240000000000002 , go to hq need fuel 39.04
🐷 Farms add waste now, cur waste(id: waste) [(1, 12), (2, 12), (3, 12), (4, 12), (5, 7)]
🐷 Farms add waste now, cur waste(id: waste) [(1, 14), (2, 14), (3, 14), (4, 14), (5, 9)]
✅ Truck 0 finish refuel in hq and its fuel is 100
🐷 Farms add waste now, cur waste(id: waste) [(1, 16), (2, 16), (3, 16), (4, 16), (5, 11)]

🎯 Final statistics 🎯
farm current waste (farm id: waste) [(1, 16), (2, 16), (3, 16), (4, 16), (5, 11)]
fuel_use_per_truck (truck id: fuel) [(0, 41.12)]
waste_collect_per_truck (truck id: waste) [(0, 5)]
```

Simulation

We start out simulation tests with all 5 farms, 2 drop-off sites, and 5 trucks, for 1000 hours using max waste strategy.

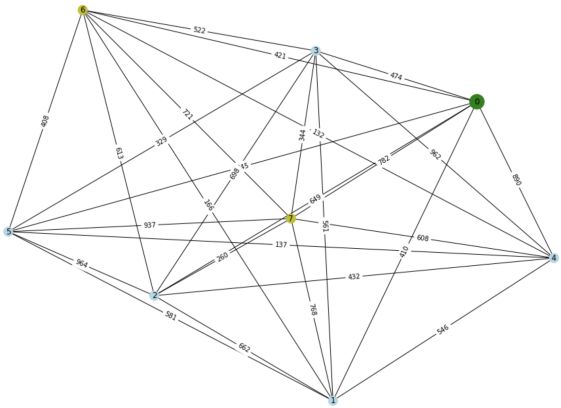

Figure 3. Initial network

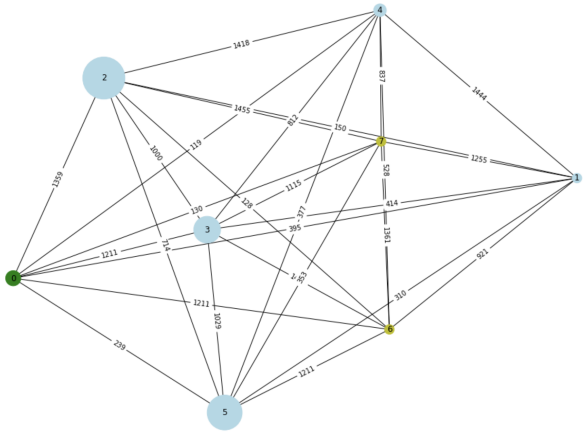

Figure 4. The network after running 1000 hours

```
🎯 Final statistics 🎯
farm current waste (farm id: waste) {1: 0.0, 2: 367.13438555425586, 3: 136.4710775161003, 4:
14.906458020995952, 5: 246.58344983146856}
fuel_use_per_truck (truck id: fuel) {0: 1768.3199999999997, 1: 1766.6399999999996, 2:
1823.2799999999997, 3: 1828.5599999999997, 4: 1766.7999999999995}
waste_collect_per_truck (truck id: waste) {0: 6711.673073258418, 1: 6206.462461783821, 2:
752.1939012694336, 3: 2420.674910555024, 4: 5600.0}
```

From the final statistic, we can see that every truck uses similar fuels (1750 - 1800 gallons), but farm current wastes and waste collected per truck vary a lot. Trucks

To understand the changes in waste for farms, we visualize the animation of waste for 1000 hours. (If you can not see the animation, you can view the video here) The more waste a farm has, the bigger the size of the farm node becomes. From the animations, we can see that with the current parameters we set, waste can grow much faster that our collection rate. However, at some moment, we can still see waste stop growing, showcasing the effect of trucks' waste collection.

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/6ff0ebfc-fcb5-4190-90b8-b08e371468a4/animation.mov

Animation 1. Road network animation visualizing waste growths for different farms.

We can also observe the total waste collected over time. We can see that a great fluctuation exists when re-running the model every time.

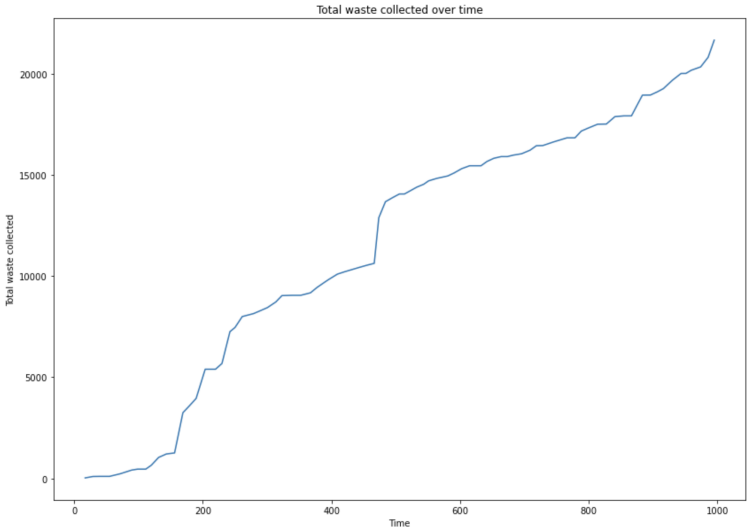

Figure 5. Total waste collected over time for one simulation (Corresponding to the final statistic summary above)

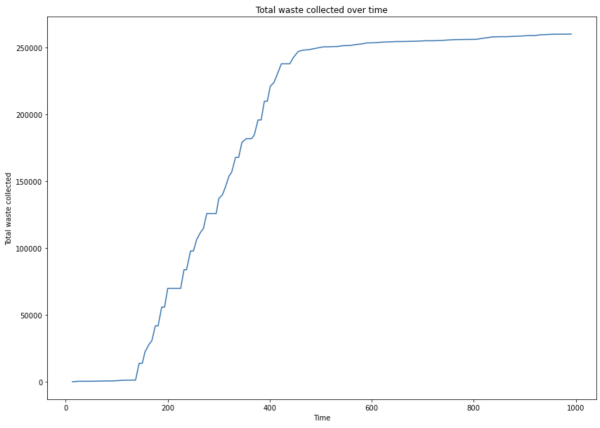

Figure 6. Total waste collected over time for another testing simulation. - Test 1

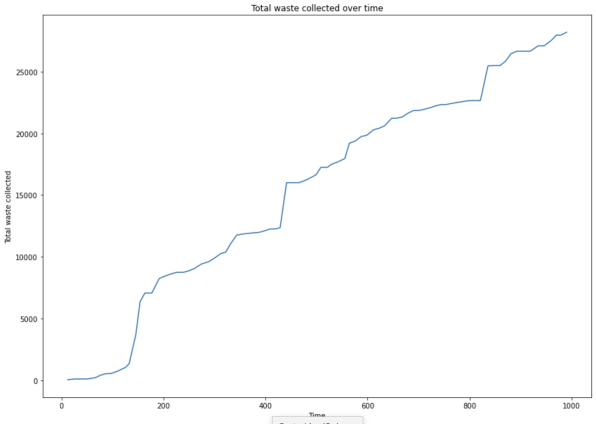

Figure 7. Total waste collected over time for another testing simulation. - Test 2

Results and Discussion

Empirical analysis

The result is obtained from 300 trials, 50 farms, 10 drop-off sites, and 5 trucks for 1000 hrs in the scheduler.

We include basic statistics on counting the number of times going back to HQ, the number of farms visited, the number of times going back to HQ and the number of drop-off sites visited for different strategies. To reach our objective, we also compare total waste collected, total fuel used, and total profits. We plot the distribution and calculate the mean, standard deviation, and 95% confidence interval to understand the spread of possible outcomes.

Basic statistics

For max waste strategy

- On average, the number of times going back to HQ for refueling: 82.0
- On average, the number of farms visited: 273.2
- On average, the number of drop-off sites visited: 98.8

For random strategy

- On average, the number of times going back to HQ for refueling: 81.9
- On average, the number of farms visited: 272.5
- On average, the number of drop-off sites visited: 99.8

For the closest farm strategy

- On average, the number of times going back to HQ for refueling: 81.4
- On average, the number of farms visited: 274.2
- On average, the number of drop-off sites visited: 102.4

Target variables statistics

Comparing total collected waste for different strategies

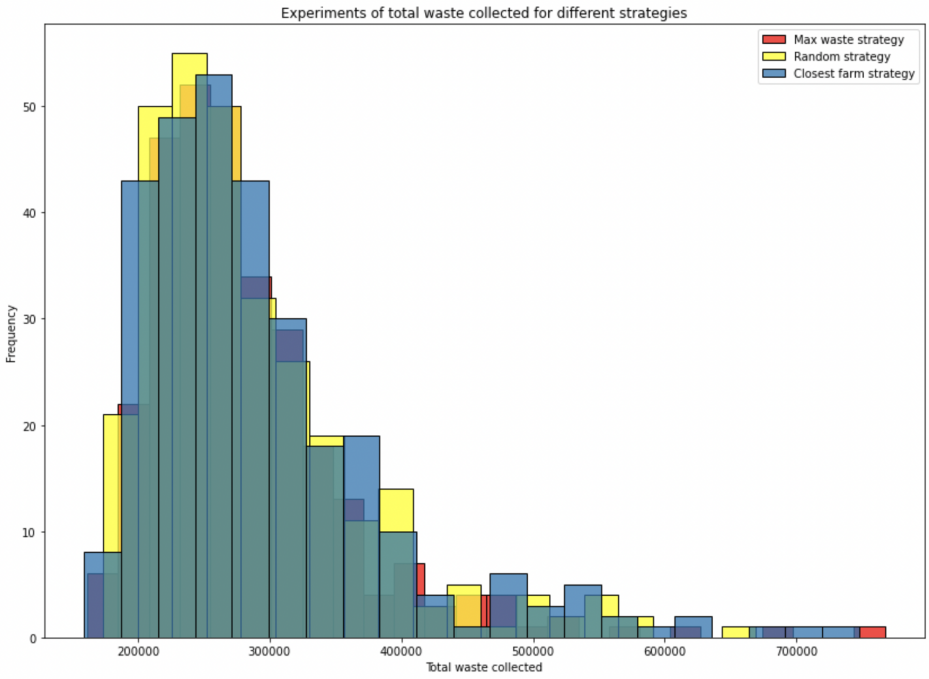

Figure 8. 300 experiments of the total waste collected comparing three strategies

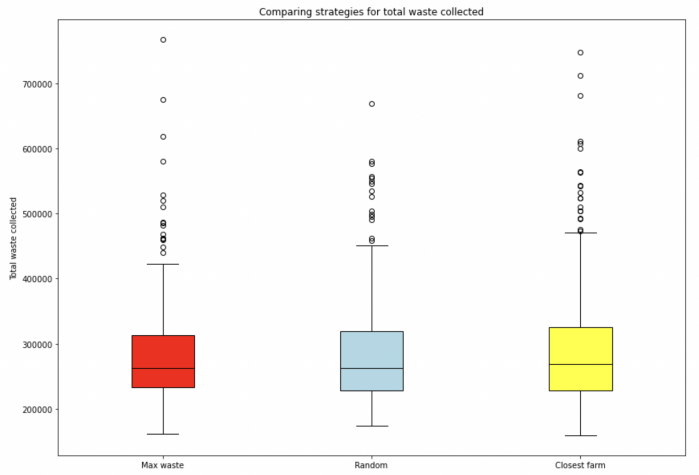

Figure 9. Box plot of the total waste collected comparing three strategies

```
Average waste for max waste strategy  283046 and its std 79917
     Its confidence interval is (273950.73570297076, 292141.06920519273)
Average waste for random strategy  285726 and its std 82845
     Its confidence interval is (276297.25391608546, 295154.05743705237)
Average waste for closest farm strategy  293326 and its std 94959
     Its confidence interval is (282519.07079111354, 304133.26217652345)
```

The result shows that total waste collected follows a right-skewed distribution, meaning that we have a small chance to collect a significant amount of waste for all three strategies. The large standard error might also come from the skewed distribution. This might be coming from the fact that the amount of waste produced follows a cuachy distribution, so a big amount of waste can occur occasionally. Overall, the three strategies have a similar amount of waste collected, but the closest farm strategy collects slightly more waste than the others.

Comparing total fuel used for different strategies

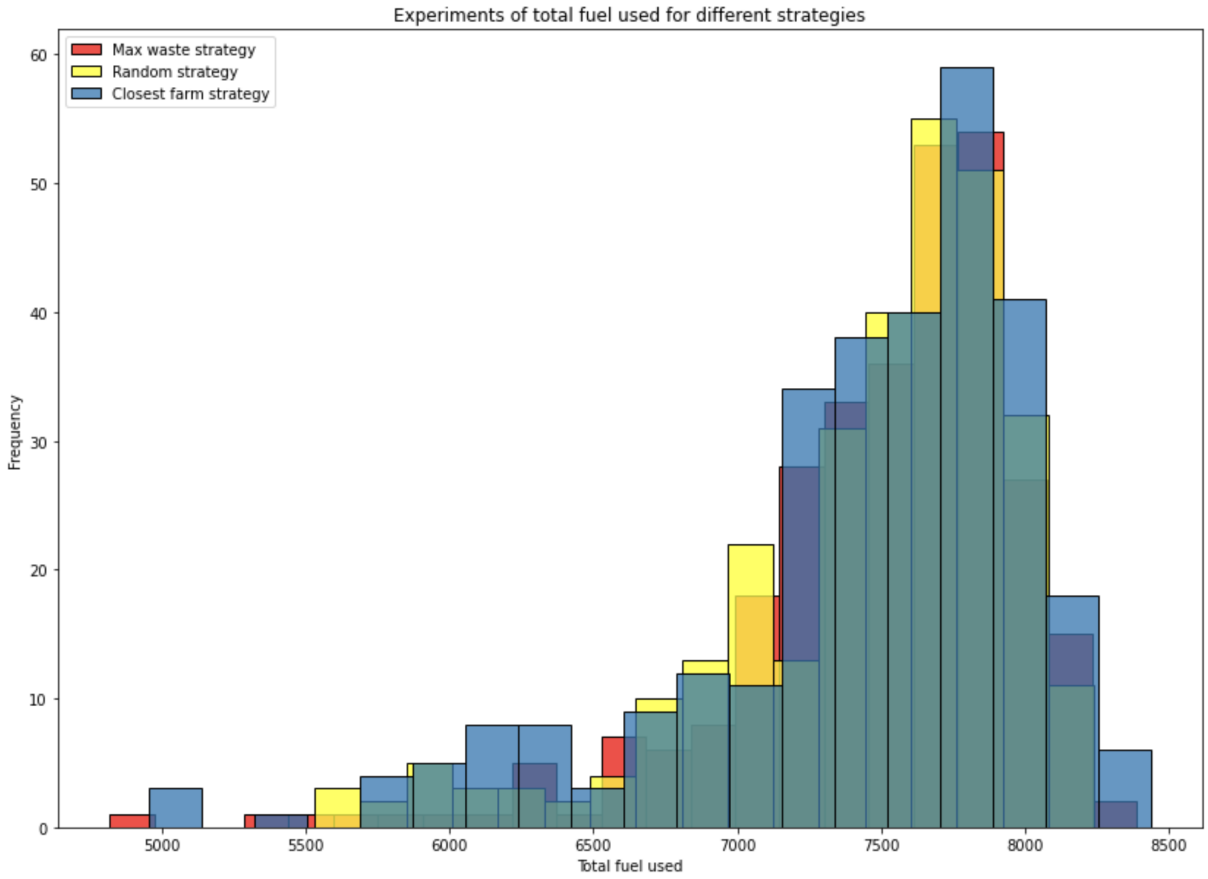

Figure 10. 300 experiments of total fuel used comparing three strategies

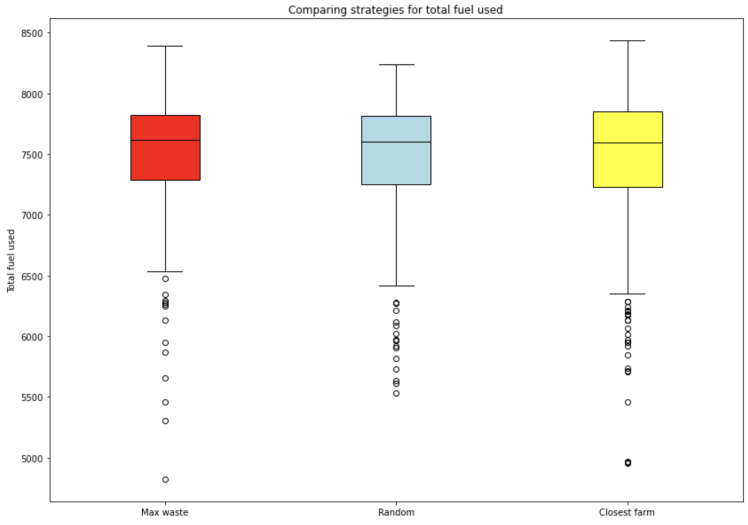

Figure 11. Box plot of total fuel used comparing three strategies

```
Average fuel used for max waste strategy  7498 and its std 502
     Its confidence interval is (7440.874711172665, 7555.106088827336)
Average fuel used for random strategy  7451 and its std 526
     Its confidence interval is (7390.771179849276, 7510.459753484058)
Average fuel used for closest farm strategy  7436 and its std 620
     Its confidence interval is (7365.750707033354, 7506.9570262999805)
```

The result shows that total waste collected follows a left-skewed distribution, meaning that we have some chance to use little fuel for all three strategies. Overall, the three strategies have a similar distribution, but the closest farm strategy seems to have a slighter smaller mean and confidence intervals compared to the two other strategies.

Compare total profits earned for different strategies

Here is the profit formula using total waste collected and fuel cost.

$$profits = waste * 200 - fuel * 0.25$$

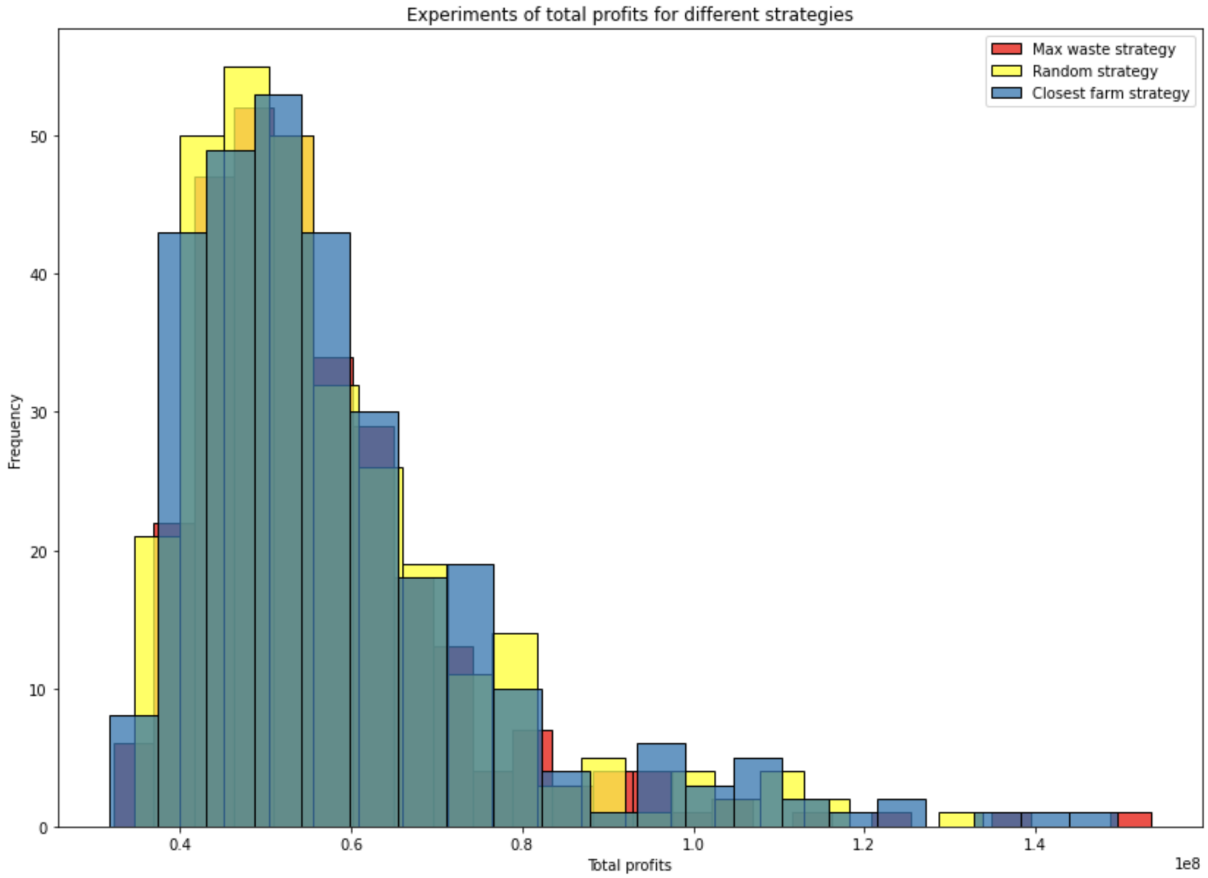

Figure 12. 300 experiments of total profits comparing three strategies

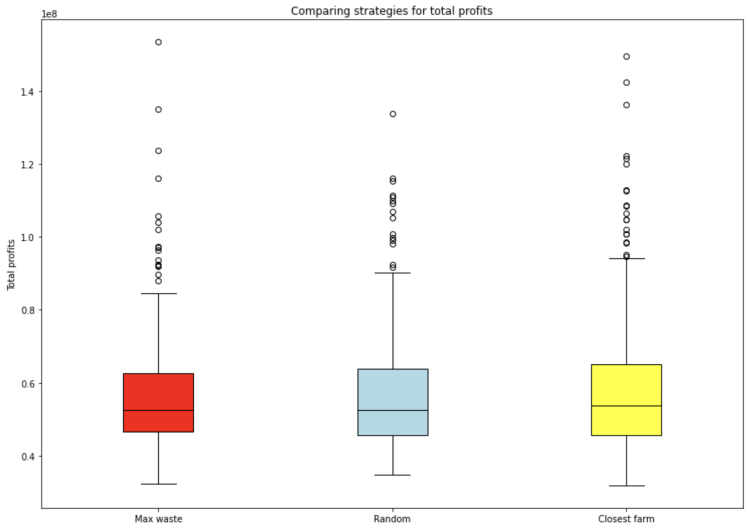

Figure 13. Box plot of total profits comparing three strategies

```
Average profits for max waste strategy  56607306 and its std 15983426
     Its confidence interval is (54788259.278452836, 58426352.707979865)
Average profits for random strategy  57143268 and its std 16569040
     Its confidence interval is (55257573.98302912, 59028962.979865156)
Average profits for closest farm strategy  58663374 and its std 18991899
     Its confidence interval is (56501938.25489457, 60824810.16169949)
```

The result shows that total profits follow a right-skewed distribution, meaning that we have some chance to earn a lot for all three strategies. Overall, the three strategies have a similar distribution, but the closest farm strategy seems to be optimal if we look at its mean profits and confidence intervals.

We are surprised that the three strategies don't have a difference from each other, especially for the underperformance of the max waste collection strategy. The reasons might be the following.

First, we can compare the strength and weaknesses of different strategies. The Max waste collection strategy doesn't collect more waste as we expect, and it might be due to the longer travel distance compared with different strategies. Though the closest farm strategy might take less waste on average, it's able to visit slightly more farms in the same amount of time. Though the random strategy outperforms the max strategy in the simulation, we see that the results differ every time we run it. Second, we can see that the volatility is high even when we only compare the outputs for one strategy. Since our results are heavily skewed, we have a very high standard deviation, showing that results can vary a lot, resulting in difficulty in comparing different strategies. Besides, we take a total fuels for all trucks but the fuel used up by individual trucks can vary quite a lot (See our Simulation section), indicating inefficiency can occur for certain trucks. that we haven't considered Lastly, performance might balance out because of refueling and throwing waste away. We assume that the more waste a truck collects, the more drop-off sites the truck needs to visit. If a strategy collects waste faster, then trucks applying this strategy will need to go to drop-off sites more often. The time and fuel used traveling to drop-off sites and waiting for the unloaded waste will allow other strategies to catch up. A similar concept applies to going back to headquarters for refueling. If a truck travels a longer distance and collects more waste, it will need to go back to the headquarter to refuel more often. Hence, we hypothesize that the dropping off and HQ refueling process balance out the output of different strategies.

Theoretical analysis

We estimate our fuel usage in our theoretical analysis, and this analysis doesn't differentiate different strategies. We choose fuel to estimate rather than waste collected because produced waste from farms has very high volatility that can be hard to provide an accurate estimate. The purpose of conducting this analysis is to understand whether our model output is valid. To ensure validity, we provide lower and upper bounds so we can provide a range of estimated values.

Fuel usage without considering waiting time

We provide a theoretical estimation of fuel usage. We start our estimation through our model setup.

- Max fuel capacity = 150 gallon
- Speed = 50 miles/ hr
- Fuel usage rate = 0.08 gallon per mile = 12 miles per gallon
- Simulation run 1000 hrs.
- Distance between sites = 100 - 1500 miles

Based on the setup, we know

- A truck with full fuel can run 150 * 12 = 1800 miles = 1800/50 = 36 hr
- Simulation run 1000 hr: 1000/36 ~ 30 times a truck with full fuel = 30 * 1800 = 54000 miles = 54000 * 0.08 = 4320 gallon.

When we don't consider any waiting time (the time waiting in a farm, drop off sites, and HQ), the max distances trucks can run is 54000 miles within 1000 hrs, costing 4320 gallons of fuel. Since our model considers waiting time, we re-calculation our estimation.

Estimate upper and lower bound for waiting time

- Mean waiting in a farm for loading waste: 5 hr
  - Distance ranging from 100 - 1500 miles to go to a farm
    - upper bound for numbers of farm visited = 54000/1000 = 54 farms
      - 54 * 5 = 270 hr
    - lower bound for numbers of farm visited = 54000/1500 = 36 farms
      - 36 * 5 = 180 hr
- Mean waiting in the drop off sites to clean trash: 3 hr
  - Distance ranging from 100 - 1000 miles to go to a drop-off site
    - upper bound for numbers of farm visited = 54000/1000 = 54 farms
      - 54 * 3 = 162 hr
    - lower bound for numbers of farm visited = 54000/1500 = 36 farms
      - 36 * 3 = 108 hr
- Mean waiting in the HQ = 5 hr
  - 4320/150 = 28.8 times (amount of time we need to go back to HQ to refuel)
    - 28.8 * 5 = 144 hr

Fuel usage including waiting time

- Fuel usage lower bound
  - 1000 hr - 144 hr (HQ) - 270 hr (farms) - 162 hr (drop off sites) = 424
  - 424 hr is an underestimation because it's impossible to go to so many farms and drop-off sites,
  - 424 hr/36 hr ~ 11.8 trucks with full fuels
    - 11.8 * 1800 = 21240 miles = 21240 * 0.08 = 1699 gallon
- Fuel usage upper bound
  - 1000 hr - 144 hr (HQ) - 180 hr (farms) - 108 hr (drop off sites) = 568
  - 568 hr/36 hr ~ 15.8 trucks with full fuels
    - 15.8 * 1800 = 28440 miles = 28440 * 0.08 = 2275 gallon

Here, we may conclude gallons we use per truck range between 1699 gallons and 2275 gallons, which is aligned with our modeling final statistics. This shows that our model output can be trusted.

```
🎯 Final statistics 🎯
farm current waste (farm id: waste) {1: 0.0, 2: 367.13438555425586, 3: 136.4710775161003, 4:
14.906458020995952, 5: 246.58344983146856}
fuel_use_per_truck (truck id: fuel) {0: 1768.3199999999997, 1: 1766.6399999999996, 2:
1823.2799999999997, 3: 1828.5599999999997, 4: 1766.7999999999995}
waste_collect_per_truck (truck id: waste) {0: 6711.673073258418, 1: 6206.462461783821, 2:
752.1939012694336, 3: 2420.674910555024, 4: 5600.0}
```

Decision recommendation

Different strategies don’t create a huge difference in our profits, but we still recommend the closest farm strategy over the other two. We will recommend a truck driver always go to the closest farm because it uses less fuel and visit more farms. Yet, based on our model assumption, we will only recommend this strategy when the waste produced by the farm varies a lot, so the waste amount collected by the farm can vary a lot too.

Nevertheless, if we look for greater performance, we will recommend adding more headquarters or refueling sites or increasing waste-carrying capacity. From the basic statistic, we can see that around 20% (80/400) of the sites we visit is the headquarter, and 25% (100/400) of the sites we visit are drop-off sites. That’s why the time and fuel cost we need to spend for going back to headquarters and drop-off sites balance out the performance of different strategies. If we can create more well-connected refueling sites or increase waste carrying capacity per truck, we can greatly improve the waste collection efficiency and have a better comparison among strategies.

Critique of the model

Though our model tries to represent the behavior of reality, we acknowledge some factors that we haven’t considered and assumptions that are violated.

For instance, we assume that all the farms produced waste at the same time, making the implementation and code testing easier but doesn’t represent the real world. In addition, we don’t consider route coordination among multiple trucks, indicating that multiple trucks can visit the same sites which reduce the waste collection efficiency.

In addition, we might always be able to deploy different types of trucks based on different speed and waste-carrying capacities. For instance, we can assign a small but quick rear-loading compactor to collect the trucks nearby the headquarter, but assign a big open-body truck to collect trucks in farther farms.

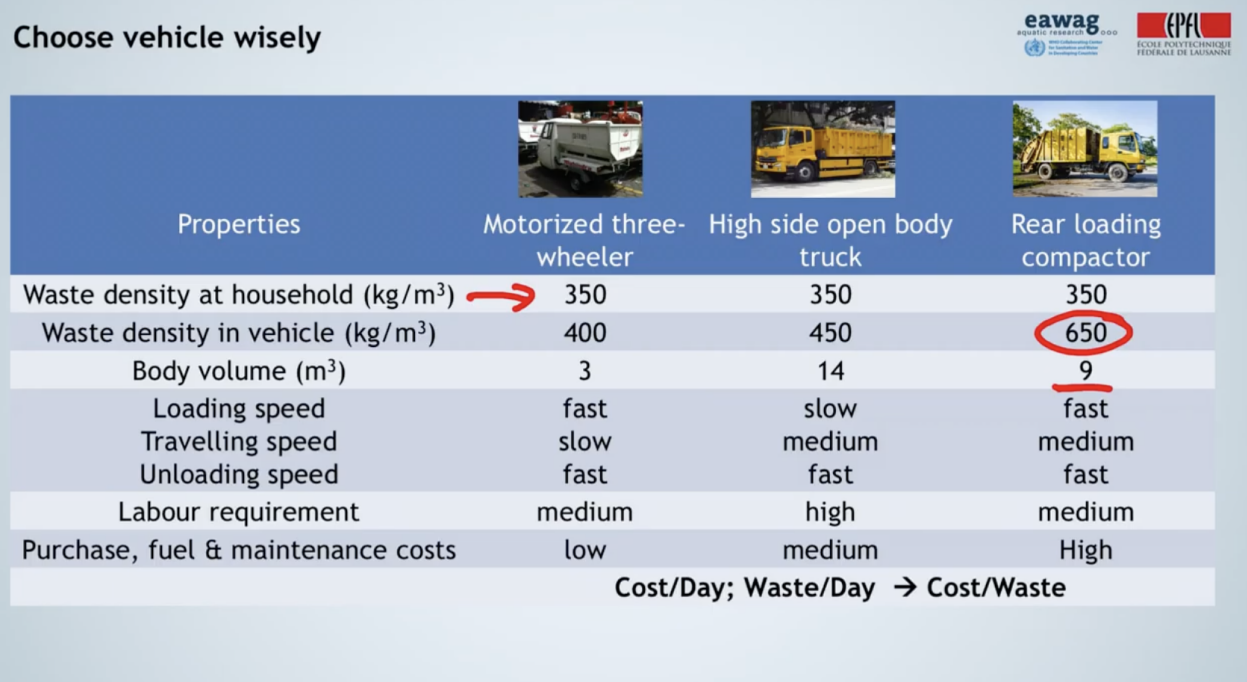

Figure 14. Screenshot of strengths and weakness of different trucks (Source)

References

Carlier, M. (n.d.). Average truck speed on U.S. metropolitan area interstates 2015. Statista. Retrieved December 15, 2022, from https://www.statista.com/statistics/195100/average-operating-truck-speed-on-selected-us-interstate-highways/

Frequently asked questions (FAQ). (n.d.). CalRecycle Home Page. Retrieved December 15, 2022, from https://calrecycle.ca.gov/bevcontainer/programinfo/faq/

Gas price per mile driven. (n.d.). Retrieved December 15, 2022, from https://archive.nytimes.com/www.nytimes.com/packages/html/business/20060510_LEONHARDT/cost_per_mile.html

How to calculate moving truck gas cost. (2018, May 9). Mymove. https://www.mymove.com/moving/moving-services/how-to-calculate-gas-usage-for-your-moving-truck/

International, C. (2021). How Singapore fixed its big trash problem [Video]. In YouTube. https://www.youtube.com/watch?v=r-q5V6LDxEY

Management, M. S. W. (2018). 1.5 Improving efficiency of waste collection and transport [Video]. In YouTube. https://www.youtube.com/watch?v=tMlvwX3TorM

Staff, F. (2020, January 3). How many gallons does it take to fill up a big rig? FreightWaves. https://www.freightwaves.com/news/how-many-gallons-does-it-take-to-fill-up-a-big-rig

HCs/LOs

#cs166-TheoreticalAnalysis: To provide high-quality theoretical analysis, I (1) lay out the theoretical assumptions (2) use the assumptions to reason through the final output (3) include upper and lower bound for the estimation to incorporate uncertainty for the output. In addition, when I cannot find data to support my assumptions (e.g. cannot find waste collection profit per ton), I research proxy data (profit for recycling glass) for a rough estimation.

#cs166-EmpiricialAnalysis: I provide both basic statistics and target variable statistics for my empirical analysis so the reader can easily compare three strategies from different perspectives. I ran 300 trials for 1000 hrs in my scheduler. I've tried 1000 trials before and the results are similar, so I output 300 trials because it already takes 15 minutes to run. Besides calculating the mean for waste collection, fuel consumption, and profits, I also provide the confidence interval and standard deviation. To understand and explain the big standard deviation, I produce the distribution plot and reason for the big standard deviation through the skewed distribution.

#cs166-CodeReadibility: I provide clear docstrings with comments for my code. In addition, I create multiple classes (Truck, farm, drop-off sites, road network) and functions to ensure modularity. In addition, for truck, farm, and drop-off sites classes, I assign them with ids so the reader can easily evaluate the correctness of the code.

#cs166-Professionalism My report includes an executive summary, model description, assumptions, parameters, updated rules, implementation of code, theoretical and empirical analysis, and the computation of output metrics to explain my code. For the data, I also include captions for the figures for clarity.

#cs166-PythonImplementation: I design very simple input variables and parameters in Code for Testing and print out the output so the reader can check the correctness of the code themselves. In the code testing, I discard all the uncertainty (fixed value rather than distribution), so we can check if every output of the schedule aligns with our expectations. In addition, I also use animation to show the growth of the waste so the reader can understand the process as well.

#estimation: I apply estimation as a plausibility check for my simulation model output. I use the max number of farms, drop-off sites, and headquarters as fuel cost upper bound, and use the min numbers of lower bound to estimate the output range for fuel cost per truck. It is a relatively wide range because max number of farms cannot achieve with max number of drop-off sites and headquarter at the same time. However, my output range of 1699 to 2275 gallons is aligned with the model output (1750 to 1850 gallons).

Appendix - code

```
import heapq
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as sts
import random
from tqdm import tqdm
import networkx as nx
import seaborn as sns
from tqdm import tqdm
import math
from pylab import rcParams
```

```
#adapt the code from classes
class Event:
    '''
    Store the properties of one event in the Schedule class defined below. Each
    event has a time at which it needs to run, a function to call when running
    the event, along with the arguments and keyword arguments to pass to that
    function.
    '''
    def __init__(self, timestamp, function, *args, **kwargs):
        self.timestamp = timestamp
        self.function = function
        self.args = args
        self.kwargs = kwargs

    def __lt__(self, other):
        '''
        This overloads the less-than operator in Python. We need it so the
        priority queue knows how to compare two events. We want events with
        earlier (smaller) times to go first.
        '''
        return self.timestamp < other.timestamp

    def run(self, schedule):
        '''
        Run an event by calling the function with its arguments and keyword
        arguments. The first argument to any event function is always the
        schedule in which events are being tracked. The schedule object can be
        used to add new events to the priority queue.
        '''
        self.function(schedule, *self.args, **self.kwargs)


class Schedule:
    '''
```

```
    Implement an event schedule using a priority queue. You can add events and
    run the next event.

    The `now` attribute contains the time at which the last event was run.
    '''

    def __init__(self):
        self.now = 0  # Keep track of the current simulation time
        self.priority_queue = []  # The priority queue of events to run

    def add_event_at(self, timestamp, function, *args, **kwargs):
        # Add an event to the schedule at a particular point in time.
        heapq.heappush(
            self.priority_queue,
            Event(timestamp, function, *args, **kwargs))

    def add_event_after(self, interval, function, *args, **kwargs):
        # Add an event to the schedule after a specified time interval.
        self.add_event_at(self.now + interval, function, *args, **kwargs)

    def next_event_time(self):
        return self.priority_queue[0].timestamp

    def run_next_event(self):
        # Get the next event from the priority queue and run it.
        event = heapq.heappop(self.priority_queue)
        self.now = event.timestamp
        event.run(self)

    def __repr__(self):
        return (
            f'Schedule() at time {self.now} ' +
            f'with {len(self.priority_queue)} events in the queue')

    def print_events(self):
        print(repr(self))
        for event in sorted(self.priority_queue):
            print(f'   {event.timestamp}: {event.function.__name__}')
```

Code for testing

```
class Farm():
    """
    Farm class to store the waste generated by the farm.

    Parameters
    ----------
    n_id : int
        The id of the farm.
    waste : int
        The amount of waste generated by the farm.
    waste_rate_dist : scipy.stats.rv_continuous
        The distribution of the waste rate.
    wait_time_dist : scipy.stats.rv_continuous
        The distribution of the wait time.
    """
    def __init__(self, n_id):
        self.waste = 0
        self.waste_rate_dist = sts.norm(5, 1)
        self.wait_time_dist = sts.norm(5, 0)
        self.n_id = n_id

    def add_waste(self, new_waste):
        """
        Add waste to the farm.
        """
        self.waste += new_waste

    def update_waste(self, recyled_waste):
        """
        Update the waste generated by the farm.
        """
        self.waste -= recyled_waste

    def clear_waste(self):
        """
        Clear the waste generated by the farm.
```

```
        """
        self.waste = 0

class Dropoff():
    """
    Dropoff class to store the waste generated by the farm.
    """
    def __init__(self, n_id):
        self.id = n_id
```

```
class RoadNetwork:
    """
    RoadNetwork class to store the road network.


    Parameters
    ----------
    g : networkx.Graph
        The graph of the network.
    num_farms : int
        The number of farms in the network.
    num_dropoff_sites : int
        The number of dropoff sites in the network.
    curr_node_id : int
        The current node id.
    w_min : int
        The minimum amount of distance between two nodes.
    w_max : int
        The maximum amount of distance between two nodes.
    time_load_waste : int
        The time it takes to load waste into the truck.
    colors : dict
        The colors of the nodes.
    n_trucks : int
        The number of trucks in the network.
    drop_time : int
        The time it takes to drop off waste at the dropoff site.
    trucks : list
        The list of trucks in the network.
    farms : list
        The list of farms in the network.
    dropoff_sites : list
        The list of dropoff sites in the network.
    """

    def __init__(self):
        self.g = nx.Graph()
        self.num_farms = 5
        self.num_dropoff_sites = 5
        self.curr_node_id = 1
        self.w_min = 100
        self.w_max = 1000
        self.time_load_waste =  5
        self.colors = {'hq': 'green', 'farm': 'lightblue', 'dropoff': 'y'}
        self.n_trucks = 1
        self.drop_time = 3
        self.trucks = [Truck(id) for id in range(self.n_trucks)]
        self.farms = []
        self.dropoff_sites = []

        self.network_setup()
        self.draw_network()


    def add_hq(self):
        """
        Add the hq to the network.
        """
        self.g.add_node(0, n_type = 'hq')

    def add_farms(self):
        """
        Add farms to the network.
        """
        new_farm = Farm(self.curr_node_id)
        self.farms.append(new_farm)
        self.g.add_node(self.curr_node_id, n_type = 'farm',
                        obj = new_farm)
```

```
        self.curr_node_id += 1

    def add_dropoff_sites(self):
        """
        Add dropoff sites to the network.
        """
        new_dropoff_sites = Dropoff(self.curr_node_id)
        self.dropoff_sites.append(new_dropoff_sites)
        self.g.add_node(self.curr_node_id, n_type = 'dropoff',
                        obj = new_dropoff_sites)
        self.curr_node_id += 1

    def network_setup(self):
        """
        Setup the network.
        """
        self.add_hq()
        for _ in range(self.num_farms):
            self.add_farms()
        for _ in range(self.num_dropoff_sites):
            self.add_dropoff_sites()
        total_nodes = self.num_farms + self.num_dropoff_sites + 1
        #connect all nodes
        for a in range(len(self.g.nodes)):
            for b in range(a+1, len(self.g.nodes)):
                self.g.add_edge(a, b, weight = random.randint(self.w_min, self.w_max))

    def draw_network(self):
        """
        Draw the network.
        """
        rcParams['figure.figsize'] = 14, 10
        pos = nx.spring_layout(self.g, scale=20, k=3/np.sqrt(self.g.order()))
        nx.draw(self.g, pos,
            with_labels=True,
            node_color = [self.colors[self.g.nodes[i]['n_type']] for i in self.g.nodes])
        edges_labels = nx.get_edge_attributes(self.g,'weight')
        nx.draw_networkx_edge_labels(self.g,pos,edge_labels=edges_labels)

        plt.show()

    def get_next_farm(self, loc):
        """
        Get the next farm to go to.

        Parameters
        ----------
        loc: int
            The current truck location

        Returns
        -------
        max_waste: int
            The maximum waste of the farms.
        farm_id: int
            The id of the farm with the maximum waste.
        """
        farm_waste_tuples = []
        for i in self.g.neighbors(loc):
            if self.g.nodes[i]['n_type'] == 'farm':
                farm_waste_tuples.append((self.g.nodes[i]['obj'].waste, i))
        max_waste, farm_id = max(farm_waste_tuples)
        return max_waste, farm_id

    def get_next_dropoff(self, loc):
        """
        Get the next dropoff to go to.

        Parameters
        ----------
        loc: int
            The current truck location

        Returns
        -------
        next_drop_dist: int
            The distance to the next dropoff.
        dropoff_id: int
            The id of the next dropoff.
        """
        dropoff_tuples = []
```

```
        for i in self.g.neighbors(loc):
            if self.g.nodes[i]['n_type'] == 'dropoff':
                next_drop_dist = nx.shortest_path_length(self.g, source = loc, target = i, weight = "weight")
                dropoff_tuples.append((next_drop_dist, i))
        next_drop_dist, dropoff_id = min(dropoff_tuples)
        return next_drop_dist, dropoff_id

    def dist_fuel_time_to_hq(self, truck, loc):
        """
        Get the distance, fuel, and time to the hq.

        Parameters
        ----------
        truck: Truck object
            The truck to get the distance, fuel, and time to the hq.
        loc: int
            The next site location that the truck might visit.

        Returns
        -------
        dist_to_hq: int
            The distance to the hq.
        fuel_to_hq: int
            The fuel to the hq.
        time_to_hq: int
            The time to the hq.
        """
        dist_to_hq = nx.shortest_path_length(self.g, source= loc, target = 0, weight= "weight")
        fuel_to_hq = dist_to_hq * truck.fuel_usage_rate
        time_to_hq = truck.get_time(dist_to_hq)
        return dist_to_hq, fuel_to_hq, time_to_hq

    def update_trucks(self, schedule):
        """
        Update the trucks.

        Parameters
        ----------
        schedule: SimPy schedule

        Returns
        -------
        None
        """
        for truck in self.trucks:
            # print("🚙 update truck count")
            self.truck_depart(schedule, truck)

    def go_to_droppoff(self, schedule, truck, next_drop_fuel, dropoff_id):
        """
        Go to the dropoff site.

        Parameters
        ----------
        schedule: SimPy schedule
        truck: Truck object
        next_drop_fuel: float
            The fuel needed to go to the dropoff site.
        dropoff_id: int
            The id of the dropoff site.

        Returns
        -------
        None
        """
        truck.waste = 0
        truck.loc = dropoff_id
        truck.cur_fuel -= next_drop_fuel
        schedule.add_event_after(
            self.drop_time,
            self.truck_depart, truck)

    def truck_depart(self, schedule, truck):
        """
        Truck depart from the current location.

        Parameters
        ----------
        schedule: SimPy schedule
        truck: Truck object
```

```
        Returns
        -------
        None
        """

        # if the capacity is full we consider go to the closest drop off sites
        if truck.waste == truck.max_waste:
            next_drop_dist, dropoff_id = self.get_next_dropoff(truck.loc)
            next_drop_fuel = next_drop_dist * truck.fuel_usage_rate
            next_drop_time = next_drop_dist/truck.speed
            dist_to_hq, fuel_to_hq, time_to_hq = self.dist_fuel_time_to_hq(truck, dropoff_id)
            # print('dist to hq', dist_to_hq)
            if  (truck.cur_fuel - next_drop_fuel) <= fuel_to_hq:
                print("Truck id", truck.id, "go back to HQ to refuel ⛽ (not drop), cur truck is at loc", truck.loc, ", cur fuel", truck.cur
                schedule.add_event_after(
                    time_to_hq, self.go_back_to_hq, truck)
            else:
                print(f"🪣 Truck {truck.id} go to dropoff")
                schedule.add_event_after(
                    next_drop_time,
                    self.go_to_droppoff, truck, next_drop_fuel, dropoff_id)

        # if not we consider going to the next farm
        else:
            farm_waste ,farm_id = self.get_next_farm(truck.loc)
            next_farm_dist = nx.shortest_path_length(self.g, source = truck.loc, target = farm_id, weight = 'weight')
            next_farm_fuel = next_farm_dist * truck.fuel_usage_rate
            next_farm_time = next_farm_dist/truck.speed
            # print("Max waste, farm id", max_waste, farm_id)

            dist_to_hq, fuel_to_hq, time_to_hq = self.dist_fuel_time_to_hq(truck, farm_id)

            #if we have enough fuel going back to hq after going to the next farm
            if  (truck.cur_fuel - next_farm_fuel) <= fuel_to_hq:
                print("Truck id", truck.id, "go back to HQ to refuel ⛽ (not farm), cur truck is at loc", truck.loc, ", cur fuel", truck.cur
                schedule.add_event_after(
                    time_to_hq, self.go_back_to_hq, truck)
            else:
                schedule.add_event_after(
                    next_farm_time,
                    self.truck_next_farm, truck, next_farm_fuel, farm_id)


    def truck_next_farm(self, schedule, truck, next_farm_fuel, farm_id):
        """
        Truck go to the next farm.

        Parameters
        ----------
        schedule: SimPy schedule
        truck: Truck object
        next_farm_fuel: float
        farm_id: int

        Returns
        -------
        None
        """
        truck.loc = farm_id
        truck_waste_capacity = truck.max_waste - truck.waste
        farm = self.g.nodes[farm_id]['obj']
        farm_cur_waste = farm.waste
        # print("🐷 Farm cur waste(id: waste)", self.farm_cur_waste())
        print(f"🚛 Truck id {truck.id} go to farm id", farm_id, "with waste", self.g.nodes[farm_id]['obj'].waste, "cur truck waste", truck.
        if truck_waste_capacity <= farm_cur_waste:
            print(f"    🛑 Truck {truck.id} finishs collecting the waste and reach full waste capacity")
            truck.load_waste(truck_waste_capacity)
            farm.update_waste(truck_waste_capacity)
        else:
            truck.load_waste(farm_cur_waste)
            farm.update_waste(farm_cur_waste)
        truck.update_fuel(next_farm_fuel)

        # print("Update farm waste", farm_id, self.g.nodes[farm_id]['obj'].waste)
        print("🚛 Truck id", truck.id, "Truck loc", truck.loc, "truck updated fuel", truck.cur_fuel, "truck waste", truck.waste)
        schedule.add_event_after(
            self.time_load_waste,
            self.truck_depart, truck)


    def go_back_to_hq(self, schedule, truck):
```

```
        """
        Truck go back to the hq.

        Parameters
        ----------
        schedule: SimPy schedule
        truck: Truck object

        Returns
        -------
        None
        """
        truck.loc = 0
        truck.refuel()
        schedule.add_event_after(
            5,
            self.truck_depart, truck)
        print(f"✅ Truck {truck.id} finish refuel in hq and its fuel is", truck.cur_fuel)

    def produce_waste(self, schedule):
        """
        This function makes sure that the farm accumulates waste constantly.

        Parameters
        ----------
        schedule: SimPy schedule

        Returns
        -------
        None
        """
        for farm in self.farms:
            farm.add_waste(2)
        schedule.add_event_after(
                5,
                self.produce_waste)
        print("🐷 Farms add waste now, cur waste(id: waste)", self.farm_cur_waste())

    def run(self, schedule):
        """
        This function runs the simulation.

        Parameters
        ----------
        schedule: SimPy schedule

        Returns
        -------
        None
        """
        schedule.add_event_after(
            0, self.produce_waste)
        schedule.add_event_after(
            0, self.update_trucks)

    def farm_cur_waste(self):
        """
        This function returns the current waste of each farm.

        Parameters
        ----------
        None

        Returns
        -------
        farms_tuples: list
            A list of tuples, each tuple contains the farm id and the current waste of the farm.
        """
        farms_tuples =[]
        for farm in self.farms:
            farms_tuples.append((farm.n_id, farm.waste))
        return farms_tuples

    def fuel_use_per_truck(self):
        """
        This function returns the fuel use of each truck.

        Parameters
        ----------
        None
```

```
        Returns
        -------
        truck_fuels: list
            A list of tuples, each tuple contains the truck id and the fuel use of the truck.
        """
        truck_fuels = []
        for truck in self.trucks:
            truck_fuels.append((truck.id, truck.fuel_use))
        return truck_fuels

    def waste_collect_per_truck(self):
        """
        This function returns the waste collected by each truck.

        Parameters
        ----------
        None

        Returns
        -------
        truck_waste: list
            A list of tuples, each tuple contains the truck id and the all the waste collected by the truck.
        """
        truck_waste = []
        for truck in self.trucks:
            truck_waste.append((truck.id, truck.all_waste))
        return truck_waste
```

```
schedule = Schedule()
road_network = RoadNetwork()
road_network.run(schedule)
while schedule.now <= 35:
    schedule.run_next_event()
    # schedule.print_events()
print()
print("🎯 Final statistics 🎯")
print("farm current waste (farm id: waste)", road_network.farm_cur_waste())
print("fuel_use_per_truck (truck id: fuel)", road_network.fuel_use_per_truck())
print("waste_collect_per_truck (truck id: waste)", road_network.waste_collect_per_truck())
```

Simulation

```
class Farm():
    """
    Farm class to store the waste generated by the farm.

    Parameters
    ----------
    n_id : int
        The id of the farm.
    waste : int
        The amount of waste generated by the farm.
    waste_rate_dist : scipy.stats.rv_continuous
        The distribution of the waste rate.
    """
    def __init__(self, n_id):
        self.waste = 0
        self.waste_rate_dist = sts.halfcauchy(0, 5)
        self.n_id = n_id

    def add_waste(self, new_waste):
        """
        Add waste to the farm.
        """
        self.waste += new_waste

    def update_waste(self, recyled_waste):
        """
        Update the waste generated by the farm.
        """
        self.waste -= recyled_waste

    def clear_waste(self):
        """
```

```
        Clear the waste generated by the farm.
        """
        self.waste = 0

class Dropoff():
    """
    Dropoff class to store the waste generated by the farm.
    """
    def __init__(self, n_id):
        self.id = n_id
```

```
class Truck:
    """
    Truck class that will be used to collect waste from farms and drop it off at the dropoff location

    Parameters
    ----------
    id : int
        The id of the truck.
    max_waste : int
        The maximum amount of waste the truck can hold.
    max_fuel : int
        The maximum amount of fuel the truck can hold.
    cur_fuel : int
        The current amount of fuel the truck has.
    loc : int
        The current location of the truck.
    speed : int
        The speed of the truck.
    fuel_usage_rate : float
        The rate at which the truck uses fuel per km.
    waste : int
        The amount of waste the truck is currently holding.
    all_waste : int
        The total amount of waste the truck has collected.
    all_fuel : int
        The total amount of fuel the truck has used.

    """
    def __init__(self, id):
        self.id = id
        self.max_waste = 2800
        self.max_fuel = 150
        self.cur_fuel = 150
        self.loc = 0 #depart from hq
        self.speed = sts.norm(50, 5)
        self.fuel_usage_rate = 0.08 #the amount of fuel spending per km
        self.waste = 0
        self.all_waste = 0
        self.fuel_use = 0

    def get_time(self, distance):
        """
        Get the time it takes to travel a certain distance
        """
        return distance / self.speed.rvs()

    def update_fuel(self, use):
        """
        Update the fuel the truck has
        """
        self.cur_fuel -= use
        self.fuel_use += use

    def refuel(self):
        """
        Refuel the truck
        """
        self.cur_fuel = self.max_fuel


    def load_waste(self, new_waste):
        """
        Load waste into the truck
        """
        self.waste += new_waste
        self.all_waste += new_waste
```

```
class RoadNetwork:
    """
    RoadNetwork class to store the road network.


    Parameters
    ----------
    g : networkx.Graph
        The graph of the network.
    num_farms : int
        The number of farms in the network.
    num_dropoff_sites : int
        The number of dropoff sites in the network.
    curr_node_id : int
        The current node id.
    w_min : int
        The minimum amount of distance between two nodes.
    w_max : int
        The maximum amount of distance between two nodes.
    time_load_waste : int
        The time it takes to load waste into the truck.
    colors : dict
        The colors of the nodes.
    n_trucks : int
        The number of trucks in the network.
    drop_time : int
        The time it takes to drop off waste at the dropoff site.
    trucks : list
        The list of trucks in the network.
    farms : list
        The list of farms in the network.
    dropoff_sites : list
        The list of dropoff sites in the network.
    strategy : str
        The strategy used to determine which farm to visit next.
    produce_waste_time : int
        The time it takes to produce waste at the farm.
    visited_farms : int
        The number of farms visited.
    visited_dropoff_sites : int
        The number of dropoff sites visited.
    visited_hq : int
        The number of times the truck has visited the hq.
    """

    def __init__(self, num_farms, num_dropoff_sites, n_trucks, strategy):
        self.g = nx.Graph()
        self.num_farms = num_farms
        self.num_dropoff_sites = num_dropoff_sites
        self.curr_node_id = 1
        self.w_min = 100
        self.w_max = 1500
        self.time_load_waste = sts.norm(5, 0.1)
        self.colors = {'hq': 'green', 'farm': 'lightblue', 'dropoff': 'y'}
        self.n_trucks = n_trucks
        self.strategy = 'max_waste'
        self.drop_time = sts.norm(3, 0.1)
        self.produce_waste_time = sts.norm(5, 0.1)
        self.trucks = [Truck(id) for id in range(self.n_trucks)]
        self.farms = []
        self.dropoff_sites = []
        self.visited_hq = 0
        self.visited_farms = 0
        self.visited_dropoff = 0

        self.network_setup()
        # self.draw_network()


    def add_hq(self):
        """
        Add the hq to the network.
        """
        self.g.add_node(0, n_type = 'hq')

    def add_farms(self):
        """
        Add farms to the network.
        """
```

```
        new_farm = Farm(self.curr_node_id)
        self.farms.append(new_farm)
        self.g.add_node(self.curr_node_id, n_type = 'farm',
                        obj = new_farm)
        self.curr_node_id += 1

    def add_dropoff_sites(self):
        """
        Add dropoff sites to the network.
        """
        new_dropoff_sites = Dropoff(self.curr_node_id)
        self.dropoff_sites.append(new_dropoff_sites)
        self.g.add_node(self.curr_node_id, n_type = 'dropoff',
                        obj = new_dropoff_sites)
        self.curr_node_id += 1

    def network_setup(self):
        """
        Setup the network.
        """
        self.add_hq()
        for _ in range(self.num_farms):
            self.add_farms()
        for _ in range(self.num_dropoff_sites):
            self.add_dropoff_sites()
        total_nodes = self.num_farms + self.num_dropoff_sites + 1
        #connect all nodes
        for a in range(len(self.g.nodes)):
            for b in range(a+1, len(self.g.nodes)):
                self.g.add_edge(a, b, weight = random.randint(self.w_min, self.w_max))

    def draw_network(self):
        """
        Draw the network.
        """
        node_sizes = []
        for node_id in self.g.nodes:
            if self.g.nodes[node_id]['n_type'] == 'farm':
                node_sizes.append(road_network.g.nodes[node_id]['obj'].waste * 10 + 200)
            elif self.g.nodes[node_id]['n_type'] == 'dropoff':
                node_sizes.append(200)
            elif self.g.nodes[node_id]['n_type'] == 'hq':
                node_sizes.append(500)

        pos = nx.spring_layout(self.g, scale=20, k=3/np.sqrt(self.g.order()))
        nx.draw(self.g, pos,
            with_labels=True,
            node_color = [self.colors[road_network.g.nodes[i]['n_type']] for i in self.g.nodes],
            node_size= node_sizes)
        edges_labels = nx.get_edge_attributes(self.g,'weight')
        nx.draw_networkx_edge_labels(self.g,pos,edge_labels=edges_labels)
        plt.show()

    def get_next_farm(self, loc):
        """
        Get the next farm to go to.

        Parameters
        ----------
        loc: int
            The current truck location

        Returns
        -------
        waste: int
            The waste of the selected farm.
        farm_id: int
            The id of the selected farm
        """
        farm_waste_tuples = []


        for i in self.g.neighbors(loc):
            if self.g.nodes[i]['n_type'] == 'farm':
                farm_waste_tuples.append((self.g.nodes[i]['obj'].waste, i))

        if self.strategy == 'max_waste':
            max_waste, farm_id = max(farm_waste_tuples)
            return max_waste, farm_id
        elif self.strategy == 'random':
            random_waste, farm_id = random.choice(farm_waste_tuples)
```

```
            return random_waste, farm_id

        elif self.strategy == 'closest_farm':
            farm_dist_tuples = []
            for i in self.g.neighbors(loc):
                if self.g.nodes[i]['n_type'] == 'farm':
                    farm_dist_tuples.append((nx.shortest_path_length(self.g, source = loc, target = i, weight = "weight"), i))
            min_dist_waste, farm_id = min(farm_dist_tuples)
            return min_dist_waste, farm_id


    def get_next_dropoff(self, loc):
        """
        Get the next dropoff to go to.

        Parameters
        ----------
        loc: int
            The current truck location

        Returns
        -------
        next_drop_dist: int
            The distance to the next dropoff.
        dropoff_id: int
            The id of the next dropoff.
        """
        dropoff_tuples = []
        for i in self.g.neighbors(loc):
            if self.g.nodes[i]['n_type'] == 'dropoff':
                next_drop_dist = nx.shortest_path_length(self.g, source = loc, target = i, weight = "weight")
                dropoff_tuples.append((next_drop_dist, i))
        next_drop_dist, dropoff_id = min(dropoff_tuples)
        return next_drop_dist, dropoff_id

    def dist_fuel_time_to_hq(self, truck, loc):
        """
        Get the distance, fuel, and time to the hq.

        Parameters
        ----------
        truck: Truck object
            The truck to get the distance, fuel, and time to the hq.
        loc: int
            The next site location that the truck might visit.

        Returns
        -------
        dist_to_hq: int
            The distance to the hq.
        fuel_to_hq: int
            The fuel to the hq.
        time_to_hq: int
            The time to the hq.
        """
        dist_to_hq = nx.shortest_path_length(self.g, source= loc, target = 0, weight= "weight")
        fuel_to_hq = dist_to_hq * truck.fuel_usage_rate
        time_to_hq = truck.get_time(dist_to_hq)
        return dist_to_hq, fuel_to_hq, time_to_hq

    def update_trucks(self, schedule):
        """
        Update the trucks.

        Parameters
        ----------
        schedule: SimPy schedule

        Returns
        -------
        None
        """
        for truck in self.trucks:
            # print("🚙 update truck count")
            self.truck_depart(schedule, truck)

    def go_to_droppoff(self, schedule, truck, next_drop_fuel, dropoff_id):
        """
        Go to the dropoff site.

        Parameters
```

```
        ----------
        schedule: SimPy schedule
        truck: Truck object
        next_drop_fuel: float
            The fuel needed to go to the dropoff site.
        dropoff_id: int
            The id of the dropoff site.

        Returns
        -------
        None
        """
        self.visited_dropoff += 1
        truck.waste = 0
        truck.loc = dropoff_id
        truck.cur_fuel -= next_drop_fuel
        schedule.add_event_after(
            self.drop_time.rvs(),
            self.truck_depart, truck)

    def truck_depart(self, schedule, truck):
        """
        Truck depart from the current location.

        Parameters
        ----------
        schedule: SimPy schedule
        truck: Truck object

        Returns
        -------
        None
        """

        # if the capacity is full we consider go to the closest drop off sites
        if truck.waste == truck.max_waste:
            next_drop_dist, dropoff_id = self.get_next_dropoff(truck.loc)
            next_drop_fuel = next_drop_dist * truck.fuel_usage_rate
            next_drop_time = next_drop_dist/truck.speed.rvs()
            dist_to_hq, fuel_to_hq, time_to_hq = self.dist_fuel_time_to_hq(truck, dropoff_id)
            # print('dist to hq', dist_to_hq)
            if  (truck.cur_fuel - next_drop_fuel) <= fuel_to_hq:
                # print("Truck id", truck.id, "go back to HQ to refuel ⛽ (not drop), cur truck is at loc", truck.loc, ", cur fuel", truck.c
                schedule.add_event_after(
                    time_to_hq, self.go_back_to_hq, truck)
            else:
                # print(f"🗑 Truck {truck.id} go to dropoff")
                schedule.add_event_after(
                    next_drop_time,
                    self.go_to_droppoff, truck, next_drop_fuel, dropoff_id)

        # if not we consider going to the next farm
        else:
            farm_waste ,farm_id = self.get_next_farm(truck.loc)
            next_farm_dist = nx.shortest_path_length(self.g, source = truck.loc, target = farm_id, weight = 'weight')
            next_farm_fuel = next_farm_dist * truck.fuel_usage_rate
            next_farm_time = next_farm_dist/truck.speed.rvs()
            # print("Max waste, farm id", max_waste, farm_id)

            dist_to_hq, fuel_to_hq, time_to_hq = self.dist_fuel_time_to_hq(truck, farm_id)

            #if we have enough fuel going back to hq after going to the next farm
            if  (truck.cur_fuel - next_farm_fuel) <= fuel_to_hq:
                # print("Truck id", truck.id, "go back to HQ to refuel ⛽ (not farm), cur truck is at loc", truck.loc, ", cur fuel", truck.c
                schedule.add_event_after(
                    time_to_hq, self.go_back_to_hq, truck)
            else:
                schedule.add_event_after(
                    next_farm_time,
                    self.truck_next_farm, truck, next_farm_fuel, farm_id)


    def truck_next_farm(self, schedule, truck, next_farm_fuel, farm_id):
        """
        Truck go to the next farm.

        Parameters
        ----------
        schedule: SimPy schedule
        truck: Truck object
        next_farm_fuel: float
```

```
        farm_id: int

        Returns
        -------
        None
        """
        self.visited_farms += 1
        truck.loc = farm_id
        truck_waste_capacity = truck.max_waste - truck.waste
        farm = self.g.nodes[farm_id]['obj']
        farm_cur_waste = farm.waste
        # print("🐷 Farm cur waste(id: waste)", self.farm_cur_waste())
        # print(f"🚛 Truck id {truck.id} go to farm id", farm_id, "with waste", self.g.nodes[farm_id]['obj'].waste, "cur truck waste", truc
        if truck_waste_capacity <= farm_cur_waste:
            # print(f"    🛑 Truck {truck.id} finishs collecting the waste and reach full waste capacity")
            truck.load_waste(truck_waste_capacity)
            farm.update_waste(truck_waste_capacity)
        else:
            truck.load_waste(farm_cur_waste)
            farm.update_waste(farm_cur_waste)
        truck.update_fuel(next_farm_fuel)

        # print("Update farm waste", farm_id, self.g.nodes[farm_id]['obj'].waste)
        # print("🚛 Truck id", truck.id, "Truck loc", truck.loc, "truck updated fuel", truck.cur_fuel, "truck waste", truck.waste)
        schedule.add_event_after(
            self.time_load_waste.rvs(),
            self.truck_depart, truck)


    def go_back_to_hq(self, schedule, truck):
        """
        Truck go back to the hq.

        Parameters
        ----------
        schedule: SimPy schedule
        truck: Truck object

        Returns
        -------
        None
        """
        self.visited_hq += 1
        truck.loc = 0
        truck.refuel()
        schedule.add_event_after(
            5,
            self.truck_depart, truck)
        # print(f"✅ Truck {truck.id} finish refuel in hq and its fuel is", truck.cur_fuel)

    def produce_waste(self, schedule):
        """
        This function makes sure that the farm accumulates waste constantly.

        Parameters
        ----------
        schedule: SimPy schedule

        Returns
        -------
        None
        """
        for farm in self.farms:
            farm.add_waste(farm.waste_rate_dist.rvs())
        schedule.add_event_after(
                self.produce_waste_time.rvs(),
                self.produce_waste)
        # print("🐷 Farms add waste now, cur waste(id: waste)", self.farm_cur_waste())

    def run(self, schedule):
        """
        This function runs the simulation.

        Parameters
        ----------
        schedule: SimPy schedule

        Returns
        -------
        None
        """
```

```
        schedule.add_event_after(
            0, self.produce_waste)
        schedule.add_event_after(
            0, self.update_trucks)

    def farm_cur_waste(self):
        """
        This function returns the current waste of each farm.

        Parameters
        ----------
        None

        Returns
        -------
        farms_tuples: list
            A list of tuples, each tuple contains the farm id and the current waste of the farm.
        """
        farms_tuples ={}
        for farm in self.farms:
            farms_tuples[farm.n_id] = farm.waste
        return farms_tuples

    def fuel_use_per_truck(self):
        """
        This function returns the fuel use of each truck.

        Parameters
        ----------
        None

        Returns
        -------
        truck_fuels: list
            A list of tuples, each tuple contains the truck id and the fuel use of the truck.
        """
        truck_fuels = {}
        for truck in self.trucks:
            truck_fuels[truck.id] = truck.fuel_use
        return truck_fuels

    def waste_collect_per_truck(self):
        """
        This function returns the waste collected by each truck.

        Parameters
        ----------
        None

        Returns
        -------
        truck_waste: list
            A list of tuples, each tuple contains the truck id and the all the waste collected by the truck.
        """
        truck_waste = {}
        for truck in self.trucks:
            truck_waste[truck.id] = truck.all_waste
        return truck_waste
```

```
num_farms = 5
num_dropoff_sites = 2
n_trucks = 5

collected_total_wastes = []
schedule = Schedule()
road_network = RoadNetwork(num_farms, num_dropoff_sites, n_trucks, strategy = 'max_waste')
road_network.run(schedule)
road_network.draw_network()
run = 0
time = []
while schedule.now <= 1000:
    run += 1
    schedule.run_next_event()
    if run % 10 == 0:
        time.append(schedule.now)
        collected_total_wastes.append(sum(road_network.waste_collect_per_truck().values()))
print()
print("🎯 Final statistics 🎯")
```

```
print("farm current waste (farm id: waste)", road_network.farm_cur_waste())
print("fuel_use_per_truck (truck id: fuel)", road_network.fuel_use_per_truck())
print("waste_collect_per_truck (truck id: waste)", road_network.waste_collect_per_truck())

road_network.draw_network()
```

```
from matplotlib import animation
from IPython.display import HTML

num_farms = 5
num_dropoff_sites = 2
n_trucks = 5


schedule = Schedule()
road_network = RoadNetwork(num_farms, num_dropoff_sites, n_trucks, strategy = 'max_waste')
road_network.run(schedule)


def simple_update(n):
    while schedule.now <= n*10:
        schedule.run_next_event()
    node_sizes = []
    for node_id in road_network.g.nodes:
        if road_network.g.nodes[node_id]['n_type'] == 'farm':
            node_sizes.append(road_network.g.nodes[node_id]['obj'].waste * 10 + 200)
        elif road_network.g.nodes[node_id]['n_type'] == 'dropoff':
            node_sizes.append(200)
        elif road_network.g.nodes[node_id]['n_type'] == 'hq':
            node_sizes.append(500)

    pos = nx.spring_layout(road_network.g, scale=20, k=3/np.sqrt(road_network.g.order()), seed = 1)
    nx.draw(road_network.g, pos,
        with_labels=True,
        node_color = [road_network.colors[road_network.g.nodes[i]['n_type']] for i in road_network.g.nodes],
        node_size= node_sizes)
    edges_labels = nx.get_edge_attributes(road_network.g,'weight')
    plot = nx.draw_networkx_edge_labels(road_network.g,pos,edge_labels=edges_labels)
    return plot


def simple_animation():
    fig, ax = plt.subplots()
    rcParams['figure.figsize'] = 14, 10
    ani = animation.FuncAnimation(fig, simple_update, frames = 100, interval=100, repeat=False)
    output = HTML(ani.to_html5_video())
    return output

simple_animation()
```

```
road_network.visited_hq, road_network.visited_farms, road_network.visited_dropoff
```

```
plt.plot(time, collected_total_wastes)
plt.title("Total waste collected over time")
plt.xlabel("Time")
plt.ylabel("Total waste collected")
```

```
trials = 300
num_farms = 50
num_dropoff_sites = 10
n_trucks = 5
def experiment(trials, num_farms, num_dropoff_sites, n_trucks, strategy):
    collected_total_wastes = []
    total_fuels = []
    total_profits = []
    visited_hq = []
    visited_farms = []
    visited_dropoff = []
```

```
    for _ in tqdm(range(trials)):

        schedule = Schedule()
        road_network = RoadNetwork(num_farms, num_dropoff_sites, n_trucks, strategy)
        road_network.run(schedule)
        while schedule.now <= 1000:
            schedule.run_next_event()

        sum_waste = sum(road_network.waste_collect_per_truck().values())
        sum_fuel = sum(road_network.fuel_use_per_truck().values())
        profit = sum_waste *200 - sum_fuel*0.25
        collected_total_wastes.append(sum_waste)
        total_fuels.append(sum_fuel)
        total_profits.append(profit)

        visited_hq.append(road_network.visited_hq)
        visited_farms.append(road_network.visited_farms)
        visited_dropoff.append(road_network.visited_dropoff)



    return collected_total_wastes, total_fuels, total_profits, np.mean(visited_hq), np.mean(visited_farms), np.mean(visited_dropoff)

max_waste_total_wastes, max_waste_total_fuels, max_waste_total_profits, max_visited_hq, max_visited_farms,max_visited_dropoff  = experiment
random_total_wastes, random_total_fuels, random_total_profits, random_visited_hq, random_visited_farms, random_visited_dropoff = experiment
closest_farm_total_wastes, closest_total_fuels, closest_total_profits, closest_visited_hq, closest_random_visited_farms, closest_visited_dr
```

```
print('Max strategy: count visited hq', max_visited_hq, ", count visited farms", max_visited_farms, ", count visited dropoff", max_visited_
print('Random strategy: count visited hq', random_visited_hq, ", count visited farms", random_visited_farms, ", count visited dropoff", ran
print('Closest farm strategy: count visited hq', closest_visited_hq, ", count visited farms", closest_random_visited_farms, ", count visite

sns.histplot(max_waste_total_wastes, label="Max waste strategy", color = 'red')
sns.histplot(random_total_wastes, label = "Random strategy", color = 'yellow')
sns.histplot(closest_farm_total_wastes, label = "Closest farm strategy")
plt.title("Experiments of total waste collected for different strategies")
plt.ylabel("Frequency")
plt.xlabel("Total waste collected")
plt.legend()
```

```
def boxplot(max_waste, random, closest_farm):
     fig, ax = plt.subplots()
     x = [max_waste,random, closest_farm]
     palette = ['red', 'lightblue', 'yellow']
     box = ax.boxplot(x, patch_artist=True)
     for patch, color in zip(box['boxes'], palette):
          patch.set_facecolor(color)
     for median in box['medians']:
          median.set_color('black')
     ax.set_xticklabels(["Max waste", "Random", "Closest farm"])

boxplot(max_waste_total_wastes, random_total_wastes, closest_farm_total_wastes)
plt.title("Comparing strategies for total waste collected")
plt.ylabel("Total waste collected")
```

```
def statistics_cal(variable, max_waste, random, closest_farm):
    print(f'Average {variable} for max waste strategy ', round(np.mean(max_waste)), 'and its std', round(np.std(max_waste)))
    print('     Its confidence interval is', sts.t.interval(0.95, len(max_waste)-1, loc=np.mean(max_waste), scale=sts.sem(max_waste)))
    print(f'Average {variable} for random strategy ', round(np.mean(random)), 'and its std', round(np.std(random)))
    print('     Its confidence interval is', sts.t.interval(0.95, len(random)-1, loc=np.mean(random), scale=sts.sem(random)))
    print(f'Average {variable} for closest farm strategy ', round(np.mean(closest_farm)), 'and its std', round(np.std(closest_farm)))
    print('     Its confidence interval is', sts.t.interval(0.95, len(closest_farm)-1, loc=np.mean(closest_farm), scale=sts.sem(closest_far

statistics_cal('waste', max_waste_total_wastes, random_total_wastes, closest_farm_total_wastes)
```

```
sns.histplot(max_waste_total_fuels, label="Max waste strategy", color = 'red')
sns.histplot(random_total_fuels, label = "Random strategy", color = 'yellow')
sns.histplot(closest_total_fuels, label = "Closest farm strategy")
plt.title("Experiments of total fuel used for different strategies")
plt.ylabel("Frequency")
plt.xlabel("Total fuel used")
```

```
plt.legend()

statistics_cal('fuel used', max_waste_total_fuels, random_total_fuels, closest_total_fuels)
boxplot(max_waste_total_fuels, random_total_fuels, closest_total_fuels)
plt.title("Comparing strategies for total fuel used")
plt.ylabel("Total fuel used")
```

```
sns.histplot(max_waste_total_profits, label="Max waste strategy", color = 'red')
sns.histplot(random_total_profits, label = "Random strategy", color = 'yellow')
sns.histplot(closest_total_profits, label = "Closest farm strategy")
plt.title("Experiments of total profits for different strategies")
plt.ylabel("Frequency")
plt.xlabel("Total profits")
plt.legend()

statistics_cal('profits', max_waste_total_profits, random_total_profits, closest_total_profits)
boxplot(max_waste_total_profits, random_total_profits, closest_total_profits)
plt.title("Comparing strategies for total profits")
plt.ylabel("Total profits")
```

```
plt.hist(list(sts.halfcauchy(0,1).rvs(10000)))
plt.title("10000 trials half cauchy distribution for producing waste")
plt.xlabel("Amount of waste produced")
plt.ylabel("frequency")
```

```
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
from matplotlib import animation
from IPython.display import HTML

num_farms = 5
num_dropoff_sites = 3
n_trucks = 2


schedule = Schedule()
road_network = RoadNetwork(num_farms, num_dropoff_sites, n_trucks, strategy = 'max_waste')
road_network.run(schedule)



def simple_update(n):
    run = 0
    while schedule.now <= 35:
        schedule.run_next_event()
    node_sizes = []
    for node_id in road_network.g.nodes:
        if road_network.g.nodes[node_id]['n_type'] == 'farm':
            node_sizes.append(road_network.g.nodes[node_id]['obj'].waste * 100 + 200)
        elif road_network.g.nodes[node_id]['n_type'] == 'dropoff':
            node_sizes.append(200)
        elif road_network.g.nodes[node_id]['n_type'] == 'hq':
            node_sizes.append(500)

    pos = nx.spring_layout(road_network.g, scale=20, k=3/np.sqrt(road_network.g.order()))
    nx.draw(road_network.g, pos,
        with_labels=True,
        node_color = [road_network.colors[road_network.g.nodes[i]['n_type']] for i in road_network.g.nodes],
        node_size= node_sizes)
    edges_labels = nx.get_edge_attributes(road_network.g,'weight')
    plot = nx.draw_networkx_edge_labels(road_network.g,pos,edge_labels=edges_labels)
    return plot


def simple_animation():
    fig, ax = plt.subplots()
    rcParams['figure.figsize'] = 14, 10
    ani = animation.FuncAnimation(fig, simple_update, frames = 5)
    output = HTML(ani.to_html5_video())
    plt.show()
    return output
```

```
simple_animation()
```