Urban Logistics for Goods Delivery

Carlos Veríssimo - 201907716
Edgar Lourenço - 201604910
Nuno Jesus - 201905477
G108

Problem description

Implement and specify a management platform for a urban logistics company with the intent of making its operation the most efficient. Different strategies for combining deliveries and delivery people registered in the platform will be explored.

General Formalization

All scenarios follow the same structure datawise:

- Two different classes have been implemented, Van and Order
- Two vectors are used throughout the code to aggregate objects from the same class.
- A generic sorter function has been implemented:
  - Receives as an argument the attribute by which we wish to order the vector.
  - Also receives a boolean that specifies ascending or descending order.

Scenario 1

In the first scenario, the company needs to fulfill all the deliveries that have been accumulating in the warehouse while minimizing the number of delivery people needed to complete such task.

Each delivery person can only make one and only one journey each day, i.e, after loading up the van and delivering all orders, they are done for the day.

Eventually, if some orders are left in the warehouse, they can be delivered in the next day.

Formalization of Scenario 1

Decision Variables:

- The used van maximum weight: $\mathbf{w_e}$
- The used van maximum volume: $\mathbf{v_e}$
- The number of used vans: **ne**
- The used order weight: $\mathbf{w_p}$
- The used order volume: $\mathbf{v_p}$
- The number of packed orders: **np**

Restrictions:

- A van can only be used once per day;
- Each van can not exceed its maximum weight: $\mathbf{\Sigma\ (w_p) <= w_e}$
- … neither its maximum volume: $\mathbf{\Sigma\ (v_p) <= v_e}$

Objective:

- Minimize the company's used vans: **ne**

Relevant Algorithms and Complexity

This scenario was approached with a **greedy** strategy:

- Firstly, the vans are sorted from highest sum(volume+weight) to lowest.
- After that, orders are ordered in the same way.

By filling the largest vans with the largest orders, we ensure that we use the minimum number of vans to complete all orders.

Complexity wise, the algorithm applied above is **O(n log(n) + n^2)** in time, since:

- we use the C++ **std::sort()** function to sort vectors: **O(n log(n))**
- a nested *for* loop is used to find the best pack deliveries for each van: **O(n^2)**

... and **O(n)** in space since we only use vectors and other trivial types of variables to store the needed data.

Results

The attribute by which each vector was ordered greatly affects the final results. These are the results of executing the program with the provided datasets and the different sorts of the van vector.

Max. Volume	DAY 1 -> Vans used: 24 -> Orders delivered: 450 -> Orders in the warehouse: 0 -> Company's efficiency: 100% -> Standard Deviation: 2.00998 DAY 2 -> Vans used: 0 -> Orders delivered: 0 -> Orders in the warehouse: 0 -> Company's efficiency: 100% -> Standard Deviation: 2.00998	Cost	DAY 1 -> Vans used: 24 -> Orders delivered: 450 -> Orders in the warehouse: 0 -> Company's efficiency: 100% -> Standard Deviation: 1.93907 DAY 2 -> Vans used: 0 -> Orders delivered: 0 -> Orders in the warehouse: 0 -> Company's efficiency: 100% -> Standard Deviation: 1.93907
Max. Weight	DAY 1 -> Vans used: 23 -> Orders delivered: 450 -> Orders in the warehouse: 0 -> Company's efficiency: 100% -> Standard Deviation: 2.22711 DAY 2 -> Vans used: 0 -> Orders delivered: 0 -> Orders in the warehouse: 0 -> Company's efficiency: 100% -> Standard Deviation: 2.22711	Max. Capacity (Volume + Weight)	DAY 1 -> Vans used: 22 -> Orders delivered: 450 -> Orders in the warehouse: 0 -> Company's efficiency: 100% -> Standard Deviation: 1.04403 DAY 2 -> Vans used: 0 -> Orders delivered: 0 -> Orders in the warehouse: 0 -> Company's efficiency: 100% -> Standard Deviation: 1.04403

Scenario 2

In this scenario we take into account that:

- for each delivered order the company receives a reward for it;
- for each van used, the company pays the corresponding fee;

Our goal is to maximize the company's net profit, that is, the difference between the total revenue made from completed deliveries and the total amount paid to the delivery people.

Formalization of Scenario 2

Decision Variables:

- The used van maximum weight: $\mathbf{w_e}$
- The used van maximum volume: $\mathbf{v_e}$
- The used van fee: $\mathbf{c_e}$
- The number of used vans: **ne**
- The used order weight: $\mathbf{w_p}$
- The used order volume: $\mathbf{v_p}$
- The used order reward: $\mathbf{r_p}$
- The number of packed orders: **np**

Restrictions:

- A van can only be used once per day;
- Each van can not exceed its maximum weight: $\mathbf{\Sigma\ (w_p) <= w_e}$
- … neither its maximum volume: $\mathbf{\Sigma\ (v_p) <= v_e}$

Objective:

- Maximize the company's profit: $\mathbf{\Sigma\ (r_p) - \Sigma\ (c_e)}$

Relevant Algorithms and Complexity

This scenario was approached with a **greedy** strategy:

- Firstly, the deliveries are sorted from highest to lowest rewarding.
- After that, vans are ordered from lowest to highest fee.

By filling the vans whose cost is the lowest with the highest rewarding deliveries, we ensure an optimal solution for any dataset.

Complexity wise, the algorithm applied above is **O(n log(n) + n^2)** in time, since:

- we use the C++ **std::sort()** function to sort vectors: **O(n log(n))**
- a nested *for* loop is used to find the best pack deliveries for each van: **O(n^2)**

… and **O(n)** in space since we only use vectors and other trivial types of variables to store the needed data.

```
Vans: index -> maxVolume maxWeight cost

0 -> 255 275 13025
1 -> 325 316 13378
2 -> 288 346 13476
3 -> 391 395 13796
4 -> 344 306 14015
5 -> 256 289 14860
6 -> 265 274 14952
7 -> 382 304 14967
8 -> 343 277 15226
9 -> 365 259 16982

Orders: index -> volume weight reward duration(s)

0 -> 3 19 900 827
1 -> 16 13 800 703
2 -> 19 16 700 865
3 -> 19 28 600 1021
4 -> 16 26 500 728
5 -> 6 26 400 318
6 -> 25 28 300 535
7 -> 11 11 200 641
8 -> 3 19 100 984
```

Results

	Orders set 1	Orders set 2	Orders set 3
Vans set 1	-> Company's profit: 196502€ -> Vans used: 18 -> Orders delivered: 341 -> Orders in the warehouse: 109 -> Company's efficiency: 75.8% -> Standard Deviation: 2.37	-> Company's profit: 18559€ -> Vans used: 3 -> Orders delivered: 50 -> Orders in the warehouse: 0 -> Company's efficiency: 100% -> Standard Deviation: 7.82	-> Company's profit: 2650€ -> Vans used: 1 -> Orders delivered: 15 -> Orders in the warehouse: 5 -> Company's efficiency: 75% -> Standard Deviation: 4.5
Vans set 2	-> Company's profit: 149889€ -> Vans used: 9 -> Orders delivered: 168 -> Orders in the warehouse: 282 -> Company's efficiency: 37.3% -> Standard Deviation: 6.24	-> Company's profit: 20226€ -> Vans used: 2 -> Orders delivered: 32 -> Orders in the warehouse: 18 -> Company's efficiency: 64% -> Standard Deviation: 6.42	-> Company's profit: 2650€ -> Vans used: 1 -> Orders delivered: 15 -> Orders in the warehouse: 5 -> Company's efficiency: 75% -> Standard Deviation: 4.5
Vans set 3	-> Company's profit: 32702€ -> Vans used: 1 -> Orders delivered: 22 -> Orders in the warehouse: 428 -> Company's efficiency: 4.89% -> Standard Deviation: 6.26	-> Company's profit: 25098€ -> Vans used: 1 -> Orders delivered: 22 -> Orders in the warehouse: 28 -> Company's efficiency: 44% -> Standard Deviation: 6.26	-> Company's profit: 7175€ -> Vans used: 1 -> Orders delivered: 20 -> Orders in the warehouse: 0 -> Company's efficiency: 100% -> Standard Deviation: 5.69

Scenario 3

In the last scenario, a new category of delivery is introduced, **express delivery.**

For this type of delivery, only a single van is used by the company and only one delivery can be made for each trip, regardless of its volume or weight.

Deliveries can only be made in the 9am to 5pm time window.

The goal is to maximize the number of express deliveries completed while also minimizing the average time per delivery.

Formalization of Scenario 3

Decision Variables:

- The used order delivery time (in seconds): $\mathbf{t_p}$
- The number of packed orders: **np**

Restrictions:

- All orders must be delivered between 9am and 5pm (time frame of 28800 seconds): $\mathbf{\Sigma\ (t_p) <= 28800}$

Objective:

- Minimize the average time of deliveries: $\mathbf{\Sigma\ (t_p)\ /\ np}$

Relevant Algorithms and Complexity

A **greedy** approach was taken in order to arrive at an optimal solution:

- Deliveries have been ordered by least to greatest time to complete.
- In order to minimize the average time to complete a delivery, while maximizing the number of completed orders, we fill the van with the least time consuming deliveries.

Complexity wise, the algorithm applied above is **O(n log(n) + n)** in time, since:

- we use the C++ **std::sort()** function to sort vectors: **O(n log(n))**
- a single *while* loop is used to pack the van inside the time frame: **O(n)**

... and **O(n)** in space since we only use vectors and other trivial types of variables to store the needed data.

```
Orders: index -> volume weight reward duration(s)

0 -> 6 26 400 318
1 -> 25 28 300 535
2 -> 11 11 200 641
3 -> 16 13 800 703
4 -> 16 26 500 728
5 -> 3 19 900 827
6 -> 19 16 700 865
7 -> 3 19 100 984
8 -> 19 28 600 1021
```

Results

Orders set 1	-> Numbers of express orders delivered: 125 -> Orders in the warehouse: 325 -> Average time to deliver: 232 seconds -> Company's efficiency: 27.8%
Orders set 2	-> Numbers of express orders delivered: 50 -> Orders in the warehouse: 0 -> Average time to deliver: 583 seconds -> Company's efficiency: 100%
Orders set 3	-> Numbers of express orders delivered: 9 -> Orders in the warehouse: 0 -> Average time to deliver: 736 seconds -> Company's efficiency: 100%

Extra Functionalities

Aside from the main scenarios, we decided to implement some of the extra functionalities. One of them was giving the chance of delivering older orders in day 2 (only implemented in scenario 1).

```
        /********************** Scenario 1 **********************/

DAY 1
-> Vans used: 10
-> Orders delivered: 124
-> Orders in the warehouse: 326
-> Company's efficiency: 27.6%
-> Standard Deviation: 1.0198

DAY 2
-> Vans used: 10
-> Orders delivered: 176
-> Orders in the warehouse: 150
-> Company's efficiency: 66.7%
-> Standard Deviation: 2.28035
```

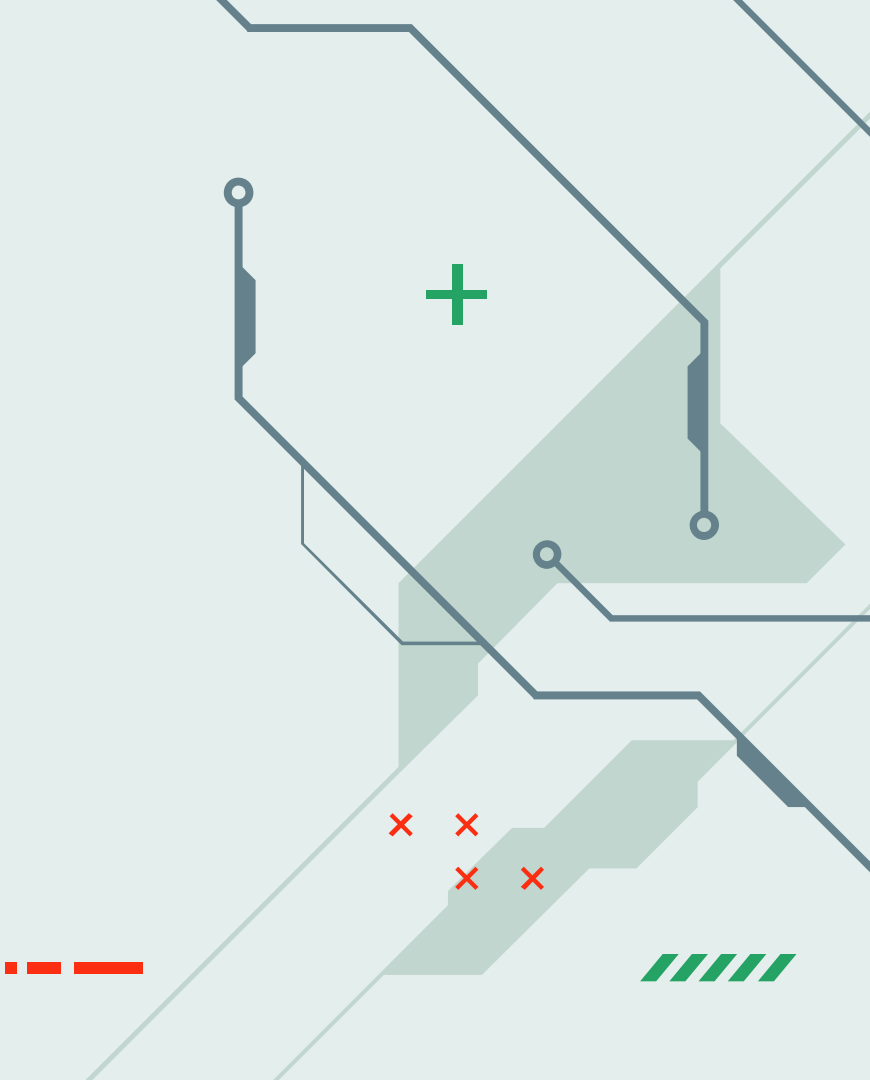
Algorithmic Solution to be highlighted

- Using the **Greedy** algorithm, we can adapt to each scenario to obtain the greatest efficiency in its different cases and objectives.
- We also highlight this algorithm for the very direct way of implementing it and allowing, through the ordering of the vans or orders, to achieve practical results.

Group self-assessment

We use a scale from 0 to 5 to make a self-assessment of the group, where 5 is the one that worked best in the group

- Edgar Lourenço - 4 / 5
- Carlos Verissimo - 5 / 5
- Nuno Jesus - 5 / 5



Conclusions

Given the required work, we were able to work through all scenarios using the requested data and achieving efficient results. Using the practical and theoretical work we were able to implement the algorithms for each of the scenarios and we were still able to implement extra features. We feel that we fulfilled everything that was proposed and that we did a communicative work and objective among us.