Team 15 – Delivery Hero's Computational Challenge

Computational Optimization at Work 2024

Laurie Boveroux Mikhail Faber Marian Stengl Su ZhaoGang

September 26, 2024

1 Challenges and Solutions

Several challenges have been encountered during the competition.

- The formulation of the MIP model in the document, which we could resolve after some discussions..
- Change of the constraints one day before submission. This forced us to change the heuristics implementation, which also came with some effort.

We have overcome that by abandoning the MIP approach (also for other reasons). Also our heuristics uses more flexibility incorporating new constraints.

2 Final Approach

The methodology implemented in this approach is based on a greedy heuristic algorithm aimed at efficiently assigning couriers to deliveries while minimizing the total time of deliveries. The main objective is to find a solution that satisfies the problem's constraints – such as delivery time windows, courier capacity, and route duration—within a set time limit. Our greedy heuristic works by iteratively assigning deliveries to couriers based on a cost matrix. The algorithm prioritizes the lowest-cost assignments and updates the status of both couriers and deliveries after each assignment. Below is a breakdown of the key steps:

2.1 Main Loop: Courier-Delivery Assignment

This loop runs until either all deliveries are assigned or the time limit is reached. It includes the following steps:

- **Pools Creation:** Two pools are created at each iteration:
  - **Courier Pool:** Includes couriers available to take on new deliveries based on their current availability. At time 0, all the couriers are available.
  - **Delivery Pool:** Includes deliveries that are ready to be assigned based on their start time.
- **Cost Matrix Calculation:** For each courier in the pool, the algorithm calculates the cost of assigning each delivery in the pool. The cost reflects:
  - The feasibility of the courier handling the delivery (based on capacity). The cost is very high when the courier cannot handle the delivery.
  - The time required to deliver it, ensuring that the total duration does not exceed the time limit (180 minutes). The cost is very high when it takes too much time to the courier and it is equal to duration when the courier can handle the delivery.
- **Greedy Assignment:** The algorithm assigns the delivery with the lowest cost to the corresponding courier, marking that delivery as completed and removing both the courier and delivery from further consideration for new assignments.

2.2 Backtracking and Stagnation Detection

Sometimes, we arrive at a point when the riders that can handle the current delivers have already did to much deliveries (i.e. 4 deliveries). To avoid getting stuck in a unfeasible situation, the algorithm monitors its progress by comparing the current state of unassigned deliveries and couriers with the previous state:

- **Backtracking:** If no progress is detected (i.e., the same number of unassigned deliveries and couriers), the algorithm restarts. In the initial pool of couriers, we do not place all the riders. We keep some of them on standby and add them later in the pool, when we are stuck. The number of riders is set with a percentage of the number of couriers. The percentage starts at zero and increases every time we start again the algorithm because we are stuck.
- **Stagnation:** When we increase the time step and the pool of riders does not change, we increase the time until a new courier is available, i.e. he finished its last delivery.

We have to note that the backtracking and the stagnation detection were useful only for the hard instances and not for the instances of the final sets.

3 Alternative Attempts

3.1 MIP

Our first attempt was the MIP solver. The corresponding implementation would have been based on the formulation given in the appendix of the Challenge Description.
Idea: Implement the MIP solver as described above and see how far we get. Then, use the built-in heuristics algorithms provided by SCIP (or FICO Xpress) to improve the solution or decrease the computation time.
Problem: There have been several issues. On the one hand, we have been working with a faulty MIP solver for quite some time. We think that we have fixed that at the time. However, we have used the Greedy algorithm to solve the problem.

3.2 Divide and Conquer

Idea: Since every courier is assigned a depot as starting point, an optimal solution will likely tend to contain many couriers picking up the delivery at a close pickup. Hence, the idea is to and separate the data set into several subsets of couriers and deliveries that are near their corresponding depots (*divide*). Then, on all of these subinstances one could apply now a solver. At the end (*conquer*) one simply combines these solutions using an n-opt strategy (as presented in the talk by Thorsten Koch on Saturday) by exchanging the deliveries between the couriers and keeping the changes if the objective improved and the solution is feasible.
Problem: One of the issues is that it depends on an already existing solver for the problem. Another one is that we ran out of time.

3.3 Assignment Problem

We solve the assignment problem using the *Hungarian algorithm.* This method solves the assignment problem by optimally matching couriers to deliveries based on their cost. This is computed when there are more couriers than deliveries. The solution from the heuristic and the assignment algorithm is compared, and the one with the lowest objective value is chosen as the final solution. In comparison, the greedy algorithm has performed better than the Hungarian method.

3.4 Simulated Annealing

The algorithm was presented by Thorsten Koch on Saturday morning. The objective was to use this local search heuristic after the greedy approach but it does not find any better solution. Some more researches must be done to define a good neighborhood.

3.5 Conclusion

The greedy heuristic provides a time-efficient and flexible approach to solving the courier assignment and delivery routing problem. Its ability to adapt by incorporating backtracking and stagnation detection helps avoid infeasible results, making it well-suited for large instances where exact algorithms may be too computationally expensive.