FAKULTÄT FÜR
WIRTSCHAFTSWISSENSCHAFT

Scheduling last-mile deliveries with truck-based autonomous robots

Author: Mr. Hamzauddin Siddiqui ([redacted])
Supervisor:[redacted]

Introduction

- Truck-based Robot Delivery is a delivery system where a truck acts as a mobile hub, carrying and launching autonomous delivery robots to serve customers in congested or restricted urban areas.
- The truck moves between designated drop-off points, launching robots that travel at walking speed to make the actual deliveries to customers, each of whom has their own deadline.
- When the truck runs out of robots, it can visit robot depots to restock.
- The key challenge is determining both the optimal truck route between drop-off points and which customers should be served from each point, with the goal of minimizing late deliveries while ensuring all customers are served exactly once.
- Think of it as a specialized version of the Traveling Salesman Problem, but with the added complexity of robot deployment and capacity constraints.


storyful

Why Autonomous Robots

- Solves the "last mile" problem in congested urban areas while avoiding the need for a costly network of local hubs
- Can operate in restricted areas where trucks can't go, while being safer than drones
- Combines the efficiency of truck transport with the flexibility of small autonomous delivery units
- Being electric based, is simply a more environmentally friendly than a traditional truck based delivery
- By-passes the unreliability of labor availability, making delivery time predictions more reliable

Why Autonomous Robots

The alternatives:

- Electric vehicles for urban delivery
- Drone-based delivery
- Traditional hub-and-spoke with local depots

The issues:

- Limited by same congestion/access issues as regular vehicles, but greener
- Safety concerns in urban areas & limited payload capacity
- Requires costly infrastructure network

While the tech is impressive at the moment..

It’s not great either..

Problem Description & Model Formulation

Objective:

- The main goal of the TBRD problem is to create such a route for the truck, as well as a robot launching schedule that it optimises for minimising the weighted number of late deliveries.

$$\text{Minimize } F(L,S,T,X,Z) = \sum_{k \in C} z_k \cdot w_k$$

Constraints

Robot constraints:

- initial robot load to not exceed the starting number of robots (δ) minus those deployed at the start position
- robot loading capacity when moving from a robot depot to a drop-off point, considering the maximum capacity K
- robot quantity consistency between consecutive drop-off points

$$l_\gamma \leq \delta - \sum_{(k \in C)} x_{\gamma,k}$$

$$l_j \leq K + M \cdot (1 - s_{i,j}) - \sum_{(k \in C)} x_{j,k} \qquad \forall\ i \in R;\ j \in D$$

$$l_j \leq l_i + M \cdot (1 - s_{i,j}) - \sum_{(k \in C)} x_{j,k} \qquad \forall\ i \in D \cup \{\gamma\};\ j \in D \setminus \{i\}$$

Constraints

Time Constraints:

- Starting time is set to 0 and is tracked
- Arrival times are bounded from below
- Late delivery detection

$$t_\gamma = 0$$

$$t_j \geq t_i - M \cdot (1 - s_{i,j}) + \vartheta^t_{i,j} \qquad \forall\, i \in D \cup R \cup \{\gamma\};\ j \in D \cup R \setminus \{i\}$$

Sub-tour Elimination Constraints:

- Ensuring the truck starts its route from at most one location
- Maintaining route continuity - number of entries equals number of exits for each location

$$M \cdot z_k \geq t_j - M \cdot (1 - x_{j,k}) + \vartheta^r_{j,k} - d_k \qquad \forall\, j \in D \cup R \cup \{\gamma\};\ k \in C$$

$$\sum_{j \in D \cup R \cup \{\gamma\hat{}e\}} s_{\gamma,j} \leq 1$$

Constraints

Location Constraints:

- Ensuring robots are only launched from locations the truck visits
- Forcing drop-off points to be used if visited
- Controlling robot depot visits - must either launch robots or connect to drop-off points

$$\sum_{k \in C} x_{j,k} \leq M \cdot \sum_{i \in D \cup R \cup \{\gamma\} \setminus \{j\}} s_{i,j} \qquad \forall j \in D \cup R$$

$$\sum_{k \in C} x_{j,k} \geq \sum_{i \in D \cup R \cup \{\gamma\} \setminus \{j\}} s_{i,j} \qquad \forall j \in D$$

$$\sum_{k \in C} x_{j,k} + \sum_{i \in D} s_{i,j} \geq \sum_{i \in D \cup R \cup \{\gamma\} \setminus \{j\}} s_{j,i} \qquad \forall j$$

Constraints

Variable domains:

- Binary variables for route segments (s), assignments (x), and late deliveries (z)

$$s_{i,j} \in \{0, 1\} \qquad \forall\, i \in D \cup R \cup \{\gamma\};\ j \in D \cup R \cup \{\gamma^e\} \setminus \{i\}$$

$$x_{i,k},\ z_k \in \{0, 1\} \qquad \forall\, i \in D \cup R \cup \{\gamma\};\ k \in C$$

- Non-negative continuous variables for robot loads (l)

$$l_j \geq 0 \qquad \forall\, j \in D \cup \{\gamma\}$$

Solution Approach

2 datasets (small & large) are created with the following parameters:

Small

Side length of the main square	2 km
Number of robot depots	4
Number of drop-off points	6
Number of customers	6
Deadline factor interval [tight, wide]	([2,4];[3,5])
Truck's robot capacity	2

Large

Side length of the main square	5 km
Number of robot depots	16
Number of drop-off points	30
Number of customers	40
Deadline factor interval [tight, wide]	([4,12];[4,15])
Truck's robot capacity	8

Solution Approach

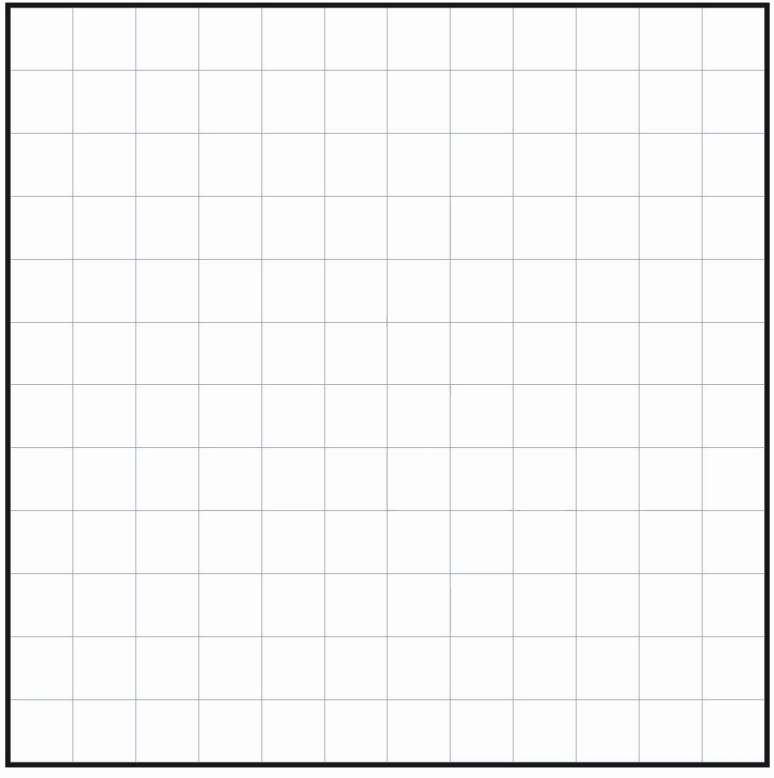
A few things remain the same between the 2 datasets:

- There is only 1 truck in both cases
- Customer weights can be either Homogeneous [1:1] or Heterogenous [1:3]
- Truck speed is fixed at 30 KMPH
- Robot speed is fixed at 5 KMPH
- A general processing time limit of 300 seconds (5 minutes) per instance is applied in both cases*

Solution Approach

- We first generate a square of side w km and divide it into a grid of squares of side length 1/6 kilometers
- We then allocate the given number of robot depots in an equidistant manner within the grid
- Then, we randomly scatter drop-off points and customers, making sure that each point is assigned only once
- A randomly chosen depot or drop-off point is set as the initial truck position
- Euclidean metric is used to compute distances between points and based on these distances and given truck and robot speeds, travel times are calculated
- Customer deadlines are drawn randomly from an interval
- And each customer is assigned a weight randomly from a different interval

Step 1: Creating Grid (2km × 2km, divided into 1/6 km squares)



Robot Depot Drop-off Point Customer Truck

Total size: 2km × 2km

Each cell: 1/6 km × 1/6 km

Gurobi Solver

Mixed-Integer Linear Programming (MILP) model

1. Parameter Setup: Gurobi is employed to solve the problem once the parameters are set up
2. Redundancy Removal: It first looks at the whole problem and removes any obviously impossible solution
3. Initial Solution Attempt: Then, tries to solve a simpler version of the problem, which gives a rough idea of what good solutions might look like
4. Improvement: It then applies a bunch of improvement techniques (in our case it used MIR cuts)
5. Branch & Bound: Splitting the problem into smaller pieces (branching) and calculating limits on how good solutions in each piece could be (bounding).

Gurobi Method

Gurobi Solver

The issue:

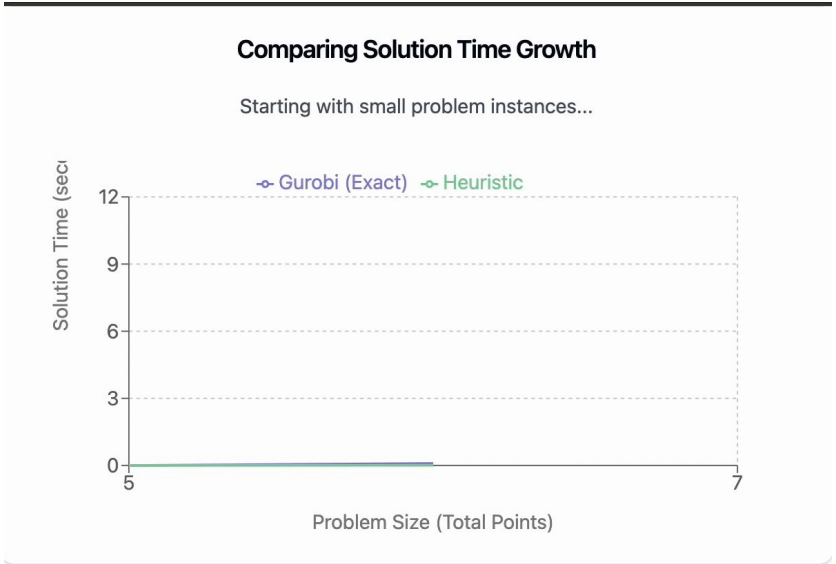
- In our problem, each location needs a yes/no decision: "Does the truck visit this location?"
- With n locations, this creates 2^n possible combinations
- This exponential growth is what makes it non-polynomial
- This is an oversimplification of the issue with using Gurobi, obviously the real issue is much more complex, given that we are juggling more than 8 parameters

Computational Complexity

- It can be proven that this scheduling problem is what we call 'NP-hard', which means it gets exponentially harder to solve as you add more locations.
- This is proven by showing that if you could solve this problem efficiently, you could also solve the famous Traveling Salesman Problem efficiently - something mathematicians have been trying to do for decades.
- This is why we need clever shortcuts (heuristics) rather than trying to find the perfect solution every time.

Computational Complexity

- It can be proven that this scheduling problem is what we call 'NP-hard', which means it gets exponentially harder to solve as you add more locations.
- This is proven by showing that if you could solve this problem efficiently, you could also solve the famous Traveling Salesman Problem efficiently - something mathematicians have been trying to do for decades.
- This is why we need clever shortcuts (heuristics) rather than trying to find the perfect solution every time.

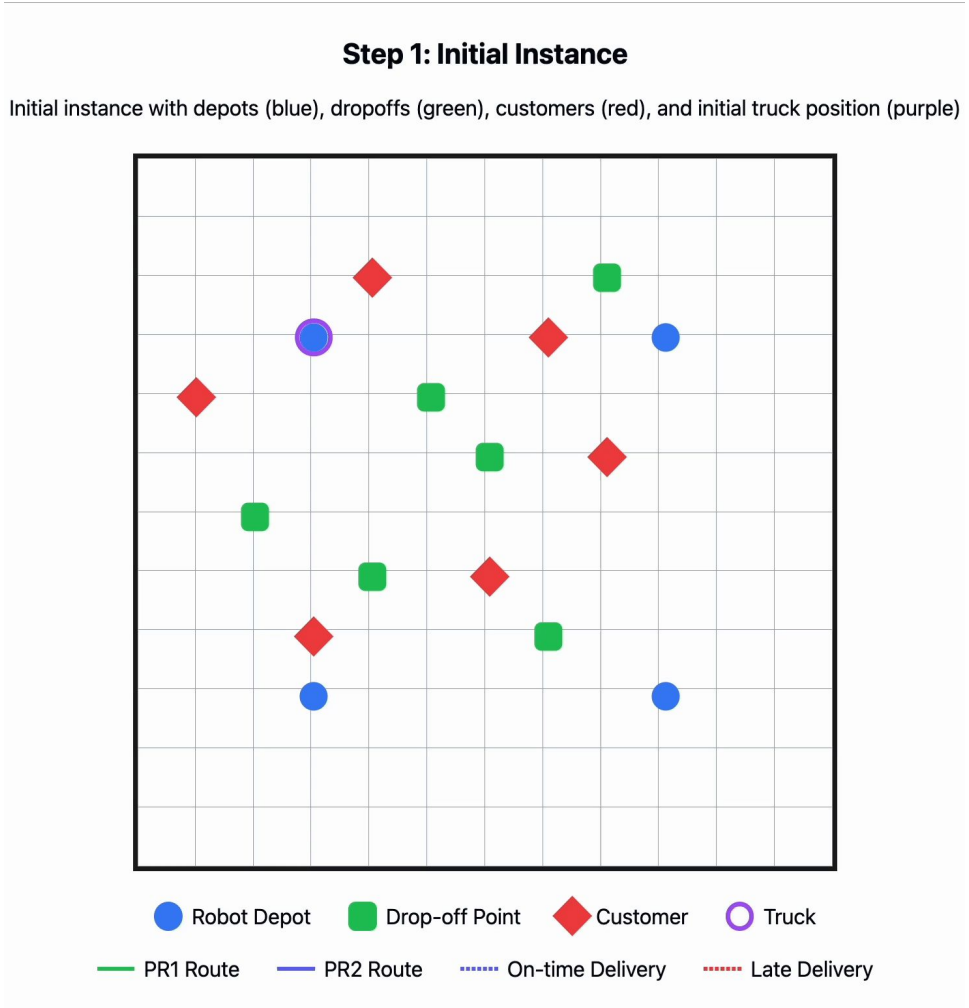


Solution Approach

Metaheuristics Optimization Model:

- Step 1: Initial Solution Generation: Two different routes are created using two different priority rules:
  - PR1: Always move to the position where the most customers can be served on time
  - PR2: Always move to the position where customers have the most urgent deadline
- Step 2: Solution Pool Creation: Take each initial solution and try to improve it
  - Look for places where there are two drop-off points next to each other
  - Try inserting depots between these drop-off points to create new possible solutions
  - Keep any new solutions that are better than the original
- Step 3: Local Search: Take the best solution found so far and try to improve it using four different types of changes:
  - N1: Remove a random position from the route
  - N2: Insert a random depot or drop-off point
  - N3: Swap two positions in the route
  - N4: Insert the closest depot before a drop-off point
  - If changes make solution better, keep it and iterate until no more improvements can be found
- Step 4: Route Evaluation: For each route generated, figure out the best way to assign robots to customers
  - Calculate which customers will be served late
  - Calculate the total penalty based on late deliveries

Heuristics Method



Results

Time Limit per Instance: 300 sec

Performance of Small Dataset					
		Gurobi		Heuristic	
Deadlines	Weights	Opt	Sec	Opt	Sec
Tight	Homo	7	0.11	25	0.45
	Hetero	7	0.09	25	0.36
Wide	Homo	18	0.11	24	0.13
	Hetero	18	0.12	24	0.14
Total/Average		50	0.1075	98	0.27

Performance of Large Dataset					
		Gurobi		Heuristic	
Deadlines	Weights	Opt	Sec	Opt	Sec
Tight	Homo	12	4.7	6	1.21
	Hetero	13	5.44	8	1.26
Wide	Homo	15	8.06	9	0.3
	Hetero	13	7.82	11	0.33
Total/Average		53	6.505	34	0.775

All opt columns are out of 25

Interpretation & Takeaways

- For the small dataset:
  - Gurobi found optimal solutions in about 50% of test cases, taking around 0.11 seconds on average
  - Our heuristic procedure found optimal solutions in about 98% of test cases, taking 0.27 seconds on average
- For the large dataset:
  - Gurobi found optimal solutions in about 53% of test cases, taking 6.5 seconds on average
  - Our heuristic found optimal solutions in about 34% of test cases, but only took 0.775 seconds
- It seems like our metaheuristic procedure is tuned to serve larger datasets better than smaller ones, hence reinforcing real life utility.
- An important thing to note here is that when 25 iterations are run for each category and lets say only 6 optimum solutions are found, it means that in 19 iterations an optimum solution couldn’t be found because of the following reasons:
  - Combination of parameters make fulfilling all constraints not possible
  - Time limit ran out

Comparison to Boysen et al. (2018)

While the overall direction of results remain the same, 2 glaring differences can be observed:

- Gurobi is able to find an optimum solution in a greater proportion of iterations in the large dataset as opposed to the small dataset.
- Average runtimes were significantly smaller for all different combinations (16) of tests run (remember that each type of combination is run 25 times)
- I would speculate that the difference in timing might be due to:
  - Hardware differences (paper has 4 × 4.0 gigahertz, my laptop has 8 x 2.8 gigahertz [windows 7 vs windows 11])
  - Gurobi version differences (paper used 7.0.2, I used 12.0)
  - Python vs C# differences

Possible improvement directions

- A lot of the assumptions that went into this model were for the purposes of making it as simple as possible.
- The assumption regarding unlimited robot availability can be disregarded and dynamic replenishment rules could be introduced to test the impact of varying robot bottleneck on delivery efficiency.
- Travel-time uncertainties could be introduced to mimic real-life delays due to traffic lights, loading & unloading times as well as customer unavailability wait times.
- Variable customer deadlines or time windows for delivery could be incorporated to make the model more applicable to the real world.

OTTO VON GUERICKE
UNIVERSITÄT
MAGDEBURG

FAKULTÄT FÜR
WIRTSCHAFTSWISSENSCHAFT

Thank you for your attention!

Extra!

- NP hardness proof
- Hitchcock transportation problem
- Sensitivity Analysis
- Policy Analysis

Proof by reduction of NP Hardness

Take any TSP problem and transform it into a TBRD problem:

1. The starting city in TSP becomes the truck's starting location
2. For each remaining city in TSP:
    - Create a drop-off point at that city's location
    - Create a customer location right next to that drop-off point (zero distance between them)
3. Give the truck n-1 robots initially (where n is number of cities)
4. Set the robot speed to be extremely slow (much slower than truck)
5. Give all customers the same deadline T (where T is the TSP tour length limit)

Why this works:

- Because robots are so slow, each customer must be served from the drop-off point right next to them. Serving from any other point would make delivery too late.
- The truck needs to visit each drop-off point exactly once (since each customer needs service)
- The truck route ends up being identical to a TSP tour through the cities
- A solution exists for TBRD only if a valid TSP tour exists

Therefore:

- If you could solve TBRD efficiently, you'd also solve TSP efficiently
- Since we know TSP is NP-hard, TBRD must also be NP-hard

Hitchcock transportation problem

The Hitchcock Transportation Problem transformation is like creating a matching game between "suppliers" (places where we can launch robots from) and "customers" (people waiting for deliveries).

Suppliers are:

- All robot depots (each can serve any number of customers)
- Each sequence of drop-off points (limited by how many robots the truck can carry)

Customers are:

- All delivery customers (each needs exactly one delivery)
- One dummy customer (to balance the math)

The "cost" of each possible match is:

- Zero if we can deliver on time from that location
- The customer's priority weight if delivery would be late
- Zero to the dummy customer (like a wastepaper basket for unused capacity)

This transformation is useful because:

1. It's a well-known type of problem that can be solved quickly
2. It tells us the best way to assign customers to different launch points
3. We can solve it repeatedly as we try different truck routes

Sensitivity Analysis

Depot Network Density:

- Since the truck has to visit the robot depot from time to time to refill itself, logic dictates that more robot depots would make detours shorter
- A denser network of robot depots increases delivery performance, however, marginal returns of additional depots diminish the more depots are added.
- In our case, on a 4 KM^2 grid, the benefit of adding anything beyond 5 depots become negligible

Drop-Off Network Density

- In the real world, stopping the truck, loading the robots and launching them safely may not be possible just anywhere. Hence, we explore the impact specifically designated parking spaces may have on delivery performance.
- Similar to previous results, increase in the number of drop-off points improves delivery performance with diminishing marginal returns. In our tests, more than 1.5 drop-off points per square kilometer could not further increase delivery performance.

Sensitivity Analysis

Robot Speed:

- In our parameters, robot speed was fixed at 5 KMPH. However, there may be some flexibility in that, albeit within a small range.
- A small increase from 5 to 6 KMPH yields almost a 75% improvement in delivery performance. However, subsequent increases in speed result in very small performance increases.

Truck Capacity:

- In the real world, the capacity of the truck has to be partitioned between the robots and the parcels
- This was interesting since it showed that increasing truck capacity mattered little, if we already have a dense network of depots. For a less dense network of depots, the importance of truck capacity increases significantly. However, diminishing returns can be observed beyond having 5 robots per truck.

Benchmarking Alternative Policies

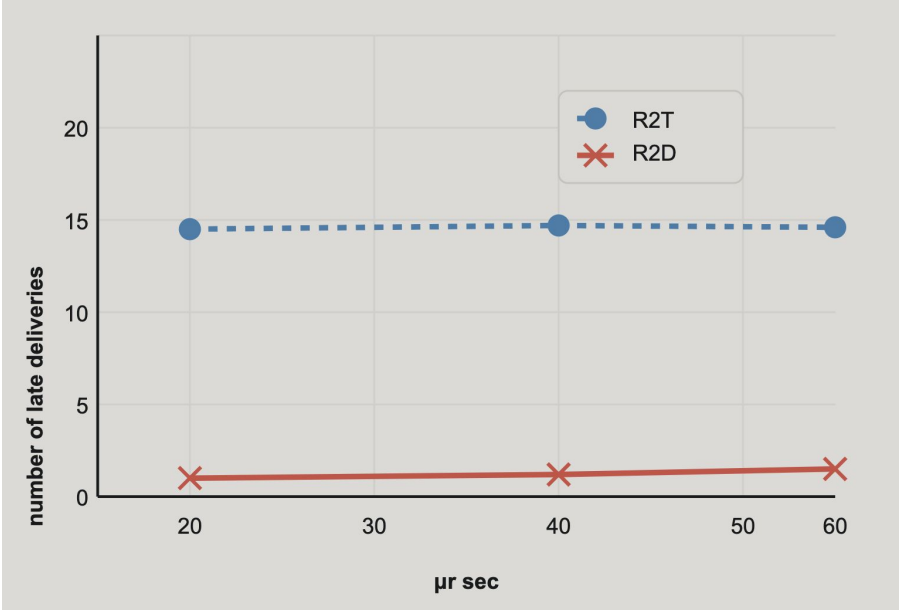
1) Return to Depot (R2D) Policy: This is the policy we have explored thus far, where the robots are deployed and the truck moves on to the next drop off point, and the robot returns to the nearest depot after completing the delivery.

2) Return to Truck (R2T) Policy: From a business perspective, a reasonable alternative policy might be to have the robots return to the truck, which should save on the investment cost required for a decentralized network.

3) TBRD w/o Autonomous Robots: This is the existing delivery model used by most delivery companies. For simplicity's sake, the existence of drop-off boxes are ignored.

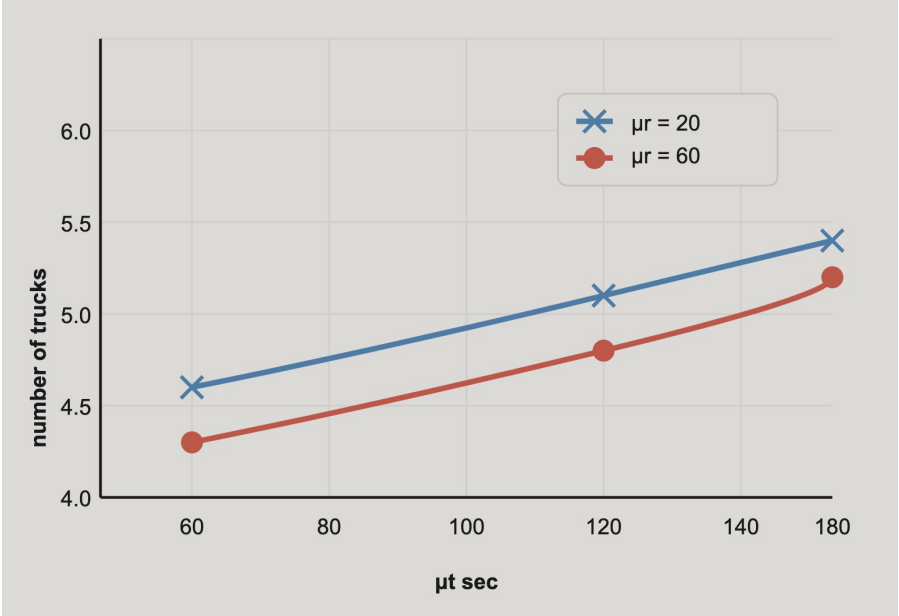
R2D vs R2T

- The R2D policy significantly outperforms the R2T policy, consistently at different test levels.
- The y-axis represents increasing delivery times, i.e. more time for loading robots, unloading robots, deploying them etc.
- As such, it is not surprising that the difference between them does not change much over increasing delivery times.
- This is because increasing delivery times affect both policies.
- Likely reasons for the poor performance of the R2T policy is that truck stops become much longer, robot delays due to longer service times or unsuccessful waiting at customer locations have to be considered



R2D vs TBRD w/o Autonomous Robots

- R2D outperforms the traditional delivery methods as well.
- Here we determine the minimum truck fleet that is required to also reach the service level of R2D.
- This means we compare how many more trucks would be needed to service the same number of customers supplied late.
- We find that to reach the same service level as R2D, at least 3 additional trucks are required if no autonomous robots are deployed.
- Even more trucks are required if the loading time of the robots is short and the delivery time of the delivery man is large.
- This is relatively easy to explain as additional driving time to each single customer is required and each single truck stop takes comparatively long time.
- The delivery man has to find a suited parking space, walk the remaining distance, wait for the customer, and then return to the vehicle.



Caveats

- While the above policy comparison may indicate a clear victory for R2D, it must be taken with a grain of salt.
- The assumption of always having the desired number of robots available at all depots play a not so insignificant part in skewing the results in favor of the R2D policy.
- This is likely not going to be the case for businesses, especially not initially when this sort of delivery system starts getting adopted.
- However, it does very clearly demonstrate the utility of having a sufficiently large robot fleet for businesses, perhaps making the investment cost worth it.