Delivery Sim: *Team 171: Ben Hendel, Rohan Nanda, Aleksandr Shirkhanyan, Haohui Ding*

Motivation

The restaurant meal delivery problem is a major logistical challenge that has gained increased attention in recent years due to the growing demand for online food delivery services. With the rise of food delivery platforms and changing consumer preferences, restaurants are facing the need to efficiently manage the delivery of meals to customers while maintaining high-quality service standards. The restaurant meal delivery problem involves coordinating the pickup of meals from restaurants and their subsequent delivery to customers' locations, considering factors such as delivery time windows, vehicle capacities, and customer preferences.

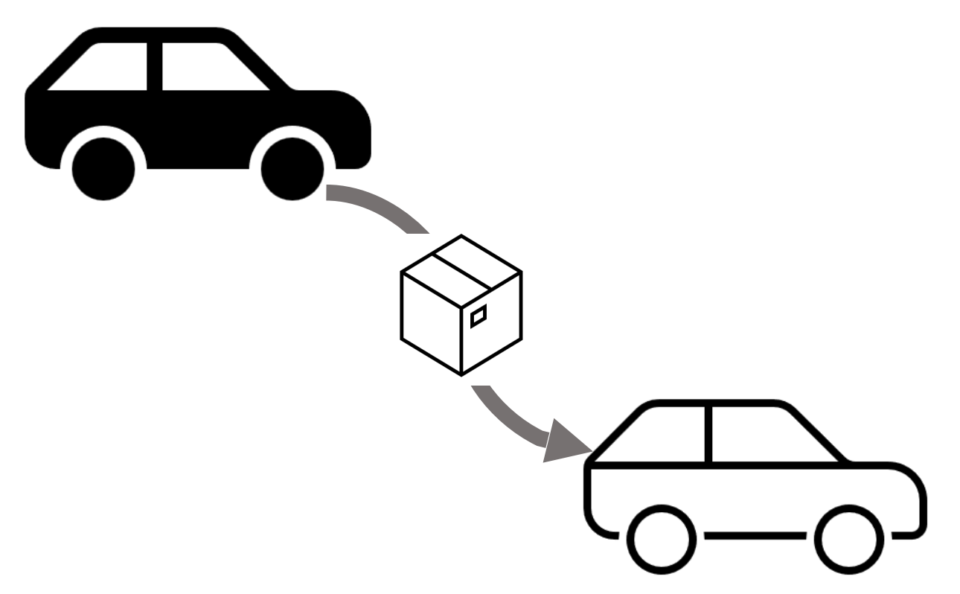
Efficiently managing restaurant meal deliveries is a complex task that requires solving multiple optimization problems simultaneously, such as vehicle routing, scheduling, and assignment. Restaurants must consider factors such as delivery distances, time windows, traffic conditions, and customer demands, while also minimizing delivery costs and maximizing customer satisfaction. Additionally, restaurants may have multiple delivery drivers or third-party delivery partners, each with their own capacity limitations and constraints.

Different studies have tried to address this problem incorporating various features such as stochastic order placement, driver availability, delay in the allocation of requests to a vehicle to allow bundling, variations in customer demand based on time and location, the waiting time encountered by couriers at restaurants, etc. However, there is one particular feature that, to the best of our knowledge, has never been studied; the possibility to hand off food to another driver during delivery. This is the main focus of our project. We incorporate the handoff system in our model and check if it improves the average food delivery time.

Problem Statement

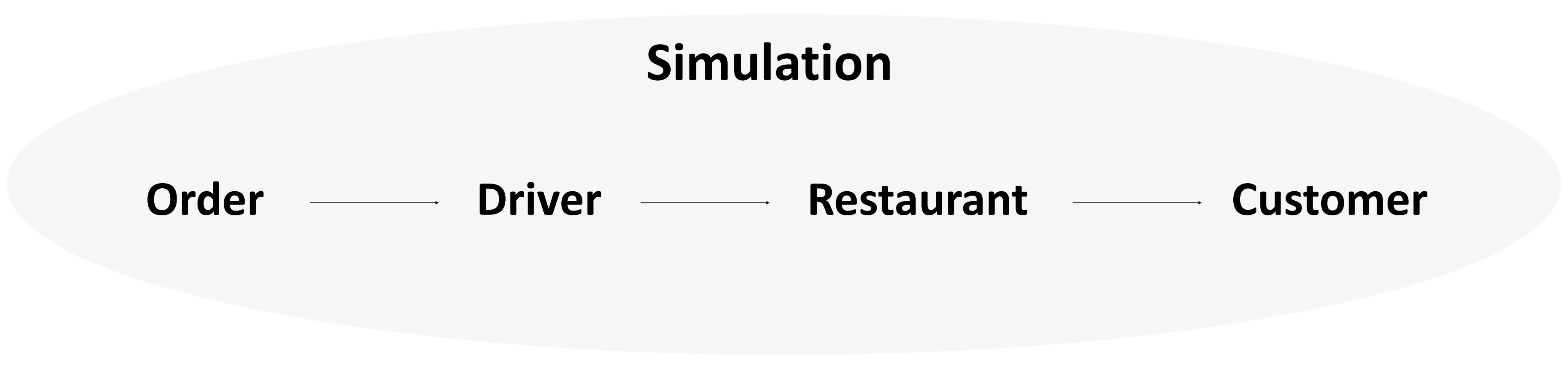
Does the handoff system decrease the average delivery time?

To answer this question, we start with a baseline simulation of the meal delivery problem, then we incorporate the handoff system. Whenever two delivery drivers' cross paths, we check if they are heading to make a delivery within a close distance from each other. If so, a second destination is added to one driver, and the other driver ends his trip and is free to complete another delivery. Finally, we compare the average delivery time for the baseline and the handoff scenarios to see if there are improvements or not.



Approach

The simulation is an agent-based model and is defined by the following classes:



Simulation is the main agent that contains the city graph and global information pertaining to all actors within it. It is therefore only instantiated once. The Driver, Restaurant & Customer agents are pre-defined as per the parameters, while the Order agents are defined dynamically as the simulation progresses. Of these, the Driver agent is the most complex as it needs to have decision-making capabilities with regards to picking/handing-over/delivering orders.
In a typical step without handoffs, the following events will occur:

- Each restaurant will receive a number of orders (following a Poisson Distribution) from random customers.
- Upon receiving an order, the restaurant will emit a call (through the method 'allocate_to_driver') to assign the order to the closest available driver. If the driver does not have the capacity to take the order, he will refuse it and the next nearest driver will be called, and so on. If none of the drivers are able to take the order, it will be added to the restaurant's backlog & an attempt to allocate will be made in the next timestep (care is taken to ensure that older orders are given priority).
- After this, the driver has to decide where to move (method 'update_target'). At any time, the driver has 2 lists of orders in mind, called the 'pending_orders' (which the driver has accepted to pick but is not yet carrying) and 'carrying_orders' (which the driver has already picked). By default, the driver will always prioritize 'pending_orders' i.e. it will update its target location to the restaurant and identify the shortest path to get there. If the 'pending_orders' is empty, the driver will look at 'carrying_orders' and update its target to the customer' location. In both cases, the nearest restaurant/customer is selected. Finally, if both lists are empty, then the driver will return to its original location and await an order.
- Following this, the 'step' method is called, whereby the driver will move along the shortest path it has identified to the next target. Concretely, he will move along the current edge at the predefined speed. If the distance to the next node is higher than the speed, he will remain along the edge and will only advance as far as his speed allows. However, if the distance to the node is less, he will snap to the node and then check to see where this node lies along the path. If it is merely a junction, he will update accordingly so that in the next step he moves along the next edge. However, if the node is a restaurant or customer, he will pick up or deliver the order and update his parameters.
- The cycle then repeats.

In a step with handoffs, the same set of events will occur, plus a check (method 'check_all_handoffs') to see if there is the opportunity for any driver to handover its order to a nearby driver who is headed in a similar direction. For this, 2 threshold parameters are defined ('handover_driver_threshold' & 'handover_customer_threshold') to ensure that they are close enough as for the handover to make sense. If there is a handover opportunity, the drivers' positions will be fixed for the duration that it would take for them to travel the distance to each other & back (in order to physically handover the order).

This is a reasonable simplification in order to avoid having to explicitly update the driver targets/steps, as this would add unnecessarily complexity. Once the handoff is complete & the drivers are free to move again, it will be observed that their orders have swapped and the driver who handed over is now free to accept a call from another restaurant rather than head to the customer.

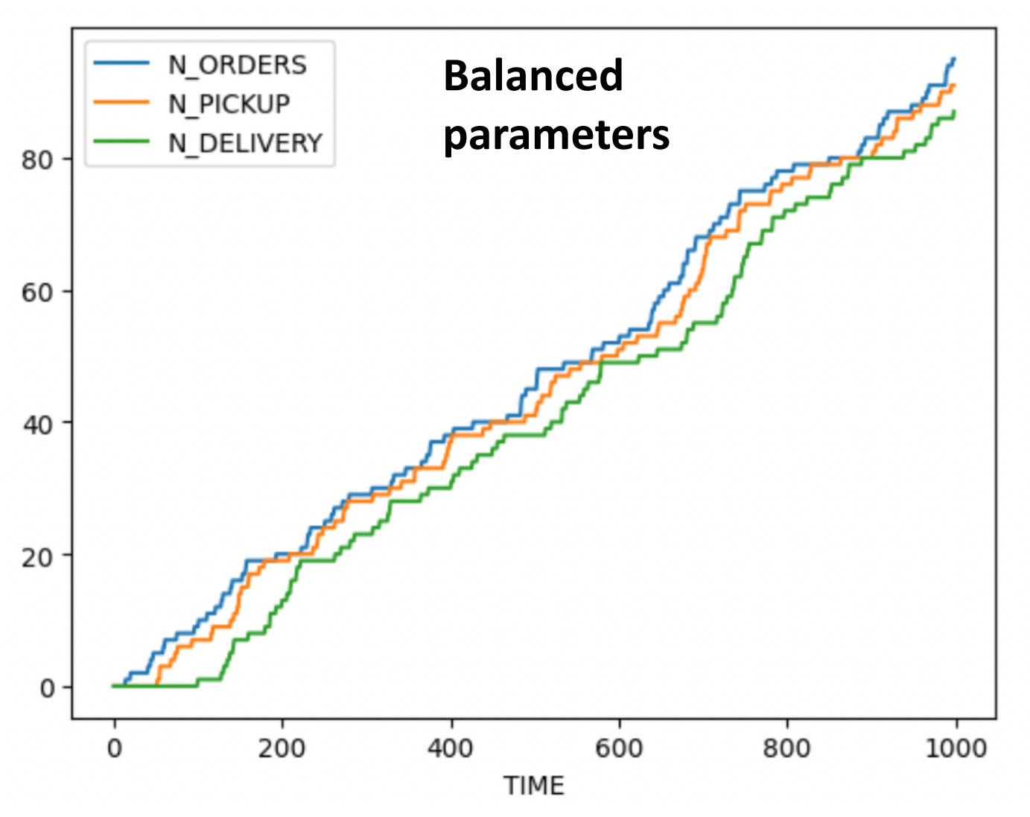
Given that d3 is not specifically designed for the movement of objects, we only visualize one simulation run with specific parameters, lest the browser crashes. We display the positions of restaurants and customers on a genuine map of Toulouse from OSM, then illustrate movement of drivers over time and the orders they carry at specific times. The user can initiate the animation by pressing the "start" button and restart it from the beginning with the "reset" button.

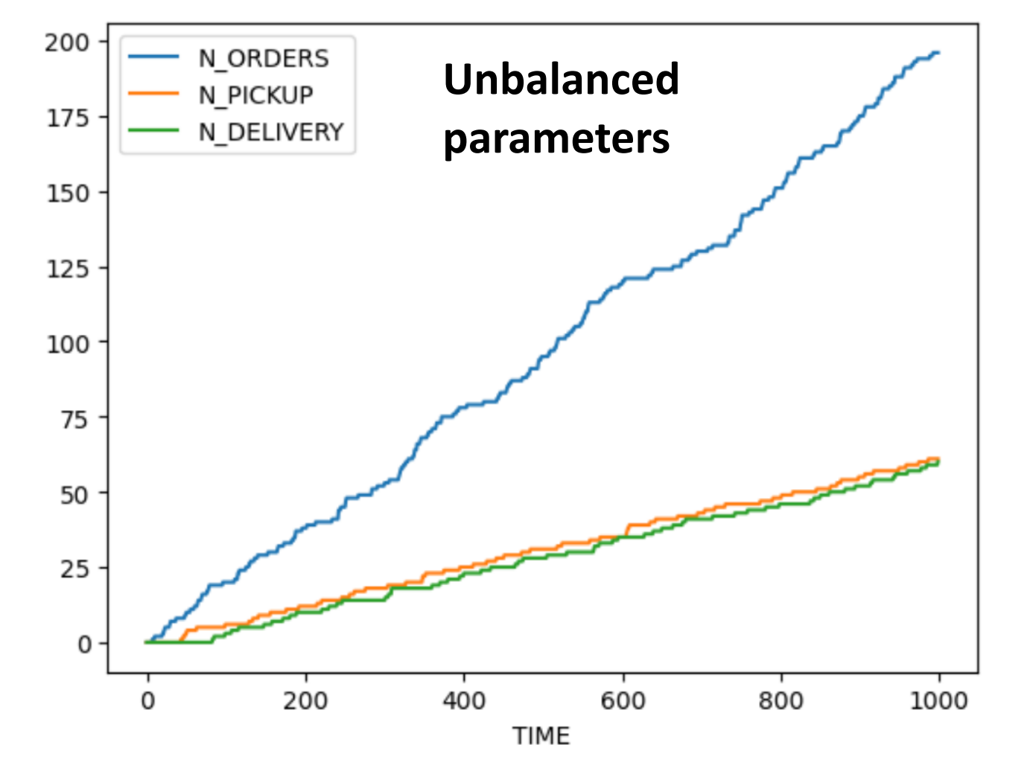
Parameter tuning

We had many parameters with too many combinations to reasonably test. Before doing our simulation runs, we needed to create a reduced parameter grid of high-medium-low values for only a subset of relevant parameters. To see if a parameter is relevant to our results, we conducted an exploratory analysis: We tested each one individually, varying it while holding everything else constant at a "base simulation value". We found the reasonable base simulation values to be:

15 restaurants, 20 drivers, 70 customers, 0.01 order rate (lambda), 0.005 speed, 0.01 miles driver threshold, 0.005 miles customer threshold

We learned that decreasing driver speed decreased average delivery time exponentially, with an elbow of around 0.005, below which deliveries became extremely sluggish. Increasing speed reduced time but not drastically. Changing the poison parameter lambda of average inter-order time yielded simulations that were either too lax (drivers sitting around with no orders) or too intense (orders grew without bound as in Figure 1). A constant lambda of 0.01 solved this problem. Likewise, varying the number of drivers, customers, and restaurants from the base value caused the orders to diverge and never be delivered. To achieve steady state, these too were held constant at 15 restaurants, 20 drivers, and 70 customers.

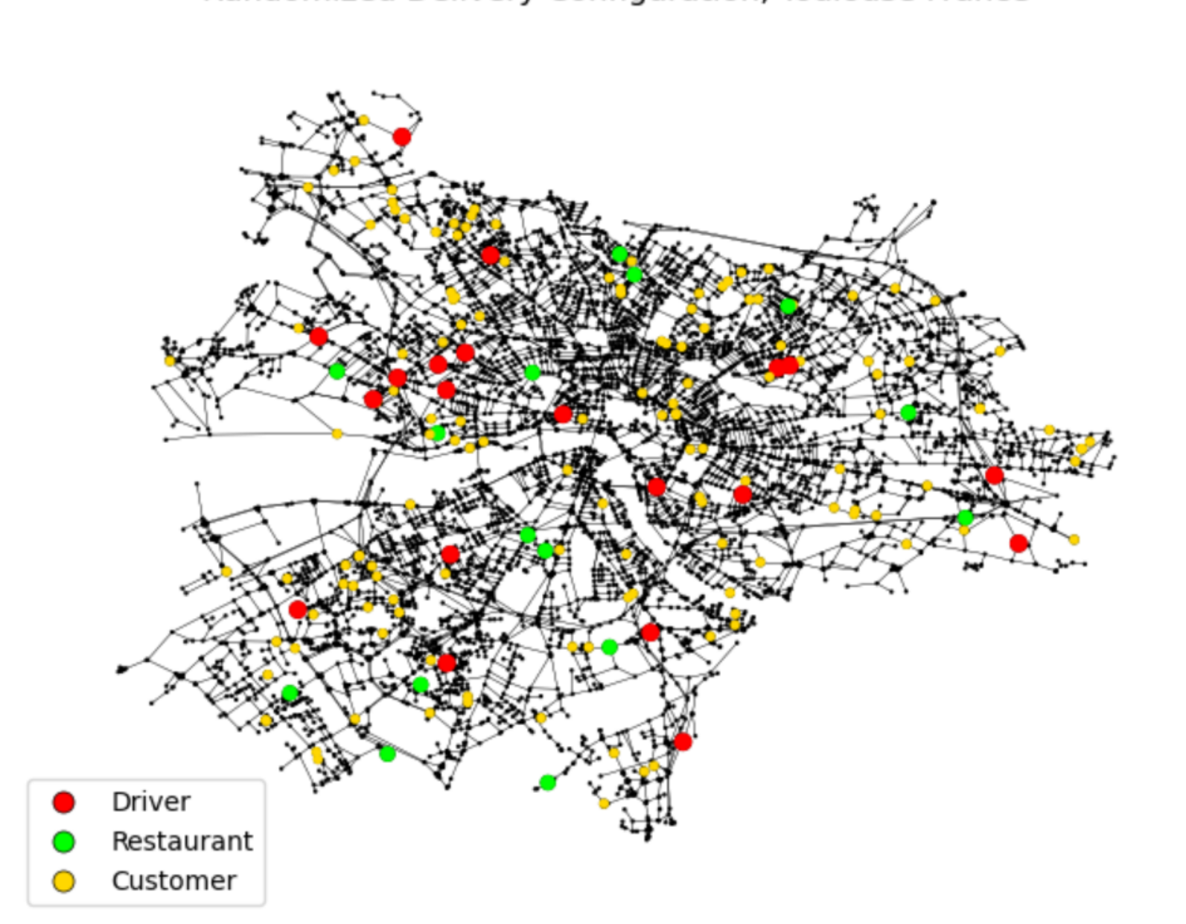


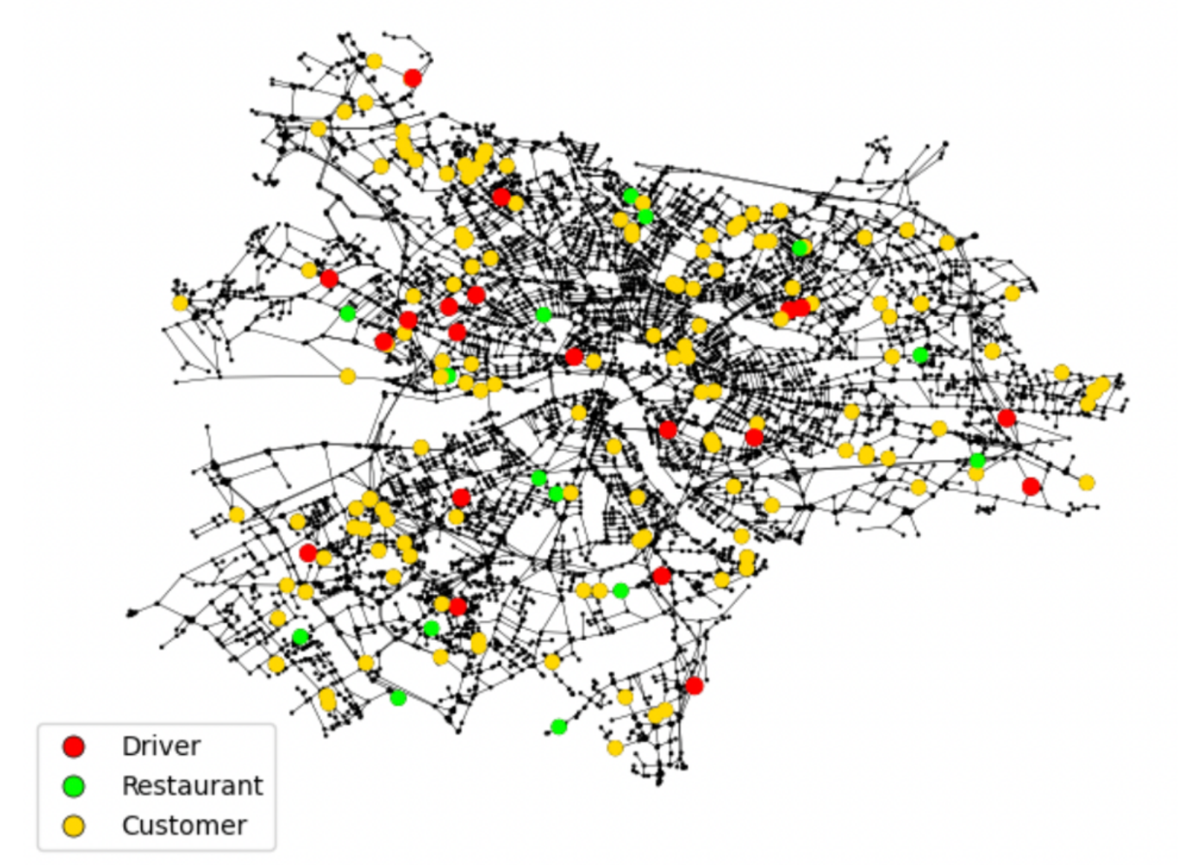


Experiments

Our simulations used some real-world data; we first downloaded the city layer of Toulouse, France. Using OpenStreetMap, we got the actual locations of the restaurants (that do delivery). The Figure on the right depicts some of the locations; yellow dots represent the locations of the customers, while green ones are the restaurants. Finally, we randomly dropped a specified number of drivers to different locations of the city (20 drivers in this case represented by red dots). These delivery drivers move at a fixed speed. Poisson random orders come in with a specified pickup business and destination home, and the closest free driver takes the shortest path to the pickup business first and then to the home. This is our baseline simulation. We compared this to handoff simulation, in which case drivers can hand the food off to another driver. The objective is to find out if the handoff simulation results in smaller average delivery time.
Upon doing many simulation runs; we realized that handoffs were simply not happening frequently enough (1 or 0 handoffs). We tried increasing the number of drivers, customers, and restaurants, but this increased the code runtime exponentially. Since we wanted to test hundreds of parameter combinations, this was a major limitation. The result is that only a few handoffs ever happened per simulation, meaning the average order delivery time wasn't different by any meaningful amount after enabling handoffs. Natural randomness was far more important than any difference due to handoffs. Simply put, the conditions for a handoff to occur (customer threshold and driver threshold) were too rare. In a large road network, the odds that two customers are initiated close to each other was very unlikely. We solved this part with a trick we will soon describe, but there is also the second condition. Even when there are two customers close to each other receiving food, drivers going to those specific locations still have to cross paths. We tried increasing both of these thresholds, but it didn't make a difference until these values were unrealistically high, resulting in delays, repeated/leapfrogging handoffs, and drivers crossing huge parts of the city to hand off deliveries to each other.
For the customer location issue, we came up with a unique solution. To artificially create more handoffs to examine, we created a specific configuration of customers that are "paired" near each other (see the Figure on the right). This makes the first condition to trigger a handoff a little more likely.

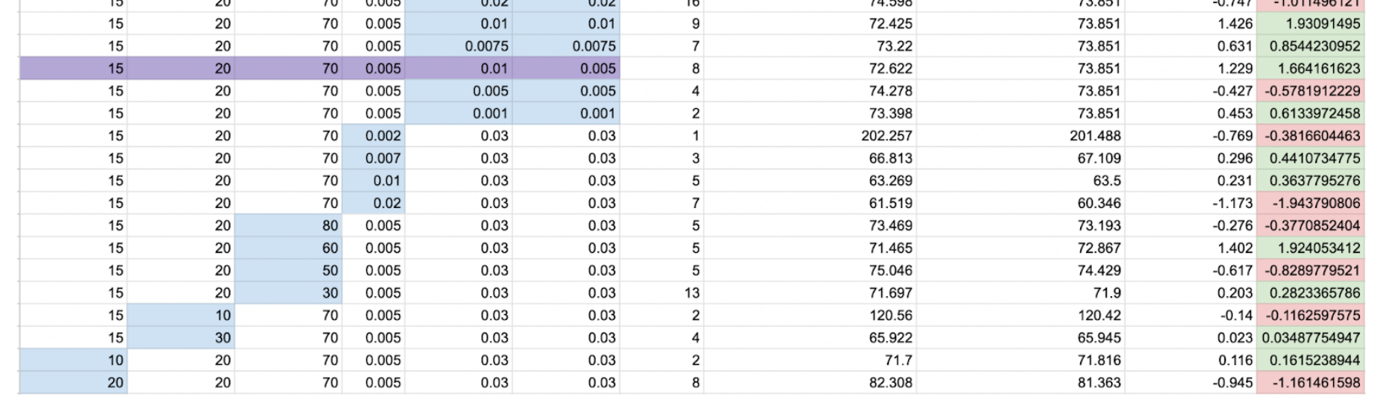
Randomized Delivery Configuration, Toulouse France





You can see in the second plot in the right that the yellow customers are near each other more often than not. With this configuration we started seeing more handoffs, but still not that many, around 5-10.

We tested all parameter combinations with this configuration, and even at the absolute best parameter combination (driver threshold=0.01, customer threshold=0.005), we got an average order delivery time reduction of only 0-2%, which was only positive within small parameter windows outside which it became negative (see the table). In the real world, these rare time saving windows would be an unrealistic best case, given that customers aren't going to start like this, and intelligent routing services will try to avoid conditions where drivers are heading to nearby locations in the first place.



n_restaurants	n_drivers	n_customers	speed	driver threshold	customer thresh	handoffs	avg order duration (handoffs on)	avg order duration (no handoffs)	average time saved	percent saved
15	20	70	0.005	0.04	0.04	42	79.77	73.851	-5.919	-8.01478653
15	20	70	0.005	0.03	0.03	31	77.102	73.851	-3.251	-4.402106945
15	20	70	0.005	0.02	0.02	16	74.598	73.851	-0.747	-1.011496121
15	20	70	0.005	0.01	0.01	9	72.425	73.851	1.426	1.93091495
15	20	70	0.005	0.0075	0.0075	7	73.22	73.851	0.631	0.8544230952
15	20	70	0.005	0.01	0.005	8	72.622	73.851	1.229	1.664161623
15	20	70	0.005	0.005	0.005	4	74.278	73.851	-0.427	-0.5781912229
15	20	70	0.005	0.001	0.001	2	73.398	73.851	0.453	0.6133972458
15	20	70	0.002	0.03	0.03	1	202.257	201.488	-0.769	-0.3816604463
15	20	70	0.007	0.03	0.03	3	66.813	67.109	0.296	0.4410734775
15	20	70	0.01	0.03	0.03	5	63.269	63.5	0.231	0.3637795276
15	20	70	0.02	0.03	0.03	7	61.519	60.346	-1.173	-1.943790806
15	20	80	0.005	0.03	0.03	5	73.469	73.193	-0.276	-0.3770852404
15	20	60	0.005	0.03	0.03	5	71.465	72.867	1.402	1.924053412
15	20	50	0.005	0.03	0.03	5	75.046	74.429	-0.617	-0.8289779521
15	20	30	0.005	0.03	0.03	13	71.697	71.9	0.203	0.2823365786
15	10	70	0.005	0.03	0.03	2	120.56	120.42	-0.14	-0.1162597575
15	30	70	0.005	0.03	0.03	4	65.922	65.945	0.023	0.03487754947
10	20	70	0.005	0.03	0.03	2	71.7	71.816	0.116	0.1615238944
20	20	70	0.005	0.03	0.03	8	82.308	81.363	-0.945	-1.161461598

Conclusion

In conclusion, we did not find evidence that a handoff-based system is more efficient in any way; Perfect handoff situations are simply too rare. The only way we achieved a noticeable improvement in delivery times was when working with artificially ideal customer locations used with the best possible input parameters, which could have been a coincidental improvement resulting from the multiple comparisons problem. Additionally, in real life, you cannot control a customer's location nor the delivery drivers' speed. The only conceivable situation where theoretically handoffs could happen more frequently would be with a huge delivery fleet of drivers that are constantly crossing paths. It is unknown if any metropolitan area has any such operations big enough. Even then, implementing a handoff system would also incorporate risks we did not account for in the scope of this project, such as implementation, complexity, cargo space, and risking orders lost. Lastly, it's likely that modern algorithmic driver routing already reduces the kinds of situations in which handoffs work (drivers headed to similar destinations crossing paths). If you would imagine an efficiently designed bus system, there would ideally be few situations in which two buses headed to nearby destinations would cross paths; if so then the route, frequency, or capacity could change. We knew these caveats when we started this project, but we believed that even with smart routing, there is always going to be at least some imperfect scenario where two drivers heading to similar situations will cross paths, meaning a handoff could save time. We still believe this is true, but we simply did not find enough of a clear improvement to keep investigating, even with ideal conditions.