Discussion 1

Routing Optimization System

- The basic structure of the algorithm would be as follows:
    - **Input:** Order, Order_Location, Restaurant_Location, sigma(radius) value, max_orders_allowed
    - **Output:** Rider, Optimized_Route
    - **Steps:**
        1. Neiching (Restaurant_Location, sigma, max_orders_allowed): possible_riders
        2. Foreach possible_rider:
            i. Calculate the fitness_function (Rider, Order_Location).
        3. Rank the rider or arange the possible_riders array on the basis of their fitness value.
        4. Select the top most rider.
- **Neiching** is the step where we select the riders which could possible minimize the total time taken to deliver a parcel.
- Following are the steps for the neiching process:
    - **Neiching (Restaurant_Location, (σ)sigma, max_order_allowed):**
    - **Input:** Restaurant_Location
    - **Output:** Possible_riders
    - **Steps:**
        1. Select all the riders present in the same area.
        2. Initialize an empty array of riders; assign to possible_riders
        3. Foreach rider:
            i. Calculate the distance of the rider from the restaurant location; assign to rider_Distance
            ii. If (ride_Distance <= sigma and rider_order_count < max_order_allowed):
                - Add rider to possible_riders
        4. Return possible_riders.
- **Step 1 (Neiching):** This step selects the riders from the overall population and filters out only those fiver riders which could possible minimize the delivery time.
- The above function works by first selecting the riders which are present in the area of the riders. Then, out of those selected riders, it selects the riders present in the radius as specified by **σ** which in our case we have fixed it to 5 km. The value of **σ** can be

increased or decreased if we are not getting the enough numbers of riders in a particular area. An empty array of possible_riders is also initialized in the beginning. At a given particular time, we should have at least five riders to compare and find the best possible rider. While filtering out the riders, we also check if the rider is allowed to take more order or not i.e. if the rider has exceeded its max order limit or not which in our case we have decided as 5 i.e. a rider is allowed to have maximum 5 orders at a time. For the time being, we are assuming the values (such as σ, max_orders and the number or riders to search for) as per used by most of the real world companies such as Uber Eats and foodpanda.

- If a rider fits the above criteria, it is added to the possible_riders array.
- The final possible_riders array is returned in the end to be used for the calculation of fitness values which is the step 2 of the basic structure of the algorithm.

- **Step 2 (Calculation of fitness function of every rider):** This part of the algorithm is not finalized yet in a proper format but the discussed points are mentioned below.
- In this step, we will use the array of riders got in step1 and **for each rider** we will do the following:
    - All the possible networks of routes will be created that a rider can opt for at a particular time. For this purpose, we might use voronoi diagram.

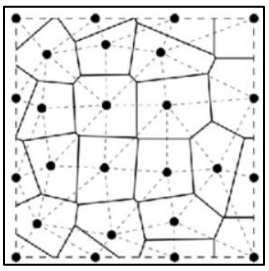

Figure 1: A simple Voronoi Diagram showing the network of possible routes.

    - After finding all the possible routes, a local search will be performed to find the best possible network that would get the job done in the least time. This part would require the parameters about the rider such as the orders it already has and the order for which we are finding the best rider, the delivery locations where the rider has to make the deliveries and other route related data such as the path length etc. For finding the best route for a rider we have decided to use A* algorithm to find the shortest path among all the possible paths.
    - In the next step, we have to calculate the fitness value of every rider using the route time got from the above step. We will also use the order_count(number of orders a rider has) and the rank (rating) of the rider to calculate its fitness value.
    - We will form a scalarization function to calculate the fitness value.

    - Order_count and the performance (rating) of the rider will be used as the tie breaker rule in case if two or more riders come equal after calculating the fitness value.
    - Still, if two riders come equal, we will choose a random rider.
- **Step 3 & 4 (Rider Selection):** After getting the fitness value we will rank the riders according to their fitness values.
- After the ranking step, the top most rider will be selected and suggested as the **best possible rider**.
