# 1 Algorithm Design

The most relevant algorithm of the myTaxiService application is the one implementing the queue management. It is important to optimize its implementation as long as it is the most complex and the one that distinguishes the system.
Another significant algorithm is the one that manages the forwarding of the requests of the users to the taxi drivers.

- **Initialization.** The class which manages the areas distribution and the queues of the taxi driver also deals with the initialization of the queues of the taxis. It starts assigning to every area as many taxi as the average number of the requests expected in that area, based on the statistics. Once all the areas have been filled with their optimal number of taxis, the remaining taxi drivers are sent to the areas that are most needy.
  The "needyArea" function, in fact, returns the index of the most needy area at the moment, calculating the area that has the maximum difference between the maximum number of requests recorded and the average of requests.

**Algorithm 1** Initialization

```
procedure initQueues(TaxiDriver[] taxiDrivers, Area[] areas)
    i ← 0
    j ← 0
    for i < areas.size() do
        k ← 0
        for k < areas[i].getAverage() do
            areas[i].addToQueue(taxiDrivers[j])
            j ← j + 1.
        end for
    end for
    if j < taxiDrivers.size() then
        for j < taxiDrivers.size() do
            areas[needyArea(areas)].addInQueue(taxiDrivers[j])
        end for
    end if
end procedure
```

COMPLEXITY: O(T) with T = number of taxis.

- **Manage requests.** After the initialization, the taxi drivers are able to take requests. If they accept the request, the function sets their availability to false and stops monitoring their position. It is also possible that a taxi driver declines the request or is not able to give an answer to the server. In this case, the first taxi driver is moved to the last position of the queue and forwards the request to the second of the queue, who is now moved to the first position. This action is iterated in the queue of that area until a taxi accept the request. After accepting the call, the taxi driver is set unavailable and he is deleted from the queue, as it is likely that he will end up in another area.

**Algorithm 2** Manage requests

```
procedure manageRequest(Position pos, Area area)
    answer ← "no"
    timer ← new Timer(60)
    while answer = "no" || answer = "timeOut" do
        sendRequest(area.getFirstOfQueue(), pos)
        answer ← waitAnswer(timer)
        if answer = "no" || answer = "timeOut" then
            area.moveToEndOfQueue()
        end if
    end while
    area.getFirstOfQueue().setUnavailable()
    area.deleteFirstFromQueue()
end procedure
```

COMPLEXITY: O(A+Q) with A = number of areas and Q = number of taxi enqueued in that area.

- **Manage queues.** The areas of the city are represented by a graph with an array of adjacencies and the queue of the taxi drivers, whose position is within its boundaries, is assigned to each area. It is a useful representation in order to decide how to distribute the taxis, in fact it is possible that an area is occupied by a number of taxi that is equal to the threshold of the maximum taxis that can be present in that area. In this case the system does not put a recently arrived taxi in the queue of that area, but advise the taxi driver that he will be moved to an adjacent area that has the minimum number of taxi, searching recursively in the adjacencies of the areas. As it is necessary to guarantee that there should always be at least one taxi driver in each area, when it occurs that the last taxi driver leaves an area, the system instantly notifies the last taxi driver of the queue that he has to move from the most populated adjacent area to the needy area. termina sicuramente dentro il while perch i taxi non sono tutti massimi.

  COMPLEXITY: O(A) with A = number of areas.

ADD DOMAIN PROPERTY: ALMENO UN TAXI ACCETTA LA RICHIESTA.

**Algorithm 3** Manage queues

```
procedure manageQueue(Area[] area, TaxiDriver taxiDriver)
    List < Area >  queue
    Area tmp
    for i < area.size() do
        if taxiDriver.position is in area[i] then
            startingArea ← i
        end if
    end for
    if area[startingArea].getNumberOfTaxi() < MAX then
        area[startingArea].addToQueue(taxiDriver)
    else
        for i < area.sie() do
            area[i].distance ← INFINITY
        end for
        area[startingArea].distance ← 0
        queue.enqueue(area[startingArea])
        while !queue.isEmpty() do
            tmp ← queue.dequeue()
            for i < tmp.adjacencies.size() do
                if tmp.adjacencies[i].getNumberOfTaxi() < avg[i] then
                    tmp.adjacencies[i].addToQueue(taxiDriver)
                    return
                end if
            end for
            for i < tmp.adjacencies.size() do
                if tmp.adjacencies[i].getNumberOfTaxi() < MAX then
                    tmp.asjacencies[i].addToQueue(taxiDriver)
                    return
                end if
                if tmp.adjacencies[i].distance = INFINITY then
                    tmp.adjacencies[i].distance ← tmp.distance + 1
                    queue.enqueue(tmp.adjacencies[i])
                end if
            end for
        end while
    end if
end procedure
```