**Loading files and initialization**

1. Network of the city (“Chicago”)
2. Crime data :
3. Crime 🡪 Segregating into index/violent and non-index/non-violent
4. Assign the no of police needed for each crime
5. Police data
6. Load police (i.e. station) geo-location (PS\_data)
7. Assign these police station geo-location to the nearest node in the “network”
8. Define police beat – (Jurisdiction of each police station on it’s district) 🡪 Import the shape file of police beats ; Result will be creation of polygons representing each district of the city “Chicago”
9. The district shape polygon created in (c), is assigned the node of its associated police station. (All crimes occurring in a particular district will the responded from its corresponding police station node)
10. Assign the police strength (i.e. no of police officers present) per district/police station

**Police Dispatch logic :** function “dispatch\_logic”

1. Input is the district shape polygon and the crime node(“pt”)
2. Crime node is checked to see which police district is the crime node coming under. The district shape polygon created defines the jurisdiction of different police station. All the crimes occurring inside a particular district polygon gets assigned to its respective police station.
3. If the crime node is outside all the district polygons, the crime node is associated to the closest district polygon
4. The output of the police dispatch logic is the police station district that the crime gets assigned to as well as its police station node and capacity(i.e. no of officers)

**Evaluation logic: function “eva\_resp\_time”**

1. Input of the function is police (i.e. PC\_node), crime data , %of BOU, police beat shapefile, city network
2. Start the loop for evaluating the response for each crime, (for i=1:length(Crime))
3. Create a list that stores the status of all the police stations, (in our case this is PC\_stalk)

PC\_ stalk contains:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Police station “node” | No of police officer engaged | No of crime in waiting queue | Time Stamp of Crime (Actively handled) | Crime node of Active Case | Time to get released from crime being actively handled | Crime node of Passive (in waiting queue) case | Time Stamp of Crime (Passively handled) | Service time of crime (Passive) |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

The 1st column , i.e. PC\_stalk[0] contains the police station nodes

1. Create a list of time stamp of crime occurrence (incident call) (in our case this is “Crime\_time\_next”). This list stores the time index of “next crime”, i.e. Crime\_time\_next[0] will store the time stamp of crime[1]. This is an indicator of when the next crime in queue occurred.
2. Call dispatch logic : Assign police station to Crime[i] (Get police district of jurisdiction , corresponding police station node, capacity of the station)
3. If the assigned police station is not maxed out in capacity (all officer’s not already engaged), the crime assigned will be actively handled (Calling these as “active crime”)
   1. Add 1 to PC\_stalk[1] to the corresponding assigned police station node PC\_stalk[0] (status of no of officer engaged in the station)
   2. Store the time stamp of crime call (Crime [i]) in PC\_stalk[3] in the corresponding police station row
   3. Store the crime node in PC\_stalk[4], (Active crime nodes)
   4. PC\_stalk[5] : Calculate the time it will take to completely handle the crime , i.e. get discharged from the Crime[i] = Travel time to and from the station to crime node + Service time
   5. Store the response time (i.e. Travel time from the station to crime node)
4. If the assigned police station is maxed out in capacity (all officers arealready engaged), the crime assigned will be handled after officers become available (Calling these as “passive crime”) :
   1. Add 1 to PC\_stalk[2] to the corresponding assigned police station node PC\_stalk[0] (status of no of crime in waiting queue: passive crime)
   2. Store the crime node in PC\_stalk[6], (Passive crime nodes)
   3. Store the time stamp of crime call (Crime [i]) in PC\_stalk[7] in the corresponding police station row
   4. Store the service time in PC\_stalk[8]
5. Moving passive nodes (crime in waiting queue) to active nodes(actively handled crime) ;(if officer becomes available)
   1. Start the loop for each police node and check if there the no of officers engaged are greater than zero
   2. Check if any of the crimes being actively handled, will get done before the next crime occurs ,i.e., “PC\_stalk[5] <Crime\_time\_next”

If yes then, move the crimes form the passive list category to active list in PC\_stalk

* + 1. E.g. , the crime nodes in PC\_stalk[6] will be moved to PC\_stalk[4]
    2. Time index needs to correctly transferred here : when moving the passive to active status, PC\_stalk[5] will be moved to the corresponding PC\_stalk[3] , representing the time index of when the passive crime was moved to active status
    3. Calculate the time spend in the waiting queue for the passive crime node , i.e. = Timestamp of passive crime node occurrence – Time when a active node became free (i.e. an active crime was finished being handled) (“PC\_stalk[5]- PC\_stalk[7] ”)
    4. For the crimes moving from passive to active status : (PC\_stalk[5] ,i.e. the time it will take to completely handle the crime , i.e. get discharged from the Crime[i] = Travel time to and from the station to crime node + Service time + time spent in waiting queue)
    5. Store the response time (i.e. Travel time from the station to crime node + waiting time)
  1. If there are no crimes in queue, and if the crimes being actively handled, will get done before the next crime occurs ,i.e., “PC\_stalk[5] <Crime\_time\_next” , free the active crime nodes, the officers and its corresponding blocks in PC\_stalk

1. For all the crimes that are still in queue at the end of the day (after 24 hours simulation), move from passive to active and calculate response time accounting for the “waiting time” (similar to the logic explained in 8)