I. Algorithm

Algorithm 1 Zone-creating algorithm

Input:

- The city's graph $G = (V, E)$ with vertices V and edges E
- expert_knowledge_edges: The list of edges identified by expert knowledge as major origin-destination points
- n, k: positive integers
- λ threshold number

Initialize graph edge labels
for edge in E **do**
  edge[zone_center] $\leftarrow$ False
  edge[zone_number] $\leftarrow$ None
end for
$i \leftarrow 0$
unassigned_edges $\leftarrow E$
for edge in expert_knowledge_edges **do**
  edge[zone_center] $\leftarrow$ True
  edge[zone_number] $\leftarrow i$
  set_of_neighbours $\leftarrow$ list of the n-th step edge-neighbourhood of edge
  for neighbour in set_of_neighbours **do**
    neighbour[zone_number] $\leftarrow i$
    Remove neighbour from list unassigned_edges
  end for
  Remove edge from unassigned_edges
  $i \leftarrow i + 1$
end for
while length(unassigned_edges) $\neq 0$ **do**
  edge $\leftarrow$ Random element from unassigned_edges
  $a \leftarrow$number of unassigned edges in the k-th step neighbourhood of edge
  $b \leftarrow$ number of edges in the k-th step neighbourhood of edge
  if $a/b >= \lambda$ **then**
    edge[zone_center] $\leftarrow True$
    edge[zone_number] $\leftarrow i$
    set_of_neighbours $\leftarrow$ k-th step neighbourhood of edge
    for neighbour in set_of_neighbours **do**
      neighbour[zone_number] $\leftarrow i$
      Remove neighbour from list unassigned_edges
    end for
    Remove edge from list unassigned_edges
    $i \leftarrow i + 1$
  else
    edge[zone_number] $\leftarrow$ Randomly selected from the zone_numbers present in the k-th step neighbourhood
  end if
end while
Output: The graph of the city with the new edge labels

Algorithm 2 Sensor-selecting algorithm

Input: The graph of the city with the edge labels created by 1, and n, the number of sensors we have to put down
sensor_places_list $\leftarrow$ []
for edge in E **do**
  edge[weight] $\leftarrow 0$
end for
zone_center_list $\leftarrow$ edges that has True label for [zone_center]
for zone_center in zone_center_list **do**
  for edge in E **do**
    if zone_center[zone_number] == edge[zone_number] **then**
      zone_center[weight] $+ =$ edge's traffic volume
    end if
  end for
end for
list_of_edges $\leftarrow$ using Dijkstra's algorithm we calculate the shortest path (in terms of time) between all of the zone center pairs and we put the edges that are part of at least one pair-connection into a set
for edge in list_of_edges **do**
  edge[list_of_connected_zone_center_pairs] $\leftarrow$ The list of zone_center pairs that it connects
  edge[weight] $\leftarrow$ The sum of the zone_center pairs' total weight that it connects
end for
$i \leftarrow 0$
while (There is an edge in list_of_edges that has a non-empty list for list_of_connected_zone_center_pairs label) AND $(n > i)$ **do**
  Sort list_of_edges in descending order based on first, the length of the label list_of_connected_zone_center_pairs and second, also in descending order based on the label weight
  The first element of the list is put into the sensor_places_list and removed from list_of_edges
  The remaining edges' list_of_connected_zone_center_pairs label is updated: all of the zone center pairs that were present in the chosen edge's label is removed from the rest of the edges' labels
  $i \leftarrow i + 1$
end while
while (length(list_of_edges) $\neq 0$) AND $(n > i)$ **do**
  Sort list_of_edges based on the weight label in descending order
  The first element is added to the sensor_places_list and removed from the list_of_edges. The first-step neighbourhood of this edge is also removed from the list_of_edges
  $i \leftarrow i + 1$
end while
if n¿i **then**
  The zone centers are chosen randomly and added to the sensor_places_list until we have n elements in this list
end if
Output: sensor_places_list