Optimal Deployment of Roadside Units for Vehicular Networks

Report for COS310: Independent Study

Ashish Gupta
2017MT10250
Mathematics & Computing
mt1170250@iitd.ac.in

Sakshi Taparia
2017MT10748
Mathematics & Computing
mt1170748@iitd.ac.in

Supervisor
Prof. S. R. Sarangi
Department of Computer Science & Engineering
Indian Institute of Technology, Delhi

1 Introduction

Research in vehicular networks has attracted much attention in recent years [3, 12, 27, 12]. A vehicular ad-hoc network (VANET) relies on three types of communication for its setup and provision of services: vehicle to vehicle (V2V) communication, vehicle to infrastructure (V2I) communication, and infrastructure to infrastructure (I2I) communication. All VANET applications depend on either one or more of these communication types. V2V communication depends on the number and location of vehicles, V2I communication depends on the number and location of roadside units (RSUs), and I2I communication relies on the availability of an interconnecting network between RSUs. During the initial deployment stages of VANET, there will be a small number of vehicles and RSUs due to the low market penetration of VANET-enabled vehicles or due to the deployment cost of RSUs.

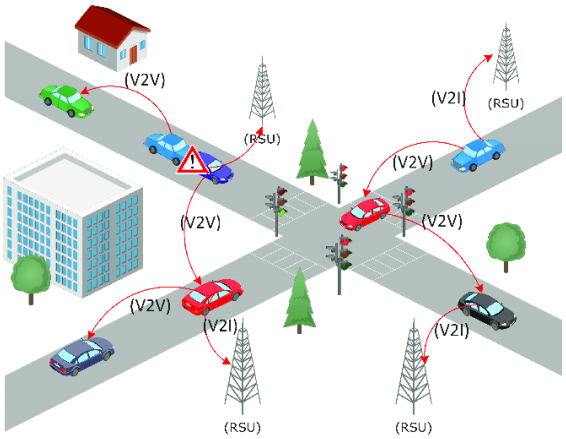


Figure 1: VANET communication architecture

Information flow in most VANET applications is either from vehicles to infrastructure or from infrastructure to vehicles. In our work, we have focused on applications that depend on information flow from vehicles to infrastructure (or RSUs), such as the collection of information from vehicles about traffic/road conditions, traffic accidents, etc. The Road Side Unit (RSU) works as an access point between VANET and other private networks. Whenever any vehicle comes into the transmission range of the RSU, it starts communicating with the RSU and exchanging information. Hence, placing RSUs efficiently is necessary for better and continuous communication between vehicle and RSU while minimizing the net deployment cost.

Deployment of RSUs depends on many factors like RSU transmission range, road length, traffic density, budget, etc. Therefore, the optimal RSU placement problem, like many other real-world optimization problems, is very challenging to solve, and optimization tools have to be used to solve them. However, there is no guarantee that an optimal solution will be obtained. In fact, for NP-hard problems, there are no efficient algorithms at all. As a result, many problems have to be solved by trial and error using various optimization techniques. Also, new algorithms have been developed to see if they can cope with these challenging optimization problems. Among these new algorithms, many algorithms, such as particle swarm optimization, cuckoo search, and firefly algorithm,

have gained popularity due to their high-efficiency [14]. In our work, we have tried to achieve an optimal strategy for the deployment of RSUs for two different scenarios using the nature-inspired ant colony optimization algorithm.

Messages between vehicles and RSUs can be classified into delay-sensitive or not delay-sensitive (non-safety) messages. We have proposed two different problem formulations for both these categories. We have selected candidate positions on road intersections as well as some locations along the streets. For each vehicle, the coverage time ratio (CTR) has been evaluated to test the performance of the network after obtaining the optimal placement scheme for the RSUs. Coverage time ratio (CTR) is the ratio of time for which a vehicle is in the coverage range of an RSU to the total time that the vehicle spends in that region. We have also experimented with parameters such as average coverage time ratio, average packet delay, and average packet loss for multiple budgets and the number of vehicles in that area. We have compared our algorithm with the degree centrality approach [27] and greedy heuristics.

Further, we have tried to achieve speedups and performance improvements over the algorithms mentioned above using optimized data structures and parallelization methods.

2 Related Work

Nowadays, RSU deployment is widely studied, as the performance of the communication network relies on the placement of roadside units. It impacts the reliability of the network and the sharing of data. There are lots of techniques for RSUs deployment and lots of limitations also.

2.1 Categories

First, we have listed a few broad categories that most of the research papers and the work done in this area of study fall under.

2.1.1 V2V vs V2I

V2V (vehicle to vehicle) enables the communication between vehicles with a slow transmission rate and short transmission distance at a low cost. However, V2I (vehicle to infrastructure) is a communication model in which vehicles communicate with roadside facilities such as Road Side Units (RSUs) using high transmission speed, long transmission distance, and topological fixedness. In the current automotive environment, these two models are combined to compensate for each model's deficiencies.

2.1.2 Spatial vs Temporal

Scenarios for RSU placement can be spatial and temporal. However, most of the recent papers combine the two. Spatio-temporal coverage approaches first extract the mobility patterns of moving vehicles and then compute the appropriate RSU locations based on the extracted patterns as finding the relevant mobility patterns is a step of paramount importance.

2.1.3 Other Scenario Variations

- Road network layouts can be in one-dimension as well as two-dimension.
- Candidate points can be selected on intersections, between road segments, or both.
- Signal degradation from surroundings may or may not be accounted for.
- RSUs with varied transmission ranges and budgets can be considered instead of a single type of RSUs.
- Multi-hop relaying and multi-homing have been incorporated in some papers.
- RSUs for larger regions and sensors for smaller regions can be combined instead of using only RSUs to cover the maximum area.

2.2 Research Papers

Advancements in the technology of transport communication systems are required to support a myriad of information exchanges related to both critical and non-critical functions. In 2011, Aslam et al. [3] proposed the balloon optimization algorithm based on the balloon expansion analogy to propose a strategy for placing the RSUs along a highway in an optimal manner, targeting to minimize the average time duration of a packet to reach a nearby RSU initiated form the vehicle. It had a limitation on the number of RSUs. Following this, there are a lot of studies focusing on the deployment of RSUs in 2-dimensional street scenarios such as an urban area with planned streets.

In 2012, Patil et al. [21] proposed to maximize vehicular network connectivity through a placement scheme using Voronoi Diagrams and Liang et al. [15] formulated an Integer Linear Programming on a two-dimensional grid for the same. Aslam et al. [2] used an urban area to study the optimal placement of the RSUs, with a focus on employing the Binary Integer Programming (BIP) method and Balloon Expansion Heuristic (BEH) methods to reduce the mean time latency of packet with a limited number of RSUs. Around the same time, Wu et al. [25] also employed the concept of Integer Linear Programming (ILP) with Capacity Maximization Placement to provide a cost-effective strategy for RSU placement. However, this study also went a step ahead to take multi-hop relay into account. Lin [16] formulated the RSU deployment for the V2I network into a binary integer programming problem with constraints and used the branch and bound heuristic approach to solve it and reduce the time complexity. In the same year, also had graph-based methods [26] that converted the problem statement to a standard NP-Hard vertex cover problem and solved it via greedy heuristics.

In 2013, Jo et al. [8] proposed an approach based on the priority of an intersection for the deployment of RSUs that minimized the installation cost of an RSU while maximizing the coverage of intersections. In 2014, Liya et al. [17] proposed a randomized algorithm for RSU placement and demonstrated that the algorithm was capable of performing reasonably well in many settings. The Binary Programming Model [9] proposed in 2016 used binary ILP formulation for information dissemination by taking candidate points on the road segments apart from intersections. Further, the advances made in the area of optimal RSU placement from 2017 onwards have been tabulated below.

Title	Objective	Approach	Novelty	Result
Centrality based RSU Deployment Approach [24]	Maximizing coverage time ratio and minimizing the total budget	Utilized the notion of centrality to formulate an ILP and used 0-1 Knapsack algorithm to solve it	Priority of candidate positions was determined by their centrality values	Gave better CTR value as compared to random deployment approach
GICA: Evolutionary strategy for RSU deployment in Vehicular Networks [11]	Maximizing spatial coverage and minimizing the total budget	Formulated a multi-objective optimization problem and employed genetic algorithm to solve it	Priority for each intersection was determined by its traffic density	Showed increased coverage and reduced overlap when compared with greedy algorithm
Efficient RSU Placement Schemes for Urban Vehicular Ad Hoc Networks [12]	Maximizing spatial coverage and minimizing the total budget	Greedily picked candidates with large impact factors and later removed the ones with low contributions to the final coverage	Utilized a combination of an algorithm used to determine vertex cover along with the greedy method	Gave better results than hybrid and genetic algorithms
MPC: RSU deployment strategy for VANET [27]	Maximize coverage ratio and minimizing the number of RSUs used	Extracted mobility patterns of vehicles from trace files and found candidate points using minimal traversal of hypergraph	Used mobility patterns to generate a hypergraph for the first time	Outperformed Centrality approach in terms of coverage ratio as well as latency values
RSU deployment framework for enhancing transportation services [4]	Maximizing coverage ratio and minimizing the number of RSUs used	Extracted mobility patterns from a sequential database and used a hypergraph to select junctions for RSUs	Extracted mobility patterns and computed crossing probabilities of vehicles at intersections	Gave better results than MPC [27] approach described in the fourth row
A Roadside Unit Placement Scheme for Vehicular Ad-hoc Networks [7]	Maximizing coverage ratio and minimizing the number of RSUs used	Generated a hypergraph from frequent mobility patterns and computed minimal traversal using dual equivalents	Proposed an alternate definition of coverage and calculated traversals of hypergraph	Only compared its results with MPC [27] and showed slight improvement

Title	Objective	Approach	Novelty	Result
Smart RSU placement for vehicular networks using evolutionary algorithms [18]	Maximizing coverage ratio and minimizing the number of RSUs used	Used floating-point numbers with an integer indicating RSU type and fraction indicating RSU location on the road	Considered every possible point on the road, and not just intersections, as candidate points	Compared its results with basic algorithms like uniform, random and greedy approach
Optimizing Infrastructure Placement with GA [22]	Maximizing coverage ratio and minimizing the number of RSUs used	Proposed a genetic algorithm that considered signal degradation caused by urban obstacles	Considered RSU range as polygon rather than circle, taking signal degradation into account	No comparison with any other work as problem formulation is very different

Further, our report considers the thesis, *Optimal Deployment of Roadside Unit for Vehicular Ad-hoc Networks*, by Ravi Shankar Singh as a starting point. We have used Ant Colony Optimization and Greedy Approach to outperform the Degree Centrality Approach described in the thesis. We have also proposed an alternate problem formulation for communication of non-safety messages and improved the performance of our algorithms using CPU based parallelization methods.

3 Frameworks & Tools

- **OpenStreetMap (OSM):** This is a tool for creating and sharing map information. We used OSM to obtain the .osm files and then used SUMO to convert them into .xml files containing information about the road networks as well as generated random traffic on these roads to calculate the traffic density values for all the road cells.

- **Simulation of Urban Mobility (SUMO):** Traffic simulations help us in estimating the changes required in infrastructures and policies before they are executed on the streets. SUMO is an open-source, free of cost, continuous, and microscopic road traffic simulation suite meant for managing large road networks. It supports tools that can handle tasks such as network imports, estimation of emission, and visualization. It also provides APIs such as Traci, to control the simulation remotely.

- **OMNeT++:** It is an extensible, modular, component-based C++ simulation library and framework, primarily used for building network simulators. In a broad sense, the word "network" includes wired and wireless communication networks, on-chip networks, queuing networks, and so on. Moreover, domain-specific functionalities such as support for sensor networks, wireless ad-hoc networks, and performance modeling are provided by model frameworks that have been developed as independent projects. OMNeT++ also offers an Eclipse-based IDE, a graphical

runtime environment, and extensions for real-time simulation, network emulation, database integration, etc.

V2I communication plays a significant role in many vehicular applications. These applications are divided into safety and non-safety use cases. Safety use cases are delay-sensitive, which is why they need low latency communication as delay above a certain threshold can lead to an accident. For instance, a pedestrian or cyclist crossing a junction that is not visible to the vehicle coming toward that intersection. On the other hand, the non-safety use cases are latency-tolerant. For instance, sharing the sensor data of a vehicle to the cloud for extracting the driving pattern of the driver. In the remaining report, we have documented ***two separate problem formulations***, namely A and B, based on the type of messages being communicated.

4 Problem Formulation - A

This formulation is useful for ***V2X communication for delay sensitive messages*** like blind intersections, overtake warnings, and road work in progress warnings, where ***we desire to maximize the duration for which every vehicle is included in the transmission range of RSUs***.

We first identify areas with high traffic density i.e. the places where the chances of vehicles crossing the roads is very high. Specifically, a road intersection is an area shared by two or more roads and hence qualifies to be one of the best candidates for placing the RSUs. However, there are some road segments that are too long to be covered adequately if the RSUs are only placed at the terminals of the road segment. In these cases, we add some more locations over that particular stretch to avoid the issue of insufficient coverage of the road, thus providing a continuous connection between RSUs and vehicles. The set of all possible locations that can serve as anchors for placing the RSUs are called the candidate positions.

Assuming that the length of a street between two connected junctions is L and the transmission range of an RSU is R_u, adding a candidate position is then determined by the rules given below.

1. If the road length L is less than or equal to twice the transmission range R_u :

$$L <= 2R_u$$

then there is no need to add any extra candidate positions on the road segment.

2. If a road length L is greater then twice the transmission range R_u :

$$L > 2R_u$$

then there is a need to add new candidate position on the road. The number of new candidate positions depends on the length of the road L and the transmission range R_u given by the equation :

$$\left\lceil \frac{L - 2R_u}{R_u} \right\rceil$$

We further filter out and remove the candidate points with zero traffic density.

Road Cells

The total area on the map is divided into cells of the same size where each cell either represents a single road segment or a non-road area. Cells containing a part of any road segment are known as *Road Cells*. Since we have made an assumption that a cell is the smallest possible area on the map, therefore, no two roads can come under a single cell.

Binary Coverage

An RSU covers all the road cells that come under its transmission range. Moreover, some road cells can be covered by more than one RSU, and some road cells might not be covered at all. Cov_{kij} is a binary matrix with value 1 when the road cell k is contained in the transmission range of RSU of type j placed at candidate position i. Mathematically, Cov_{kij} is defined as :

If (Euclidean Distance of $[RC_k, CP_i]$ <= Transmission range of type j RSU):

$Cov_{kij} = 1$

Else:

$Cov_{kij} = 0$

CP_i	Candidate position i where i ranges from 1 to N
RC_k	Road cell k where k ranges from 1 to K
deg_i	Number of intersecting roads at candidate position i
C_{ij}	Cost of deploying RSU of type j at candidate position i
$Cov_{k,i,j}$	Value is 1 when RC_k is covered by RSU of type j placed at CP_i
$X_{i,j}$	Value is 1 when RSU of type j is placed at CP_i
w_k	Weight of RC_k depending on traffic density
B	Maximum Budget for deployment

Table 1: List of variables used in the description of algorithms

Hence, the objective function of the ILP that we wish to solve is :

Maximize:

$$\sum_{k=1}^{T} min(1, \sum_{i=1}^{N} \sum_{j=1}^{R} w_k Cov_{kij} X_{ij})$$

Subject to constraints:

$$\sum_{j=1}^{R} X_{ij} \leq 1$$

$$\sum_{i=1}^{N} C_{ij} X_{ij} \leq B$$

The first constraint allows for the placement of at most one RSU at every candidate position. The second constraint restricts the maximum deployment budget to B. Further, we consider the minimum of 1 and $(\sum_{i=1}^{N} \sum_{j=1}^{R} w_k Cov_{kij} X_{ij})$ for every RC_k to avoid a particular road cell from being counted multiple times in the total coverage even if it is covered by multiple RSUs.

Floating Point Coverage

Signal strength is maximum and packet delay is minimum near the source antenna and the signal strength tends to degrade as we move away from it. To incorporate this property into coverage, we have defined a different form of coverage which is not a binary value. Instead, we have used a Gaussian Distribution to assign coverage values to the road cells surrounding the source road cell (the one with the antenna), giving the maximum value (= 1) at the source road cell and the decreasing values as we go away from it.

4.1 Algorithms & Results

This section considers **homogeneous RSUs** ($j = 1$) with a transmission Range of 243 metres. Further, the cost of deployment is approximately $ 4,000 per RSU with very slight variations depending on whether the candidate point has a traffic light or not. Hence, a budget of $ 50,000 units implies that $12 - 13$ RSUs have been deployed. We have considered three algorithms, namely Degree Centrality, Greedy approach and Ant Colony Optimization for comparison.

4.1.1 Degree Centrality

Zhenyu Wang et al. [24] studied a deployment approach based on computing the centrality value of a candidate position without considering the traffic knowledge of the area. This centrality value, used to evaluate the importance of every candidate position for RSU placement, was defined as :

$$\eta_{di} = \frac{deg_i}{N-1}$$

where N is the total number of candidate positions in the network and η_{di} is degree centrality of CP_i. They formulated the problem as a 0-1 Knapsack problem with the maximum budget as the capacity of the bag, the deploying cost as the weight of the item, and the centrality of the node as an item. Finally, the candidate positions were selected to maximize the total degree centrality of the selected set under a maximum budget B using dynamic programming.

$$m(i,b) = \begin{cases} max\{m(i+1,b), m(i+1,b-C_i) + \eta_{di}\}, & \text{if } b \geq C_i, \\ m(i+1,b), & \text{if } 0 \leq b \leq C_i \\ & i = 1,2,...,N \end{cases}$$

Here, $m(i,b)$ is the total centrality of a sub-solution for candidate positions $i, i+1, .., N$ given the budget B. C_i denotes the deployment cost of an RSU at position candidate i. The value of the optimal solution is therefore $m(1,B)$, which calculated iteratively starting from $m(N,b)$ as follows :

$$m(N,b) = \begin{cases} \eta_{dN}, & \text{if } b \geq C_N \ , \\ 0, & \text{if } 0 \leq b \leq C_N \end{cases}$$

4.1.2 Ant Colony Optimization

Ant Colony Optimization (ACO) takes inspiration from the foraging behavior of some ant species that deposit pheromones on the ground in order to mark particular favorable paths to be followed by other members of their colony. Ant Colony Optimization algorithm exploits meta heuristics with similar mechanisms for solving optimization problems.

The first step of ACO is to create the *Ant Class*. The *Candidate Positions* list extracts all candidate positions and the cost of installing RSUs at every location. Further, the *Coverage Matrix* defines whether a given road cell will be covered by the RSU, if placed at a given candidate position.

Algorithm 1: Ant Class

CandidatePositions stores the trail of the ant
CandidatePositions = {}

CurrentCoverage is defined similar to Cov matrix for individual ants
CurrentCoverage = null matrix

CoverageValue = 0
CurrentBudget = 0
FreezeAnt = false

Transition probabilities and trails are updated in Ant Colony Optimization
Probability = null matrix
Trial = null matrix

Functions defined in this class are called by Algorithm 2 and 3.

Function `AddCandidatePosition(`*index*`):`

> CandidatePositions = CandidatePositions.add(index)
> CurrentCoverage[:,index] = Cov[:,index]
> CurrentBudget = CurrentBudget + Cost(index)
>
> CoverageValue = CoverageValue + FindCoverageChange(index)[0]
> FindCoverageChange dynamically computes additional coverage obtained on adding RSU at the *index*

return

Function `FindCoverageChange(`*index*`):`

> Temp = CurrentCoverage
> CurrentCoverage value is extracted from CurrentCoverage matrix. This counts the number of road cells covered by the ant
>
> Temp[index] = Coverage[index]
> Temp is updated to CurrentCoverage plus coverage addition by position the *index*
>
> addition = Temp - CurrentCoverage
> additionNormalized = Normalized(Temp - CurrentCoverage)

return addition,additionNormalized

The *Find Coverage Change* function computes the effective change in overall coverage value after placing the RSU at a candidate position identified by the given *index* value. This function is dynamically calculated. The term *dynamic* means that the coverage contribution of every candidate position is variable and is dependent on set of selected candidate positions generated till then. ***Dynamic coverage changes are the most crucial part of our algorithm***.

Algorithm 2: Ant Colony Optimization Functions

α, β**, Evaporation and AntFactor are hyper parameters**

Function `SetupAnts()`**:**

TotalAnts = AntFactor * size(CP)
Random initialization of Ant[i] for i in range (0,TotalAnts)

return

Function `SelectNextCandidate(`*Ant, curr*`)`**:**

For all i not in Ant.CandidatePositions list
Probability[curr,i] = FindCoverageChange(i)[1]$^{\alpha}$ + Trail[curr,i]$^{\beta}$

rand = random number between 0 to 1
sum = 0

For all i not in Ant.CandidatePosition
sum = sum + Probability[curr,i]
if *sum* > *rand* **then**
  return i
end

return ERROR!

Function `MoveAnts(`*index*`)`**:**

Only applicable when Ant[i].FreezeAnt = False
Current = latest element added to Ant[index] trail

Temp = SelectNextCandidate(Ant[index], Current)

if *Budget + Cost(Temp)* > *B* **then**
  Ant[index].FreezeAnt = True
else
  Ant[index].AddCandidatePosition(Temp)
end

return

Function `UpdateTrails(`*index*`)`**:**

Ant[index].Trail = Ant[index].Trail * Evaporation

For all i in Ant.CandidatePositions list
Ant[index].Trail[i,i+1] += Ant[index].FindCoverageChange(i)

return

Firstly, the *Setup Ants* function assigns random starting positions to the ants. Total number of ants in the system is determined by the *Ant Factor*. Then for *Max Iteration* times, the following steps are performed in a loop :

Move Ants Function: Every possible candidate position is defined as a node of a fully connected graph. The *Select Next Candidate* function works by probabilistically selecting a candidate position that is not present in the solution set of the ant. Chances of a candidate position j being selected just after placing an RSU at candidate position i is directly proportional to the value of Probability $[i, j]$. If the budget limit B is not exceeded by placing the RSU at selected position, then the *Add Candidate Position* function adds this position to solution set of the ant and updates the *Coverage Matrix*.

Update Trails: The *Trail Matrix* represents the weights given to each path which defines the likelihood of the path being followed by other ants. The *Evaporation Factor* is multiplied to the trails of each ant. For every candidate position, the *Coverage Change* factor is also added to its trail values.

Update Best: Among all the ants, the ant's trail with the maximum coverage value is selected and the global best coverage value is updated. The current budget is also updated accordingly. Hence, the algorithm (upon termination) returns the best coverage obtained till then. It also returns the corresponding ant trail which gives a list of all the selected candidate positions for placing the RSUs.

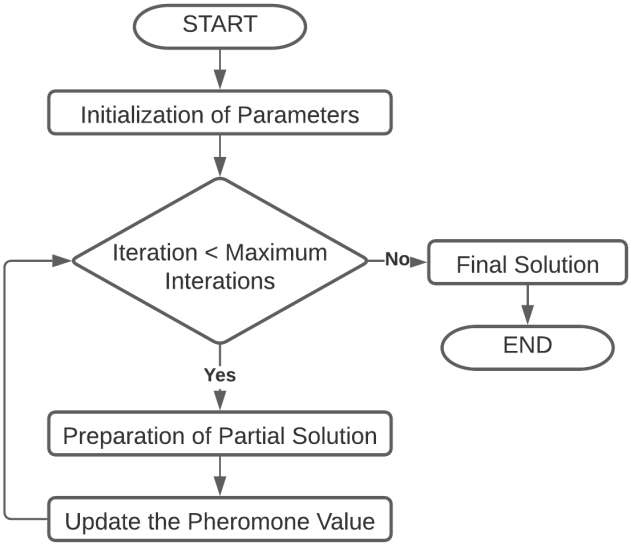


Figure 2: Overall Flow of the Ant colony Optimisation Algorithm

In Ant Colony Optimization algorithm, the ants' route preferences depend on the amount of pheromone deposited, which, in turn, is determined by the change in the value of the function to be optimized. The evaporation of the pheromone on all of the routes occurs at a certain rate, keeping the amount of pheromone on the routes that the ants have already passed (i.e., the past solutions) inversely proportional to the total relative imbalance level. The evaporation ratio causes a reduction in the importance of the previous solutions. A pheromone increase that is inversely proportional to the fitness of the route ensures the importance of the fine solutions.

Algorithm 3: Ant Colony Optimization Execution

Function `AntColonyOptimization()`:

Input: CP, B and Cov matrix
Set: α, β, AntFactor, Evaporation and MaxIterations

SetupAnts()

while *Up to MaxIterations* **do**

MoveAnts(i) for all i in range(0, TotalAnts)
UpdateTrails(i) for all i in range(0, TotalAnts)
Update BestSolution after every step

end
return BestSolution

4.1.3 Greedy Approach

Algorithm 4: Greedy Algorithm Functions

Input: Cost of each RSU at candidate position C = $\{c_1,...,c_N\}$
Output: Set of RSU deployment positions X and TotalCoverage

Function `CoverageAddition(`*index*`)`:

RC is a binary list indicating if a road cell is covered by any of the selected candidate positions or not.
while $j <$ *size(RC)* **do**
if *Cov[j,index] = 1 & RC[j] = 0* **then**
addition = addition + 1
end
j = j+1
end
return addition

Function `FindMaxWeightElement()`:

index = -1
while $i < N$ **do**
if *X[i] = 1* **then**
continue
else
if *CoverageAddition(i)* $>$ *maximum* **then**
index = i
end
end
i = i+1
end
return index

Greedy algorithm selects the candidate position with max weight in every iteration and increases the current budget appropriately. Here, the weight of a candidate position is defined as the ratio of coverage addition after placing an RSU at that particular candidate position to the deployment cost of placing the RSU. The algorithm terminates when the maximum budget B is reached.

Algorithm 5: Greedy Algorithm Execution

```
Budget = 0
TotalCoverage = 0
while Budget <= B do
    Index, NewCoverage = FindMaxWeightElement()
    if index = -1 then
        return: ERROR!
    end
    if (Budget + c_i) > B then
        break
    end
    Update Budget and list X
    X[Index] = 1
    Budget = Budget + c_i
    TotalCoverage = TotalCoverage + CoverageAddition(index)
    Update the RC list as described in Algorithm 4.
end
```

4.2 Initial Results

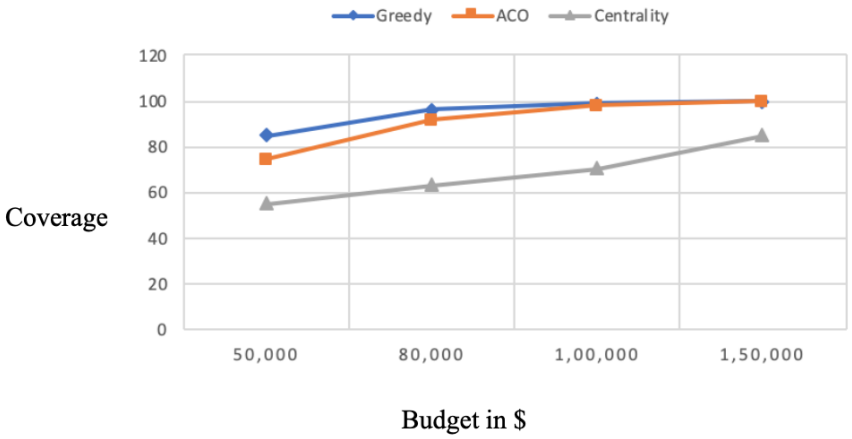


Figure 3: Comparison of Greedy and Degree Centrality with original ACO

Coverage indicates the percentage of total road cells that are covered by any RSU placed at the selected candidate positions. This quantity is proportional to the Coverage

Time Ratio (CTR) which indicates the fraction of the total duration for which a vehicle is covered by an RSU.

We can see that ACO outperforms Degree Centrality in terms of the percentage of road cells covered by the RSUs at selected candidate positions. Greedy performs much better initially but as the coverage value increases above 90%, it starts to saturate because it is unable to change the RSUs already placed and our algorithm that shows consistent improvement in coverage addition also comes at par with greedy. However, ACO has not been able to beat the results obtained by the greedy approach at any point.

4.3 Improvements in ACO

We proceeded by including the improvements listed below along with using the ***Floating Point Coverage*** definition proposed in Section 4 this time.

- Initialize some ants after the rest so that there has been sufficient time lapse for the trail values to develop properly.
- Drop the nodes that have less contribution to final coverage after new nodes with overlapping coverage regions have been added.
- Set maximum and minimum trail values so that no trail is dominating. If trail values leading to some candidate positions are significantly larger than those for others, all the ants tend converge to these candidate positions and other paths end up remaining unexplored.
- Maximum Iterations will become another hyper-parameter if ants start at different iterations.

4.4 Visual Representation

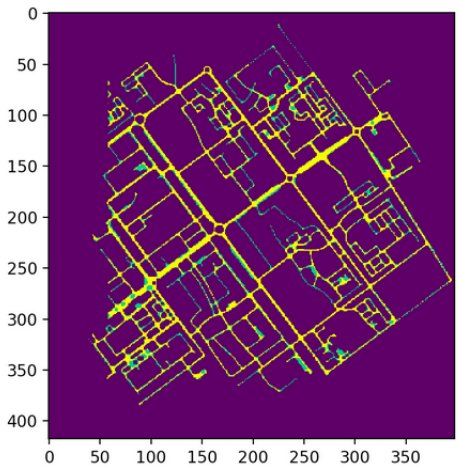


Figure 4: Map of an area of Chandigarh obtained from OSM with non-uniform traffic densities along roads generated using SUMO.

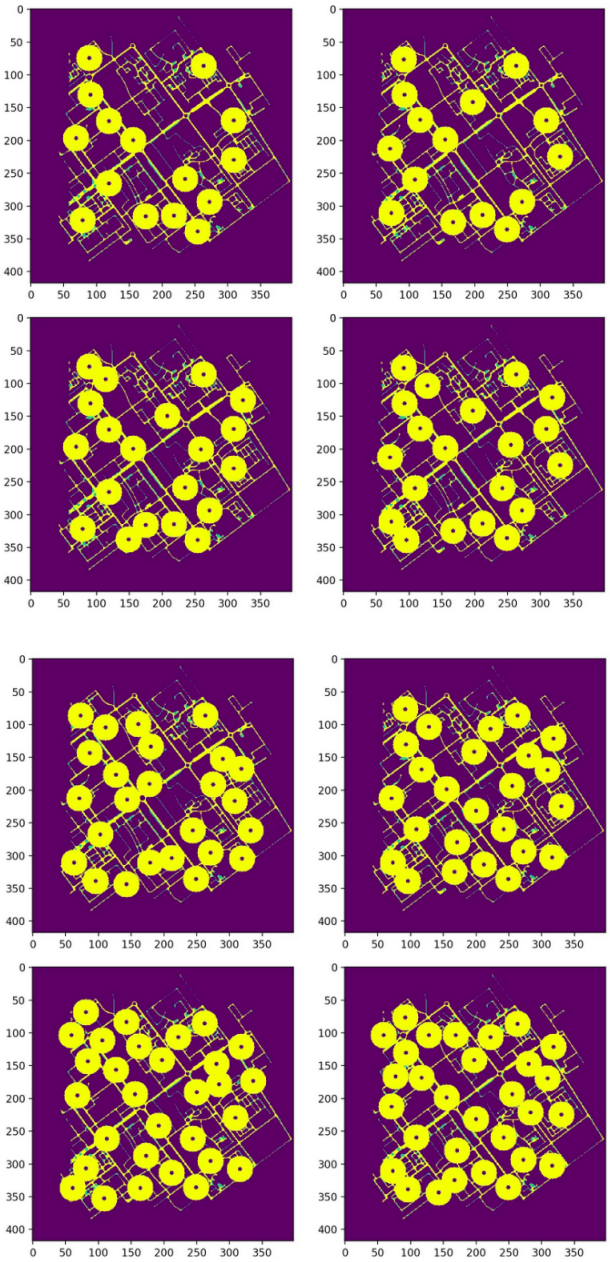


Figure 5: Maps after placing 15, 20, 25 and 30 RSUs using improved ACO (left) and Greedy Approach (right) on the sample map taken above.

The size of the yellow circle indicates the area covered by the transmission range of the RSU placed at the center of that circle.

Further, the table below tabulates the Floating Point Coverage values for greedy approach and ACO algorithm. The number beside ACO indicates the ratio of the two hyper parameters, α and β. We can see that ***ACO approach is able to outperform greedy method by a small margin when this greedy approach utilizing the dynamic coverage function that we have proposed has already outperformed Degree Centrality by a huge margin*** as per the graph shown in Figure 3. However, adjustment of hyper parameters is required to achieve these results.

Budget	Greedy	ACO – 2	ACO – 4	ACO – 6	ACO – 8	ACO – 10	ACO – 12	ACO – 14
50,000	37.72	37.38	37.18	37.31	37.63	36.99	37.04	36.36
80,000	49.49	50.00	49.48	49.54	48.91	47.83	47.39	
90,000	53.48	53.73	53.67	52.93	52.61			
1,00,000	55.79	56.15	55.96	55.34	53.64			
1,10,000	58.93	59.13	58.96	59.02	56.95			
1,20,000	60.89	61.07	60.6	60.30	59.18			
1,50,000	66.81	66.52	66.06	65.46	62.39			

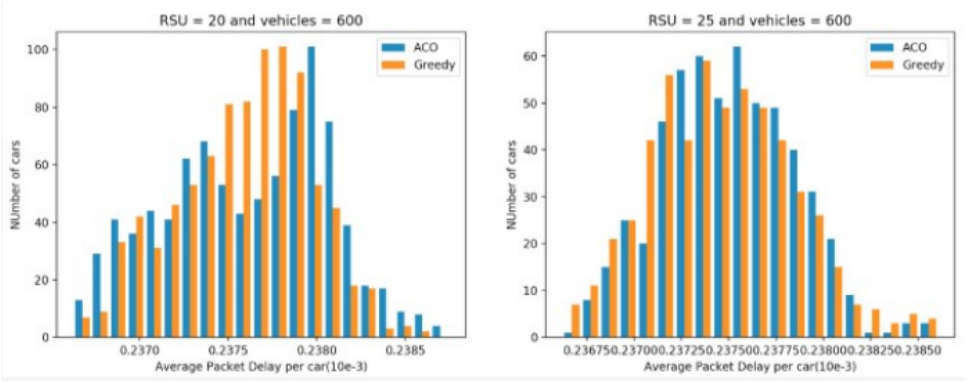
Figure 6: Comparison of coverage values obtained by ACO with greedy algorithm. Highlighted numbers represent cases when coverage obtained by ACO is better than that obtained by greedy method. Numbers in red represent the global best coverage for each budget.

4.5 Traffic Network Simulation

Network simulation was done using OMNeT++ and the traffic was simulated using SUMO. The simulation setup was such that vehicles were run between randomly selected start and end points on a given map. Each vehicle generated message packets after a fixed interval of time. Further, all the messages generated from vehicles were received by RSUs as well as other vehicles in the range of that vehicle. Each message received by an RSU was then duplicated and sent into the channel with all the vehicles i.e. it was broadcast to all the vehicles. However, messages received by vehicles were not duplicated.

Number of RSU	ACO - Packet Delay Mean	ACO - Packet Delay Variance	Greedy - Packet Delay Mean	Greedy - Packet Delay Variance
15	237.61322	2.39	237.61756	1.45
20	237.61485	2.14	237.58360	1.39
25	237.50342	1.18	237.46272	1.43
30	237.50767	1.02	237.49608	1.15

Table 2: Packet delay mean and variance



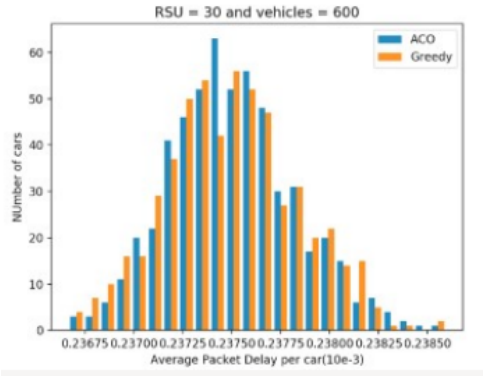


Figure 7: Distribution of packet delay values

Number of RSU	ACO - Packet Loss Mean	ACO - Packet Loss Variance	Greedy - Packet Loss Mean	Greedy - Packet Loss Variance
15	0.576	0.0044	0.565	0.0049
25	0.565	0.0018	0.559	0.0019
30	0.572	0.0024	0.570	0.0018

Table 3: Packet loss mean and variance

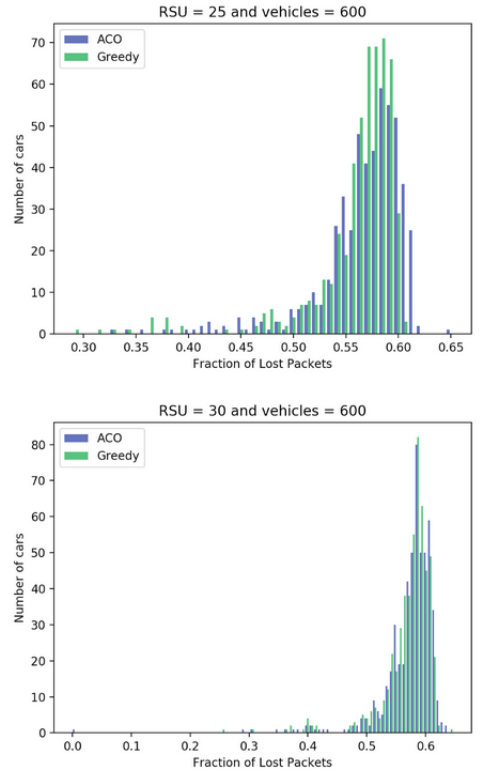


Figure 8: Distribution of packet loss values

We also considered different number of vehicles, but the trend in the graphs obtained were similar. In terms of packet loss and packet delay, ACO and greedy are very close as it can be seen in Figure 9. In fact, in some cases, greedy algorithm even gave better results.

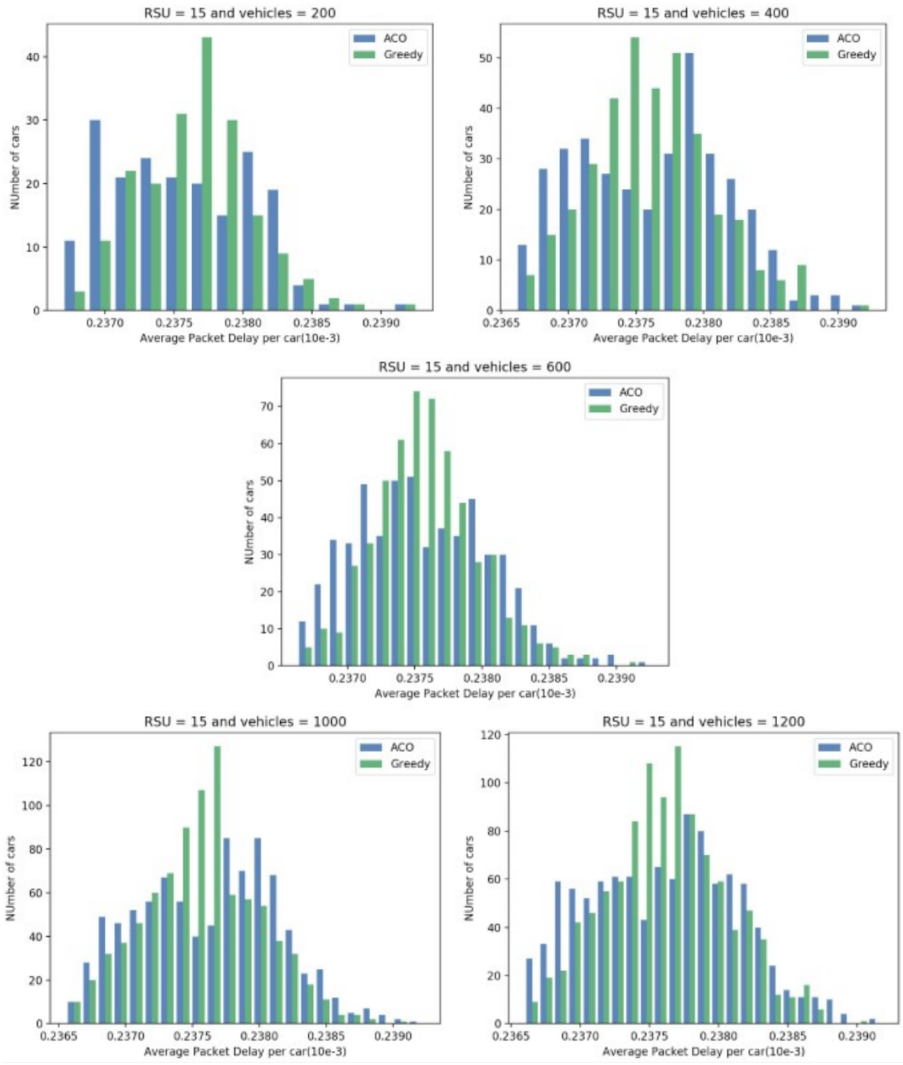


Figure 9: Packet Delay for different number of vehicles

4.6 Heterogeneous RSUs

The cost of RSU deployment varies at different locations, depending on the infrastructure or support for RSU installation. We have summarized the appropriate values in Table 4 and Table 5.

Antenna Brand with Gain	**Wireless Range**	**Cost**
Omni Site Antenna, 6dBi	243 m	$ 121.70
Omni Site Antenna, 9dBi	338 m	$ 139.20
Omni Site Antenna, 12dBi	503 m	$ 227.50

Table 4: Antenna description and prices [6]

The total cost of RSU installation is the sum of antenna cost, RSU cost and deployment cost. For instance, the net installation charge of an RSU with antenna type I is $\$(3900 + 140 + 121.7) = \$4161.7$, of type II is $\$ (3900 + 140 + 139.40) = \$4179.4$ and of type III is $\$ (3900 + 140 + 227.50) = \$4267.5$.

Parameter	Value
RSU Cost	$ 3900
RSU deployment cost at an intersection with traffic lights	$ 140
RSU deployment cost at an intersection without traffic lights	$ 190
RSU deployment cost on a road segment	$ 190

Table 5: RSU Deployment Cost [1]

The data shows that for the values provided to us, the ***effective total budgets for deploying RSUs with different types of antennas are very close, and hence the antenna with maximum coverage gets chosen every time when the algorithm is run***. This converts the scenario with heterogeneous RSUs back to the scenario with homogeneous RSUs having an antenna transmission range equal to the maximum antenna range. Hence, using heterogeneous RSUs did not provide any significant variation in results.

5 Problem Formulation - B

This RSU deployment strategy is designed for various **non-safety VANET applications** such as driving assistance, traffic information collection, and commercial announcements, to cite a few. In this respect, our goal is to maximize the coverage ratio while minimizing the number of used RSUs, which results in minimizing the total cost of RSUs deployment. Enhancing the coverage stands for maximizing the number of moving vehicles that make contact with at least one RSU during their trips in a given city.

The road network in converted into a planar undirected graph with roads as edges and intersections as vertices by writing a parser for the raw OSM file corresponding to the region including the desired roads.

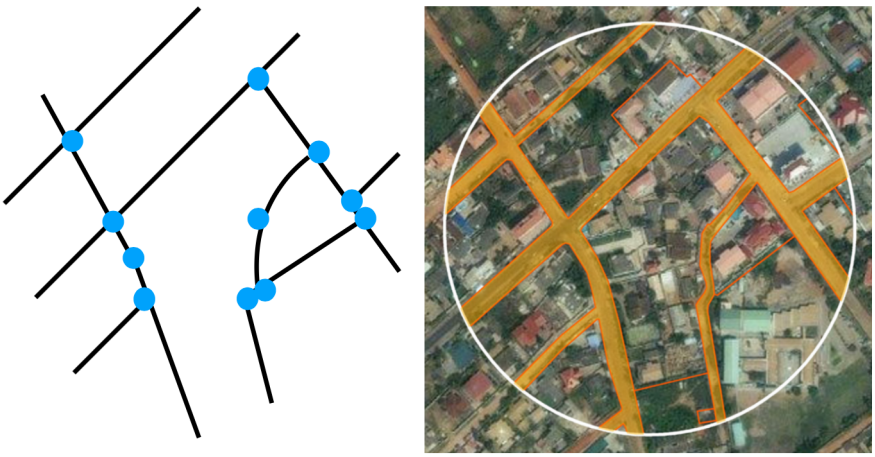


Figure 10: Modelling road network into a graph for using vertex cover

Further, all the degree 0 and degree 1 nodes are simply removed from the graph. For the degree 2 nodes, the two edges emerging from them are merged into one before they are removed. This process is repeated till all the degree 0, 1 or 2 nodes are removed.

$Coverage = V'/V$

V' = Number of vehicles that came in contract with RSU

V = Total number of vehicles in the system

Under this formulation, the maximum coverage is achieved when the RSU placement scheme ensures that the graph has at least one RSU on every possible path. Basically, for a given ***undirected graph*** $G = (V, E)$ ***and a given weighting function defined on the vertex set*** V***, we wish to find a vertex set*** S ***whose total weight is minimized; subject to a constraint that every edge of*** G ***has at least one end point (vertex) in*** S***. This problem is also known as the minimum weighted vertex cover problem or MWVCP.***

5.1 Algorithms & Results

Nodes	Greedy (weight)	Greedy (degree)	Greedy (ratio)	ACO (best)
10	3.02	2.42	**2.02**	2.02
10	3.02	3.05	**2.66**	2.06
30	12.86	**6.33**	6.51	6.29
30	11.04	**6.06**	6.13	6.06
50	22.95	**14.29**	14.33	13.90
50	20.04	19.48	**18.39**	17.57
100	49.05	**21.44**	21.91	21.22
100	49.05	46.42	**44.36**	42.79
100	49.05	46.57	**44.79**	43.9
200	91.3	89.5	**86.50**	86.09
200	91.35	87.36	**85.51**	85.12
300	149.51	145.55	**143.19**	141.28
300	149.50	**70.63**	74.56	70.67
500	245.41	241.04	**236.6**	237.72
500	245.41	**116.03**	120.04	117.19
1000	498.5	495.5	**489.3**	491.4
1000	498.5	**254.8**	259.5	271.5

Figure 11: Comparison of Minimum Weight Vertex Cover for graph with different number of nodes obtained using Greedy vs ACO. Numbers which are bold represent the lowest weight of cover obtained using different kinds of Greedy algorithm. Numbers highlighted with green represent overall best results out of results obtained using both ACO and Greedy algorithms for each case.

We compare ACO with three types of greedy heuristics:

- **Greedy over Degree of Nodes (Greedy_D:** In this approach, we greedily select the node with highest degree and add it to the vertex cover set. We then remove all the edges covered by this node and update the degree of the surrounding nodes before selecting the next unselected node with highest degree. This is done till the vertex cover is achieved.
- **Greedy over Weight of Nodes (Greedy_W):** We simply keep adding the unselected nodes with the lowest weight and add them to the vertex cover set till a vertex cover is achieved.
- **Greedy over the Degree to Weight Ratio of Nodes (Greedy_R):** This is similar to the Greedy_D approach, except that the unselected node with highest degree/weight ratio is selected before updating the degrees of the remaining nodes.

Further, we run the ACO algorithm for a fixed number of times for each graph and the best solution obtained among all these cases has been reported.

We have experimented with graphs from size 10 - 1000 nodes. We have 2 types of graphs for every number of nodes - one contains nodes from zero to very high degrees (i.e. variation in the degree of nodes is very high. Some nodes are densely connected while others are sparsely connected); the second type contains nodes with somewhat similar degrees throughout (i.e. variation in degrees is very less). As an example, the first graph type is somewhat a mixture of fully connected graphs and non-connected nodes (with a few additional edges) and the second type is similar to a tree with few extra edges. Weights per node have been generated completely randomly. The papers we followed have biased the graphs by adding extra weight to the heavily connected nodes so that Greedy_D does not perform well but we have not done that part.

We observe that Greedy_D performs better than Greedy_R when the degrees of the nodes have a lot of variation. However, for nearly similar degrees, greedy over ratio performs better. Further, when weights are randomly generated, ***ACO is able to outperform all the three greedy heuristics described above till up to 300 nodes***.

Several papers show that incorporating a positive correlation between the weight of the node and the degree of the node in the graph can make greedy approaches perform poorly. Hence, in those cases, ACO has been shown to outperform greedy till up to 1000 nodes as well. But, ***we have only considered random weights while documenting results because it is a closer representation of the real world scenario*** as a real world scenario may or may not have the bias in weights that is desired for ACO to perform better than greedy approach.

6 Speedups & Parallelization

We have implemented CPU based parallelization methods to speed up both greedy and ACO based algorithms. We have also used data structures like heaps and lists of lists instead of arrays wherever possible to reduce the searching time complexity as well as overall space complexity in our codes. Note that all the experiments in this section were done on system with Intel® Core™ i5-7200U CPU @ 2.50GHz × 4 Processor.

6.1 Greedy Approach

Greedy algorithm is an algorithmic paradigm mainly used for discrete optimization problems. The basic idea of this algorithm is to iteratively populate the solution space by adding the best known candidate at each step. From the algorithmic viewpoint, the key is the heuristic used for the selection process, which should lead to the desired solution. From the complexity viewpoint, the key is the efficiency of the selection, which should be implemented at the best using optimal memory and the best processing time [23].

For our greedy algorithm code, the implementation is already $O(n^2)$ and straightforward. However, Algorithm 4 and 5 shows us that ***the*** $FindMaxWeightElement()$ ***function can be parallelized directly. This function involves the computation of the*** $CoverageAddition(index)$ ***term for every possible candidate position that has not yet been selected. This value can be computed in parallel for all candidate positions.*** Basically, we have parallelized the code by updating the local maximum of smaller chunks of data in parallel and then taking the global maximum out of them to utilize multiple cores simultaneously.

Number of RSUs	Without parallelization	With parallelization
20	34.53 s	12.92 s
25	43.10 s	18.34 s
30	55.31 s	21.19 s

Table 6: Time taken by parallelized and non-parallelized Greedy Algorithm to determine candidate positions for placement of N RSUs.

We were able to achieve a ***speedup of nearly 2.5 - 3 times using 4 cores*** with respect to the time complexity of the original code.

6.2 Ant Colony Optimization

We studied multiple papers [10, 5, 19, 20] to conclude that the implementation strategies for parallelized ACO on multiple cores of a CPU follow two broad approaches:

- **Parallel Ants:** This approach maintains a master thread to coordinate the best solution out of local searches and update trails accordingly. Ants are run on multiple threads (called the slave threads) and they update the main thread about their solution obtained [10]. ***Our current code already implements this strategy***. Parallelizing the $UpdateTrails$ step (if possible) and skip iterations while updating the trails matrix with respect to the best solution [5] are two further improvement strategies that we found. However, parallelizing the $UpdateTrails$ step is not possible in our code because all the ants write their values into a single global Trail Matrix. Also, the $MoveAnts$ function for our code is far costlier in terms of time complexity (approx. $O(n^3)$ per ant) than updating the Trail Matrix (approx. $O(n)$ per ant). Hence, skipping the crucial step of updating the Trail Matrix after some iterations will provide negligible performance benefits.

- **Multiple Ant Colonies:** Alternatively, we can create multiple instances of Ant Colony Systems on different cores (using same or even different hyper parameter

values). We either do not exchange any information among these colonies and simply take the best solution out of them after k steps, or employ an information exchange strategy to improve solutions in each system [19]. Some information exchange strategies include: Exchanging of global best solution, exchanging the trail values, exchange of locally best solutions in certain pre-defined order, exchanging the values of transition probabilities between colonies, etc. Some papers also experiment with swapping the complete pheromone matrices or ant trails between the ant colonies running on different cores. However, as per the results of [20], we found that ***if the required solution quality is high (our focus is on accuracy), then information exchange between the colonies may be useful but otherwise no information exchange tends to perform better***. Hence, we have implemented independent ant colony systems without information exchange in our code.

To summarize, we have employed multiple ants following a master-slave thread based parallel ant method (as described in the first point) with no information exchange between them. Instead we run independent ant colony systems on different cores with varied hyper parameters (as described in the second point) and select the best solution out of these independent colonies.

Further, we implemented a technique called ***SEE (suspicious element exclusion)*** [13]. The intuition behind this technique is to remove the old nodes from the ant trails if they do not contribute significantly to the solution generated later. This can happen when the new nodes added to the ant trails have a high overlap with the old nodes. This method can lead to faster convergence as well as a better solution as compared to the version of ACO that does not remove any nodes from an ant trail once they are selected. ***All the details regarding the implementation of SSE on top of ACO are provided in [13].*** We have used the exact same formulation and hyper parameter values as mentioned in the referenced paper as they were giving good results in over our data as well. Further, we have removed functions performing repetitive calculations and incorporated optimized data structures, min/max heaps, parallel cores, etc. in our code to make it both space and time efficient.

Number of Nodes	ACO (weight of the vertex cover)	ACO (iterations used)	ACO + SEE (weight of the vertex cover)	ACO + SEE (iterations used)
100	43.28	291	42.79	161
200	86.92	308	86.11	219
300	142.09	337	141.28	275

Table 7: Final weight of vertex cover set and number of iterations required to obtain that solution for ACO with and without SEE when solving problem formulation B. ACO + SEE requires about two-third of the iterations required by ACO alone to reach a better solution. Since ACO is extremely randomized, the output values tend to be fluctuating. Hence, we have reported the best values out of 3 independent runs of the algorithm in every case.

Using SSE, we were able to attain a ***speedup of nearly 1.5 - 2 times*** over ACO alone with similar accuracies.

7 Conclusion

We have experimented with multiple formulations and nature-inspired algorithms for improving the current RSU placement approaches that mostly employ greedy heuristics and dynamic programming based methods. We have compared our results with degree centrality [24], which has a similar scenario and outperformed it. We have further compared the results obtained using the Greedy algorithm with those obtained using ACO. Although ACO outperforms greedy marginally in medium-sized maps, greedy gives better results with larger maps. We also proposed another formulation for the RSU placement problem, which makes it equivalent to the Minimum Weight Vertex Cover Problem. In this scenario again, ACO marginally outperforms greedy in maps of not very large size (no. of nodes 300). We have also improved the run time of ACO and Greedy using parallelization and other techniques.

Bibliography

[1] AliExpress. *VivienneWang*. `https://www.aliexpress.com/item/32852825724.html`.

[2] Baber Aslam, Faisal Amjad, and Cliff C Zou. Optimal roadside units placement in urban areas for vehicular networks. In *2012 IEEE Symposium on Computers and Communications (ISCC)*, pages 000423–000429. IEEE, 2012.

[3] Baber Aslam and Cliff C Zou. Optimal roadside units placement along highways. In *2011 IEEE Consumer Communications and Networking Conference (CCNC)*, pages 814–815. IEEE, 2011.

[4] Seif Ben Chaabene, Taoufik Yeferny, and Sadok Ben Yahia. A roadside unit deployment framework for enhancing transportation services in maghrebian cities. *Concurrency and Computation: Practice and Experience*, page e5611, 2019.

[5] Bernd Bullnheimer, Gabriele Kotsis, and Christine Strauß. Parallelization strategies for the ant system. In *High Performance Algorithms and Software in Nonlinear Optimization*, pages 87–100. Springer, 1998.

[6] Cetacea. *DSRC Intelligent Transportation*. `http://shop.cetacea.com/5-9-GHz-V2V-V2I-s/1181.htm`.

[7] Seif Ben Chaabene, Taoufik Yeferny, and Sadok Ben Yahia. A roadside unit placement scheme for vehicular ad-hoc networks. In *International Conference on Advanced Information Networking and Applications*, pages 619–630. Springer, 2019.

[8] Jeonghee Chi, Yeongwon Jo, Hyunsun Park, and Soyoung Park. Intersection-priority based optimal rsu allocation for vanet. In *2013 Fifth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 350–355. IEEE, 2013.

[9] Hamid Reza Eftekhari, A Jalaeian Bashirzadeh, and Mehdi Ghatee. Binary programming model to optimize rsu placement for information dissemination. In *2015 International Conference on Connected Vehicles and Expo (ICCVE)*, pages 112–115. IEEE, 2015.

[10] Issmail Ellabib, Paul Calamai, and Otman Basir. Exchange strategies for multiple ant colony system. *Information Sciences*, 177(5):1248–1264, 2007.

[11] Abderrahim Guerna and Salim Bitam. Gica: An evolutionary strategy for roadside units deployment in vehicular networks. In *2019 International Conference on Networking and Advanced Systems (ICNAS)*, pages 1–6. IEEE, 2019.

[12] Shiow-Fen Hwang, Wei-Chen Chen, Chyi-Ren Dow, and Nhut-Lam Nguyen. Efficient rsu placement schemes in urban vehicular ad hoc networks. *Journal of Information Science & Engineering*, 35(5), 2019.

[13] Raka Jovanovic and Milan Tuba. An ant colony optimization algorithm with improved pheromone correction strategy for the minimum weight vertex cover problem. *Applied Soft Computing*, 11(8):5360 – 5366, 2011.

[14] Iztok Fister Jr., Xin-She Yang, Iztok Fister, Janez Brest, and Dusan Fister. A brief review of nature-inspired algorithms for optimization. *CoRR*, abs/1307.4186, 2013.

[15] Yingsi Liang, Hui Liu, and Dinesh Rajan. Optimal placement and configuration of roadside units in vehicular networks. In *2012 IEEE 75th Vehicular Technology Conference (VTC Spring)*, pages 1–6. IEEE, 2012.

[16] Po-Chiang Lin. Optimal roadside unit deployment in vehicle-to-infrastructure communications. In *2012 12th International Conference on ITS Telecommunications*, pages 796–800. IEEE, 2012.

[17] Xu Liya, Huang Chuanhe, Li Peng, and Zhu Junyu. A randomized algorithm for roadside units placement in vehicular ad hoc network. In *2013 IEEE 9th International Conference on Mobile Ad-hoc and Sensor Networks*, pages 193–197. IEEE, 2013.

[18] Renzo Massobrio, Santiago Bertinat, Sergio Nesmachnow, Jamal Toutouh, and Enrique Alba. Smart placement of rsu for vehicular networks using multiobjective evolutionary algorithms. In *2015 Latin America Congress on Computational Intelligence (LA-CCI)*, pages 1–6. IEEE, 2015.

[19] Martin Middendorf, Frank Reischle, and Hartmut Schmeck. Multi colony ant algorithms. *Journal of Heuristics*, 8(3):305–320, 2002.

[20] Martin Middendorf, Frank Reischle, and Hartmut Schmeck. Multi colony ant algorithms. *Journal of Heuristics*, 8:305–320, 01 2002.

[21] Prithviraj Patil and Aniruddha Gokhale. Maximizing vehicular network connectivity through an effective placement of road side units using voronoi diagrams. In *2012 IEEE 13th international conference on mobile data management*, pages 274–275. IEEE, 2012.

[22] Matheus Ferraroni Sanches, Allan M de Souza, Wellington Lobato, and Leandro A Villas. Optimizing infrastructure placement with genetic algorithm: A traffic management use case. In *2019 IEEE Latin-American Conference on Communications (LATINCOM)*, pages 1–6. IEEE, 2019.

[23] Claude Tadonki. Openmp parallelization of dynamic programming and greedy algorithms. In *Distributed, Parallel, and Cluster Computing*. arXiv, 2020.

[24] Zhenyu Wang, Jun Zheng, Yuying Wu, and Nathalie Mitton. A centrality-based rsu deployment approach for vehicular ad hoc networks. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–5. IEEE, 2017.

[25] Tsung-Jung Wu, Wanjiun Liao, and Chung-Ju Chang. A cost-effective strategy for road-side unit placement in vehicular networks. *IEEE Transactions on Communications*, 60(8):2295–2303, 2012.

[26] Yongping Xiong, Jian Ma, Wendong Wang, and Jianwei Niu. Optimal roadside gateway deployment for vanets. *Przeglad Elecktrotechnicznyt*, 88(7B):273–276, 2012.

[27] Taoufik Yeferny and Sabri Allani. Mpc: A rsus deployment strategy for vanet. *International Journal of Communication Systems*, 31(12):e3712, 2018.