

# Bosch Route Optimization Challenge

[Abstract](#)

## Problem Statement

Design an efficient employee shuttle service for a company of 1000 employees.

The objectives of the challenge are:

1. Minimize the number of busses required
2. Minimize the operational and fuel costs
3. Employee satisfaction is of prime importance
4. Improve Bus usage efficiency

## Solution

### Reading Input

The Input provided had a list of the employees interested in availing the campus shuttle service.

### Challenge

1. Random Address: 11/1, Opp Ram Lila Gate No 1, Geeta Colony Some address were difficult to geocode.
2. API Limits for Geocoding

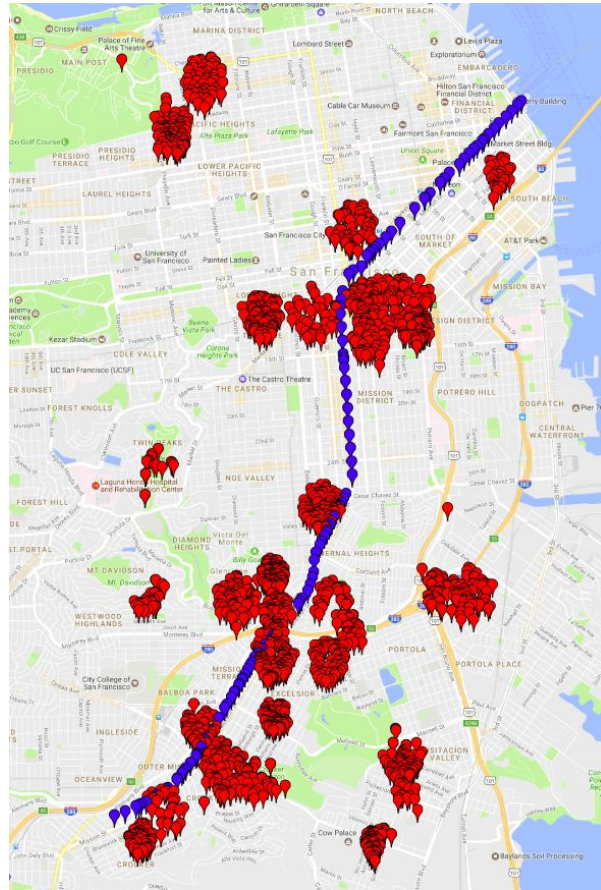
### Solution

1. Scraped the list of Localities and landmarks of Bangalore and its neighbours from this site <http://www.nivalink.com/localities-and-landmarks/bangalore>. For address which failed the geocoding process, checked whether they contained the above localities or landmarks. If so then replaced them with the geocodes of these localities. Else removed them
2. Used Try/Catch Statements with many API credentials

Bangalore Neighbourhood Data File:

<https://www.kaggle.com/rmenon1998/bangalore-neighborhoods>

## Sample Data



As per our analysis, the data was concentrated in a few parts of Bangalore and also contained a few outliers of the employees who were coming from the outer region surrounding Bangalore.

## Pre-Processing

Given a network  $N$  composed of a set of roads  $R$ ,  $R = \{1, 2, \dots, r\}$ , a set of employees  $E$ ,  $E = \{1, 2, \dots, i\}$ , a set of equally spaced points  $P$ ,  $P = \{1, 2, \dots, j\}$  for installing bus stops and a maximal home-to-bus-stop walking distance  $\lambda$ , our goal is to find the minimal set of bus stops  $B$ ,  $B \subseteq P$ , which is able to attend all employee.

To reduce the operational cost we decided to create and assign bus stations to employees in order to reduce the number of data points to the VRP solver.

## Clustering

The **DBSCAN (Density-based spatial clustering of applications with noise)** algorithm was used to cluster employee into groups. The key idea is that for each point of a cluster, the neighbourhood of a given radius has to contain at least a minimum number of points.

DBSCAN Algorithm requires two parameters:

1. **EPS (epsilon):** It defines the neighbourhood around a data point i.e. if the distance between two points is lower or equal to 'eps' then they are considered as neighbours. If the eps value is chosen too small then a large part of the data will be considered as outliers. If it is chosen very large then the clusters will merge and the majority of the data points will be in the same clusters. *One way to find the eps value is based on the k-distance graph.*
2. **Min-Points:** Minimum number of neighbours (data points) within eps radius. Larger the dataset, the larger value of Min-Points must be chosen. *As a general rule, the minimum MinPoints can be derived from the number of dimensions D in the dataset as,  $\text{MinPts} \geq D+1$ . The minimum value of MinPoints must be chosen at least 3.*

In this algorithm, we have 3 types of data points.

1. **Core Point:** A point is a core point if it has more than Min Points points within epsilon.
2. **Border Point:** A point which has fewer than Min Points within epsilon but it is in the neighbourhood of a core point.
3. **Noise or outlier:** A point which is not a core point or border point.

Pseudo Code for DBSCAN is Attached below:

The implementation of DBSCAN was taken from a research paper which used a QuadTree for implementation of DBSCAN. <https://github.com/Sentimentron/DBSCAN-Cluster>

We used the Haversine Distances for the calculation of the clusters.

```

DBSCAN(D, eps, MinPts)
  C = 0
  for each unvisited point P in dataset D
    mark P as visited
    NeighborPts = regionQuery(P, eps)
    if sizeof(NeighborPts) < MinPts
      mark P as NOISE
    else
      C = next cluster
      expandCluster(P, NeighborPts, C, eps, MinPts)

expandCluster(P, NeighborPts, C, eps, MinPts)
  add P to cluster C
  for each point P' in NeighborPts
    if P' is not visited
      mark P' as visited
      NeighborPts' = regionQuery(P', eps)
      if sizeof(NeighborPts') >= MinPts
        NeighborPts = NeighborPts joined with NeighborPts'
    if P' is not yet member of any cluster
      add P' to cluster C

regionQuery(P, eps)
  return all points within P's eps-neighborhood (including P)

```

### Why DBSCAN?

1. It could find Clusters of Arbitrary shapes
2. It was not affected by outliers and noise
3. We did not know the number of clusters that existed

### Why Not KMEANS?

1. K-Means clustering fails when the data is not spherical i.e. when the data doesn't have the same variation all around.
2. Cannot handle outliers and noise.

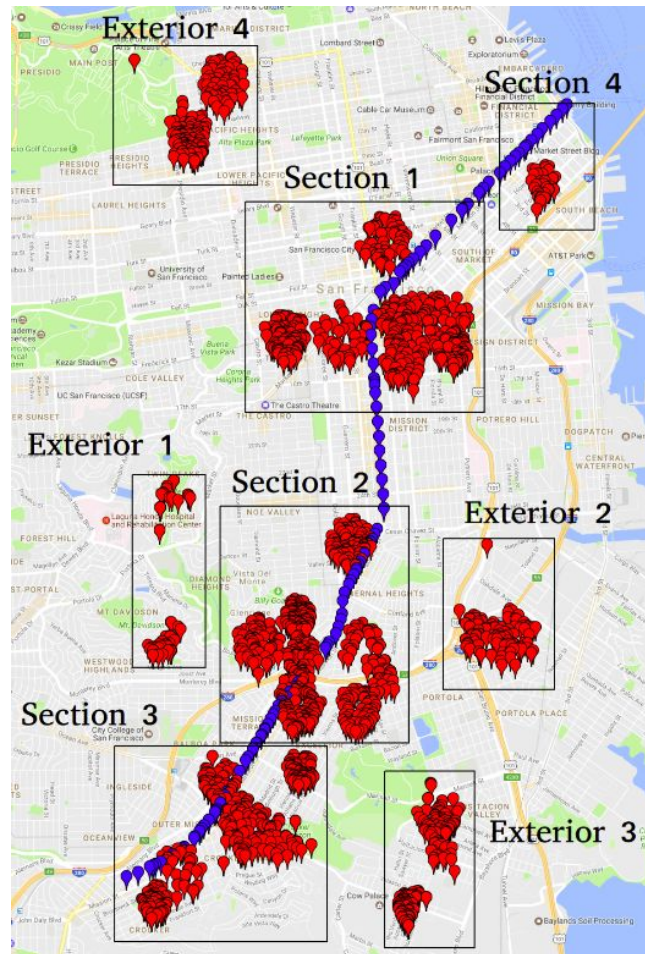
### Challenges

1. Deciding the correct value of the Epsilon and the Min Points for the results

### Solution

1. We did it by running the DBSCAN algorithm on the example data set provided of 50 points and chose arbitrary values of

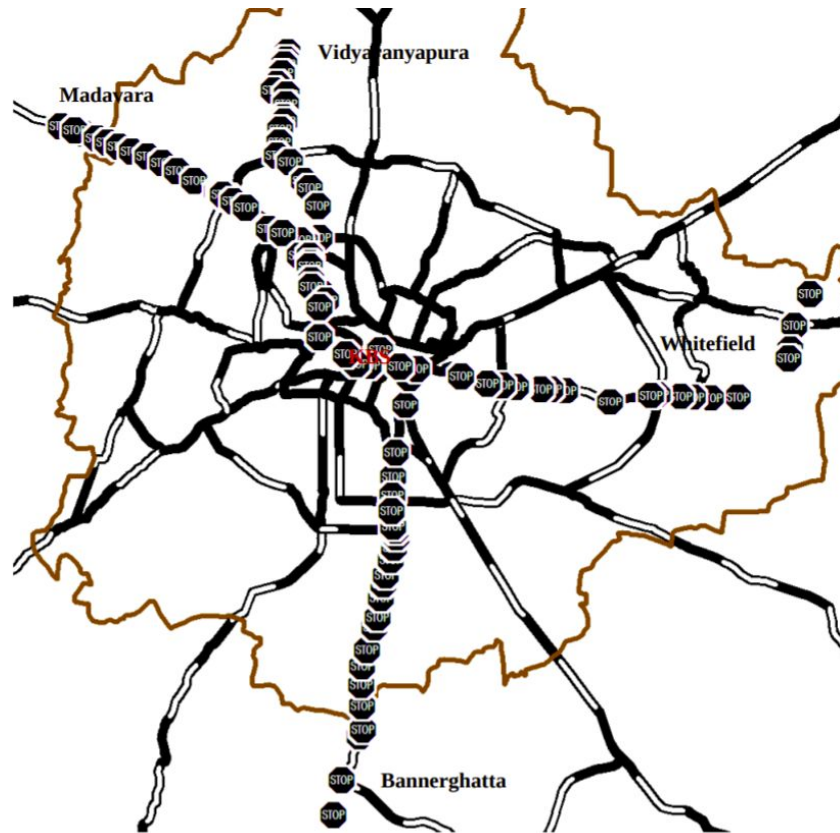
## Clustering Results



All the employee points were classified into sections and exteriors

1. Sections were the regions where the points were very densely populated.
2. Exteriors consisted of the noise or outliers that were single employees and took more than 1km to reach the nearest bus stop.

## Bus Stops in Bangalore



**This is the Bus stops map of Bangalore**

We decided to tackle both these groups separately

1. For each of the sections,
  - a. Get the List of Bus stops in that section
  - b. We then generated combinations of 2, 3 and 4 Bus stops
  - c. Then the Haversine distance of each employee from the nearest bus stop was calculated and these distance were summed to get the total distance for the bus stop.
  - d. Finally, the best combination was chosen for each Section
2. For the Exteriors, we suggested alternate methods or public transportation to help the employees to get to the nearest bus station.



## Vehicle Routing Problem

### Why Routing?

On close analysis of the data, we can see that the clusters do not contain an equal amount of people. Some clusters (near the centre) contain about 110 employees while the exteriors contain only about 20 employees.

Hence for the efficient usage of the busses routing algorithm has to be used.

### Why Genetic Algorithms?

1. On testing, we discovered that it got better results as compared to linear programming for large datasets
2. It could be easily hybridised with other local search algorithms.
3. It was easy to implement and experiment with Solomons benchmark instances

## **SOLVED AS A CAPACITATED VEHICLE ROUTING PROBLEM**

### Representation

The Representation of Solutions i.e. chromosomes is in the form of a set of tours that visit the employees. Two situations may arise:

1. The Number of Employees of a particular tour may exceed the capacity of the bus
2. The Number of Employees of a particular tour may fall below 85% occupancy of a bus

Both of the above cases make the solution infeasible. So we used a penalty term for such solutions that affect their fitness values rather than discarding the solution. This helps in better variations and the generation of potentially better offsprings.

### Initial Population

For creating an initial solution We choose a random node from the set of unserved nodes and perform Nearest Neighbour Greedy search on this node until the capacity of the bus is not violated. This generated route is passed through a 2-OPT operator. The process is repeated for generating the complete solution until all the nodes are serviced.

We generate a pool of such solutions using the Greedy Method and this becomes our Initial Population



### Calculation of Fitness

Fitness was calculated as a combination of the chromosome cost and its penalty. The penalty was calculated as follows :

$$\sum \text{Math.max}(0, \text{route.demand} - \text{bus.capacity})$$

**The cost of a chromosome is simply the sum total of the distance travelled by each of its routes.**

### Selection

Roulette Wheel Selection was used for the selection of the Mating parents. Fitness values were rounded off in the form of probabilities.

### Crossover

Partially Matched Crossover was used for performing the Crossover

PMX is similar to Two Point Crossover, A substring is swapped like in two-point crossover and values are retained in all other non-conflicting places. Next, the conflicting points are replaced by the values which replaced them.

Example :

P1 ( 1 2 3 4 5 6 7 8 9 )


P2 ( 4 5 2 1 8 7 6 9 3 )

Suppose we choose the indices as mentioned in blue :

Next, we get

O1 ( \* 2 3 | 1 8 7 6 | \* 9 )

O2 ( \* \* 2 | 4 5 6 7 | 9 3 )



Finally for filling the conflicting positions

O1 ( 4 2 3 | 1 8 7 6 | 5 9 )

O2 ( 1 8 2 | 4 5 6 7 | 9 3 )

### Mutation

The Inversion Operator was chosen for performing Mutation which inverses the nodes in between two random points chosen in an individual solution.