Sterlite Tech

Delivering
Smarter Networks

# Research Paper

## Cell Site Optimization

# Table of Contents

CONFIDENTIALITY CLAUSE 2

ALL RIGHTS RESERVED 2

TRADEMARKS 2

TABLE OF CONTENTS 3

1. INTRODUCTION 4

2. ALGORITHMS AND METHODS

   A. GENETIC ALGORITHM
   B. CROSSOVER
   C. VALUE ADDED MUTATION
   D. MISCELLANEOUS SHAPE'S AREA CALCULATION USING BOX METHOD

3. CELL SITE OPTIMIZATION USING GENETIC ALGORITHM

   A. TARGET AREA
   B. RADIUS OF BASE STATIONS
   C. COORDINATES OF INITIAL BASE STATIONS
   D. ACTUAL AREA COVERED BY BS
   E. NUMBER OF BS REQUIRED
   F. GENETIC ALGORITHM(GA)

4. STEPS OF GA

   A. SELECTION
   B. CROSSOVER
   C. MUTATION
   D. UPDATE
   E. TERMINATION CHECK

5. RESULT

6. LIMITATIONS AND POSSIBLE SOLUTIONS

Milin Dholakia Keval Pambhar Namas Mankad

**Abstract :**

In the telecom market, with increase in competency, the companies constantly aim at increasing and retaining their current user base for which they strive to improve their Quality of Service and Coverage while keeping the cost as low as possible. In the order to achieve this, the first step is to establish a well connected network of Base stations (BS) that ensure connectivity and quality for the users. But we often face a problem in which there are more than required BSs in a particular area. Hence, this paper is an attempt to solve this problem by calculating and determining the minimum number of BS to keep out of the existing ones without losing any coverage. The methods of genetic algorithm have been used. We have also covered the limitations and how we can attempt to solve them in the future.

## 1. Introduction :

In the telecom market the first most priority of a company is to have a well established network of BSs in order to provide maximum coverage and optimum quality of service(QoS) to its users. The companies install the BSs in the target area while aiming at achieving maximum coverage and optimum quality of service. They keep in mind the population density and BS capacity while installing the towers(BSs). The companies usually have set budgets to establish the network. But after installing the BSs we can determine the actual amount of BSs required to achieve the same coverage while providing the same QoS to its users. Hence the problem of network optimization comes in where we have to identify the actual amount of BSs required out of the initially installed BSs, without compromising the QoS and coverage to a large extent. We have to find that point where in we get a significant amount of optimization(much fewer towers) for little to no compromise in QoS and coverage of that area. We attempt to solve this problem by using genetic algorithm and box method to identify the coverage area for each configuration of BSs(configuration of k towers that are selected out of the initial amount.)

## 2. Algorithms and methods used :

A. Genetic algorithm :

The process of genetic algorithm has been inspired from the process of biological evolution :

1. initialize n number of chromosomes
    a. We have each chromosome as random selection of k BSs from the 50 existing BSs.
    b. These set of chromosomes are our initial population.

2. define a fitness function - fitness function determines the 'fit' of the chromosomes and decides on the complexity, accuracy and efficiency of the whole algorithm.

3. Calculate initial fitness of the population

4. Repeat the steps for a large number of 'generations' (say 5000,50000 etc...)
    a. Calculate fitness of the initial population.
    b. selection of two chromosomes
    c. crossover of these two chromosomes
    d. mutation of the resultant chromosomes of step b.
    e. calculate fitness of new offsprings yielded after mutation
    f. replace the 'bad' chromosomes from initial population (if any)
        i. bad chromosomes are the one with lower fitness value than the offsprings
5. termination criteria :
    a. It can be such that the step 4 repeats until the best chromosome in a particular generation population has a desired % of fitness. Sometimes we don’t know the % fitness required or whether it is achievable or not, we use a fix, very large number of generations

B. N-point crossover(probability of crossover = 0.9)

Consider two chromosomes A and B each with k = 10 number of genes :

A = [(1,2),(3,2),(5,4),(6,5),(8,7),(1,9),(3,3),(5,4),(11,7),(13,5)]

B = [(8,3),(7,2),(6,1),(4,9),(7,7),(2,2),(3,4),(2,7),(12,1),(11,3)]

1. Create crossover array
    a. Determine length of crossover array
        i. Length L = k/2
            1. If L is odd, L = L + 1
            2. If L is even, no change
        ii. For eg, k = 10, L = 6

2. Create the crossover array of length 'L=6' where each value is randomly generated in the range of (1 to k):
    a. Crossover array = [2,6,4,8,1,9]
    b. Sort the array = [1,2,4,6,8,9]

3. Perform crossover as follows :
    a. For each consecutive pair of sorted crossover array, we swap the values in the selected chromosomes A and B
        i. So index 1 to 2 is swapped in A and B
        ii. Index 4 to 6 is swapped in A and B
        iii. Index 8 to 9 is swapped in A and B

4. We get final A and B as:
    a. A = [(8,3),(7,2),(5,4),(4,9),(7,7),(2,2),(3,3),(2,7),(12,1),(13,5)]
    b. B = [(1,2),(3,2),(6,1),(6,5),(8,7),(1,9),(3,4),(5,4),(11,7),(11,3)]

C. Value added mutation(probability of mutation = 0.1)

Consider the above two chromosomes A and B (k = 10), that we yield after crossover :

A = [(8,3),(7,2),(5,4),(4,9),(7,7),(2,2),(3,3),(2,7),(12,1),(13,5)]
B = [(1,2),(3,2),(6,1),(6,5),(8,7),(1,9),(3,4),(5,4),(11,7),(11,3)]

1. Randomly generate the 'index' at which to perform mutation inside the chromosome.
    a. Index = random(between 1 and 10)
        i. Index = 3

2. Randomly generate a number to add or subtract to the value at that index of the chromosome.
    a. Random value = random(between 0.0 and 2.0)
        i. Random value = 0.7

3. Randomly decide to whether add / subtract the ‘random value’
    a. A[index] = A[index] + random value

4. We get final chromosome A and B as:
    a. A = [(8,3),(7,2),(5,4),(4.7,9.7),(7,7),(2,2),(3,3),(2,7),(12,1),(13,5)]
    b. B = [(1,2),(3,2),(6,1),(6.7,5.7),(8,7),(1,9),(3,4),(5,4),(11,7),(11,3)]

The mutation can be carried out on a single chromosome with different ‘random values’

D. Miscellaneous shape’s area calculation using box method

1. Consider a rectangular/square box around any shape for which area is to be calculated.
2. Plot large number of random points in the whole box
3. Count the number of points inside the shape
4. Area = (count / 1000000) * area of the box

Higher are the number of random points considered, higher is the accuracy of the value of the area.

## 3. Cell Site optimization using Genetic Algorithm :

### 3.1 Target Area :

The first step was to consider a target area and a set of BSs established in that area.

- the target area taken is a 100x100 unit^2 geographical area with 50 BSs(x[50]) established as shown in [ ].
- The figure shows the target region in eucildean axes.
- we also store the target area for future use as, target area = 100*100 = 10000.

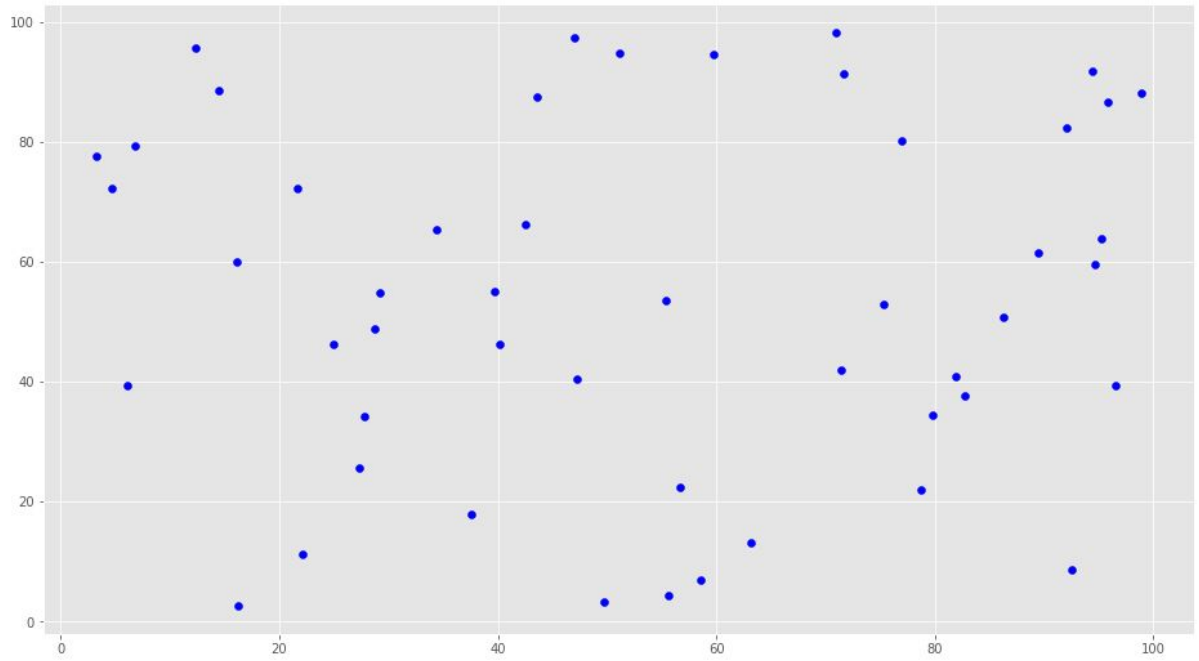
### 3.2 Radius of BSs :

- we consider an urban-suburban setting with not too sparse and not too crowded population density.
- hence we consider the radius of each tower to be constant.
- We take BS's radius r = 10 units

3.3 The coordinates of the 50 existing BSs :

The points have been plotted on a 2-D plane of 100x100.

| | | | | |
|---|---|---|---|---|
| [ 43.60193044, 87.39491363] | [ 81.9290237 , 40.83102805] | [78.76914877, 22.02647689] | [ 16.16441515, 59.99510453] | [ 40.19178726, 46.17210731] |
| [ 71.0014435 , 98.25126363] | [ 47.07314995, 97.30881677] | [92.02520728, 82.41281236] | [34.38713063, 65.35574296] | [ 71.61506076, 91.28754073] |
| [ 49.67114135, 3.24704176] | [28.73282417, 48.74153125] | [ 27.2884754 , 25.6839164 ] | [ 27.7768188 , 34.33790469] | [ 86.34737891, 50.7673117 ] |
| [ 16.21722215, 2.77652374] | [79.80309661, 34.46344822] | [ 59.7054293 , 94.47639451] | [ 56.65731173, 22.37289703] | [ 24.98416091, 46.30324155] |
| [ 51.11503537, 94.6759556 ] | [ 6.10970525, 39.28283245] | [ 12.3897041 , 95.72728891] | [47.22829451, 40.44405577] | [ 71.45362931, 41.99839814] |
| [ 4.67247945, 72.17824552] | [94.45203128, 91.72052527] | [ 58.55608459, 6.97829599] | [96.60504031, 39.47552244] | [ 6.81889906, 79.34774706] |
| [ 82.80019815, 37.63289182] | [ 14.40943535, 88.64701602] | [ 3.22133448, 77.5847233 ] | [ 55.42796483, 53.45328774 | [ 42.47549373, 66.21943596] |
| [ 94.67096157, 59.67520035] | [ 37.55122269, 17.8268646 ] | [63.22244682, 13.2989283 ] | [ 89.46209971, 61.44582705] | [ 55.68177263, 4.35596846] |
| [ 95.27287005, 63.76451511] | [ 76.97115462, 80.19381054] | [ 75.29586815, 52.90407055] | [ 22.16823961, 11.28230332] | [ 29.23945654, 54.79851388] |
| [ 39.72825814, 55.03739796] | [ 21.62869319, 72.19709092] | [ 92.60020217, 8.64174249] | [ 95.89670462, 86.56486252] | [ 98.94248385, 88.11227138] |




The above image represents the initial 50 BSs with their position.

### 3.4 Actual area covered by the BSs

It is of no use for us to consider the area that the current 50 BSs are not covering. The first step would be to see how much area out of the 100x100 the already existing towers cover. To determine this we use a box method. The algorithm of box method would be as follows:

1. consider a circle of radius r = 10 around each data point(already existing BSs)

2. plot 1000000 random non-overlapping points(x_random[1000000]) on the whole target area of 100x100

3. generate a distance matrix of [1000000 x 50] where it stores each random point's[x_random] distance to each data point[x]
    a. matrix1 = use scipy.spatial.distance.cdist(x_random,x,metric = 'euclidean')

4. //count the number of those random data points inside the circles

5. for each random point generated[x_random[i]]:
    a. find the data point[x] closest to the random data point x_random[i]
    b. index = np.argmin(matrix1[i])
    c. //calculate distance between the random point and that data point

d. dist = np.linalg.norm(x[index] - x_random[i]) (//euclidean distance)
e. if dist < radius(10)
Count++ //random point falls into the circle
Else
//do nothing

- Count = 788723. Therefore, 788723 number of random points lie inside the circles assumed around each BS.

- actual coverage = (count / 1000000) * target area = (788723 / 1000000) * 10000
= 7887.23 unit^2

- actual coverage = 7887.23 (out of 10000)

3.5 Number of BSs required

We now have to find the number of minimum BSs(k) required to cover the 'actual coverage' area. We do this simply by dividing the actual coverage by the area of one BS. Therefore, we get
k = number of BSs
k = actual coverage / math.pi*radius*radius
k = 7887.23 / 3.14*10*10
k = 21.27
we round it off to k = 21, since number of clusters has to be a whole number.

3.6 Genetic Algorithm

Now we apply genetic algorithm to determine those k BSs that we need to keep out of the 50 total by the algorithm of genetic algorithm :

1. We define a initial population of n chromosomes (here, n = 500), where each chromosome is an array of k genes where each gene is a BS coordinate and is randomly selected from x[50]. Hence, each chromosome is a randomly selected array/set of k BS points. We define n = 500 such chromosomes so as to cover as many as combinations possible. each combination is the random selection of 21 BSs out of the 50 present.

a. The chromosomes randomly formed in our run are:

[[[ 96.60504031, 39.47552244],
[ 63.22244682, 13.2989283 ],
[ 3.22133448, 77.5847233 ],
..., (k genes/coordinates)
[ 16.16441515, 59.99510453],
[ 43.60193044, 87.39491363],
[ 95.89670462, 86.56486252]],

[[ 71.0014435 , 98.25126363],
[ 16.21722215, 2.77652374],
[ 28.73282417, 48.74153125],
..., (k genes/coordinates)
[ 92.02520728, 82.41281236],
[ 96.60504031, 39.47552244],
[ 94.67096157, 59.67520035]],

[[ 71.0014435 , 98.25126363],
[ 4.67247945, 72.17824552],
[ 79.80309661, 34.46344822],
..., (k genes/coordinates)
[ 96.60504031, 39.47552244],
[ 51.11503537, 94.6759556 ],
[ 21.62869319, 72.19709092]],

……..., (500 such chromosomes)

[[ 94.67096157, 59.67520035],
[ 47.22829451, 40.44405577],
[ 95.27287005, 63.76451511],
..., (k genes/coordinates)
[ 71.0014435 , 98.25126363],
[ 98.94248385, 88.11227138],
[ 14.40943535, 88.64701602]],

[[ 16.16441515, 59.99510453],
[ 71.0014435 , 98.25126363],
[ 56.65731173, 22.37289703],
..., (k genes/coordinates)
[ 40.19178726, 46.17210731],
[ 39.72825814, 55.03739796],
[ 92.02520728, 82.41281236]],

[[ 24.98416091, 46.30324155],

[ 95.27287005, 63.76451511],
[ 98.94248385, 88.11227138],
..., (k genes/coordinates)
[ 39.72825814, 55.03739796],
[ 94.67096157, 59.67520035],
[ 81.9290237 , 40.83102805]]]

2. Most important step is to define a fitness function since it decides the complexity, efficiency and accuracy of the algorithm. Fitness values correspond to chromosomes. To calculate fitness of a chromosome[size k] :

    a. We calculate the 'area of coverage of the chromosome' by the box method demonstrated above.
        i. Consider a circular area of 10 unit radius around each gene of the chromosome.
        ii. Plot 1000000 random data points(x_random) in the whole target area of 100x100 unit^2.
        iii. generate a distance matrix of [1000000 x 50] where it stores each random point's(x_random) distance to each gene.
            1. Matrix1=scipy.spatial.distance.cdist(x_random,chromosome,metric = 'euclidean')
        iv. //count the number of those random data points inside the circles
        v. for each random point generated[x_random[i]]:
            1. //find the gene closest to the random data point x_random[i]
            2. index = np.argmin(matrix1[i])
            3. //calculate distance between the random point and that gene
            4. dist = np.linalg.norm(x[index] - x_random[i]) (//euclidean distance)
            5. if dist < radius(10)
                Count++ //random point falls into the circle
                Else
                //do nothing
        vi. Area of coverage by the chromosome = (count / 1000000) * total area

    b. Return fitness = area of coverage by the chromosome / actual coverage

3. now calculate the initial fitness of all the chromosomes in the initial population. (initial population = initial 500 chromosomes)

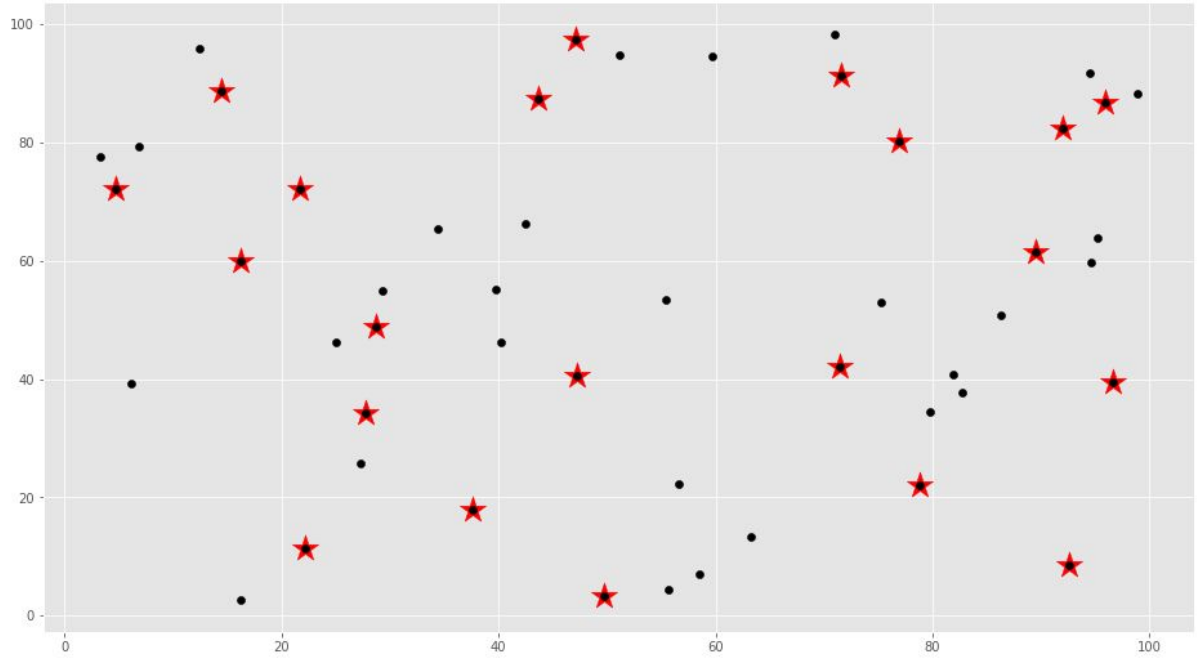
[ 4216.5, 4512.5, 4625.5, 4660. , 4413.3, 4732.2, 5052.9, 4122.6, 4364. , 4670.5, 4530.6, 4968.8,.....500 values..... 4292.2, 4345. , 4369.2, 4432.1, 4429.1, 4986.5, 4528.8, 4630.1, 4380.1, 4841.3, 4265.6, 4495.7, 4297.6]

4. repeat for large number of generations (here, generations = 1000)
    a. Selection
    b. Crossover
    c. Mutation
    d. update chromosomes and fitness (offspring population)

5. Termination criteria : There are various criterions that we can follow to terminate this process of step 4. One of which being, looping the above steps 4 until we get a population in which the best chromosome has a certain desired percentage of fitness (fitness that we desire). But since in our case, we don't know the percentage fitness that is actually achievable and we don't know how many generations it will take to get the highest fitness possible, we run the above steps 4 for a fix number of 1000 generations. The number of generations should not be too low and not too high because if it is too low, GA would not be able to check sufficient amount of combinations and hence we won't achieve global maxima(chromosome with higher/supposedly best fitness). If it is too high, GA takes a long time to run and hence increases the time complexity. It is highly recommended that the number of generations are as high as possible.

6. After the GA executes for fix number of generations, we take the chromosome with best fitness out of the final population of chromosomes.

    a. The chromosome with best fitness gives us coverage of: 5450.4




The above image shows us the genes(BSs) of the best chromosome from the final population. These give us maximum coverage of 5450.4 unit^2 (in 1000 generations for which the algorithm runs).

## 4. Steps of GA in detail :

A. Selection: (two chromosomes) : There exists many methods of selection of ranging from random wheel method to tournament selection. It may seem that taking two chromosomes of best fitness(two good chromosomes) would further yield to offsprings of better fitness. But that is not at all necessary since the crossover and mutation of 'a good and a bad chromosome' or even ' two bad chromosomes' can yield to a much improved off springs. For our scenario we have taken two chromosomes randomly from the initial population (chrom_s1 and chrom_s2).

Consider a particular generation in which the two chromosomes chosen are :

Chrom_s1 =

[[ 58.55608459, 6.97829599], [ 34.38713063, 65.35574296],
[ 37.55122269, 17.8268646 ],[ 22.16823961, 11.28230332],
[ 49.67114135, 3.24704176],[ 6.81889906, 79.34774706],
[ 76.97115462, 80.19381054],[ 71.61506076, 91.28754073],
[ 94.67096157, 59.67520035],[ 47.07314995, 97.30881677],
[ 43.60193044, 87.39491363],[ 29.23945654, 54.79851388],
[ 39.72825814, 55.03739796],[ 86.34737891, 50.7673117 ],
[ 59.7054293 , 94.47639451],[ 16.21722215, 2.77652374],
[ 95.89670462, 86.56486252],[ 71.45362931, 41.99839814],

[ 3.22133448, 77.5847233 ],[ 75.29586815, 52.90407055],
[ 82.80019815, 37.63289182]]

Chrom_s2 =
[[ 27.7768188 , 34.33790469],[ 21.62869319, 72.19709092],
[ 94.67096157, 59.67520035],[ 75.29586815, 52.90407055],
[ 58.55608459, 6.97829599],[ 81.9290237 , 40.83102805],
[ 22.16823961, 11.28230332],[ 98.94248385, 88.11227138],
[ 43.60193044, 87.39491363],[ 14.40943535, 88.64701602],
[ 34.38713063, 65.35574296],[ 76.97115462, 80.19381054],
[ 29.23945654, 54.79851388],[ 82.80019815, 37.63289182],
[ 96.60504031, 39.47552244],[ 92.60020217, 8.64174249],
[ 79.80309661, 34.46344822],[ 12.3897041 , 95.72728891],
[ 39.72825814, 55.03739796],[ 24.98416091, 46.30324155],
[ 37.55122269, 17.8268646 ]]

B. Crossover: (n-point crossover) : The crossover corresponds to the process of generating offsprings by the exchange of genes among the selected chromosomes. We implement the process of crossover as follows :

a. we set the probability of crossover as 0.9

b. initiate a random number between 0.0 and 1.0

c. if random number < 0.9
   i. //carry out crossover
   ii. //determine length of crossover array
   iii. length of crossover array L = k/2, if k = 21; L = 10
      1. If L = odd; L = L + 1
   iv. //randomly generate crossover array
   v. each element of crossover array should be randomly generated between the range of 1 and k
   vi. crossover array = random.sample(range(1,k),n)
      1. Crossover array = [4,2,7,9,11,16,19,20,12,14]
   vii. sort the crossover array in ascending order
      1. Crossover array = [2,4,7,9,11,12,14,16,19,20]
   viii. //execute crossover as stated in 3.B to get offsprings
   ix. //store the two new chromosomes as offspring.

Chrom_s1 =

[[ 58.55608459, 6.97829599],[ 21.62869319, 72.19709092],
[ 94.67096157, 59.67520035],[ 75.29586815, 52.90407055],
[ 49.67114135, 3.24704176],[ 6.81889906, 79.34774706],
[ 22.16823961, 11.28230332],[ 98.94248385, 88.11227138],
[ 43.60193044, 87.39491363],[ 47.07314995, 97.30881677],
[ 34.38713063, 65.35574296],[ 76.97115462, 80.19381054],
[ 39.72825814, 55.03739796],[ 82.80019815, 37.63289182],
[ 96.60504031, 39.47552244],[ 92.60020217, 8.64174249],
[ 95.89670462, 86.56486252],[ 71.45362931, 41.99839814],
[ 39.72825814, 55.03739796],[ 24.98416091, 46.30324155],
[ 82.80019815, 37.63289182]]

Chrom_s2 =

[[ 27.7768188 , 34.33790469],[ 34.38713063, 65.35574296],
[ 37.55122269, 17.8268646 ],[ 22.16823961, 11.28230332],
[ 58.55608459, 6.97829599],[ 81.9290237 , 40.83102805],
[ 76.97115462, 80.19381054],[ 71.61506076, 91.28754073],
[ 94.67096157, 59.67520035],[ 14.40943535, 88.64701602],
[ 43.60193044, 87.39491363],[ 29.23945654, 54.79851388],
[ 29.23945654, 54.79851388],[ 86.34737891, 50.7673117 ],
[ 59.7054293 , 94.47639451],[ 16.21722215, 2.77652374],
[ 79.80309661, 34.46344822],[ 12.3897041 , 95.72728891],
[ 3.22133448, 77.5847233 ],[ 75.29586815, 52.90407055],
[ 37.55122269, 17.8268646 ]]

d. if random number > 0.9
    i. //skip crossover

- Crossover : (values in chromosome 1 from crossover array[0] to crossover array[1] should be swapped with chromosome 2) and (values in chromosome 1 from crossover array[2] to crossover array[3] should be swapped with chromosome 2) and so on…

C. Mutation: (value added mutation) : Just like crossover we perform this with a set probability but here the probability is very low(0.1 or 0.05 etc) as compared to crossover (0.9). We implement the process of mutation as follows:

a. We set the probability of mutation as 0.1

b. Initiate a random number between 0.0 and 1.0

c. if random number < 0.1

i. carry out value added mutation
ii. generate a 'random value' which is to be added or subtracted
iii. generate another 'random number' between 0.0 and 1.0 to decide whether the 'random value' is to be added or subtracted
iv. if random number < 0.5
    1. generate a random index of chromosome at which to add the 'random value'
    2. add the 'random value' at that index of chromosome
v. if random number > 0.5
    1. generate a random index of chromosome at which to subtract the 'random value'
    2. subtract the 'random value' at that index of chromosome

d. if random number > 1.0
    i. //skip mutation

Chrom_s1 =

[[ 58.55608459, 6.97829599],[ 21.62869319, 72.19709092],
[ 94.67096157, 59.67520035],[ 75.29586815, 52.90407055],
[ 49.67114135, 3.24704176],[ 6.81889906, 79.34774706],
[ 22.16823961, 11.28230332],[ 98.94248385, 88.11227138],
[ 43.60193044, 87.39491363],[ 47.07314995, 97.30881677],
**[ 35.58713063, 66.55574296],[ 78.17115462, 81.39381054],**
[ 39.72825814, 55.03739796],[ 82.80019815, 37.63289182],
[ 96.60504031, 39.47552244],[ 92.60020217, 8.64174249],
[ 95.89670462, 86.56486252],[ 71.45362931, 41.99839814],
[ 39.72825814, 55.03739796],[ 24.98416091, 46.30324155],
[ 82.80019815, 37.63289182]]

Chrom_s2 =

[[ 27.7768188 , 34.33790469],[ 34.38713063, 65.35574296],
[ 37.55122269, 17.8268646 ],[ 22.16823961, 11.28230332],
[ 58.55608459, 6.97829599],[ 81.9290237 , 40.83102805],
[ 76.97115462, 80.19381054],[ 71.61506076, 91.28754073],
[ 94.67096157, 59.67520035],[ 14.40943535, 88.64701602],
**[ 44.80193044, 88.59491363],[ 30.43945654, 55.99851388],**
[ 29.23945654, 54.79851388],[ 86.34737891, 50.7673117 ],
[ 59.7054293 , 94.47639451],[ 16.21722215, 2.77652374],
[ 79.80309661, 34.46344822],[ 12.3897041 , 95.72728891],
[ 3.22133448, 77.5847233 ],[ 75.29586815, 52.90407055],
[ 37.55122269, 17.8268646 ]]

- Crossover and mutation balance : crossover is the convergence criterion which almost at each generation yields new offsprings that are more fit and hence the probability is kept more.On the other hand, the process mutation is a divergence criterion and hence the probability for it to be carried out in a particular generation is kept very low. Generally the probability of crossover + probability of mutation is kept 1.0 (specially, in tpot libraries). however we have implemented from scratch and hence we can try using different values.

D. Update chromosomes and fitness : calculate the fitness of the offsprings generated from the above steps (f_new1, f_new2). Also calculate the two chromosomes with least fitness from the initial population (f1,f2). Sort the values f1,f2,f_new1,f_new2 and take the two largest values. Hence, the chromosomes with low fitness have been replaced and the population has been revised. The new population becomes the input for the next generation.

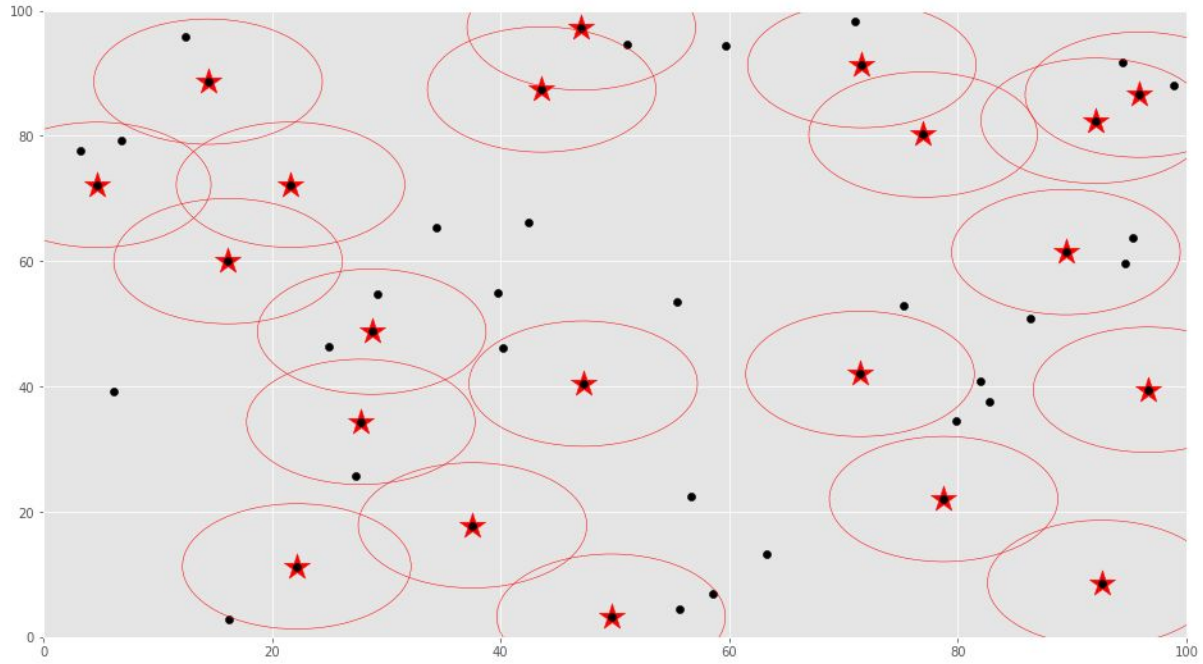
E. Check for termination criteria : If the generation < 1000; re-iterate above steps
Else If generation >=1000; terminate

**5. Results :**

1. The genes in the final chromosome give us the best points that give us maximum coverage.

   Coverage by best chromosome = 5769.5 unit^2
   we calculate the coverage % as,
   **Coverage %** = coverage by best chromosome / actual coverage
   = 5769.5 / 7887.23
   = 73.15%

   The optimization % can be calculated as,
   **Optimization %** = (50-21) / 50 = 62%




this image shows us the coverage area final chromosome's 21 BSs(red) selected out of the 50 initial. we get coverage % of 73.15%

2. Now we can see that we have reduced the number of BSs to 21 from 50 using a fitness function that uses euclidean coverage as its metric. Although the coverage % is low(75.31%), the model has been optimized hby a whopping 62%. But in practical and real world scenario, optimizing it by 10-30% would be a great feat in itself.Hence we decide to manually add 7 more BSs to increase the coverage from the remaining 29 BSs.

   The 7 points added are:

   [ 6.10970525, 39.28283245],
   [ 42.47549373, 66.21943596],
   [ 82.80019815, 37.63289182],
   [ 63.22244682, 13.2989283 ],
   [ 55.42796483, 53.45328774],
   [ 59.7054293 , 94.47639451],
   [ 56.65731173, 22.37289703].

   Upon adding 7 more BSs we have 28 BSs(21+7) in total we get the following coverage value:

Coverage by best chromosome with the new 7 BS points = 6986.21 unit^2

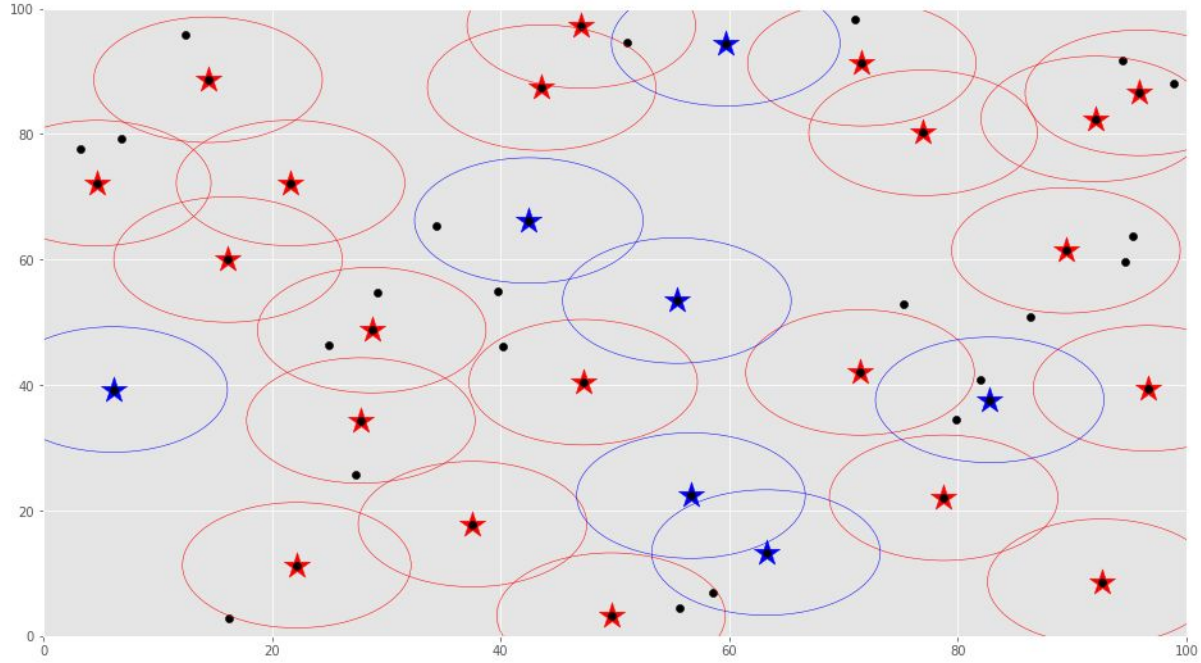
Hence the new coverage % is,

**Coverage %** = 6986.21 / 7887.23
= 88.34%

The optimization % can be calculated as,

**Optimization %** = (50-28) / 50 = 44%

We get 88.34% coverage with 44% optimization.




this image shows us the final coverage area after adding 7 more BSs(blue, 28 total) in the same configuration. we get coverage % of 88.34%.

## 6. Limitations and their possible solutions:

1. The number of clusters(k=21) are calculated considering only the coverage as the criteria. It doesn't count the fact that the initial 50 BSs were installed the way they are for some reason in the first place. The initial installers must have considered the load and population density of that area and then installed those 50 initial BSs accordingly.

   a. The number of clusters should be calculated by keeping in mind not only the coverage as criteria but also the utilization of each tower. We should find minimum amount of BSs required to handle the existing load while having as much coverage.

2. The fitness function uses 'max coverage' as the criteria i.e it keeps the chromosome that has more coverage and doesn't take in the consideration of population density.
   a. One way to consider the population density and users' requirements is to take into account the utilization of each of the 50 BSs and then modifying the fitness function in such a way that it takes into account not only the coverage but also the utilization of BS. The fitness function should keep those chromosomes that have max coverage as well as no BS(gene) in that chromosome should go beyond its max capacity. Then only, we would say that the 'reduced network' can handle all the load and have as much coverage while keeping the BS count lower than initial.

3. The radius of each BS is taken as constant while it can vary from urban to suburban region, depending on the population density. The radius of each BS would be different and should be calculated by keeping in mind the population density and tower utilization.
   a. One way to consider the population density in calculating the radius of each BS is to check the distance to the neighboring BSs. The reason behind calculating the distance between two BSs is that, If the initial installation any two BS is close, then that would be because of high population density in that area(area covered by these two BSs). Similarly, two BSs must have been placed relatively far from each other because the population density in that area(area covered by these two BSs) must have been low.
   b. So if one BS is closer to its neighboring BS then that BS must be in highly population density area and hence its radius must be smaller.

4. Number of generations for which the Genetic algorithm runs is significantly less. Due to hardware limitations, we were not able to test it for more than 1000 generations since this itself takes about an hour to run. Increasing more number of generations would result in much more accurate results. Deploying the algorithm on high performance servers would enable us to run with more number generations without taking more time.