Using Genetic Algorithm for Dynamic Allocation of Facility Location.

***Abstract*—Since industrial developments, there has been an alarming rise in fire accidents, specifically in highly dense and industrial areas. Due to the lack of well-planned rescue location-allocation, damage caused by these incidents are significant. To solve this problem, the approach in this project is to apply Genetic algorithm on the dataset containing potential demand locations and facility locations for Mumbai, India. The data provided has been used to predict the potential locations for the fire stations and number of vehicles required in them to handle an emergency situation in the nearby area. This has been done on the basis of the fact that the demand locations can be of different types, commercial, residential or marketplace. Depending on the location, the demand for the number of fire stations and rescue vehicles required. Genetic algorithm is used to accurately predict the optimal location of rescue stations. The results obtained are convincing enough to be used for the fire station location planning in Mumbai and various other locations facing such a problem.**

I. Introduction

In India, many deaths are caused due to delays in rescue vehicles reaching the required location. While not having enough resources is a problem, suboptimal planning of the location of current rescue facilities (fire stations for this project) poses a bigger challenge as it leads to unfortunate loss of life while having resources at our disposal.

The various factors causing delays that prevent successful rescue in such times of need are:

- Bad state/ narrowness of the roads and presence of obstacles.
- Traffic at various times of the day.
- Total travel time from the rescue station to the rescue spot.
- Multiple rescue locations requiring assistance from one fire station.
- Frequency of emergency cases.

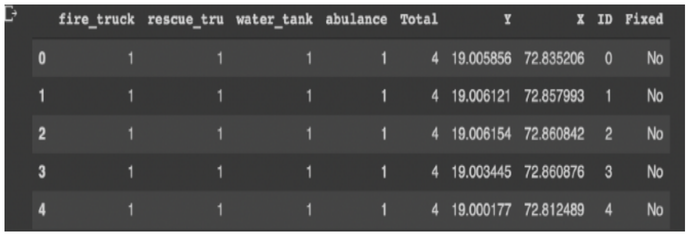
There are some cases when due to the abovementioned problems the fire fighter squad needs to commute on foot to execute the rescue operation.

A possible solution for such situations is setting up of dynamic facilities, mini fire stations or hubs, at a distribution point with a potential for reaching various rescue locations with the constraint that any demand point is served by its nearest station and is served solely by that particular station, accounting for factors of travel route and time taken.

II. Data used:

A. Demand data:

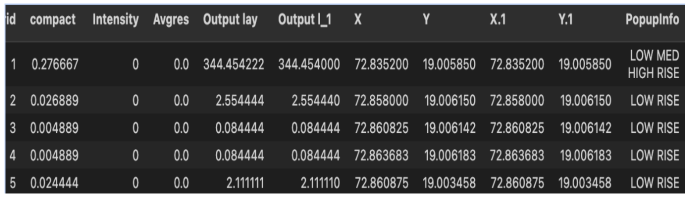
Contains information about the rescue vehicles requirements (fire trucks, ambulances, rescue tanks and water tanks) of various points across three different time intervals 4am - 8 am, 8 am - 12pm and 12 pm - 16 pm.



	fire_truck	rescue_tru	water_tank	abulance	Total	Y	X	ID	Fixed
0	1	1	1	1	4	19.005856	72.835206	0	No
1	1	1	1	1	4	19.006121	72.857993	1	No
2	1	1	1	1	4	19.006154	72.860842	2	No
3	1	1	1	1	4	19.003445	72.860876	3	No
4	1	1	1	1	4	19.000177	72.812489	4	No

B. Potential locations data:

Contains information regarding various points that can be a potential location. Along with the compactness and population density information of various travel routes to them.



id	compact	Intensity	Avgres	Output lay	Output l_1	X	Y	X.1	Y.1	PopupInfo
1	0.276667	0	0.0	344.454222	344.454000	72.835200	19.005850	72.835200	19.005850	LOW MED HIGH RISE
2	0.026889	0	0.0	2.554444	2.554440	72.858000	19.006150	72.858000	19.006150	LOW RISE
3	0.004889	0	0.0	0.084444	0.084444	72.860825	19.006142	72.860825	19.006142	LOW RISE
4	0.004889	0	0.0	0.084444	0.084444	72.863683	19.006183	72.863683	19.006183	LOW RISE
5	0.024444	0	0.0	2.111111	2.111110	72.860875	19.003458	72.860875	19.003458	LOW RISE

C. Shapefiles data:

Plotting this data using a 3D visualization tool or some computational geometry method gives an idea of the location and how the points are distributed over a map.

D. Normalizing the data:

Data for location and time travel was normalized to reduce the computation cost while applying the genetic algorithm.

III. Genetic Algorithm, Formulation and Python implementation.

Objective function:

Equation 1- (Where d denotes the density/compactness of location i per hub at i, di $\in [0, 1]$).

$$\sum_i d_i x_i + \sum_j \sum_i y_{j_i} t_{j_i}$$

The first term signifies the property of an establishment cost of a hub at that point based on the area's compactness; the second term signifies the cost in terms of travel time for each establishment. The fitness function is the inverse of the objective function.

$$x_i = \begin{cases} 1, & \textit{hub present at } i \\ 0, & \textit{hub absent at } i \end{cases}$$

$$y_{ji} = \begin{cases} 1, & \textit{location at } j \textit{ is served by hub at } i \\ 0, & \textit{otherwise} \end{cases}$$

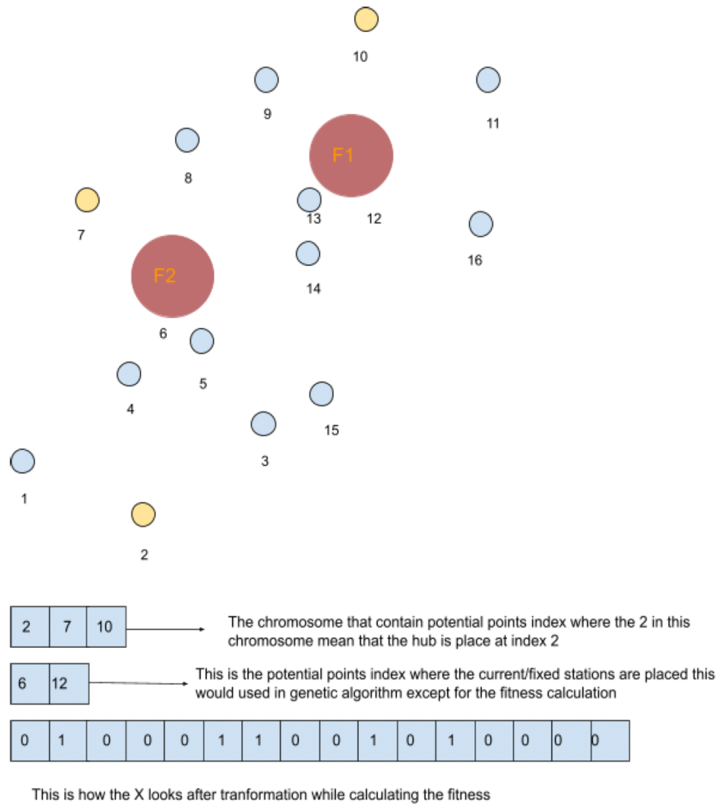
tji = time taken by hub at i to some location at j

Constraint:-

$$\sum_i y_{ij} = 1, \qquad \forall j$$

A. Genetic Algorithm:

Genetic algorithms (GAs) can be defined as meta-heuristics based on the evolutionary process of natural systems. They have been applied to a lot of optimization problems giving us a great result. GAs are a new approach to solving complex problems such as determination of facility layout. GAs became known through the work of John Holland in the 1960s. The GAs contain the elements of the methods of blind searching for the solution and of directed and stochastic searching and thus give compromise between the utilization and searching for a solution. At the beginning, they search in the entire search space and afterwards, by means of crossover, they search only in the surrounding of the promising solutions. So GAs employed random, yet directed search for locating the globally optimal solution. The starting point in GA presented in this work was an initial population of solutions (which was randomly generated). The facility layout and its randomly generated chromosome are shown on figure1. This population undergoes a number of transformations designed to improve the solutions provided. Such transformations are made in the main loop of the algorithm, and have three basic stages: selection, crossover, and replacement, as discussed below. Each of the selection-transformation cycles that the population undergoes, constitutes a generation. After a certain number of generations, the population evolves towards the optimum solution to the problem, or at least to a near-best solution. The selection stage consists of sampling the initial population, thereby obtaining a new population with the same number of individuals as the initial one. This stage aims at improving the quality of the population by favoring those individuals that are more adequate for a particular problem (the quality of an individual is gauged by calculating its fitness, using equation 1, which indicates how good a solution is).



The chromosome that contain potential points index where the 2 in this chromosome mean that the hub is place at index 2

This is the potential points index where the current/fixed stations are placed this would used in genetic algorithm except for the fitness calculation

This is how the X looks after tranformation while calculating the fitness

Fig. 1. Type of layout used in calculations and its chromosome representation (actual data is larger)

The selection, mutation, and crossover functions are used to create the new generation of solutions. A fitness function evaluates the likelihood of a design to be in the next generation. Each chromosome is put into a mating several times based on the fitness making it more likely to be selected. Mutation is the process of randomly changing the genes in the chromosome and it prevents GAs from stagnating during the solution process. Crossover is responsible for introducing most new solutions by selecting two parent strings at random from the mating pool and exchanging parts of the strings.

B. Formulation:

A GA cycle used in this work operates as follows:

- Generate an initial population consisting of 200-1000 members using a random number generator. The initial population's chromosome size varies from 0, 40 i.e. the total number of hubs.
- Calculate the fitness f (Eq. (1)) of all population members. And append it to a new list with the matching index.
- Place each member in a mating pool, more number of times if the fitness value is higher.
- Randomly select two parents from the mating pool. So the chromosome with the highest fitness value has a

high probability to be selected.

- Now the two parent chromosomes are crossed over with a midpoint randomly chosen.
- Now for each member in the population each gene is mutated with a low probability. And replaces the previous gene's population.

The starting chromosome in the new iteration isn't randomly generated. It is the chromosome obtained by crossover of two parent chromosomes discussed above. Consider a pair of parent chromosomes (P1, P2) shown below:

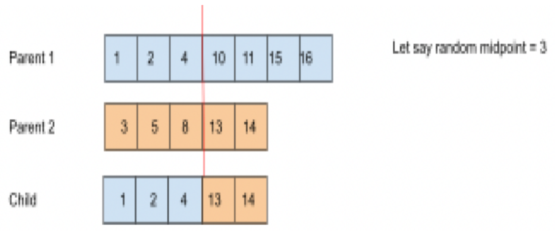
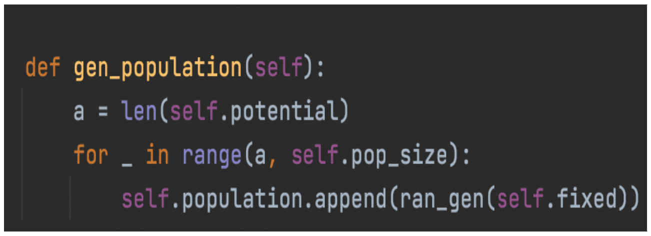


Fig 2. The crossover

The way of crossover implementing in this work was choosing a midpoint randomly based on the parents size and take the first half from the first parent and the second half from the second parent i.e. for random midpoint of 3, from P1(1, 2, 4) is taken an from P2 (13, 14) is taken and the child formed is (1, 2, 4, 13, 14). Which is shown in the above figure. The mutation function is used to replace each gene with a certain low probability .

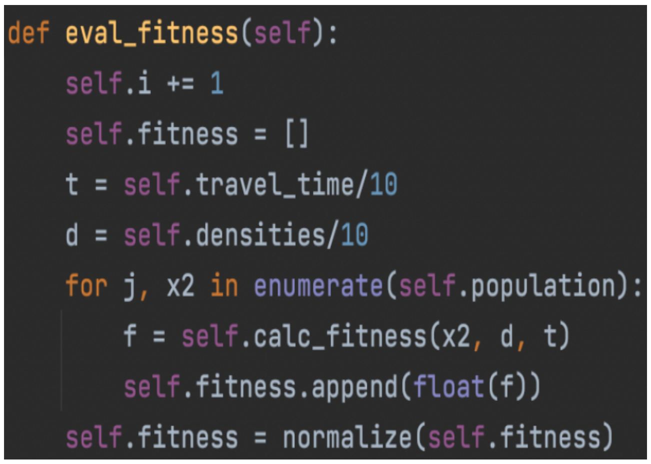
C. Python implementation

1) Population Initialisation: The gen_population method generates loops for the pop_size mentioned and triggers a function called "ran_gen" which randomly generates chromosomes of random size which do not contain any indexes from fixed points.



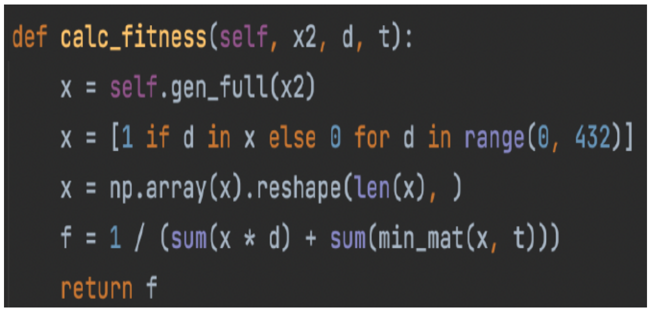
```
def gen_population(self):
    a = len(self.potential)
    for _ in range(a, self.pop_size):
        self.population.append(ran_gen(self.fixed))
```

2) Fitness Evaluation: The eval_fitness method loops over every population chromosome, then transforms the chromosome to a list of 0's and 1's where 1 means that there is a facility at the location in that facility and calculates their fitness.



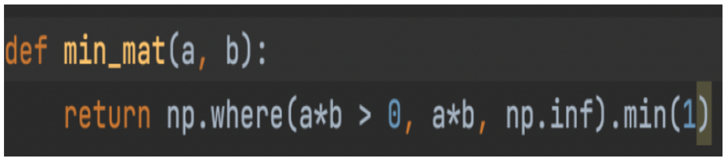
```
def eval_fitness(self):
    self.i += 1
    self.fitness = []
    t = self.travel_time/10
    d = self.densities/10
    for j, x2 in enumerate(self.population):
        f = self.calc_fitness(x2, d, t)
        self.fitness.append(float(f))
    self.fitness = normalize(self.fitness)
```

The calc_fitness calculates the value for the fitness. Here the min_mat function finds the y matrix i.e where a particular demand location is served by the hub where the rows are demands which adds to 1 which is as the constraint mentioned.



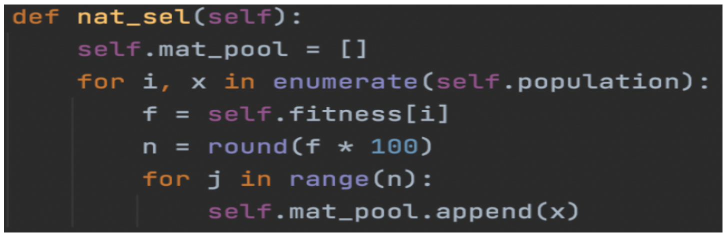
```
def calc_fitness(self, x2, d, t):
    x = self.gen_full(x2)
    x = [1 if d in x else 0 for d in range(0, 432)]
    x = np.array(x).reshape(len(x), )
    f = 1 / (sum(x * d) + sum(min_mat(x, t)))
    return f
```

In the code block below, after x(i.e a) is multiplied with t(i.e b) by the numpy broadcasting each row in t has dot product with x(i.e. Without summing them up),then the minimum of each row is taken and turned into a numpy array, which is what the function return later it is summed in the fitness function which basically the summing of all elements in y*t matrix. And this fitness value is added to the respective index in the fitness list.



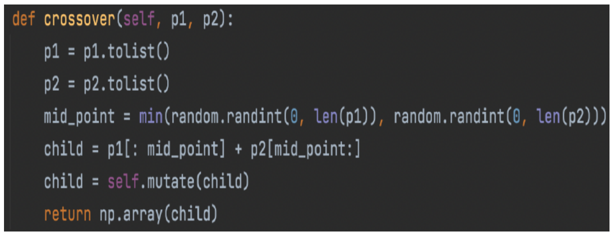
```
def min_mat(a, b):
    return np.where(a*b > 0, a*b, np.inf).min(1)
```

3) Selection: In nat_sel method mat_pool is emptied and looped over the entire population and adds the chromosomes to the mat_pool for the respective fitness times 100.



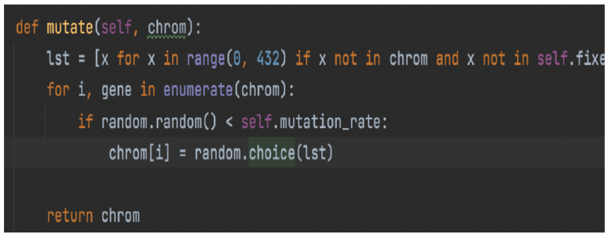
```
def nat_sel(self):
    self.mat_pool = []
    for i, x in enumerate(self.population):
        f = self.fitness[i]
        n = round(f * 100)
        for j in range(n):
            self.mat_pool.append(x)
```

4) Crossover Mutation: In the new_pop method we loop over the population and two parents are randomly chosen and are crossed over.



```
def crossover(self, p1, p2):
    p1 = p1.tolist()
    p2 = p2.tolist()
    mid_point = min(random.randint(0, len(p1)), random.randint(0, len(p2)))
    child = p1[: mid_point] + p2[mid_point:]
    child = self.mutate(child)
    return np.array(child)
```

Then we call over the mutate method which loops over the child's gene and each gene is changed to a different value with a low probability mut_rate.



```
def mutate(self, chrom):
    lst = [x for x in range(0, 432) if x not in chrom and x not in self.fixe
    for i, gene in enumerate(chrom):
        if random.random() < self.mutation_rate:
            chrom[i] = random.choice(lst)

    return chrom
```

The resultant child is returned as a numpy array which then replaces a chromosome in the population. And the whole process is repeated.

IV. Results

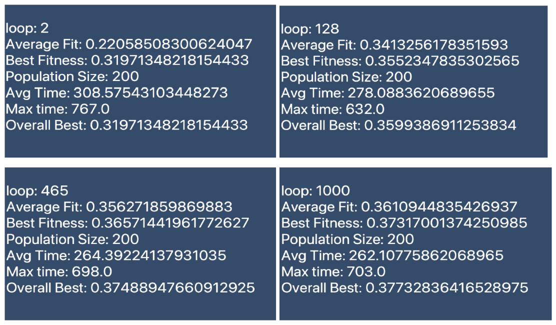
- The Genetic algorithm ran as expected the average fitness value of the population increased over loops. It was done with a max facility size of 50 including 22 fixed stations and a population size of 200. The finding for the 8-12 dataset is provided below. It is over a loop of 1000.

- Here we can see that we started with around 0.22 for average fit with reaching to 0.34 in the 128th loop and 0.34 on the 400th loop subsequently reaching to 0.36 in the 1000th loop.

- Same trend can be seen with best fitness and average time as well. But for max time, i.e. the point where the facilities are farest from, seems to increase this could be mainly due to the density factor that is included in the objective function. And the other reason could be that the algorithm tries to find ones with some location having very small time values ignoring the other ones.

- We were able to run the file on all the time period data sets of 4-8, 8-12 and 12-16, and the findings for each of them is discussed below.

- Looking at the final findings(Fig. 4) for the different time stamps we observe that fitness value for data set on time 4-8 seems to have very high compared to other ones this is mainly due to reason that, during this time the road traffics are very less so most places can be reached in less time.



```
loop: 2
Average Fit: 0.22058508300624047
Best Fitness: 0.31971348218154433
Population Size: 200
Avg Time: 308.57543103448273
Max time: 767.0
Overall Best: 0.31971348218154433
```

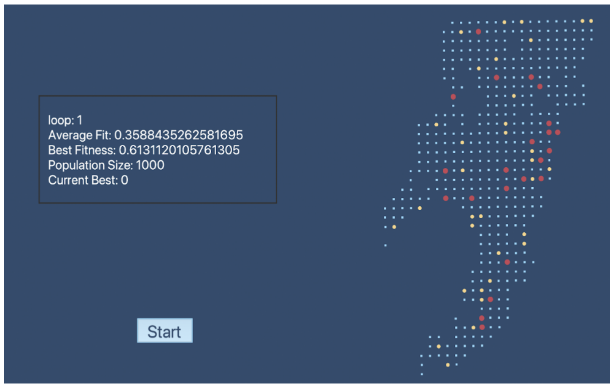
```
loop: 128
Average Fit: 0.3413256178351593
Best Fitness: 0.3552347835302565
Population Size: 200
Avg Time: 278.0883620689655
Max time: 632.0
Overall Best: 0.3599386911253834
```

```
loop: 465
Average Fit: 0.356271859869883
Best Fitness: 0.36571441961772627
Population Size: 200
Avg Time: 264.39224137931035
Max time: 698.0
Overall Best: 0.37488947660912925
```

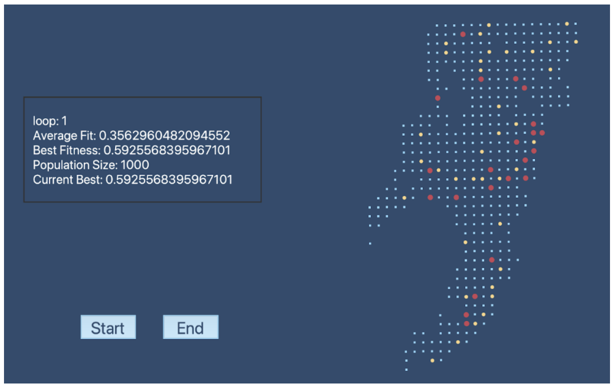
```
loop: 1000
Average Fit: 0.3610944835426937
Best Fitness: 0.37317001374250985
Population Size: 200
Avg Time: 262.10775862068965
Max time: 703.0
Overall Best: 0.37732836416528975
```

Fig 3. Findings over course of 1000 loop

- Below are two examples of changing the allocation with changing needs.



```
loop: 1
Average Fit: 0.3588435262581695
Best Fitness: 0.6131120105761305
Population Size: 1000
Current Best: 0
```



```
loop: 1
Average Fit: 0.3562960482094552
Best Fitness: 0.5925568395967101
Population Size: 1000
Current Best: 0.5925568395967101
```

Legend for the above two mappings. (The dashboard displaying the allocation is made using tkinter library).

● Fixed Location (Physical Fire stations)

● Dynamic Hubs (Movable Fire stations hubs)

■ Potential Locations

V. Conclusion

- The dynamic allocation approach is an extension of the static one as it considers the changes over multiple periods and the cost of rearranging the layout as per the requirements. Also genetic algorithm can cater to the needs of dynamic approach without taking much help from dynamic programming.

- Were able to find out potential locations, assign dynamic hubs for each and were able to visualise which fixed location will serve them.

- The allocation is such that the hub may serve multiple demand points but each demand point is served only by a single hub.

VI. Further works

- Complexity of each genetic cycle is very high. We could try to improve it to improve the speed of the program.

- Can find better value to normalize the fitness. Because it would give us better results.

- Solve the problem under additional constraints of vehicle availability.

References

[1] K .F. Man, K.S. Tang, S. Kwong, “Genetic Algorithm: concepts and applications”, IEEE Transactions on Industrial Engineering, (Oct 1996).

[2] Chawis Boonmee, Mikiharu Arimura, Takumi Asada, “Facility location optimization model for emergency humanitarian logistics”, International Journal of Disaster Risk Reduction

[3] Holland H.J., “Adaptation in Natural and Artificial Systems”, 1975 (University of Michigan Press: Ann Arbor).