TERRITORY OPTIMIZATION PROJECT

By- Megha Dhandhania

PROBLEM: Algorithm should perform Multivariate Territory Optimization

GOAL: The goal is to achieve:

1. Reduced travel time for reps
2. Balancing the potential and workload between territories
3. Improving the performance
4. Identifying and reallocating overlapping territories

GENETIC ALGORITHM

1.1 The Essence of Genetic Algorithm:

The Genetic Algorithm is a heuristic search technique which finds approximate solutions to optimization problems. It is an independent optimizer which performs consistently well on a broad range of problem types. In other words, there are no prerequisites on the problem before using the algorithm.

1.2 Different Scenarios in Sales Territory Planning:

Nowadays, marketers and decision makers face three different scenarios in sales territory planning. These scenarios are the following:

1. We have neither the location of sales representatives nor the existing sales territory structure. In this case, the number of needed territories should be determined by decision makers. Then the algorithm consists of two parts, respectively:

- Finding the best location for each sales representative. The number of sales representatives are equal to the number of territories.
- Performing a heuristic search algorithm for finding an acceptable sales territory structure.

This scenario is usually called *Green Field Planning* in Geo-marketing.

2. We have a number of sales representatives with fixed locations, but we do not have any existing territory structure.
3. We have either fixed sales representatives or existing sales territory structure. But, the territories should be optimized due to the new strategy of the sales organization.

For each scenario, a respective strategy is chosen. But, we are going to design a generic approach for all scenarios. So we should have a workflow which supports all scenarios.

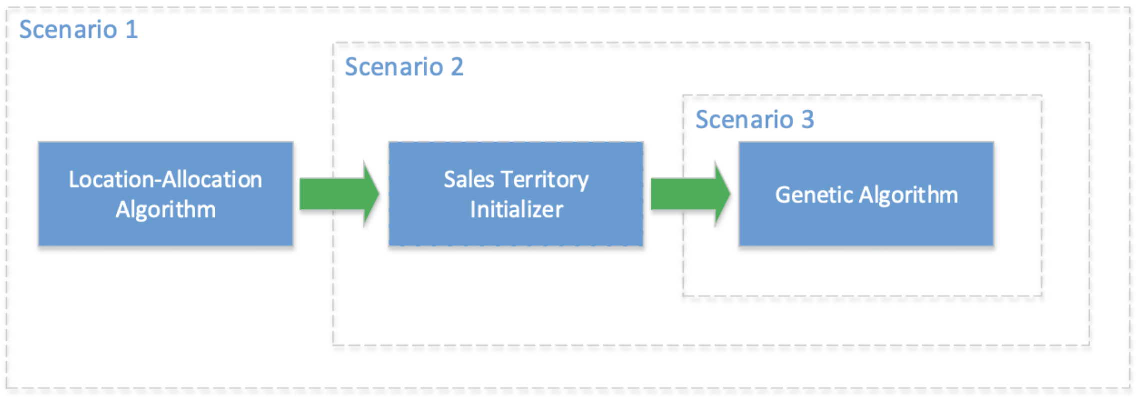


Fig 1: Generic Workflow

1.3 Objective:

There are several criteria that can be considered to group basic areas into the sales territories. Some of them search to balance income or potential turnover. Some criteria attempt to balance workload between salesmen. Another objective is to maximize the total profit contribution . In our model, we formulate an objective to balance the potential customers and workload among sales representatives. In order to obtain this objective, we should minimize the total standard deviation of potential in each territory from the mean potential value (target value).

1.4 Algorithm Design:

We firstly introduce a novel approach to handle the location-allocation problem. Secondly, we define an algorithm for initializing a raw structure for sales territories based on the travel time. Ultimately, we design a customized genetic algorithm for solving our multi-objective sales territory design problem. As we discussed before, one of the main advantages of the genetic algorithm is that the genetic algorithm does not depend on the problem type. So, we only have to design a respective fitness function for each scenario and we do not need to change the main body of the algorithm which will be defined later.

1.4.1 A simple Approach for Location-Allocation

Basic Area, also called the Sales Coverage Unit, is considered as a relatively small geo-graphical area which depends on the specific problem to be solved. Examples include, Zip-Code, Trading area or municipality. Firstly, it is important to find a set of basic areas which are the starting point for the territory planning. In our approach we will be using the distribution of potential among basic areas to meet our objectives.

Algorithm #1 :

This algorithm classifies basic areas into several classes and determines how many basic areas must be selected for each class.

Algorithm 1 Location Allocation Algorithm

```
Inputs:
    $Candidates = [c_1, c_2, \ldots, c_N]$    # list of capital municipalities as candidate location
    $Size_s \leftarrow$ size of sales team
Outputs:
    $Locations \leftarrow$ empty array list    # empty array to store the locations of sales team
$Classes \leftarrow$ NaturalBreaks($Candidates$)    # classifying the candidates by using NaturalBreaks
$Size_c \leftarrow$ size of $Classes$    # number of classes
for $i = 1$ to $Size_c$ do
    $weight \leftarrow i$    # the weight of class
    $identifier \leftarrow (Size_c * (2 + (Size_c - 1)))/2$
    $capacity \leftarrow Round(weight * (Size_s/identifier))$    # the number of persons can be located in class[i]
    for $j = 1$ to $capacity$ do
        $tmp \leftarrow$ getRandomCandidate($Classes[i]$)    # get a candidate municipality from class[i] randomly
        addItemToList($Locations$ , $tmp$)    # adding a municipality to Locations array
    end for
end for
return $Locations$
```

1.4.2 Sales Territory Initialization

Search Space: The Genetic Algorithm always looks for the best solution among a set of possible solutions. The space of all feasible solutions is called the search space which also contains the global optimum solution.

Since the search space is so large due to the number of basic areas, we have to minimise the search space before running the Genetic Algorithm (GA). Therefore, we attempt to cluster the basic areas based on the nearest sales/service man to make a raw structure for sales territories.

Algorithm #2

This algorithm shows the clustering process of the basic areas.

Algorithm 2 Clustering Algorithm

```
Inputs:
  $G = [g_1, g_2, \ldots, g_N]$          # set of basic areas
  $S = [s_1, s_2, \ldots, s_N]$          # set of salesmans basic areas
  $DM$                                   # distance matrix ($SCU_a$ to $SCU_b$)
Initialize:
  $Size_1 \leftarrow$ size of $G$
  $Size_2 \leftarrow$ size of $S$
  $Chromosome \leftarrow$ empty array list # array list to store territory genomes

for i = 1 to $Size_1$ do
  $tmp \leftarrow 0$                     # variable to store nearest salesman id
  $gid \leftarrow G[i]$
  for j = 1 to $Size_2$ do
    $t \leftarrow$ Maximum value of travel time
    $sgid \leftarrow S[j]$               # basic area of salesman
    $traveltime \leftarrow DM(gid, gsid)$
    if $t > traveltime$ then
      $t \leftarrow traveltime$
      $tmp \leftarrow$ getSalesmanId($gsid$)
    end if
  end for
  $Genome \leftarrow$ createGenome($gid$ , $tmp$)   # instantiating the territory genome object

  addItemToList($Chromosome$ , $Genome$)  # adding Genome to Chromosome
end for
return $Chromosome$
```

1.4.3 The Genetic Algorithm for Sales Territory Planning

The sales territory planning is the problem of grouping basic Geographic areas (SCU) such as counties, zip codes, municipalities into larger clusters called sales territories. The goal of sales territory planning is mostly maximising the total profit of the sales organization. As the sales territory planning is classified as an NP-hard problem, using a meta-heuristic search algorithm is beneficial. Moreover, the sale territory planning is a multi-objective problem and is non-continuous and thus cannot be solved via linear programming. The genetic algorithm is a search heuristic which has proven efficient and effective for solving spatial problems. So we have chosen the genetic algorithm for our solution approach.

The abstract scheme of the genetic algorithm is defined in a pseudo-code fashion as follows.

Algorithm #3

Algorithm 3 The Genetic Algorithm

```
initialize a population with random candidate solutions;
while (termination condition is satisfied) do
    evaluate each candidate;
    select fitter candidates and copy into the mating pool;
    while (size of new generation = size of population) do
        select pair of parents from mating pool;
        recombine parents to make offspring;
        mutate the generated offspring;
        copy the offspring into the next generation;
    end while
end while
decoding the best found solution
```

Check Appendix 1 for a detailed explanation of each step of the genetic algorithm

1.5 Overview

Genetic Algorithm is trying to emulate the genetic evolution process to find a solution.
It uses:

- ➢ Chromosome
- ➢ Cross-over
- ➢ Mutation
- ➢ Survival of the fittest (natural evolution)

to find the solution.

Biggest Advantage: You do not need to know how to solve the problem; we just have to be able to evaluate the quality of generated solutions coming and through iterations you get good solutions.

- ➢ Evolution and Selection process - almost same for all kind of problem
- ➢ Fitness Function and Chromosome Design - has to be problem specific

1.6 Advantages and Disadvantages of Genetic Algorithm:

1.6.1 Advantages

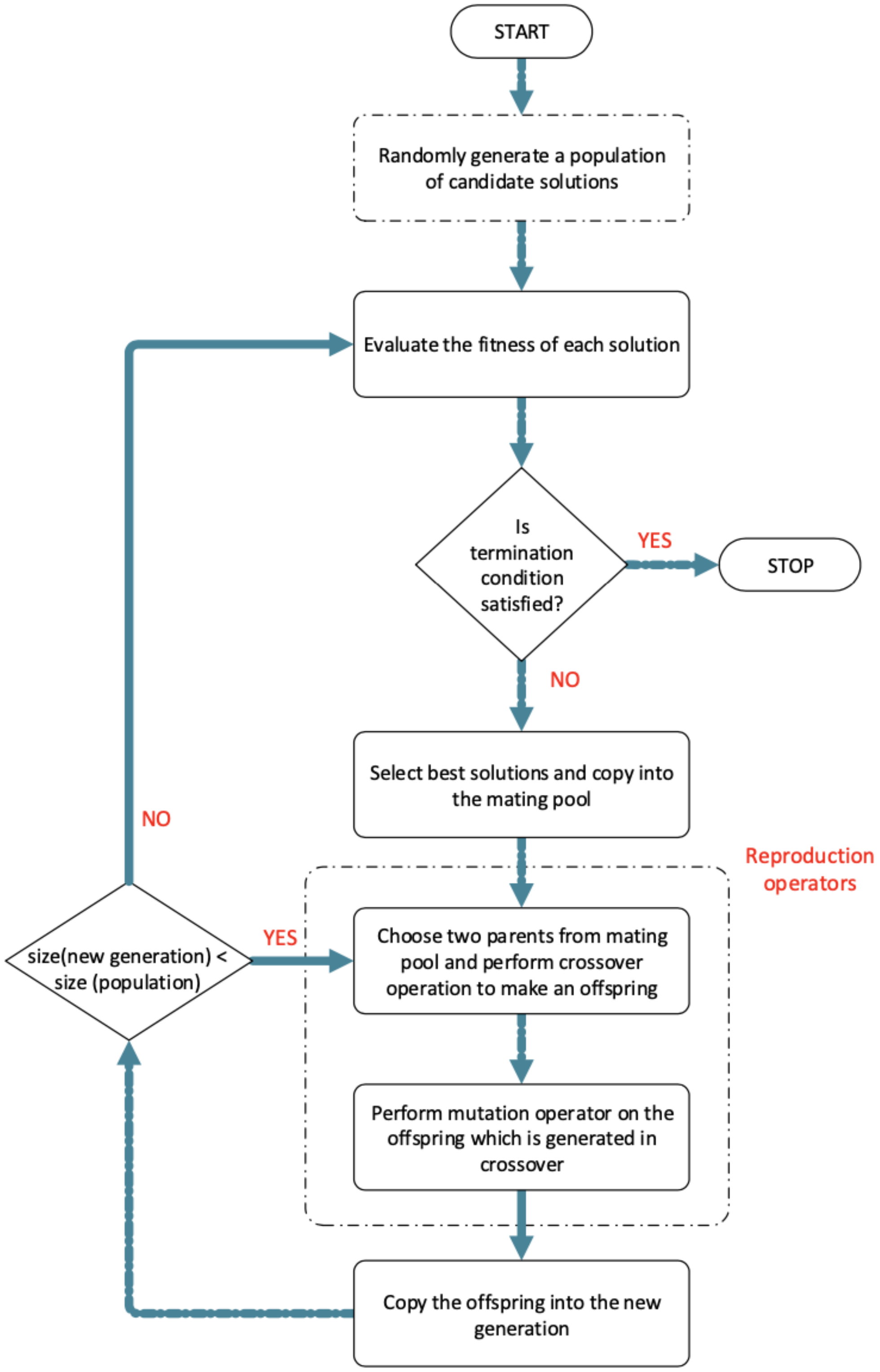
- It is possible to run the algorithm in parallel
- The algorithm is able to escape from local optimum
- It is able to discover global solution
- It can solve multi-objective problems
- It is totally independent from the problem domain
- The fitness function can be anything which can be evaluated by computers
- It can be easily customized for different problems
- Its concepts are easy to understand
- It always returns an answer for the problem and the answer gets better with time – It can be very fast for some special problems (e.g. TSP)

1.6.2 Disadvantages/Limitations

- Identifying the fitness function can be difficult
- Definition of representation (mapping from Phenotype to Genotype) for the problem is not straightforward
- Premature convergence happens

- It is always difficult to find the best value for various parameters such as: selection pressure, crossover and mutation probabilities, population size and so on
- GA cannot always find the exact solution, but finds the best solution
- It does not assure constant optimization response time

1.7 Flowchart of Genetic Algorithm


START
Randomly generate a population of candidate solutions
Evaluate the fitness of each solution
Is termination condition satisfied?
YES
STOP
NO
Select best solutions and copy into the mating pool
Reproduction operators
Choose two parents from mating pool and perform crossover operation to make an offspring
Perform mutation operator on the offspring which is generated in crossover
Copy the offspring into the new generation
size(new generation) < size (population)
YES
NO

ITERATIVE PROJECTION APPROACH

Aim: Using iterative approach to solve the 'Territory Business and Sales Optimization Problem'

Synopsis: The problem is formulated as a combination of assignment, scheduling and routing optimization problems. The plan contains the three main aspects.

1. Grouping of customers located in a certain region to a sellers with capacity limitation
2. The weekly schedule plans for visits to fulfill the demand of the customers. Due to the customers requirement, the visits should be spread over the week.
3. The daily route of the sellers to visit the customers

Case Study #1

Problem: Analyzing a logistics districting problem for package pickup and delivery within a region, motivated by a real-world application. The region is divided into districts, each served by a single vehicle that departs from a central depot.

Aim: To optimise workload balance and compactness as a single objective problem.

General Approach:
1. Clustering of customers using nearest neighbour approach (districting)
2. Scheduling the visits (using effective queuing)
3. Minimisation of the complete distance travelled by the vehicles while servicing all the customers

Note : Particularities of the modelling approach include scheduling constraints of visits spread over the week, service and travelling time; as well as the time capacity to ensure the fulfillment of the customers demand.

Mathematical Formulation:

Check Appendix 2 for mathematical functions used.

Solution Method:

The above model is NP-Complete. Therefore the time required to achieve the optimal solution of the whole model increases exponentially with the growth of the problem size. Therefore, this difficulty can be solved by dividing the problem into three sub-problems.

Phase 1 : Projection Method : minimising the distance between customers and sellers.

Also known as the m-dimensional method that maps m components of the approximate solution vector onto a subspace determined by k < = m.

This is a randomised block projection method combined with the Johnson-Lindenstrauss dimension
Reduction, given as follows:

Algorithm

1. Take a m × n matrix A, and a column vector b, choose an integer parameter $s = 8log(m)\varepsilon^{2}$, ε is the desired accuracy, which corresponds to the probability level $1 - m^{-2}$ in the J-L theorem, and the initial approximation x0.
2. We aim at an approximation of the solution x to Ax = b.
3. Set k = 0, generate an n × s random matrix R satisfying the independence and symmetry condition, say, a Gaussian matrix $R = \{g_{ij}\}$ where the entries g_{ij} are independent zero mean Gaussian random numbers with variance 1/s , and calculate the rows $hi = (AR)_{i} = a_{i}R$.
4. 4. Sample a set of N = m rows at random, say, according to $p_{i} = ||a_{i}||_{2}^{2} = ||A||_{F}^{2}$. Uniform sampling is also possible. Note that N can be taken less than m which implies that the random search of rows is carried out only among a part of all the rows. For each row, calculate the distance $\Delta_{i} = (|b_{i} - (h_{i}; R^{T}x_{k})|) / ||h_{i}||_{2}$ and choose a_{j} as the row with the maximal Δj . This row is used to calculate the next projection step in our iteration process shown by (Equation 1).

A hedging step can be introduced here to prevent the case that the chosen *j* is worse than the random row actually used in the calculation of the projection. So along with the chosen a_j take also an arbitrary row a_p in the set of chosen rows, and calculate k+ k +1 , go to 4.

$$\overline{\Delta}_j = \frac{|b_j - (a_j, x_k)|}{\|a_j\|_2}, \qquad \overline{\Delta}_p = \frac{|b_p - (a_p, x_k)|}{\|a_p\|_2}$$

If $\overline{\Delta}_p \geq \overline{\Delta}_j$ set $j = p$.
Calculate the projection

$$x_{k+1} = x_k + \frac{b_j - (a_j, x_k)}{\|a_j\|_2^2} a_j^T$$

Equation (1)

Phase 2: Scheduling plan for visits

Minimizes the total days used for visiting subject to the constraints (8)-(10) in appendix 2.

Objective Function:

$$min \sum_i \sum_t t * \cdot v_i^t$$

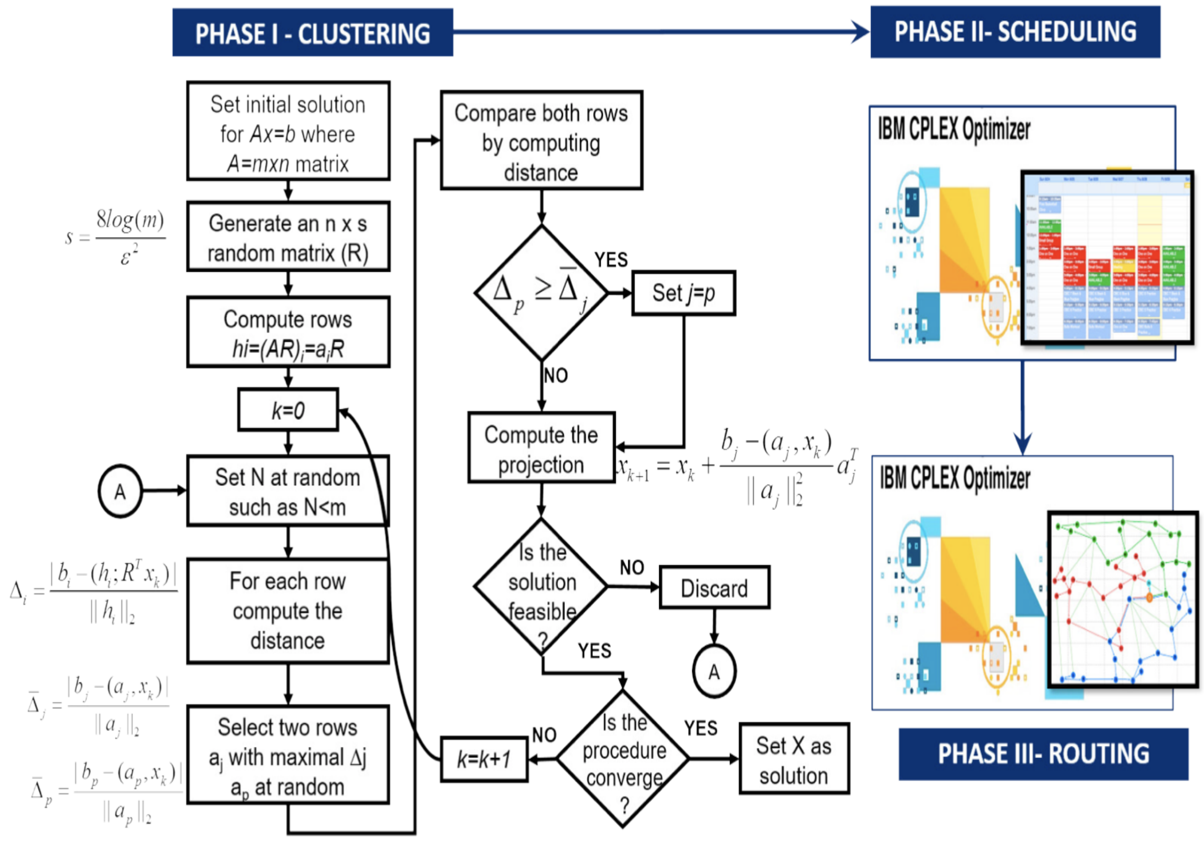
This function helps to schedule the nearest neighbours per day.

Phase 3: **Routing Problem Formulation**

Objective Function:

$$min \sum_i \sum_j d_{i,j} \cdot x_{i,j}$$

Proposed Flowchart



PHASE I - CLUSTERING
PHASE II- SCHEDULING
Set initial solution for Ax=b where A=mxn matrix
$s = \frac{8\log(m)}{\varepsilon^2}$
Generate an n x s random matrix (R)
Compute rows hi=(AR)_i=a_iR
k=0
A
Set N at random such as N<m
$\Delta_i = \frac{|b_i - (h_i; R^T x_k)|}{\|h_i\|_2}$
For each row compute the distance
$\bar{\Delta}_j = \frac{|b_j - (a_j, x_k)|}{\|a_j\|_2}$
$\bar{\Delta}_p = \frac{|b_p - (a_p, x_k)|}{\|a_p\|_2}$
Select two rows a_j with maximal Δj a_p at random
Compare both rows by computing distance
$\Delta_p \geq \bar{\Delta}_j$
YES
Set j=p
NO
Compute the projection
$x_{k+1} = x_k + \frac{b_j - (a_j, x_k)}{\|a_j\|_2^2} a_j^T$
Is the solution feasible ?
NO
Discard
A
YES
Is the procedure converge ?
NO
k=k+1
YES
Set X as solution
IBM CPLEX Optimizer
IBM CPLEX Optimizer
PHASE III- ROUTING

Fig. 1: Proposed solution for territorial business plan

Conclusion

The first phase uses a projection method to assign customers to sellers, the method proved an efficient solving time, with gaps from 0 to 28% in the objective function. The described approach also allows us to tackle the uncertainties stemming from practical problems such as different sizes of territory and particular features of the demand such as the distance and the service time.

SMARTALIGN - OPTIMIZATION ALGORITHM

SmartAlign designs sales territories by dividing a set of sales coverage units (SCU), such as accounts and zip-codes or salesforce activities, among sales territories.

Algorithm:

Each iteration of the algorithm consists of the following steps:

1. Generate the 'greedy' algorithm i.e each zip code is assigned to its closest territory center via its shortest path
2. Calculate the violation of the workload constraint set (i)
3. Modify distances from each zip code to each center in such a way that the "light" centers, i.e those with insufficient workload appear closer to each zip code than at the previous iteration, and "heavy" centers appear further away.

This process is repeated with diminishing distance adjustments at successive sets of iterations as described in Zoltners and sinha (1983). In order to set up the contiguity constraints, the shortest path from each zip code of its potential territories centers are calculated. To facilitate this step, we can develop a cross zip-code distance database that makes the calculation a simple table look up.

Termination:

The process ends when adjustments are below a set tolerance limit.

Objective Function:

$$\text{Minimize} \ \sum_i \sum_j w_j d_{ij} x_{ij}$$

subject to:

$$\text{(i)} \quad l_i \leq \sum_j w_j x_{ij} \leq u_i \quad \text{for each } i,$$

$$\text{(ii)} \quad x_{ij} \leq \sum_{p \in A_{ij}} x_{ip} \quad \text{for each } i,\ j,$$

$$\text{(iii)} \quad \sum_i x_{ij} = 1 \quad \text{for each } j,$$

$$\text{(iv)} \quad x_{ij} = 0 \text{ or } 1 \quad \text{for each } i,\ j,$$

where :

j is the index of sales coverage units (zip codes)

i is the index of sales territory centers

xij is the decision variable which takes on the value 1 if zip code j is assigned to territory center i, 0 otherwise;

dij is the distance of center i from zip code j;

wj is the workload in zip code j;

l is the lower bound for the total workload for territory center i;

ui is the upper bound for the total workload for territory center j;

p ∈ Aij is the set of zip codes that immediately precede zip code j in the shortest path to territory center i.

The objective function is the sum of workload weighted SCU distances from the territories centers and helps us to ensure compact territories. Constraint set (i) is the balancing criterion, and set (ii) ensures contiguity of sales territories.

CASE STUDY 1

Problem: Design a territory by zip codes in four states such that it contains a population of at least 75,000.

Approach 1: Using the K Means Clustering Algorithm

The essence of the approach is as follows:

1. Choose one state and map the population in that state
2. Form clusters in the densely populated area throughout the state using the K-means Clustering Algorithm.
3. Check the population contained within the clusters
4. If the population_count > = 75,000 then the zip codes contained within the cluster forms a territory within that state
5. If the population count is less than 75,000, we expand the boundary of the cluster.
6. We do so by using the shortest path algorithm to find the nearest zip code with the next maximum population and include that in the cluster.
7. We check again if our termination condition is met. If no, we go to Step #6 Otherwise we move on to do the same thing for a different cluster in the same state.
8. We can repeat step#1 to step#7 for the other three states given to us.

Approach 2: Using the clustering application in Genetic Algorithm

The essence of the approach is as follows:

1. Again we will be choosing a state and map the population in that state
2. Using minimum population requirement >= 75,000 as our constraint for each territory on the clustering method under Genetic algorithm, we will get a set of possible solutions in the search space.
3. In other words, the clustering algorithm will create a set of different combinations of possible zip codes which would give a total of 75,000 in population at least.
4. Now, we will use the genetic algorithm to find the optimal combination from the given search space which would then constitute a territory
5. We keep iterating until we reach the desired number of territories for each state.

Approach 3:

BIBLIOGRAPHY

Genetic Algorithm

1. J. Kalcsics, S. Nickel, M. Schröder. Towards a Unified Territory Design Approach – Applications, Algorithms and GIS Integration. https://d-nb.info/1027387403/34
2. Wenfei Xu. 2018. Balancing Territories for Equity and Efficiency: A Field Sales Data Study. https://carto.com/blog/sales-territory-balancing-optimization/
3. Carrie Ka Yuk Lin, "Solving a Location, Allocation, and Capacity Planning Problem with Dynamic Demand and Response Time Service Level", *Mathematical Problems in Engineering*, vol. 2014, Article ID 492340, 25 pages, 2014. https://doi.org/10.1155/2014/492340

Iterative Projection Approach

1. L. H. Escobar, V. Alexandrov, "Iterative Projection Approach for solving the Territorial Business Sales Optimization Problem", 2017, https://www.sciencedirect.com/science/article/pii/S1877050917327308

APPENDIX 1

Genetic Algorithm Detailed Description

1.1 Building Blocks for Genetic Algorithm: Mathematical Modelling

1.1.1 Basic Area:

The basic area (also called Sales Coverage Unit) is considered as a relatively small Geo-graphical area which depends on the specific problem to be solved. Zip code, municipality and trading areas are the most common examples of basic areas [32].

C: a set of basic areas (1)

$j \in |C|$ *: index of basic area which is often represented by its central point*

1.1.2 Location of sales representatives:

The location of salespersons is considered as the centroid of the basic area where the sales-person lives, because our distance matrix consists of travel time and distance between the centroid of all municipalities in Germa*ny*.

V *: a set of basic areas for locating salespersons* $(V \subseteq C)$

(2)

$i \in |V|$ *: index of sales representative*

1.1.3 Number of territories:

The number of sales territories is often defined by decision makers before planning. Thus, we assume that the number of territories is N .

1.1.4 Non-overlapping:

Each basic area must be assigned only to one salesperson.

$Xij \in \{0, 1\}$ *: a binary variable equal to 1 if salesperson located in SCU i*

(3) *is assigned to SCU j ; Otherwise equal to 0*

1.1.5 Complete assignment of basic areas:

Every basic area must be assigned to exactly one salesperson (a territory).

$Ti \subset C$: i-th territory (4)

$$\sum_{i=1}^{N} T_i = c\,,\; T_i \cap T_k = \varnothing,\; for\, all\, i \neq k,\; 1 \leq i,\, k \leq N$$

1.1.6 Contiguity:

Territories should be created in such a way that all basic areas are connected. In order to obtain contiguity, neighborhood information for each basic area is required.

$Ji \subseteq C$: a set of SCUs which are assignable to salesperson in SCU i

(5)

$Ai \subseteq C$: a set of SCUs are adjacent to SCU j

q_{ivj}: quantity of flow from v to j with origin in location i

$$X_{ij} + \sum_{v\in(A_i\cap Ji)} (q_{ijv} - q_{ivj}) - \sum_{v\in Ji \mid i=j} X_{iv} = 0 \qquad i \in I\,,\; j \in J_i$$

$$|J_i|.\, X_{ij} - \sum_{v\in(A_i\cap Ji)} q_{ijv} \geq 0$$

1.1.7 Compactness:

A territory is compact if it is relatively round-shaped and undistorted. In our approach we consider the total travel time from the centroid of territory Tj (usually the basic area where the salesperson lives) to each basic area assigned to territory Tj . Then, the following problem should be solved in order to obtain the compactness:

minimize $f(x) = \sum_{i=1}^{N} K_i$ *(i = 1, …, N)*

(6)

subject to

$$K_i = \sum_{j \in T_i} (D_{ij}) \qquad (i = 1, \ldots, N)$$

where :

Dij = travel time or travel distance from the salesman basic area of territory i to SCU j

Ti = a set of basic areas belong to territory i

1.1.8 Objective :

We aim to balance the potential customers and workload among the sales representatives. Therefore, we should minimise the total standard deviation of potential in each territory from the mean potential value (targeted value)

Minimize $f(x) = \sum_{i=1}^{N} | R_i - \bar{r} |$

(7)

Subject to

$$\bar{r} = \frac{1}{N} \sum_{j \in T_i} r_j \qquad (i = 1,\ldots,N)$$

$$| R_i - \bar{r} | \leq \delta \bar{r} \qquad (i = 1,\ldots,N)$$

where :

n : number of basic areas

N : number of territories

T_i : i-th Territory

r_j : the amount potential contained in SCU j

R_i : the total potential in territory i

δ : is the maximum allowable percentage ($0 \leq \delta \leq 1$)

1.2 Detail Description of the Genetic Algorithm functions:

1.2.1 Mapping Phenotype and Genotype: The GA has two distinguished elements which are individuals and populations. Individual is a single candidate solution to the optimization problem while the population is a set of individuals involved in the optimization process. Observable properties of the individual (Phenotype) should be encoded to the raw genetic information (Genotype). Genotype is also called chromosome in the GA. The mapping between genotype and phenotype is necessary, because the solution set should be converted from the model into the form that the GA can work with. In other words, through mapping of phenotype and genotype we attempt to encode all decision variables of our problem in such a way that it can be applied to the genetic algorithm and can be simply decoded in order to be shown inside the application.

We choose a Generic Array List data structure to store an arbitrary size of Genotype in any form and manipulate it easily. The main advantage of a generic array list is that it is capable of storing different data types such as Integer, String, Object, Arrays, etc. Since array list is index-based data structure, iterating through an array of *N* elements is faster than other data structures with the same size.

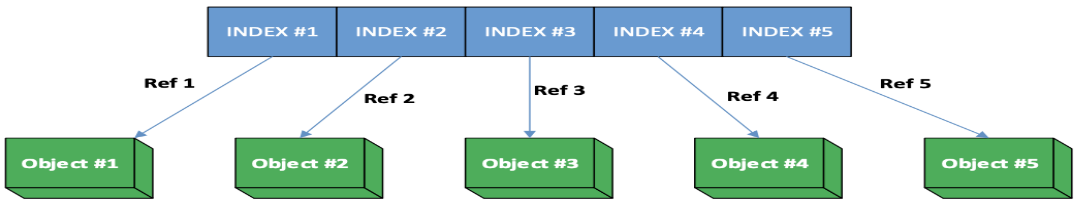


Fig 2: Generic Array List for Genotype

1.2.2 Initializing a Random Population: In the GA, we maintain many individuals (candidate solutions) in a population. The size of the population is usually defined before running the algorithm. Each individual (also called chromosome) is composed of genomes. As we discussed before, we are able to add an arbitrary number of genomes into the generic array list. In our model, the size of the array list shows the number of basic areas. The algorithm below creates a population of random candidate solutions by means of an iterative process. The process will be stopped when the number of generated individuals are equal to the size of population which is given by the user.

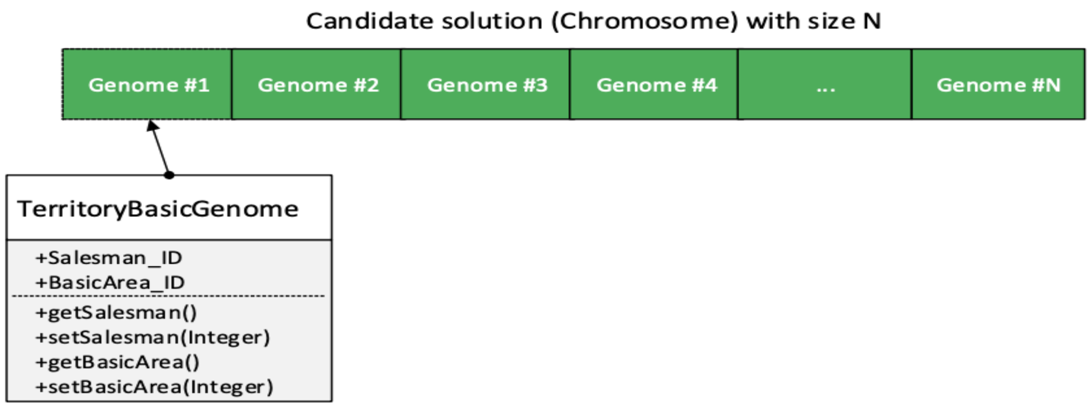


Fig 3: Territory Genome Object

Algorithm #3.1 Making Random Population

```
1:  **Inputs**:
      $Size \in \mathbb{N}$                    # size of the population
      $S = [s_1, s_2, \ldots, s_N]$            # $S$: set of salesmans
      $G = [g_1, g_2, \ldots, g_L]$            # $G$: set of basic areas
2:  **Initialize**:
      $N \leftarrow$ size of $S$
      $L \leftarrow$ size of $G$
      $Chromosome \leftarrow$ empty array list
      $Population \leftarrow$ empty array list
3:  for i = 1 to $Size$ do
4:    for i = 1 to $L$ do
5:      $gid \leftarrow G[i]$
6:      $rnd \leftarrow$ getRandomNum(1 , $N$)   # making random index between 1 and $N$

7:      $sid \leftarrow S[rnd]$                  # choose one sales man randomly
8:      $Genome \leftarrow$ createGenome($gid$ , $sid$)   # instantiating the territory genome object

9:      addItemToList($Chromosome$ , $Genome$)   # adding Genome to Chromosome
10:   end for
11:   addItemToList($Population$ , $Chromosome$)   # adding Chromosome to Population
12: end for
13: return $Population$
```

The generated random population by the algorithm above will be used as an input for the selection process discussed below. **If you go to Algorithm #3, initializing random population is Step #1.**

1.2.3 Selection Process: The GA goes through a main loop. The selection process is supposed to simulate natural selection by giving fitter individuals a higher chance to survive and make offspring into the next generations. The selected individual will be added into the temporary storage (also called mating pool) to be used for breeding via crossover and mutation operations. So the selection process is considered as the main component of the GA for converging towards a global optima.

As filter individuals will be selected based on their fitness, we use the fitness function to evaluate the fitness of each individual.

We use the Tournament Selection method for the selection process which takes the local information into account. In other words, it tries to select the best individual in within a small group of individuals instead of the whole population

Algorithm #3.2 Tournament Selection

```
1:  Inputs:
      $Population = [ind_1, ind_2, \ldots, ind_N]$ # the population of candidate solutions
      $K$                                          # tournament size
2:  Initialize:
      $Size \leftarrow$ size of $P$               # size of population
      $Selection \leftarrow$ empty array list      # array list of selected individuals
3:  for i = 1 to $Size$ do
4:     $List \leftarrow$ empty array list           # temporary list for each tournament
5:     addItemToList($List$ , $Population[i]$)
6:     for j = 1 to $(K - 1)$ do
7:        $indv \leftarrow$ getRandomItem($Population$) # random individual($indv \notin List$)

8:        addItemToList($List$ , $indv$)
9:     end for
10:    $fittest \leftarrow$ getFittestIndividual($List$) # fittest individual wins the tournament

11:    addItemToList($Selection$ , $fittest$)
12: end for
13: return $Selection$
```

All selected individuals in the selection process will be copied into the mating pool to be used in the next operations. **If you go to Algorithm #3, the selection process Step involves step #5 - Step#6.**

1.2.4 Fitness Function : The fitness function (also called objective function in optimization problems) is the most important component of genetic algorithms. It measures the goodness of candidate solutions and makes a basis for selection, and thereby it simplifies improvements. In other words, the fitness function defines what improvement means for a specific problem.Since the sales territory planning is a multi-criteria problem intrinsically, we should formulate a multi-criteria decision making model to simultaneously minimize the components of criterion vector F(x):

minimize F (x) = (f1(x), f2(x))

subject to

f1(x) = equation (6)

f2(x) = equation (7)

In the following mathematical model we normalise our objective components and simplify them into a single objective function.

$$fitness = \left(\left(\frac{f_1(x) - a}{b - a}\right) * w_1\right) + \left(\left(\frac{f_2(x) - c}{d - c}\right) * w_2\right)$$

subject to

$$a \le f1(x) \le b$$

$c \le f2(x) \le d$

where :

f1(x) , f2(x) : control variables

w1 , w2 : weights of control variables

a , c : lower bounds of control variable

b , d : upper bounds of control variables

In this part, we only consider the first object defined in our objective section above.

Algorithm #3.3 Fitness Function

```
1:  **Inputs**:
      $Chromosome = [genome_1, \dots, genome_N]$ # candidate solutions
      $S = [s_1, s_2, \dots, s_N]$               # $S$: set of salesmans
      $DM$                                       # distance matrix ($SCU_a$ to $SCU_b$)
      $W_1, W_2$                                 # criteria weights
      $a, c$                                     # lower bounds
      $b, d$                                     # upper bounds
2:  **Initialize**:
      $Size_1 \leftarrow$ size of $S$
      $Size_2 \leftarrow$ size of $Chromosome$
      $potentials \leftarrow$ empty array list   # array list to store potential values
3:  $dist \leftarrow 0$                          # total distance (second critera: $f_2(x)$)
4:  $fitness \leftarrow 0$
5:  **for** i = 1 to $Size_1$ **do**
6:    $potential \leftarrow 0$
7:    $sid \leftarrow S[i]$
8:    **for** j = 1 to $Size_2$ **do**
9:      $Genome \leftarrow Chromosome[j]$
10:     $gsid \leftarrow$ salesman_id of $Genome$
11:     **if** $gsid = sid$ **then**
12:       $potential \leftarrow potential +$ getPotential($Genome$)
13:       $SCU1 \leftarrow$ get $SCU$ of $S[i]$
14:       $SCU2 \leftarrow$ get $SCU$ of $Genome$
15:       $dist \leftarrow dist + DM(SCU1, SCU2)$ # distance from $SCU1$ to $SCU2$
16:     **end if**
17:   **end for**
18:   addItemToList($potentials$ , $potential$)
19: **end for**
20: $sd \leftarrow$ calculateDeviation($potentials$) # standard deviation (first criteria: $f_1(x)$)

21: $fitness \leftarrow$ calculateFitness($sd, dist, w_1, w_2, a, b, c, d$) # equation 3.2.3

22: **return** $fitness$
```

If you go to Algorithm #3, the fitness function is shown from Step#3-Step#4.

1.2.5 Elitism: The generated offspring in the GA are supposed to inherit good characteristics of its par- ents, but sometimes offspring are weaker than their parents. So good candidate solutions can be lost after crossover and mutation operations.Although the GA is capable to re-discover the lost candidate solutions again, it can increase the calculation time. For solving this issue, we will use a technique which is called *Elitism* .

In this technique, we copy a proportion of the best candidate solution from the mating pool into the next generation without any changes. Most of the time, *Elitism* extremely improves the performance of the GA, because it does not waste the time to discover the lost candidate solutions in the previous generations and helps to navigate the search direction towards the global optimal solution. In addition, unchanged candidate solutions which are copied by elitism also remained in the mating pool for the reproduction process.

1.2.6 The Cross-Over Operation: The first operation of the reproduction process is *Cross-Over*. This operation picks up two parents from the mating pool and combines them in such a way that the generated off- spring inherits the characteristics of its parents. This operation gets the probability value 0 ≤ P ≤ 1 as the first parameter and the number of cross-over points as the second param- eter. In our model, we use a *one point crossover*.

The algorithm below depicts how the cross-over operation makes two offspring from two parents:

Algorithm #3.4 CrossOver Operation

```
1:  Inputs:
      $Chromosome1 = [genome_1, \dots, genome_N]$ # a candidate solutions as parent1
      $Chromosome2 = [genome_1, \dots, genome_N]$ # a candidate solutions as parent2
      $P$                                        # cross-over probability: $0 \leq P \leq 1$
2:  Initialize:
      $Size_1 \leftarrow$ size of $Chromosome1$
      $Size_2 \leftarrow$ size of $Chromosome2$
      $Result \leftarrow$ empty array list        # an array list to store 2 offspring
3:  $rnd \leftarrow$ getRandomNum(0 , 1)          # random number between 0 and 1
4:  if $rnd \geq P$ then
5:    $Max \leftarrow$ Min($Size_1$ , $Size_2$)
6:    if $Max > 1$ then
7:      $index \leftarrow$ getRandomNum(1 , $Max$) # random index between 1 and $Max$

8:      for j = 1 to $index$ do
9:        $Genome \leftarrow Chromosome1[j]$
10:       $Chromosome1[j] \leftarrow Chromosome2[j]$
11:       $Chromosome2[j] \leftarrow Genome$
12:     end for
13:   end if
14: end if
15: addItemToList($Result$ , $Chromosome1$) # adding first offspring

16: addItemToList($Result$ , $Chromosome2$) # adding second offspring

17: return $Result$
```

If you go to Algorithm #3, the crossover operation function is shown by Step #7.

1.2.7 The Mutation Operation: This operation helps in converging the search direction to the global optima. We just mutate the offspring on the border of the existing territories. It means that the basic area which is located on the border can be randomly assigned to the neighboring territory based on a very small probability. The basic areas inside the territories remain unchanged.

Algorithm #3.5 Mutation Operation:

```
1:  Inputs:
      $Chromosome = [genome_1, \ldots, genome_N]$ # offspring
      $P$                                          # mutation probability: $0 \leq P \leq 1$
      $NM \leftarrow$ Neighborhood matrix          # $NM[i,j] = 1$ if $SCU_i$ is neighbor of $SCU_j$; Otherwise 0
2:  Initialize:
      $Size \leftarrow$ size of $Chromosome$
3:  for i = 1 to $Size$ do
4:    $genome \leftarrow Chromosome[i]$
5:    $list \leftarrow$ get the neighbors of $genome$ # temporary salesman ID

6:    $rnd \leftarrow$ getRandomNum(0 , 1)  # random number between 0 and 1
7:    if $rnd \geq P$ then
8:      $nbr \leftarrow$ getRandomGenome($list$) # selecting random neighbor

9:      $nsid \leftarrow$ getSalesmanID($nbr$)  # salesman Id of neighbor genome
10:     $sid \leftarrow$ getSalesmanID($genome$) # salesman Id of $genome$

11:     $gid \leftarrow$ getScuID($genome$)      # SCU Id of $genome$
12:     if $nsid \neq sid$ then
13:       $genome \leftarrow$ createGenome($gid$ , $sid$)
14:       $Chromosome[i] \leftarrow genome$
15:     end if
16:   end if
17: end for
18: return $Chromosome$
```

If you go to Algorithm #3, the mutation operation function is shown by Step #8.

1.2.8 Minimising the search space: In the Genetic Algorithm, the whole search space is explored to get the best solution. Although we can minimise the search space before running GA, we can also minimise it within the algorithm. The treatment algorithm below helps in minimising the search space by avoiding invalid or useless solutions. A useless solution is a valid candidate solution which is generated by the algorithm after several generations, but does not lead the search direction to the global optimum solution. The algorithm below helps in the treatment of the invalid solutions. It detects the invalid assignments of basic areas and assigns them to the surrounding territory. This can be included as a part of the mutation operation mentioned above.

Algorithm #3.6 Treatment Operation

```
Inputs:
  $Chromosome = [genome_1, \ldots, genome_N]$ # offspring
  $NM \leftarrow$ Neighborhood matrix    # $NM[i, j] = 1$ if $SCU_i$ is neighbor of $SCU_j$; Otherwise 0
Initialize:
  $Size1 \leftarrow$ size of $Chromosome$
for i = 1 to $Size1$ do
   $genome \leftarrow Chromosome[i]$
   $list \leftarrow$ get the neighbors of $genome$ # temporary salesman ID

    $Size2 \leftarrow$ size of $list$
   for j = 1 to $Size2$ do
      $nsid \leftarrow$ getSalesmanID($list[nbr]$) # salesman Id of neighbor genome

      $sid \leftarrow$ getSalesmanID($genome$) # salesman Id of $genome$

      $flag \leftarrow 0$
      if $(nsid = sid)$ then
         $flag \leftarrow 1$                # basic area is not island
         break;                 # stop the loop
      end if
   end for
   if $(flag = 0)$ then
      $gid \leftarrow$ getScuID($genome$)      # SCU Id of $genome$
      $genome \leftarrow$ createGenome($gid$ , $sid$)
      $Chromosome[i] \leftarrow genome$     # marge the island into the surrounding territory
   end if
end for
return $Chromosome$
```

1.2.9 Termination Condition: Genetic Algorithm attempts to minimise the standard deviation of potential among the territories as much as possible. However, soon it is seen that the progressive decrement of standard deviation has stopped after a certain number of iterations. Therefore, if a certain number of generations pass without any improvement, then the genetic algorithm is terminated.

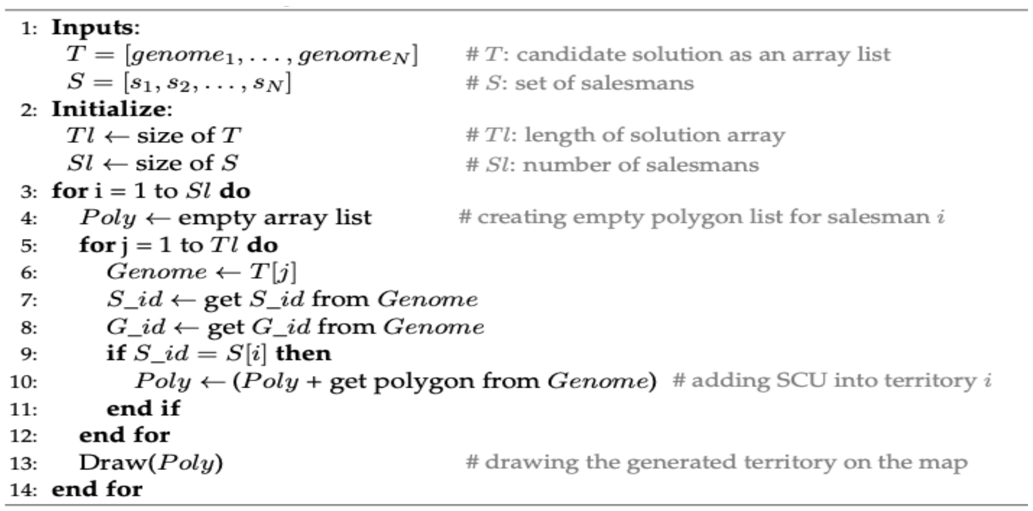
Algorithm #3.7 Termination Algorithm

```
1:  Inputs:
      $generationNumber$                # generation number
      $generationLimit$                 # limit of generations without improvement
2:  $fitness \leftarrow$ getFitness();
3:  if ($generationNumber = 0$) | | hasFitnessImproved($fitness$) then
4:      $fittestGeneration \leftarrow generationNumber$
5:  end if
6:  if ($generationNumber - fittestGeneration \geq generationLimit$) then
7:      return $true$
8:  else
9:      return $false$
10: end if
```

If you go to Algorithm #3, the termination operation function is shown by Step #2.

1.2.10 Decoding and Visualization: Ultimately, when the termination criteria is satisfied, the algorithm stops and returns the best solution as an array list. Each item in the array list is a specific object that we have already defined as territory genomes. The territory genome determines a basic area which belongs to a certain sales representative. At this step, we have to decode our result such that it can be visualized in the user interface. The algorithm below describes how the array list will be decoded.

Algorithm #3.8 Decoding Final Result



```
Inputs:
  $T = [genome_1, \ldots, genome_N]$          # $T$: candidate solution as an array list
  $S = [s_1, s_2, \ldots, s_N]$               # $S$: set of salesmans
Initialize:
  $Tl \leftarrow$ size of $T$                 # $Tl$: length of solution array
  $Sl \leftarrow$ size of $S$                 # $Sl$: number of salesmans
for i = 1 to $Sl$ do
    $Poly \leftarrow$ empty array list       # creating empty polygon list for salesman $i$
    for j = 1 to $Tl$ do
        $Genome \leftarrow T[j]$
        $S\_id \leftarrow$ get $S\_id$ from $Genome$
        $G\_id \leftarrow$ get $G\_id$ from $Genome$
        if $S\_id = S[i]$ then
            $Poly \leftarrow (Poly$ + get polygon from $Genome)$  # adding SCU into territory $i$
        end if
    end for
    Draw($Poly$)                              # drawing the generated territory on the map
end for
```

APPENDIX 2

Mathematical Formulation Detailed Description

Consider a set of customers *C* = {1, 2, ..., *N*} and a set of potential sellers *S* = {1, 2, ..., |*S* |} dispersed in a given region with geographical coordinates (*long*, *lat*). It is desired to design a business plan that defines a minimum number of sellers required to fulfill the customer's demand β. The sellers will attend the demand of customers during the weekdays *W* = {1, 2, 3, 4, 5, 6} denoted by index *t* in the scheduling plan per week. Finally, it is desired to get the optimal daily routing. The distance between two location is computed using the Haversine formula given in the following equation:

$$d_{i,j} = 2R\arcsin\left(\sqrt{\sin^2\left(\frac{lat_j - lat_i}{2}\right) + \cos\left(lat_i\right)\cos\left(lat_j\right)\sin^2\left(\frac{lon_j - lon_i}{2}\right)}\right)$$

where latitude (*lat*) and longitude (*long*) are given in radians, and *R* is earth's radius (mean radius = 6,371km)

The mathematical formulation of the model is defined using the following equations:

The objective function (1) represents the sum of three goals, the minimization of the distance between customers assigned to the sellers $d_{s,i}$, the traveling distance to visit each customer for each routing plan $d_{i,j}$, and the assignment of visits during the first days of the week.

This equation has three binary decision variables: y_i^s with value of 1 if the customer *i* is assigned to seller *s*, and $x_{i,j}^{s,t}$ with value of 1 is customer *j* is visited after customer *i* by seller *s* on day *t* and, $v_i^{s,t}$ with the value of one if customer *i* is visited by seller *s* during the day *t.*

$$min \sum_{s\in S}\sum_{i\in\mathbb{N}} d_{s,i}y_i^s + \sum_i\sum_j\sum_s\sum_t d_{i,j}x_{i,j}^{s,t} + \sum_i\sum_s\sum_t t * v_i^{s,t} \qquad \mathbf{(1)}$$

Constraint (2) ensures that the customer is attended by only one seller.

$$\sum_{s\in S} y_i^s = 1; \qquad \forall i \tag{2}$$

Equation (3) relates two binary variables y and v in order to guarantee that the customer assigned to the seller is actually visited.

$$v_i^{s,t} \leq y_i^s; \qquad \forall i, t, s \tag{3}$$

Equation (4) and (5) link the scheduling variables to the routing ones.

$$x_{i,j}^{s,t} \leq v_i^{s,t}; \qquad \forall i, j, t, s; \qquad i \neq j \in N \tag{4}$$

$$x_{i,j}^{s,t} \leq v_j^{s,t}; \qquad \forall i, j, t, s; \qquad i \neq j \in N \tag{5}$$

Equation (6) establishes that the number of incoming links to a client node must be equal to the number of outgoing links.

$$\sum_i x_{i,j}^{s,t} = \sum_j x_{i,j}^{s,t}; \qquad \forall i, j, t, s; \qquad i \neq j \in N \tag{6}$$

Equation (7) sets for each client one arrival and one departure at the time.

$$\sum_i x_{i,j}^{s,t} + \sum_j x_{i,j}^{s,t} = 2v_i^{s,t}; \qquad \forall i, j, t, s; \qquad i \neq j \in N \tag{7}$$

Equation (8) ensures that the sum of the service time per customer θ_i during the day does not exceed the available time of the seller.

$$\sum_i \theta_i v_i^{s,t} \leq \tau^s; \qquad \forall t, s \tag{8}$$

Equation (9) establishes the number of visits carried out per customer according to the given frequency. The frequency is computed by dividing the demand β and capacity ω and is given by σ

$$\sum_{t}\sum_{s} v_i^{s,t} = \sigma_i; \qquad \forall i \tag{9}$$

Equation (10) avoids consecutive visits to those customers whose frequency is less than 4 visits per week.

$$v_i^{s,t} + v_i^{s,t+1} \leq 1; \qquad \forall i, s, t; \qquad t \leq 5 \qquad \sigma_i \leq 3 \tag{10}$$

Finally, Equation (11) and (12) allow to assign proper order of visits to the customers during the routing plan to avoid sub-tours. The continuous variable e denotes the order in which customer i is visited in the route plan by seller s during day t. Equation (11) ensures that the difference between the order of visits between two customers is one. Equation (12) limits the maximum order of visits to the customer.

$$e_i^{s,t} - e_j^{s,t} + N x_{i,j}^{s,t} \leq N - 1; \qquad \forall i, j, t, s; \qquad i \neq j \in N \tag{11}$$

$$e_i^{s,t} \leq \sum_{j} v_j^{s,t}; \qquad \forall i, j, t, s; \qquad i \neq j \in N \tag{12}$$