Animal rescue and evacuation given a natural disaster scenario

Catalina Valle

July 8, 2022

Abstract

Animals have different hearing capacity than humans, this allows them to feel changes in frequencies and infra-sounds that are not audible to people, which is why they tend to run desperately in the face of natural events such as earthquakes and tsunamis causing a high index of lost pets and traffic accidents. In this work we study the evacuation problem with petFriendly shelters.

1 Introduction

According to [7] all animals have different reactions given a stressful situation, in this case an earthquake. Most of them will tend to run and get lost, generating greater stress and risk for the animal. It could also generate accidents since most of them are not used to the traffic rules as the stray dogs. The main objective of this research work is to minimize the rescue time of the animals, reducing their stress and avoiding accidents. This is due to a possible post-earthquake tsunami.

For this, we first present an study of some related works. Moreover, we present a formal problem description and its corresponding mathematical model. We test this using a set of randomly generated problem instances. Finally we propose an heuristic algorithm to solve large problem instances in reduced computational times.

2 Related work

The Bus Evacuation Problem was first proposed by [2] who formulated two mixed-integer programming formulations for a bus-based evacuation planning. In their work, the results obtained by the proposed models are compared to the well-known Vehicle Routing Problem (VRP) results.

The vehicle routing problem was proposed in 195 by Dantzig and Ramser [5] who modeled the problem of an heterogeneous fleet of trucks serving the oil demand of a number of service stations from a central depot while minimizing the total travel distance. Five years after that work, in 1964, Clarke [3] would generalize this concept by wrapping it around the vehicle routing problem. The vehicle routing problem considers a fixed starting and ending point for all the vehicles considered in the routing process. The goal of the vehicle routing problem can be defined as the determination of a set of routes for the given vehicles that visit exactly once each client minimizing the total distance traveled. The vehicle routing problem is a variant of the Traveling Salesman Problem (TSP) [6]. The traveling salesman problem defines the problem of an agent who must travel through determined places by the shortest feasible route.

Loyola et al. [12] solve the same bus evacuation problem explained using an heuristic method. They work with a fleet of buses with a fixed capacity. The objective of this problem is to evacuate all people in the shortest time. This, considering a fire scenario in Valparaiso, Chile. The problem here solved is composed by evacuees, buses, rescue blocks, shelters and yards. Unlike to the vehicle routing problem, the possible trips are only those from shelters to the blocks. In their work they decided the trips performed by each bus. A bus can only take one trip at time. Each bus must leave the yard in its first trip. If a bus does not leave the yard at the beginning of the evacuation, it cannot leave later (it does not participate in the evacuation process). The last trip of a bus cannot finish in a collection point. Neither bus nor shelter capacity must not be exceeded. All evacuees must be picked up from the rescue blocks and moved to the shelters. The goal is to schedule the evacuation route of each bus, minimizing the duration of the evacuation, while covering all the demand, without breaking the capacity constraints of shelters and buses.

At [9] Kocatepe et al. propose a model that consider the evacuation of 85+ population with special needs cause of their vulnerabilities as having pets, or having physical and mental limitations [11]. Authors indicates that the shelter characteristics play a critical role in terms of their functions (e.g., pet friendly shelters), and designated capacities. Pet ownership also presents limitations for old adults. Even though pets offer benefits such as being associated with better health, family values, reduce risk of loneliness, depression or cardiovascular disease death [4], they are a major concern during evacuations. In [9] the exact number of pets is not considered, but people that accompany them are counted as the demand. They also consider that every person in the evacuations zones must be evacuated. No road disruptions are assumed. In their work they minimize the total traveled distance to rescue all the people in rescue blocks respecting the shelters capacity. A minimum number of pet-friendly shelters is considered. Moreover, some classical shelters can also be considered to satisfy the rescue demand.

3 Problem description

The Bus Evacuation Problem (BEP) is a route planning problem, in the context of an evacuation in an emergency situation [2]. In this case, earthquakes and tsunamis could cause lost of pets because of the emergency stress. This situation may be prevented by immediate rescue and evacuation.

An animal evacuation scenario is composed by the following elements:

- **Blocks:** group of people in an specific location that requires to get taken to a shelter. These people may have pets.
- **Shelters:** group of safe places that can receive rescued people. These people may have pets. In our problem, shelters can be *petFriendly* or not.

Blocks and shelters locations can be represented as a (directional) connected graph.

The problem here formulated considers a set of constraints:

1. A set of rescue vehicles can only make one trip at time.
2. The rescue vehicles must leave a shelter in its first trip.
3. The last trip of rescue vehicles must finish in a shelter.
4. Shelters capacity must not be exceeded by the demand of blocks allocated.
5. All blocks must be allocated to a shelter.

The objective here is to minimize the total transportation costs of people in evacuation blocks to shelters respecting the capacity constraints.

The model of the problem here solved comes from [9] and is presented below.

Minimize:

$$\sum_{j\in J}\sum_{i\in I} d_{ij} * X_{ij} * P_i \tag{1}$$

Subject to:

$$Y_j = 1, \quad \forall j \in K \tag{2}$$

$$X_{ij} \leq Y_j, \quad \forall i \in I, \forall j \in J \tag{3}$$

$$\sum_{j\in J} X_{ij} = 1, \quad \forall i \in I \tag{4}$$

$$\sum_{j\in J} Y_j \leq N, \quad \forall j \in J \tag{5}$$

$$\sum_{i\in I} P_i * X_{ij} \leq C_j, \quad \forall j \in J \tag{6}$$

where,
I = Set of census block groups
J = Set of all shelters.
K = Set of all designated shelters (pet-friendly or SpNS).
d_{ij} = travel cost between the population block group i and the shelter at location j.
N = number of shelters to be used, defined by the operator.
P_i = demand at point i.
C_j = capacity of the shelter j, by total number of people.

Variables:
X_{ij} = binary variable, 1 if the population block group is assigned to the shelter at j, 0 otherwise.
Y_j = binary variable, 1 if the population block group is assigned to a shelter, 0 otherwise.

Objective function in equation (1) intends to maximize the accessibility of shelters to populations by minimizing the total travel cost (travel time) of population. Equation (3) control that census block groups cannot be assigned to a shelter that is not designated for use. Constraints (4) control that every population block group assigned to exactly one shelter. Equation (5) controls the number of designated shelters to N. Equation (6) controls that the assigned demand (in persons) should not exceed the capacity of the shelter.

4 Instances generation algorithms

We will use three lists and a dictionary to represent the blocks list, shelters list, petFriendly shelters list and the distances dictionary (Block, Shelter):distance.

The number of blocks and shelters, the quantity of shelters to use (N), the percentage of petFriendly shelters and the seed are specified by the user.

The classes Block and Shelter have their own name and a random location defined as their Cartesian coordinates. These coordinates are selected uniformly between 1 and 500. From these, distance between shelters and blocks can be easily computed as euclidean distances. The demand of the blocks are also randomly selected between 25 and 60 as shown in line 4 of algorithm 1.

Algorithm 1 blocksGeneration

```
Require: $BlocksAmount$
Ensure: $I$
1: $I \leftarrow empetyList$
2: $i \leftarrow 0$
3: for $i < BlocksAmount$ do
4:     $P \leftarrow Block(BlockName, RandomCoordX, RandomCoordY, RandomDemand)$
5:     $I[i] \leftarrow P$                                    ▷ The object P is added to the list I
6:     $i \leftarrow i + 1$
7: end for
```

The generation works by calling the functions described below (blocksGeneration shown in algorithm 1, sheltersGeneration shown in algorithm 2, sheltersCapacityGeneration shown in algorithm 3, petFriendlySheltersSelection shown in algorithm 4 and BlockToShelterDistance shown in algorithm 5.

Once the blocks have been generated, we proceed to create the shelters as shown in sheltersGeneration algorithm 2. The capacity of shelters is generated after this proceed.

Algorithm 2 sheltersGeneration

```
Require: $SheltersAmount$
Ensure: $J$
1: $J \leftarrow empetyList$
2: $j \leftarrow 0$
3: for $j < SheltersAmount$ do
4:     $S \leftarrow Shelter(ShelterName, RandomCoordX, RandomCoordY, 0)$
5:     $J[j] \leftarrow S$                                  ▷ The object S is added to the list J
6:     $j \leftarrow j + 1$
7: end for
```

The shelters capacity generation algorithm 3 sets the capacity of shelters according to the Blocks number and the number of shelters to be used (N). For this, we first calculate how many blocks are we going to put on each shelter (amountBxS). Then, we get the number of shelters that will be assigned with their capacity (shelterAs). When the blocks demand are already distributed between shelters, we proceed to set the capacities to the shelters. This in order to promote the feasibility of problem instances. Shelters capacities are set between 80 and 120.

Algorithm 3 sheltersCapacityGeneration

```
Require: $BlocksAmount, SheltersAmount, N, I, J$
Ensure: $Jelementswithcapacityassigned$
1: $amountBxS \leftarrow TRUNCATED(BlocksAmount/N) + 1$
2: $shelterAs \leftarrow TRUNCATED(BlocksAmount/amountPxS) + 1$
3: $i \leftarrow 0$, $j \leftarrow 0$, $z \leftarrow 0$
4: for $j < shelterAs$ do
5:     $dem \leftarrow 0$
6:     for $i < amountBxS$ do
7:         $z \leftarrow 0$
8:         if $z < BlocksAmount$ then
9:             $dem \leftarrow dem + I[z].getDemand()$   ▷ getDemand() returns the demand of the the object
10:            $z \leftarrow z + 1$
11:            $i \leftarrow i + 1$
12:        else break
13:        end if
14:    end for
15:    $J \leftarrow J[j].setCapacity(dem)$            ▷ setCapacity() sets the capacity of the the object
16:    $j \leftarrow j + 1$
17: end for
                        ▷ Not all Shelters will get a capacity in previous steps, these are set below
18: $k \leftarrow SheltersAmount - shelterAs - 1$
19: while $k < SheltersAmount$ do
20:    $cap = RandomCapacity$
21:    $J \leftarrow J[j].setCapacity(cap)$
22:    $k \leftarrow k + 1$
23: end while
```

Once the list of shelters is completely generated, we proceed to randomly choose the shelters that will be petFriendly as shown in line 3 of petFriendlySheltersSelection algorithm.

Algorithm 4 petFriendlySheltersSelection

```
Require: $J$
Ensure: $K$
1: $count \leftarrow 0$
2: while $count < PetFriendlySheltersAmount$ do
3:     $k \leftarrow$ random element from $J$
4:     if $k \notin K$ then
5:         $K \leftarrow k$                    ▷ The object k is added to the list K
6:         $count \leftarrow count + 1$
7:     end if
8: end while
```

Finally, we calculate the distances from each block to the shelters as shown in the Distance from Block to Shelter algorithm 5.

Algorithm 5 BlockToShelterDistance

```
1: for $i \in I$ do
2:     for $j \in J$ do
3:         $t = (Block, Shelter)$
4:         $x = coordXShelter$
5:         $y = coordYShelter$
6:         $dist$ = distance from the Block $i$ to the Shelter $j$ calculated with 7
7:         $t \leftarrow d$  ▷ The tuple t is added to the dictionary d, associated to the distance between them
8:     end for
9: end for
```

The distance between blocks and shelters are calculated using the euclidean distance shown in equation 7.

$$d(X, Y) = \sqrt[2]{(coordXShelter - coordXBlock)^2 + (coordYShelter - coordYBlock)^2} \tag{7}$$

Once blocks and shelters with their respective characteristics are generated, we proceed to write the text file, later transformed to .dat file to solve it with the adaptation of Kocatepe at al. model shown in section 3.

The generator was implemented in Python 3.7. The out file is .dat extension file as shown in the instance example figure 1, hence it can be read by AMPL. The first three lines are generated blocks, shelters and petFriendly shelters. Line five, is N. Lines 7 to 11 are the shelters capacities. Lines 13 to 19 are the demands of each block. Lines 21 to 27 are the distance from each block to each shelter. Finally, lines 29 to 38 are the coordinates of each block and shelters.

5 Experimental results

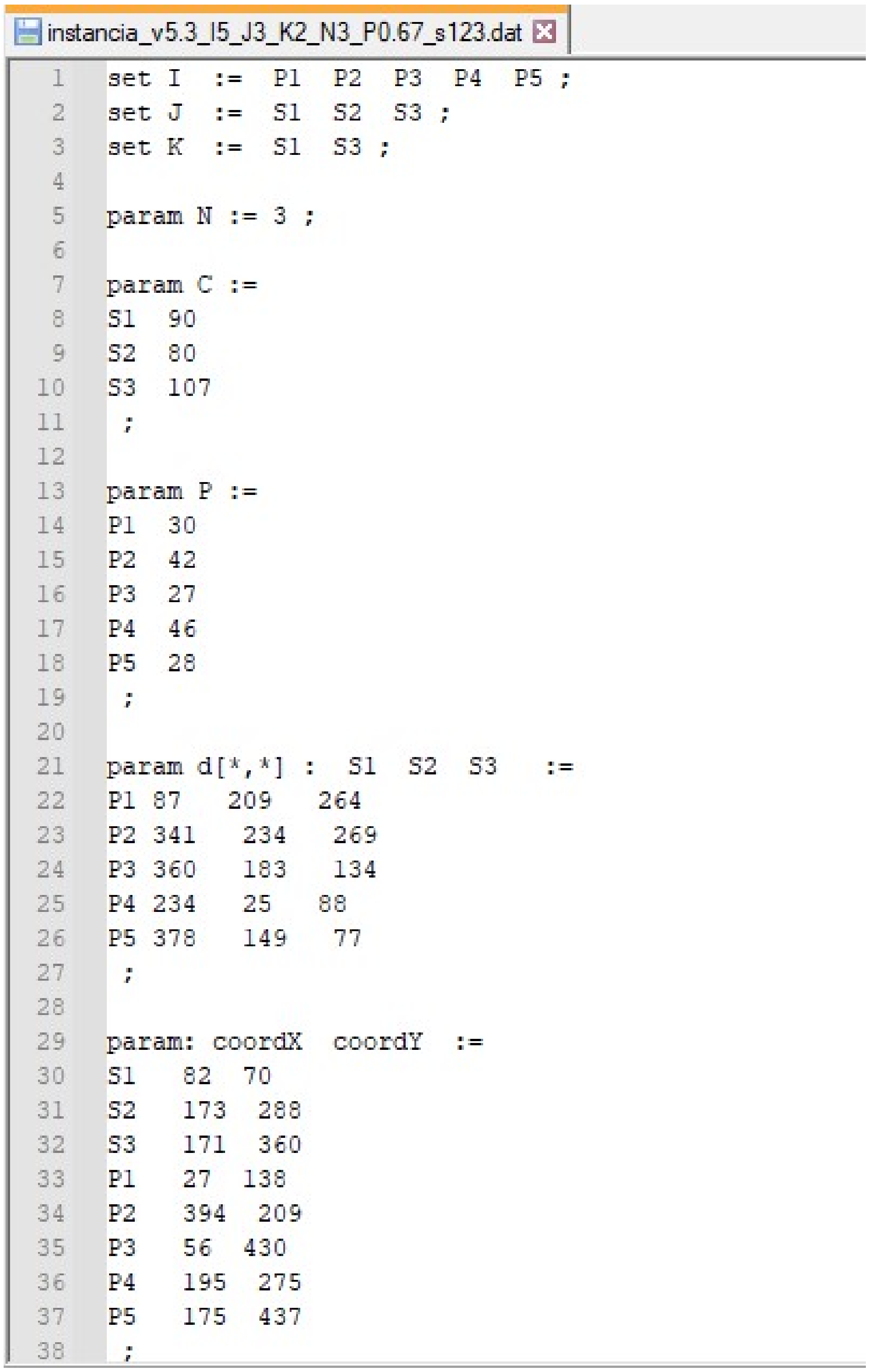
In this section we test the mathematical model using a set of problem instances generated by the generation algorithm described above. These problem instances instances have 100 blocks and 50 shelters in total. In each one, a specific number of shelters must be selected (N) and of these a fixed percentage (per) must be *petFriendly*.

The model was executed using ampl solver. For each instance a maximum total time of 3600 seconds was set. A total of 1,373 instances were tested, of which 151 were feasible. Table 1 summarizes the number of instances that are feasible and comply with the N and percentage of petFriendly shelters in the corresponding row or column.

The lower the percentage of petFriendly shelters, the more likely it is that feasible solutions can be found. A feasible solution is more likely to be obtained if between 60% and 65% of the total shelters are to be used.

The feasible instances that took the longest (most difficult) are described in table 2.

As shown in table 2, the instance that took the longest time required 13,656 seconds, that is, approximately 3 hours and 48 minutes. As can be seen in the table, the instance that took the longest


instancia_v5.3_I5_J3_K2_N3_P0.67_s123.dat

```
set I  :=  P1  P2  P3  P4  P5 ;
set J  :=  S1  S2  S3 ;
set K  :=  S1  S3 ;

param N := 3 ;

param C :=
S1  90
S2  80
S3  107
 ;

param P :=
P1  30
P2  42
P3  27
P4  46
P5  28
 ;

param d[*,*] :  S1  S2  S3   :=
P1 87   209   264
P2 341   234   269
P3 360   183   134
P4 234   25   88
P5 378   149   77
 ;

param: coordX  coordY  :=
S1   82  70
S2   173  288
S3   171  360
P1   27  138
P2   394  209
P3   56  430
P4   195  275
P5   175  437
 ;
```

Figure 1: Instance generated by 5 blocks, 3 shelters, N equals to 3, per equals to 0.67 and using 123 as seed.

time took 13,656 seconds, that is, 3 hours and 48 minutes approximately, making it necessary to use other resolution methods. This will be discussed in more detail in the next section, presenting an heuristic proposal.

Table 1: Feasibility by N and per

feasible	0.6	0.65	0.7	0.75	Total
24	2	1	1	0	4
28	6	2	1	1	10
30	19	19	15	15	68
32	20	20	20	0	60
38	9	0	0	0	9
Total	56	42	37	16	151

Table 2: Instances that took the longest time to solve

Blocks	Shelters	PetFriendly	N	per	Time (sec)
100	50	16	28	0.6	13656
100	50	18	28	0.65	13588
100	50	16	28	0.6	13553
100	50	19	28	0.7	13508
100	50	21	28	0.75	12501
100	50	22	38	0.6	12368
100	50	21	30	0.7	12226
100	50	22	30	0.75	9080
100	50	16	24	0.7	8636
100	50	19	30	0.65	5867

6 Heuristic proposal

Heuristic algorithms have shown being able to solve efficiently a large range of optimization problems. There is a lot of evidence in literature supporting their capabilities [10] [1].

This section presents a local search based heuristic algorithm formulated to solve the Bus routing evacuation with petFriendly shelters problem studied.

6.1 Instance Reader

First, we need to read the problem instances. To do it, we set our global variables, I and J as empty lists and d as an empty dictionary. Then we use the same classes that we used for the generation of Blocks and Shelters (Block and Shelter). The only difference is that the Shelter class has a new attribute called type which defines if it is petFriendly or not.

The reader algorithm works calling the following functions at main function:

- **Obtain Blocks:** gets the list of blocks (I).
- **Obtain Shelters:** gets the list of shelters (J). All shelters are defaulted set as notPetFriendly.
- **Obtain PetFriendly Shelters:** gets the list of petFriendly shelters and set them as PetFriendly.
- **Obtain N:** gets N value.
- **Obtain Capacities:** gets a list of capacities, then, it set the shelters capacities.
- **Obtain Demands:** gets a list of demands, then, it set the blocks demands.
- **Obtain Distances:** gets a list of distances from each block to each shelter, then, it transforms it to a dictionary (d)
- **Obtain Coordinates:** gets the coordinates of all blocks and shelters, then, it sets them as required.

The reader main function gets an instance generated by the Instances Generation Algorithm section 4 and returns I (blocks), J (shelters), d (distances) and N.

6.2 Random Solutions Constructor

To create initial solutions, we used a class called Solucion that contains a constructor with the following attributes:

- **Blocks:** list of blocks, received from the instance reader.
- **Shelters:** list of shelters, received from the instance reader.
- **N**: amount of shelters to be used, received from the instance reader.
- **Distances:** dictionary format (Shelter, Block):distance, received from the instance reader.
- **Solved:** dictionary format Shelter:[Blocks assigned], starts empty.
- **Total Distance:** total distance of the solution it self, starts as zero.
- **Assigned Shelters:** list of shelters that are used in the solution, starts empty.

For the construction of a random solution we use a python file that receives I (blocks), J (shelters), N and d (distances). Then, it calls the RandomSolutionConstructor who creates temporal variables as shown at lines 1 and 2 of the algorithm 6. Solved is the solution dictionary that relates each block to a shelter as Shelter:[Blocks], at line 5 it initializes the shelters with no assigned blocks. Then, it gets into a loop that stops when all blocks are assigned to a shelter, returning True, or when a block cannot be assigned to any possible shelter and the assigned shelters amount reach N, returning False. It is controlled by the RandomSolutionsConstructorMain algorithm 7 looking for possible solution until the RandomSolutionConstructor returns True.

To distribute all the blocks between petFriendly shelters preferably and then in the non petFriendly shelters without exceeding N, first we get a random shelter, a random block and set aggregated as False (lines 9 to 11), then, we check if the selected shelter has no blocks assigned and if N has not been reached (line 12). If it is true, the selected shelter gets into the assigned shelters list, the selected block gets assigned to the shelter in solved, block is removed from the temporal list of blocks and aggregated gets set as True (lines 13 to 16). If not, then we try to assign the block to a shelter that already has assigned blocks, to do this, we check all the assigned shelters until we find one that does not exceed its capacity when we assign the new block (lines 18 to 25).

If at this point it is still not possible to assign the block to a shelter and N has not been reached, we get a list with the shelters that have not been assigned yet (line 27), then, for each shelter in that list, we check if the demand do not exceed the capacity, the block has been assigned and that the shelter is petFriendly (line 28). If it is true, then the selected block gets assigned to the shelter in solved, the shelter gets into the assigned shelters list, the block is removed from the temporal list of blocks and aggregated gets set as True (lines 29 to 34).

If the block is not assigned yet and N is reached, we return False (lines 36 and 37), since it is not possible to find a solution and it will try all again.

At finishing each iteration of while, we set solved as solution (line 40).

Algorithm 6 RandomSolutionConstructor

```
Require: $solutionobjectSolution$
Ensure: $solved$
 1: $tempBlocksList \leftarrow I$
 2: $tempSheltersList \leftarrow J$
 3: $assignedShelters \leftarrow [\ ]$
 4: $solved \leftarrow \{\ \}$
 5: for $(j \in J)$ do
 6:     $solved \leftarrow jrelatedwith[]$
 7: end for
 8: while $(len(tempBlocksList) \neq 0)$ do
 9:     $block \leftarrow RandomChoice(tempBlocksList)$
10:     $shelter \leftarrow RandomChoice(tempSheltersList)$
11:     $aggregated \leftarrow False$
12:     if $((noblocksassignedtoshelter)$ and $(len(assignedShelters) < N))$ then
13:         shelter added to assignedShelters
14:         block added to solved[shelter]
15:         block removed from tempBlocksList
16:         $aggregated = True$
17:     else
18:         for $(shelter$ in $assignedShelters)$ do
19:             $tempCapacity \leftarrow usedCapacity + blockDemand$
20:             if $((tempCapacity \leq shelterCapacity)$ and $(aggregate == False))$ then
21:                 block added to solved[shelter]
22:                 $aggregated = True$
23:                 block removed from tempBlocksList
24:             end if
25:         end for
26:         if $((aggregated == False)$ and $(len(assignedShelters) < N))$ then
27:             $noAssigned \leftarrow$no assigned shelters list
28:             for $(shelter$ in $noAssigned)$ do
29:                 if $((shelterCapacity \geq blockDemand)$ and $(aggregated == False)$ and
        $(shelterType == petFriendly))$ then
30:                     block added to solved[shelter]
31:                     shelter added to assignedShelters
32:                     block removed from tempBlocksList
33:                     $aggregated = True$
34:                 end if
35:             end for
36:         else$((aggregated == False)$ and $(len(assignedShelters) == n))$
37:             $return\ False$
38:         end if
39:     end if
40:     solved set as solucion solution
41: end while
42: $return\ True$
```

Algorithm 7 RandomSolutionsConstructorMain

```
1: $flag \leftarrow False$
2: while $(flag == False)$ do
3:     $flag = RandomSolutionsConstructor()$
4: end while
5: $return$
```

6.3 Solution Reparation

To repair the solution created with the RandomSolutionsConstructor, we use a Hill climbing approach.

Hill climbing algorithm is a technique which is used for optimizing the mathematical problems throw a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value [8].

To apply the Hill Climbing Algorithm we had to create a few functions that helped us to get the Neighbours solutions:

- **get Blocks:** receives the assigned Shelters list and the solved dict. Returns a list with one random blocks of each shelter and a temporal solved dictionary that has removed the blocks mentioned before.
- **reassignBlocksController:** receives the list with one random blocks of each shelter, the assigned Shelters list and the temporal solved dictionary. Calls the Re-assign Far Blocks function with the parameters mentioned before. It will try to get a new solved until the function stops returning False. Always returns a solution.
- **reassignBlocks:** receives the the parameters mentioned before. Returns false if it cannot solve it and returns the found solution if it can be solved.

Algorithm 8 getNeighbours

```
Require: $solutionobjectSolution$
Ensure: $neighboursList$
 1: $neighbours \leftarrow [\,]$
 2: $deletedBlocks, tempSolved \leftarrow GetBlocks(assignedShelters, solution)$
 3: $i = 0$
 4: while $(i < len(neighboursAmount))$ do
 5:     $solved \leftarrow reassignBlocks(deletedBlocks, assignedShelters, tempSolved)$
 6:     $totalDistance \leftarrow solvedTotalDistanceCalculation$
 7:     $newNeighbour \leftarrow SolucionObject(I, J, n, d, solved, totalDistance, assignedShelters)$
 8:     $neighboursList \leftarrow newNeighbour$
 9:     $i = i + 1$
10: end while
11: $return\ neighboursList$
```

7 Conclusions and future work

In this paper, we studied the Bus evacuation problem with petFriendly shelter. For this, we implemented a mathematical model based on [9]. We implemented a instances generator and tested the model solving these randomly generated instances. The results were analyzed, obtaining that the lower the percentage of petFriendly shelters, the more likely it is that feasible solutions can be found. Also, it is more likely to be obtained if between 60% and 65% of the total shelters are to be used. From these results, we obtain that the average distance was 6258 and a standard deviation of 1303, on the other side, the average time was 1114 seconds (19 minutes) and a standard deviation of 2929 seconds (49 minutes) were from a total of 151 feasible instances, 61 took less than a minute, 10 more than a hour and 80 in between.

Moreover, we proposed an heuristic algorithm to improve the resolution time of these problem instances. A Hill Climbing based algorithm as proposed. Preliminary results show the ability of the heuristic approach to generate and consistently improve the quality of solutions.

As future work, we plan to test the proposed heuristic more deeply and obtain robust results to backup the proposal. Moreover, this analysis can motivate the design of new components to improve the results obtained by the initial version here proposed.

References

[1] Emile Aarts and Jan K Lenstra. *Local Search in Combinatorial Optimization*. 2003.

[2] Douglas R. Bish. Planning for a bus-based evacuation. *OR Spectrum*, 33:629–654, 2011.

[3] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

[4] Susan Phillips Cohen. Can pets function as family members? *Western Journal of Nursing Research*, 24(6):621–638, 2002. PMID: 12365764.

[5] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Manage. Sci.*, 6(1):80–91, October 1959.

[6] M. M. Flood. The traveling-salesman problem. *Operations Research*, 4(1), 1956.

[7] Dick Green. Chapter 10 - animal behavior. In Dick Green, editor, *Animals in Disasters*, pages 93–104. Butterworth-Heinemann, 2019.

[8] JavaTpoint. Hill climbing algorithm in ai - javatpoint.

[9] Ayberk Kocatepe, Eren Erman Ozguven, Mark Horner, and Hidayet Ozel. Pet- and special needs-friendly shelter planning in south florida: A spatial capacitated p-median-based approach. *International Journal of Disaster Risk Reduction*, 31:1207–1222, 2018.

[10] Zbigniew Michalewicz and David B Fogel. *How to solve it: Modern heuristics*. Springer, Berlin, Germany, 2 edition, March 2013.

[11] Eren Erman Ozguven, Mark W. Horner, Ayberk Kocatepe, Jean Michael Marcelin, Yassir Abdelrazig, Thobias Sando, and Ren Moses. Metadata-based needs assessment for emergency transportation operations with a focus on an aging population: A case study in florida. *Transport Reviews*, 36(3):383–412, 2016.

[12] Javiera Loyola Vitali, María-Cristina Riff, and Elizabeth Montero. Bus routing for emergency evacuations: The case of the great fire of valparaiso. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 2346–2353, 2017.