Multiple Heterogeneous Vehicle Routing Problem allowing Simultaneous Delivery and Pick-up from Single Depot while minimizing the Distance travelled by all vehicles to complete the entire operation

June 9, 2022

Introduction

In this problem, we have a single depot with multiple types of vehicles which shall cater to demands at the relief centres (nodes) as well as pickup evacuees in case of emergency and disasters. Each Vehicle Type (all distinct) consists of homogeneous vehicles.

Sample of the Inputs required

1. Table 1. Indicating Vehicle Specifications for each type present at the depot and being considered for the operation

Sample Vehicle details at the Depot					
Vehicle Type [VT]	Number of this type of Vehicles [VN]	Capacity [VQ]	Variable Cost [VS]	Fixed Cost [VC]	Layers differentiating attributes w.r.t. OpenStreetMaps (Road Vehicle Compatibility for generating appropriate layers from the original Network)
V1	25	350	1	50	highway = Motorway, Trunk, Primary, Secondary, Tertiary
V2	15	450	1.04	80	highway = Motorway, Trunk
V3	15	600	1.08	120	highway = Motorway, Trunk, Primary, Secondary, Tertiary, Unclassified, Residential
V4	10	800	1.14	150	highway = Motorway, Trunk, Primary

2. Table 2. Indicating PickUp & Delivery details at respective Nodes with their locations

Sample Node Details				
Node Number	Latitude	Longitude	PickUp	Delivery
0 [Warehouse / Depot]	81.5397	58.2791	[p]	[d]
1	98.8912	87.9014	69.9888	69.5949
2	86.5439	0.0522	3.3604	63.8531
⋮	⋮	⋮	⋮	⋮
149	14.2041	10.5709	76.55	26.9062
150	62.0959	16.646	28.7498	18.8662

3. Table 3. Distance Matrices for each type of vehicle (i.e. for each k). Here certain distance matrices shall be repeated for multiple vehicles in the same category/type.

Formulation:-

This formulation allows only single vehicles to visit each node {similar to the original disaggregated formulation considering homogeneous vehicles as mentioned in Amico, Righini & Salani 2006 as well as for heterogeneous vehicles as in Avci Topaloglu 2016}. Network layers are specific to vehicle types such that each type of vehicle may be imagined to travel in its Type-Layer. Each layer can be be further improved with their individual vehicle-type specific variable costs.

1. Notations, Sets & Decision Variables:

   (a) N refers to the set of all Relief Centres, where $|N| = n$

   (b) N_0 refers to the extended set of all Relief Centres as well as including the Depot, where $|N_0| = n + 1$

   (c) k refers to the set of all Vehicle Types [VT]

   (d) x_{ijk} is a binary variable which refers to the k^{th} type of vehicle going from node i to node j, $(i \neq j)$ on layer k, when it equals 1. The network layers are generated for specific vehicles types according to their compatibility with the road.

   (e) y_{ijk} refers to the amount of pickup (humans) being carried by a vehicle of type k on its specific layer, also type k, between nodes i and j, $(i \neq j)$.

   (f) Similarly z_{ijk} refers to the amount of delivery (Food, Water, Medicine, Sanitary Items) being carried by a vehicle type of k, on its specific layer k, between nodes i and j, $(i \neq j)$.

   (g) C_{ijk} is the cost of vehicle of type k travelling from node i to node j, $(i \neq j)$. This may be imagined as the normal cost matrix/distance matrix being extended to different layers specific to vehicle types. These types will help in differentiating road types to check compatibility between vehicle and road. The general cost matrix will be generated by finding the shortest distance between two nodes in the specific layer. The initial network layer may be therefore filtered according to vehicle types so that the network layer for say Vehicle Type V2 may contain the road segments on which only V2 type vehicles will be able to travel. The cost matrix, which is to be generated will be the shortest distances for each of these specific layers which is further indicated by the suffix k in C_{ijk}.

2. Distance Minimization Objective Function:

$$\sum_{k \in VT} \sum_{i \in N_0} \sum_{j \in N_0,\, (i \neq j)} C_{ijk} x_{ijk} VS_k + \sum_{j \in N} \sum_{k \in VT} x_{0jk} VC_k$$

   Here VS_k is a variable cost which may be compared to Distance dependant elements like fuel or emissions. VC_k is the fixed cost which may be considered as the per tour expense towards maintaining and sending a crew consisting of the driver and personnel helping in Disaster Relief which may be considered the same for similar types of vehicles.

3. Constraints:

   (a) At most a single vehicle may attend any relief centre/node.

$$\sum_{j \in N_0,\,(i \neq j)} \sum_{k \in VT} x_{ijk} \leq 1 \qquad \forall i \in N$$

   Removing this constraint shall not allow complete split delivery (for split delivery, each vehicle should have its own layer variables which is also necessary for time minimization), but allow some number of vehicles to tend to a node through different paths. This may also be considered since the result of this formulation without this constraint is better; and may be chosen as per real-scenarios by the operator.

   (b) Ensuring the same number of each type of vehicles entering any node also leaves it.

$$\sum_{j \in N_0,\,(i \neq j)} (x_{ijk} - x_{jik}) = 0 \qquad \forall k \in VT \quad \& \quad \forall i \in N_0$$

   (c) Ensuring at most VN_k vehicles (where VN denotes the Vehicle Number for the particular layer of vehicle type as per Table 1. above) are allowed to exit the depot for the specific vehicle layer types

$$\sum_{j \in N} x_{0jk} \leq VN_k \qquad \forall k \in VT$$

   (d) Flow limitation constraints of PickUp and Delivery are provided to get the exact flows within the respective flow variables of pickup and delivery.

      i. Assigning the value 0 to all outgoing pickup values from the Node

$$y_{0jk} = 0 \qquad \forall j \in N \quad \& \quad \forall k \in VT$$

      ii. Assigning the value 0 to all incoming delivery values from the Node

$$z_{i0k} = 0 \qquad \forall i \in N \quad \& \quad \forall k \in VT$$

   (e) Flow Equations for each Relief Centre

      i. Ensuring the pickup constraints are satisfied

$$\sum_{k \in VT} \sum_{j \in N_0,\,(i \neq j)} (y_{ijk} - y_{jik}) = p_i \qquad \forall i \in N$$

ii. Ensuring the delivery constraints are satisfied

$$\sum_{k\in VT}\sum_{j\in N_0,\,(i\neq j)}(z_{jik}-z_{ijk})=d_i \qquad \forall i\in N$$

(f) Ensuring the sum of flows from and to the origin (Depot) is equal to the total Pickup and Delivery

i. Ensuring the sum of inflow into the origin is equal to the total Pickup demand

$$\sum_{i\in N}\sum_{k\in VT} y_{i0k}=\sum_{i\in N} p_i$$

ii. Ensuring the sum of outflow from the origin is equal to the total delivery demand

$$\sum_{i\in N}\sum_{k\in VT} z_{0ik}=\sum_{i\in N} d_i$$

(g) Limiting the vehicle capacities according to the vehicle types

$$y_{ijk}+z_{ijk}\leq x_{ijk}VQ_k \qquad \forall i,j\in N_0,(i\neq j) \quad \& \quad k\in VT$$

(h) Variable Constraints

$$y_{ijk},z_{ijk}\geq 0 \qquad \forall i,j\in N_0,\ (i\neq j) \quad \& \quad k\in VT$$

$$x_{ijk}=\{0,1\} \qquad \forall i,j\in N_0,\ (i\neq j) \quad \& \quad k\in VT$$

Some Sample Results:-

Table 3. Indicates some sample optimal while considering an increasing number of Nodes starting from 1. {The Depot Node is indicated with number 0 and is always present.}

Upto Node	VN = 1 for all VT		VN = 2 for all VT		VN = 3 for all VT	
	Objective	Time (s)	Objective	Time (s)	Objective	Time (s)
1	118.674	0.376	118.674	0.035	118.674	0.040
2	231.483	0.170	231.483	0.099	231.483	0.146
3	231.874	0.577	231.874	0.585	231.874	0.507
4	272.807	0.882	272.807	0.900	272.807	0.858
5	331.802	0.618	331.802	0.691	331.802	0.700
6	335.277	2.293	335.277	1.463	335.277	1.404
7	377.043	3.770	377.043	3.001	377.043	4.385
8	379.861	13.562	379.861	4.467	379.861	50.275
9	431.271	27.643	413.154	7.927	413.154	25.219
10	442.615	72.627	423.658	30.537	423.658	30.922
11	449.578	179.491	423.861	25.523	423.861	32.524
12	451.893	4956.401	425.906	90.578	425.906	751.899
13	467.649	1105.888	462.218	14437.194	462.218	27730.046
14	511.349	16624.971	511.349	28803.394	511.349	5054.949
15	615.053	55478.563				

Indicating Optimal Solution (Objective Values after reported Time) obtained from Coin-OR CBC Engine when used in PuLP using PYTHON Programming. The coloured values indicate same optimal results.

Developing the Heuristic

We name the Heuristic as ATSAH which refers to ATS-Algorithm-Heuristic where ATS refers to the same ATS heuristic as described in the 2016 paper by Avci Topaloglu. The Algorithm here refers to the implementation of the heuristic Algirithm as per the logic in the referred paper's algorithm. Avci Topaloglu has subtle differences between the implementation of their described heuristic in their flowchart, algorithm and text. We have implemented both the HLS and ATS heuristics using the three different logics as mentioned in their algorithm, flowchart and text and have chosen the algorithm's logic. The Tabu list cannot be included here since this heuristic is not deterministic in terms of the output of the Decoding Mechanism. This however seems to make this superior. Below is the description of the heuristic components and their integration:-

1. Neighbourhood Structures

   To generate new Node Sequences from the set of all Nodes we use some Neighbourhood Structures. All Node Sequences always starts with the Depot Node which is the 0^{th} Node. The importance of a Node Sequence is mentioned further in the Encoding Mechanism. Here we use six different Neighbourhood Structures as mentioned below. The importance of these six Neighbourhood Structures is immense. The

ATSAH Heuristic is extremely fast and gives better results than the referred paper of Avci Topaloglu in one-tenth of the average time for the toughest instances of 150 Nodes with 4 Vehicle Types. However since the base algorithm for the ATSAH is the ATS Decoding Mechanism itself; ATSAH could never give better solutions than the ATS. This is because the working of the ATSAH is exactly the same as that of ATS with the minor difference of ATSAH using a probabilistic/-greedy/andomised approach to accept the Edges of the Node Sequence as generated during the encoding mechanism. For a given Node Sequence, these same edges would also be generated in the ATS; and it would find the best combination of these edges using the shortest path algorith (as mentioned in their paper, they used Dijkstra's dynamic programming algorithm.) Therefore, we understand that ATSAH's success cannot lie in the ability to select good edges. Therefore it must then lie in the generation of the Node Sequences. Here we need to understand that the Node Sequences generated using Avci Topaloglu's Decoding Mechanisms were focusing on Neighbouring Node Sequences with small changes. Therefore drastic changes in the Node Sequences were not possible. However once we provided new Neighbourhood Structures, the combined effect of the greedy approach and capability to generate Node Sequences from fresh territories started improving the solutions solutions. The initial Node Sequence is the sequential sequence of Nodes starting from 0 to the maximum Node Number.

(a) Shuffle Random:- Generating a randomly shuffled Node Sequence from the existing Node Sequence
(b) Reversal:- Reversing an arbitrary length (we kept it ¿3) of the Node Sequence, not including the first Node
(c) Single Insertion:- Taking one Node from the Node Sequence and placing it anywhere else
(d) General Swap:- Swapping any two Nodes of the Node Sequence
(e) Adjascent Swap:- Swapping adjacent Nodes of the Node Sequence
(f) SubArray Transplant allowing intermediate Operations:-
    i. Generate the SubArray randomly using the below methods:-
        A. Take a random length of consecutive elements from the Node Sequence
        B. Take elements from arbitrary locations in the Node Sequence, such that the number of elements may be equal to a generated random length less than the length of the Node Sequence

        . For the ATSAH solutions discussed below, the probability of using the first method was double that of using the second method.

ii. Remove the Sub-Array from the Node Sequence. Now we have a SubArray of some random length and the shorter Node Sequence without the Sub-Array elements.

iii. Perform any 1 of the below operations on the SubArray:-

A. Reverse the SubArray

B. Shuffle the SubArray elements randomly

For the ATSAH considered; either of the above operations were allowed to happen with one-third probability; thereby allow the SubArray to remain intact as well.

iv. Perform Operations on the shortened Node Sequence without SubArray elements:-

A. Reverse the Node Sequence

B. Shuffle the SubArray elements randomly

For the ATSAH considered; either of the above operations were allowed to happen with one-third probability; thereby allow the shortened Node Sequence to remain intact as well.

v. Insertion of the SubArray within the shortened Node Sequence is done using either of the following ways:-

A. Single Bulk Insertion of the SubArray, in a reversed manner, at a random location in the shortened Node Sequence.

B. Insert each element of the SubArray at random locations in the shortened Node Sequence

In our case, the first method had double the probability of being used than the second. At the end of this process, we get a new Node Sequence containing all the original elements only once.

In all the above cases, it needs to be ensured that the first Node remains the Depot.

2. Decoding Mechanism:- For a Node Sequence, which is the giant tour representation itself as mentioned in Avci Topaloglu 2016, we use a similar decoding mechanism to generate feasible routes. We first describe the nomenclature used and then the process.

   (a) Edges:- Each edge is a directed connection from one Node of the Node sequence to another. Since the Node sequence is an array, the Edges always start from a lower positional number and end at any Node which is in a higher array position. An Edge contains minimum 2 Nodes, one starting and one stopping Node and its uniqueness is identified by the following information:-

   i. Starting Node

ii. Stopping Node

iii. Vehicle Type Used:- Each edge represents a tour being traversed by a Vehicle of among the specified types.

iv. Cost of the Edge:- This is an additional information we retain to calculate the objective value. To determine the cost of the Edge we explain what the Edge represents. Each Edge represents a tour which starts from the Depot, sequentially covers all Nodes which are present at higher array positions than the starting Node of the Edge, upto the last Node of the edge which is the stopping Node itself, and then back to the Depot. If this route is not feasible for any Vehicle Type, then the Edge would not exist. In case this is feasible, the cost of this tour is the cost of the Edge. In this SDP problem, we allow any Node to be visited by only one Vehicle and therefore use the following process to check feasibility of an Edge:-

   A. Initialise a DynamicVehicleCapacityLeft variable which would have the initial value to be equal to the capacity of the concerned Vehicle Type. We shall keep track of this variable at each Node of the route, including the Depot. If it's value becomes negative ever, then the route is infeasible and the Edge is not constructed. Otherwise we get a feasible route.

   B. We see that at the Depot, we need to load all the Delivery Quantities into the Vehicle and therefore update the DynamicVehicleCapacityLeft variable by substracting the sum of all demands of the route (i.e. for all Nodes sequentially at higher array positions that the starting Edge position upto the least Node where the Edge ends).

   C. In each subsequent Node, we see that first the delivery quantity gets reduced from the Vehicle's capacity and the respective PickUp quantity get's added. Therefore for each Node we update the DynamicVehicleCapacityLeft variable by adding the delivery quantity of the Node and substracting the PickUp quantity.

(b) Vehicles remaining to be used of each Type:- Initially, this is the information regarding the number of vehicles of each type which are available to be used for the operation. As we use up some of these Vehicles for creating some routes (Edges) we shall keep updating this. We will therefore only be creating Edges for those Vehicle Types which are available here. This information is unique for each Graph and therefore stored differently.

(c) Mother Graph:- Each Mother Graph consists of some Edges. One of these Edges must start from the first Depot Node and end at another Node down the array of the Node Sequence. The next Edge starts from this end Node, and ends further down the array of the same Node Sequence. Thus the Mother Graph is a continuous series of such connected Edges, the first of which starts at the Depot. Completed Mother Graphs are moved to a set of Final Graphs. By completed, we mean that the Mother Graph's set of Edges are such that one of them starts at the Depot, and ends at another Node, from which another Edge starts; and this process ultimately continues such that no more Nodes are left, i.e. there is an Edge starting from the end Node of another Edge, and ending at the last Node of the Node Sequence. These completed Mother Graphs will not not give rise to new Daughter Edges and are therefore moved to the Final Edge Graphs.

Each Mother graph also has another information of the number of Vehicles left to be used of each type. Whenever a new Daughter Edge is added to its Mother Graph, this information gets updated, (i.e. the number of Vehicles of the Vehicle Type as used by the Daughter Edge, is reduced by 1).

Each Mother Graph has another additional information regarding the ending Node position of its last Edge. This is updated every time new Daughter Edges are added, which become equal to the ending Node of the inserted Daughter Edge. This information helps to create the next Edge as this Node will serve as the starting Node for the next Edge(s). When Mother Graphs become completed and are moved to the Final Edge Graph set, this information is dropped.

(d) Final Graph:- Each Final Graph is a set of Edges. One of these Edges of a Final Graph starts at the Depot and ends at another Node from where the next Edge starts; and this process continues until the last Edge ends at the last Node of the Node Sequence. Unlike Mother Graphs, Final Graphs have no leftover Nodes to consider further.

The leftover Vehicles for each Final Graph represents the final un-utilized Vehicles; in case this solution is accepted.

(e) Mother Edge Graphs:- This is a set, where each element represents Mother Graphs. Each of these Mother Graphs generate multiple Daughter Edges. Few of these Daughter Edges are selected according to the Edge Selection criterias. Each selected Daughter Edge is added to a copy of its Mother Graph and this updated Mother Graph is added to either Final Edge Graphs or Next Mother Edge Graphs, depending on whether the selected Daughter Edge ended at the last Node of the Node Sequence or not respectively.

(f) Next Mother Edge Graphs:- This is a set, where each element represents Mother Graphs. Every time new Daughter Edges are selected and are added to it's respective copy of it's Mother Graph, it is checked whether the Mother Edge should become next Mother Edge, or whether it may be moved to the set of Final Graphs. Therefore this set serves as a dynamic temporary set of next Mother Graphs. So after all operations on Mother Edge Graphs is over, we initialize Next Mother Edge Graphs to be the set Mother Edge Graphs and continue until no Mother Graphs enter the Next Mother Edge Graphs.

(g) Final Edge Graphs:- This is the set of Final Graphs.

Initially all three sets, (i.e. Mother Edge Graphs, Next Mother Edge Graphs and Final Edge Graphs) are empty. The algorithm starts generating Mother Graphs and keeps populating the set of Mother Edge Graphs. It expands each Mother Graph present in the Mother Edge Graphs by generating Daughter Edges for a Mother Graph, selecting few Daughter Edges greedily, adding these selected Daughter Edges to its respective copy of it's Mother Graph, dumping each newly updated Mother Graph into Next Mother Edge Graphs/Final Edge Graphs. After all Mother Graphs in the set of Mother Edge Graphs have been considered; subsequently the Mother Edge Graphs is updated to become equal to Next Mother Edge Graphs and this process is repeated until the set of Mother Edge Graphs become null; in which case the Final Edge Graphs is taken into consideration. Each Final Graph consists of a number of Edges which have its correspondingly associated costs. All these Edge costs are added to get the total cost of it's corresponding Final Graph. Then the cost of all Final Graphs, in the set of Final Edge Graphs, is compared and the least cost Final Graph is chosen as the solution of the Decoding Mechanism.
In case the number of Vehicles are not sufficient, the left-over Vehicles remaining to be used for each Type will attain 0 values for a Mother Graph even when it's not complete, (i.e. there are left Nodes). Such Mother Graphs need not be proceeded further with, and would be dropped off.

(a) Starting the Decoding Mechanism:- We start by iterating over all Mother Graphs present in Mother Edge Graphs. At the beginning, since the set of the Mother Edge Graphs would be null, we may create an initial Mother Graph with no Edges, left-over Vehicles to be used as equal to the original availability, and the starting Node Location as the Depot Node. From this information, new Edges are created for this initial Mother Graph and dumped to the set of Daughter Edges from which some are selected using Edge Selection Strategies. When these selected Edges are added to new copies of this initial Mother Graph for each selected Edge, to generate new Mother Graphs for each new selected Edge, the newly generated Mother Graphs have updated information regarding the left-over Vehicle Types as well as the next starting Node, which would be the ending Node of the selected Edge. It must be noticed that new Mother Graphs are created for each selected Edge and then each of these Mother Graphs are further allowed to generate Daughter Edges and create new Mother Graphs for each of the few selected Daughter Edges. If the number of Daughter Edge creation is big (even a constant 2); the processing time taken would increase enormously. We will innovatively estimate the number of combinations of Mother Graphs which may be created when considering all Daughter Edges subsequently and thereby discuss the Edge selection strategies used and it's importance in speeding up the solution.

(b) Edge selection strategy:- From among the set of Daughter Edges, we select a few Edges using a host of Edge selection strategies explained later. It is to be noted that these Edge selection strategies must play a key role in improving the solution time. For each of the selected Daughter Edges, it is checked whether the stopping Node is the last Node of the Node sequence. If the last Node of the Node Sequence is also the ending Node of the Edge, then this respective selected Daughter Edge is added to a copy of its Mother Graph and this updated Mother Graph is added to a set of Final Graphs, termed Final Edge Graphs. If the Daughter Edge doesn't end at the last Node of the Node Sequence, then this respective selected Daughter Edge is added to a copy of its Mother Graph and this updated Mother Graph is added to a set of the Next Mother Edge Graphs.

(c) Continuing the Decoding Mechanism:- For each Mother Graph in Mother Edge Graphs, we generate new Daughter Edges as follows:- (Each Daughter Edge starts from the starting Node, only using each Vehicle Type remaining to be used, both information being available in each Mother Graph.)

   i. Iterate over available Vehicle Types (i.e. Vehicle Types which have positive number of Vehicles available to be used)

   ii. For each Vehicle Type available we start with creating the smallest Edge from the starting Node and keep increasing the Edge length keeping the same starting Node. The starting Node is the same as the stopping Node of the last Edge. In case there are no pre-existing Edges, the starting Node is the first Depot Node of the Node-sequence.

   iii. We start from the starting Node and it's next Node in the Node Sequence as the first Edge. If this is found to be feasible, this Edge is retained in a dump/set of Daughter Edges. We then include the next Node of the Node Sequence and see whether this increased Edge is feasible. If this is also found feasible, this is similarly dumped into the set of Daughter Edges. This process of increasing the Edge Length by taking in more consecutive Nodes is carried out until we find an infeasible Edge. Once we find an infeasible Edge, it is rejected, and we move on to the next Vehicle Type in the iteration.

   iv. It is to be noted that the starting Node has not yet changed. The starting Node and remaining Vehicles of each Type to be used, are updated during addition of the Graph into the Next Mother Edge Graphs (or dropped when adding the Graph into Final Edge Graphs). From the above iterations, we get multiple Edges from the same starting Node for some/all of the available Vehicle Types. All these Edges are dumped into the set of Daughter Edges.

   v. Some Daughter Edges are selected to be added to their respective copy of the Mother Graphs with updated information of left-over Vehicles of each type and the next starting Node.

   vi. These Mother Graphs are either added to Next Mother Edge Graphs; or to the Final Edge Graphs if the Mother Graph contains all Nodes in the sequence.

   vii. The set of Mother Edge Graphs is equalled to the Next Mother Edge Graphs for further generation of Daughter Edges for the new Mother Edges.

(d) The Decoding Mechanism is continued till the set of Mother Edge Graphs becomes null. After that, each Final Graph in Final Edge Graphs is valued and the lowest cost Final Edge is given as the output of the Decoding Mechanism. The value of a Final Graph is the sum of the cost of all it's Edges, which is also the Objective Function Value.

3. Edge Selection Strategies:-
   Logically accepting the best Daughter Edges is a challenging task and further improvement must be done in this regard.We employ the following selection strategies:-

   (a) Accepting only the Longest Edge:- The length of a Edge is equal to the number of Nodes it encompasses within its starting and ending Nodes. Since these Edge selection criterias are only applicable for Daughter Edges from the same Mother Graph, the longest Daughter Edge must have its ending Node position in the array of Node Sequence as the highest among the other Daughter Edges. The length of an Edge may be calculated by finding the difference between the Edge's ending Node position in the Node Sequence and the Edge's starting Node position in the Node Sequence.

(b) Accepting the minimum Cost Edge w.r.t the Number of Nodes in that Edge:- The Edge Cost is divided with the Edge Length and the Edge which has the lowest of this value is selected.

(c) Accepting the minimum Cost Edge w.r.t the cumulative Number of Nodes:- Cumulative Number of Nodes refers to the addition of integers starting from 1 upto the length of each respective Edge. The Edge Cost is divided by this number and the Edge with the lowest value is taken.

(d) Accepting the Edge with the minimum Distributed Cost, where Distributed Cost is computed by adding the below two different values for a concerned Edge:-

   i. Variable Cost of the Edge divided by the Cumulative Number of Nodes
   ii. Fixed Cost of the Edge divided by the Number of Nodes

   If:-
   L is the Edge Length equal to the number of Nodes in the Edge, and
   CN is Cumulative Number of Nodes
   Then, $CN = L * (L + 1)/2$

   The Variable Cost of an Edge is obtained by subtracting the Fixed Edge Cost from the Total Edge Cost which is equal to $VC[VT]$, where VT is the Vehicle Type of that Edge

(e) Using any of the above (a, b, c, d) Edge selection methods with uniform probability of strategy selection.

(f) Accepting either the Longest or the Second Longest Edge: In case of the ATSAH for which the results have been reported, the Longest Edge is selected with double probability than that of selecting the second Longest Edge.

(g) Accepting the Edge with the minimum Cost w.r.t. number of Nodes, or the Second-Minimum: In case of the ATSAH for which the results have been reported, the Edge with this minimum value was selected with double probability than that of selecting the Edge with the second-minimum value.

(h) Accepting the Edge with the minimum Cost w.r.t. cumulative number of Nodes, or the Second-Minimum: In case of the ATSAH for which the results have been reported, the Edge with this minimum value was selected with double probability than that of selecting the Edge with the second-minimum value.

(i) Accepting the Edge with the minimum Distributed Cost, or the Second-Minimum, with the Edge with this minimum values being selected with double probability.

(j) Using any of the above (f, g, h, i) Edge selection methods with uniform probability of strategy selection.

(k) Using any of the above (f, g, h, i) Edge selection methods with uniform probability of strategy selection, and selecting both the minimum and the second minimum Edges with certain probabilities. These probabilities may be adjusted further and for the ATSAH solutions reported, the probability of accepting the second-minimum-Edge was 22/27 and that of accepting the Edge with the minimum value was kept at 21/27.

(l) Random Acceptance:- This strategy randomly accepts one Daughter Edge.

(m) Random Acceptance 2:- This strategy randomly accepts atleast one Daughter Edge. It may accept another random Daughter Edge, with a small probability. In case of the ATSAH solutions discussed here-in-below, this probability was taken 10%. Increasing this probability will increase solution time drastically and may not improve the Objective Value much.

(n) Using any of the above Edge selection methods with uniform probability of strategy selection. In this case some of the probabilities of the strategy mentioned in (k) was updated, so that the probability of accepting the Edge with minimum-cost as used for the below solutions, was 407/500, and the probability of accepting the Edge with second-minimum value as used was 11/20. Also the probability of accepting another random Daughter Edge in the strategy l was updated to be 25%.

Each of the Edge selection strategies (a to m) are used for the Decoding Mechanism for each evaluation of a new Node Sequence. The last Edge selection strategy n is used to generate the initial solution, as well as used once during every iteration for updating the best solution, which is also highlighted in the flowchart.

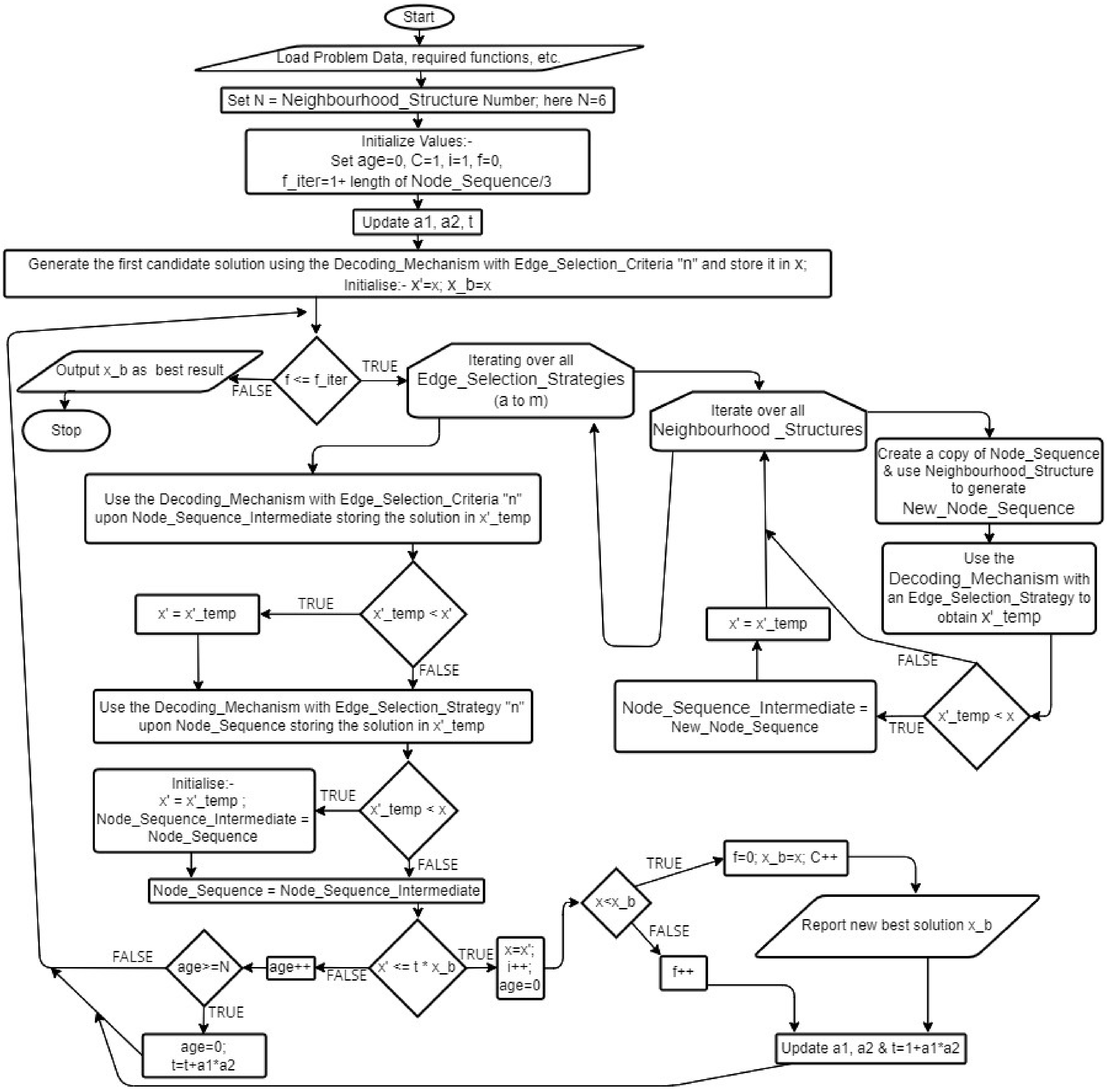
Start
Load Problem Data, required functions, etc.
Set N = Neighbourhood_Structure Number; here N=6
Initialize Values:- Set age=0, C=1, i=1, f=0, f_iter=1+ length of Node_Sequence/3
Update a1, a2, t
Generate the first candidate solution using the Decoding_Mechanism with Edge_Selection_Criteria "n" and store it in x; Initialise:- x'=x; x_b=x
f <= f_iter
TRUE
FALSE
Output x_b as best result
Stop
Iterating over all Edge_Selection_Strategies (a to m)
Iterate over all Neighbourhood _Structures
Create a copy of Node_Sequence & use Neighbourhood_Structure to generate New_Node_Sequence
Use the Decoding_Mechanism with an Edge_Selection_Strategy to obtain x'_temp
x'_temp < x
FALSE
TRUE
Node_Sequence_Intermediate = New_Node_Sequence
x' = x'_temp
Use the Decoding_Mechanism with Edge_Selection_Criteria "n" upon Node_Sequence_Intermediate storing the solution in x'_temp
x'_temp < x'
TRUE
x' = x'_temp
FALSE
Use the Decoding_Mechanism with Edge_Selection_Strategy "n" upon Node_Sequence storing the solution in x'_temp
x'_temp < x
TRUE
Initialise:- x' = x'_temp ; Node_Sequence_Intermediate = Node_Sequence
FALSE
Node_Sequence = Node_Sequence_Intermediate
x' <= t * x_b
TRUE
FALSE
x=x'; i++; age=0
x<x_b
TRUE
f=0; x_b=x; C++
FALSE
f++
Report new best solution x_b
Update a1, a2 & t=1+a1*a2
age++
age>=N
FALSE
TRUE
age=0; t=t+a1*a2

Figure 1: Flowchart representing the Logic

References

1. Avci, Mustafa, and Seyda Topaloglu. "A hybrid metaheuristic algorithm for heterogeneous vehicle routing problem with simultaneous pickup and delivery." Expert Systems with Applications 53 (2016): 160-171.