

# ### Background The modeling and solution process for fuzzy propositions in the context of the Green Vehicle Routing Problem with

User:

## Background

The modeling and solution process for fuzzy propositions in the context of the Green Vehicle Routing Problem with Time Windows and Capacity Constraints (GCVRPTW) (focusing on fairness and sustainability for delivery drivers, customers). Consider the sustainability influenced by the route fairness and load fairness of delivery drivers, as well as the service fairness and service satisfaction of customers.

Modeling contexts of fairness and sustainable, consider relationships of customers, drivers, load and routes distance.

For delivery drivers, factors such as differences in working hours, differences in vehicle load levels, and differences in the length of assigned routes may be considered. For customers, factors like differences in service satisfaction and service quality may be taken into account. Additionally, the context of sustainability in enterprise development, which is influenced by both drivers and customers, should be considered.

```
$\mathcal{F}_s := \left\{ \begin{array}{l} p_1: \text{implies}(\text{or}(\text{loadDiff} \text{ Low}, \text{routeLenDiff} \text{ Low}), \text{High}), \\ \quad \text{where } \text{loadDiff} = (\text{Driver Load Difference}), \text{routeLenDiff} = (\text{Route Length Difference}), \text{High} = (\text{Fairness}) \\ p_2: \text{and}(\text{driver} \text{ Low}, \text{driver} \text{ Medium or Low}), \\ \quad \text{where } \text{driver} = (\text{Driver Overtime Hours}), \text{driver} = (\text{Driver Count}) \\ p_3: \text{implies}(\text{custAvg} \text{ High}, \text{service} \text{ High}), \\ \quad \text{where } \text{custAvg} = (\text{Customer Avg Satisfaction}), \text{service} = (\text{Service Quality}) \\ p_4: \text{Capacity} \in [0.5-0.2, 0.5+0.2], \\ \quad \text{where } \text{Capacity} = (\text{Capacity Utilization}) \end{array} \right.
```

Original Matlab code for `CalObj2`:

```
function [PopObj, PopCon, routesList, C1, C2, C3, C4, ALPHA] = CalObj2(obj, PopDec)
```

```

..... % CalObj2 - Calculate bi-criteria objectives, constraint violations, and route lists

..... % Input Parameters:

..... % ... obj - Problem object containing distance matrix, demands, time windows, etc.

..... % ... PopDec - Population decision matrix (each row is a chromosome encoding)

..... % Output Parameters:

..... % ... PopObj - Bi-criteria objective matrix [total cost, -total customer satisfaction]

..... % ... PopCon - Constraint violation value (penalty function sum)

..... % ... routesList - Cell array of routes (customer nodes per route)

..... % Initialize output parameters

N = size(PopDec, 1); % Population size

PopObj = zeros(N, 2); % Dual objectives: [total cost, -total customer satisfaction]

PopCon = zeros(N, 1); % Constraint violation values

routesList = cell(N, 1); % List of routes (only non-empty routes)

C1 = cell(N, 1); % Economic cost (€)

C2 = cell(N, 1); % Environmental cost (€)

C3 = cell(N, 1); % Social cost (accident risk, €)

C4 = cell(N, 1); % Total customer satisfaction

ALPHA = cell(N,1); % Truth values for fuzzy proposition constraints
(reserved)

..... % Core parameters (revised fuel efficiency definition: 0.052 L/KM)

depotIdxs = obj.depotIdxs; % Depot node indices

maxVehicles = obj.car; % Maximum number of vehicles

Q = obj.capacity; % Vehicle capacity constraint

```

```

FC = 67.62; ..... % Fixed vehicle cost (€/trip)

DW = 9.91 / 60; ..... % Driver cost (€/minute, converted from 9.91€/hour)

CF = 1.58; ..... % Fuel price (€/l)

fuelEfficiency = 0.052; ..... % Fuel efficiency (L/KM, 0.052 liters consumed per kilometer)

Ce = 0.02; ..... % Carbon emission cost (€/kg CO2)

gamma = 0.75; ..... % Carbon emission factor (kg CO2/l)

a = 0.0005; ..... % Accident risk coefficient (€/kg·km)

% Penalty factors

capPenalty = 10000; ..... % Penalty for capacity constraints

timePenalty = 10000; ..... % Penalty for time window constraints

% Iterate through each solution in the population

for i = 1:N

    % 1. Parse chromosome and extract non-empty routes

    chrom = PopDec(i, PopDec(i, :) ~= -1); ..... % Remove placeholder -1

    if isempty(chrom), chrom = depotIdxs(1); end

    % Ensure routes start from depot (assume vehicles depart from depot)

    if ~ismember(chrom(1), depotIdxs)

        firstDepot = find(ismember(chrom, depotIdxs), 1);

        chrom = [chrom(firstDepot), chrom(1:firstDepot-1), chrom(firstDepot+1:end)];

    end

```

```

% Split routes and filter empty ones (only keep routes with customers)

depotPos = find(ismember(chrom, depotIdxs));

depotPos = [depotPos, length(chrom)+1];

validRoutes = {};

for r = 1:length(depotPos)-1

    startIdx = depotPos(r);

    endIdx = depotPos(r+1)-1;

    routeCustomers = chrom(startIdx+1:endIdx);

    routeCustomers = routeCustomers(~ismember(routeCustomers, depotIdxs));

    if ~isempty(routeCustomers)

        validRoutes{end+1} = routeCustomers;

    end

end

routesList{i} = validRoutes;

numValidRoutes = length(validRoutes);

% 2. Initialize cost and constraint variables

totalEconomic = 0; % Economic cost

totalEnvironmental = 0; % Environmental cost

totalSocial = 0; % Social cost

totalSatisfaction = 0; % Customer satisfaction

capViolation = 0; % Capacity constraint violation

timeViolation = 0; % Time window constraint violation

served = zeros(1, length(obj.demand)); % Mark served customers

current_alpha = 0; % Current Global Truth value for fuzzy propositions (reserve
d)

```

```

% 3. Calculate costs and constraints for non-empty routes

for r = 1:numValidRoutes

    route = validRoutes{r};

    routeWithDepot = [0, route, 0]; % Start and end at depot (closed route)

    routeLen = length(routeWithDepot);

    % Total demand of the route (initial load)

    demandIdxs = route + 1;

    totalDemand = sum(obj.demand(demandIdxs));

    currLoad = totalDemand; % Delivery scenario: load decreases after delivery

    % Check capacity constraint

    if totalDemand > Q

        capViolation = capViolation + (totalDemand - Q);

    end

    % Initialize route variables

    distTotal = 0; % Total distance (km)

    fuelTotal = 0; % Total fuel consumption (l)

    timeTotal = 0; % Total driving time (minutes)

    arrivalTime = 0; % Current arrival time (minutes)

    % Iterate through each node pair in the route (i→j)

    for k = 1:routeLen-1

```

```

..... from = routeWithDepot(k);

..... to = routeWithDepot(k+1);

..... dist = obj.C(from+1, to+1); % Distance (km)

..... % calculate driving time (hours)
..... v = 60; % km/h average speed (km/h)

..... % Calculate driving time (minutes): (distance/speed)×60 (convert hours to minutes)
..... travelTimeMin = (dist / v) * 60; % minutes

..... timeTotal = timeTotal + travelTimeMin;

..... arrivalTime = arrivalTime + travelTimeMin;

..... % calculate fuel consumption (core: based on 0.052 L/KM)
..... fuel = fuelEfficiency * dist; % Fuel (l) = consumption per km × distance

..... fuelTotal = fuelTotal + fuel;

..... distTotal = distTotal + dist;

..... % Carbon emission cost (environmental cost)
..... emissionCost = Ce * fuel * gamma;

..... totalEnvironmental = totalEnvironmental + emissionCost;

..... % Accident risk cost (social cost)
..... if to > 0

..... accidentCost = a * dist * currLoad;

```

```

totalSocial = totalSocial + accidentCost;

end

% Process customer node service

if to > 0 && ~served(to)

    served(to) = 1;

    % Check time window constraints (unit: minutes)

    custIdx = to + 1;

    ready = obj.readyTime(custIdx); % Earliest customer time (minutes)

    due = obj.dueTime(custIdx); % Latest customer time (minutes)

    % Handle waiting/lateness

    if arrivalTime < ready

        arrivalTime = ready; % Wait until earliest time

    elseif arrivalTime > due

        timeViolation = timeViolation + (arrivalTime - due); % Lateness time

    end

    % Calculate customer satisfaction (based on time window deviation)

    prefTime = (ready + due) / 2; % Preferred time (minutes)

    timeDiff = abs(arrivalTime - prefTime); % minutes

    windowHeight = due - ready; % minutes

    if windowHeight > 0

        satisfaction = 100 * max(0, 1 - 2*timeDiff/windowHeight);

    end

```

```

        . . .
    else
        . . .
        satisfaction = 0;
    end

    totalSatisfaction = totalSatisfaction + satisfaction;

    . . .

    % Reduce load after delivery (only for delivery scenarios)

    currLoad = currLoad - obj.demand(custIdx);

    . . .

    % Add service time (minutes, directly accumulated)

    arrivalTime = arrivalTime + obj.serviceTime(custIdx);

    . . .
    end

    . . .
    end

    . . .

    % Calculate economic cost

    vehicleCost = FC; % Fixed cost counted once for non-empty routes

    driverCost = DW * timeTotal; % Driver cost (€/minute × minutes)

    fuelCost = Cf * fuelTotal; % Fuel cost (€)

    economicCost = vehicleCost + driverCost + fuelCost;

    totalEconomic = totalEconomic + economicCost;

    . . .
    end

    . . .

    % 4. Assign constraint violations and objective functions

    PopCon(i) = capPenalty * capViolation + timePenalty * timeViolation;

    PopObj(i, :) = [totalEconomic + totalEnvironmental + totalSocial, -totalSatisfaction];

```

```

..... % Store costs for each dimension

..... C1{i} = totalEconomic;

..... C2{i} = totalEnvironmental;

..... C3{i} = totalSocial;

..... C4{i} = totalSatisfaction;

..... ALPHA{i} = current_alpha;

..... end

..... end

```

## 1. Modeling Foundations for Fuzzy Propositions

- \*\*Select the universe of discourse for the variables of the propositions to ensure Problem could be solved.
- **Variable Standardization:** Variables are linearly mapped to the [0,1] range.
- **Linguistic Term Definition:** Fuzzy terms such as "High," "Medium," and "Low" are used to describe variables.
- **Logical Connectives:** Include operations like AND, OR, implies, NOT, with connective families such as Łukasiewicz, Zadeh, Gödel, and Product.
- **Proposition Structures:**
  - Conditional and qualified propositions (e.g., "V is F and (W is G or Y is not H)").
  - Conditional and unqualified propositions (e.g., "If V is F, then W is G").
  - Unconditional and unqualified propositions (e.g., "V is F").

## 2. Workflow for Fuzzy Proposition Integration

The process starts with a context  $\mathcal{F}_s$ , which is defined by a set of fuzzy propositions ( $F_s$ ). These propositions are integrated using t-norm operators (e.g., Zadeh's min) to form an aggregated context. The truth value of each proposition is computed, and all truth values are further aggregated into an overall truth degree via t-norm. A global threshold  $\alpha = 0.6$  (minimum acceptable overall truth for  $\mathcal{F}_s$ ) is introduced. If the aggregated overall truth degree is less than  $\alpha$ , the fuzzy constraint violation is calculated, multiplied by a penalty factor  $10^5$ , and used to update the total constraint violation (PopCon).

## Task Requirements

**Objective:** Design membership functions based on  $F_s$ , and embed them into the `CalObj2` function to generate the final code.

### Output (In English):

1. Logical proposition set  $F_s = \{p_1 : \dots, p_2 : \dots, p_3 : \dots, p_4 : \dots\}$ .
2. Regarding the constraints on the universe of discourse, please reasonably select the universe of discourse for the variables of the membership function.

3. Definitions and descriptions of membership functions for each proposition in  $F_s$ .
4. A complete `CalObj2` function (Matlab code) that integrates the above elements
5. Please use the correct MATLAB syntax (Dont use ternary operator and correctly use if-else-end).
6. If the membership function is implemented in the form of a helper function, it should inherit from 'obj'. If use auxfunctions, please correctly define and when invoking it, use obj.helperFunction (eg. output = Auxfunctions(obj,...))..

#### Deepseek-R1:

## Modeling Foundations for Fuzzy Propositions in GCVRPTW

### 1. Universe of Discourse Selection

For fair standardization across all fuzzy variables, each variable is linearly mapped to  $[0, 1]$ . Domain bounds are set as follows:

Variable	Symbol	Domain	Standardization Formula
Driver Load Difference	$D_{loadDiff}$	$[0, Q]$	$\frac{\max(loads) - \min(loads)}{Q}$
Route Length Difference	$D_{routeLenDiff}$	$[0, D_{\max}]$	$\frac{\max(dists) - \min(dists)}{D_{\max}}$
Driver Overtime	$O_{driver}$	$[0, 240]$ minutes	$\min\left(1, \frac{\text{max}(maxOvertime, overtime) - 240}{240}\right)$
Driver Count	$N_{driver}$	$[0, \text{maxVehicles}]$	$\frac{\text{numValidRoutes}}{\text{maxVehicles}}$
Customer Avg Satisfaction	$S_{custAvg}$	$[0, 100]$	$\frac{\text{totalSatisfaction}}{\text{n_served} \times 100}$
Service Quality	$Q_{service}$	$[0, 1]$	$S_{custAvg}$
Fairness	$F$	$[0, 1]$	$1 - 0.5D_{loadDiff} - 0.5D_{routeLenDiff}$
Capacity Utilization	$U_{capacity}$	$[0, 1]$	$\text{mean}(loads/Q)$

#### Notes:

- $D_{\max}$  = Total solution distance (sum of all route distances)
- $Q$  = Vehicle capacity
- Overtime calculated as excess over 480 minutes (8-hour workday)
- $S_{custAvg}$  standardized from satisfaction scores (0-100 per customer)

### 2. Membership Function Definitions

Triangular/trapezoidal membership functions defined for **Low (L)**, **Medium (M)**, and **High (H)** linguistic terms. Parameters standardized across all variables in  $[0, 1]$ :

#### Linguistic Terms:

- **Low (L):** Trapezoidal,  $[0, 0]$  core,  $[0, 0.3]$  support

$$\mu_{low}(x) = \begin{cases} 1 & x \leq 0 \\ (0.3 - x)/0.3 & 0 < x < 0.3 \\ 0 & x \geq 0.3 \end{cases}$$

- **Medium (M):** Triangular, [0.3, 0.5, 0.7]

$$\mu_{\text{medium}}(x) = \begin{cases} (x - 0.3)/0.2 & 0.3 \leq x < 0.5 \\ (0.7 - x)/0.2 & 0.5 \leq x < 0.7 \\ 0 & \text{otherwise} \end{cases}$$

- **High (H):** Trapezoidal, [0.7, 1] core, [0.7, 1] support

$$\mu_{\text{high}}(x) = \begin{cases} 0 & x \leq 0.7 \\ (x - 0.7)/0.3 & 0.7 < x < 1 \\ 1 & x \geq 1 \end{cases}$$

### Proposition-Specific Applications:

Proposition	Evaluation Logic
$p_1$	$\mu_{\text{high}}(\text{P1}) \wedge (\neg \mu_{\text{high}}(\text{P2})) \wedge (\neg \mu_{\text{high}}(\text{P3})) \wedge (\neg \mu_{\text{high}}(\text{P4}))$
$p_2$	$\mu_{\text{high}}(\text{P1}) \wedge (\neg \mu_{\text{high}}(\text{P2})) \wedge (\neg \mu_{\text{high}}(\text{P3})) \wedge (\neg \mu_{\text{high}}(\text{P4}))$
$p_3$	$\mu_{\text{high}}(\text{P1}) \wedge (\neg \mu_{\text{high}}(\text{P2})) \wedge (\neg \mu_{\text{high}}(\text{P3})) \wedge (\neg \mu_{\text{high}}(\text{P4}))$
$p_4$	$\mu_{\text{medium}}(U_{\text{capacity}})$

### Operators:

- AND: min (Zadeh t-norm)
- OR: max (Zadeh s-norm)
- Implies:  $\max(1 - \text{antecedent}, \text{consequent})$  (Kleene-Dienes)
- Aggregation: min() across all  $p_1$  to  $p_4$  truth values
- Penalty: Applied if aggregated truth < 0.6

### Complete **CalObj2** MATLAB Code

```

function [PopObj, PopCon, routesList, C1, C2, C3, C4, ALPHA] = CalObj2(obj, PopDec)
    % CalObj2 - Calculates objectives, constraints, and integrates fuzzy propositions
    N = size(PopDec, 1);
    PopObj = zeros(N, 2);
    PopCon = zeros(N, 1);
    routesList = cell(N, 1);
    C1 = cell(N, 1); C2 = cell(N, 1); C3 = cell(N, 1); C4 = cell(N, 1);
    ALPHA = cell(N, 1);

    % Core parameters
    depotIdxs = obj.depotIdxs;
    maxVehicles = obj.car;
    Q = obj.capacity;
    FC = 67.62; DW = 9.91 / 60; Cf = 1.58; fuelEfficiency = 0.052;
    Ce = 0.02; gamma = 0.75; a = 0.0005;
    capPenalty = 10000; timePenalty = 10000;
    fuzzyPenaltyFactor = 1e5; alphaThreshold = 0.6;

    % Iterate through each solution
    for i = 1:N
        chrom = PopDec(i, PopDec(i, :) ~= -1);
        if isempty(chrom)

```

```

chrom = depotIdxs(1);
end

if ~ismember(chrom(1), depotIdxs)
    firstDepot = find(ismember(chrom, depotIdxs), 1);
    chrom = [chrom(firstDepot), chrom(1:firstDepot-1), chrom(firstDepot+1:end)];
end

% Split chromosome into valid routes
depotPos = find(ismember(chrom, depotIdxs));
depotPos = [depotPos, length(chrom)+1];
validRoutes = {};
for r = 1:length(depotPos)-1
    startIdx = depotPos(r);
    endIdx = depotPos(r+1)-1;
    routeCustomers = chrom(startIdx+1:endIdx);
    routeCustomers = routeCustomers(~ismember(routeCustomers, depotIdxs));
    if ~isempty(routeCustomers)
        validRoutes{end+1} = routeCustomers;
    end
end
routesList{i} = validRoutes;
numValidRoutes = length(validRoutes);

% Initialize cost and constraint variables
totalEconomic = 0; totalEnvironmental = 0; totalSocial = 0;
totalSatisfaction = 0; capViolation = 0; timeViolation = 0;
served = zeros(1, length(obj.demand));
loads = zeros(1, numValidRoutes);
routeTimes = zeros(1, numValidRoutes);
routeDists = zeros(1, numValidRoutes);

% Process each route
for r = 1:numValidRoutes
    route = validRoutes{r};
    routeWithDepot = [0, route, 0];
    routeLen = length(routeWithDepot);

    % Calculate total demand
    demandIdxs = route + 1;
    totalDemand = sum(obj.demand(demandIdxs));
    loads(r) = totalDemand;
    currLoad = totalDemand;
    if totalDemand > Q
        capViolation = capViolation + (totalDemand - Q);
    end

    % Route processing variables
    distTotal = 0; fuelTotal = 0; timeTotal = 0; arrivalTime = 0;
    serviceTimeTotal = 0;

```

```

..... for k = 1:routeLen-1
.....     from = routeWithDepot(k);
.....     to = routeWithDepot(k+1);
.....     dist = obj.C(from+1, to+1);
.....     travelTimeMin = (dist / 60) * 60; % v=60 km/h
.....     timeTotal = timeTotal + travelTimeMin;
.....     arrivalTime = arrivalTime + travelTimeMin;
.....     fuelTotal = fuelTotal + fuelEfficiency * dist;
.....     distTotal = distTotal + dist;

..... % Environmental cost
..... emissionCost = Ce * fuelEfficiency * dist * gamma;
..... totalEnvironmental = totalEnvironmental + emissionCost;

..... % Accident cost
..... if to > 0
.....     accidentCost = a * dist * currLoad;
.....     totalSocial = totalSocial + accidentCost;
..... end

..... % Process customer node
..... if to > 0 && ~served(to)
.....     served(to) = 1;
.....     custIdx = to + 1;
.....     ready = obj.readyTime(custIdx);
.....     due = obj.dueTime(custIdx);

.....     if arrivalTime < ready
.....         arrivalTime = ready;
.....     elseif arrivalTime > due
.....         timeViolation = timeViolation + (arrivalTime - due);
.....     end

..... % Customer satisfaction calculation
..... prefTime = (ready + due) / 2;
..... timeDiff = abs(arrivalTime - prefTime);
..... windowWidth = due - ready;
..... if windowWidth > 0
.....     satisfaction = 100 * max(0, 1 - 2*timeDiff/windowWidth);
..... else
.....     satisfaction = 0;
..... end
..... totalSatisfaction = totalSatisfaction + satisfaction;

..... % Update load and service time
..... currLoad = currLoad - obj.demand(custIdx);
..... serviceTime = obj.serviceTime(custIdx);
..... arrivalTime = arrivalTime + serviceTime;
..... serviceTimeTotal = serviceTimeTotal + serviceTime;

```

```

        end
    end

    % Store route metrics
    routeDists(r) = distTotal;
    routeTimes(r) = timeTotal + serviceTimeTotal; % Total = travel + service

    % Economic costs
    vehicleCost = FC;
    driverCost = DW * timeTotal; % Only travel time for driver cost
    fuelCost = Cf * fuelTotal;
    totalEconomic = totalEconomic + vehicleCost + driverCost + fuelCost;

end

% Calculate constraint violations
PopCon(i) = capPenalty * capViolation + timePenalty * timeViolation;
PopObj(i, :) = [totalEconomic + totalEnvironmental + totalSocial, -totalSatisfaction];
C1{i} = totalEconomic; C2{i} = totalEnvironmental; C3{i} = totalSocial;
C4{i} = totalSatisfaction;

% --- FUZZY PROPOSITION EVALUATION ---
% Compute fuzzy variables
n_served = sum(served);
totalDist = sum(routeDists);

if numValidRoutes > 0
    % D_loadDiff: Driver load difference
    D_loadDiff = (max(load) - min(load)) / Q;

    % D_routeLenDiff: Route length difference
    if totalDist > 0
        D_routeLenDiff = (max(routeDists) - min(routeDists)) / totalDist;
    else
        D_routeLenDiff = 0;
    end

    % Fairness index F
    F_val = max(0, min(1, 1 - 0.5*D_loadDiff - 0.5*D_routeLenDiff));

    % O_driver: Maximum overtime
    maxRouteTime = max(routeTimes);
    overtime = max(0, maxRouteTime - 480); % 480 min = 8 hours
    O_driver = min(1, overtime / 240); % 240 min = 4 hours max overtime

    % N_driver: Driver count
    N_driver = numValidRoutes / maxVehicles;

    % S_custAvg: Average customer satisfaction
    if n_served > 0
        S_custAvg = totalSatisfaction / (n_served * 100);
    end
end

```

```

    else
        S_custAvg = 0;
    end

    % Q_service: Service quality (same as satisfaction)
    Q_service = S_custAvg;

    % U_capacity: Capacity utilization
    utilizations = loads / Q;
    U_capacity = mean(utilizations);

    % Compute membership values
    ante_p1 = max(obj.low_mu([D_loadDiff, D_routeLenDiff])); % OR operation
    cons_p1 = obj.high_mu(F_val);
    truth_p1 = max(1 - ante_p1, cons_p1); % Implication

    ante_p2_1 = obj.low_mu(O_driver);
    ante_p2_2 = max(obj.medium_mu(N_driver), obj.low_mu(N_driver)); % OR
    truth_p2 = min(ante_p2_1, ante_p2_2); % AND

    ante_p3 = obj.high_mu(S_custAvg);
    cons_p3 = obj.high_mu(Q_service);
    truth_p3 = max(1 - ante_p3, cons_p3); % Implication

    truth_p4 = obj.medium_mu(U_capacity);

    % Aggregate truth values
    truths = [truth_p1, truth_p2, truth_p3, truth_p4];
    alpha_global = min(truths);

    % Apply penalty if below threshold
    if alpha_global < alphaThreshold
        fuzzy_penalty = fuzzyPenaltyFactor * (alphaThreshold - alpha_global);
    else
        fuzzy_penalty = 0;
    end
    PopCon(i) = PopCon(i) + fuzzy_penalty;
    ALPHA{i} = alpha_global;
    else
        ALPHA{i} = 0; % No routes -> lowest truth
    end
end
end

% --- Membership Function Subfunctions ---
function mu = low_mu(~, x)
    a = 0.0; b = 0.3;
    if x <= a
        mu = 1.0;
    elseif x >= b

```

```

        mu = 0.0;
    else
        mu = (b - x) / (b - a);
    end
end

function mu = medium_mu(~, x)
    a = 0.3; b = 0.5; c = 0.7;
    if x < a
        mu = 0.0;
    elseif x >= a && x < b
        mu = (x - a) / (b - a);
    elseif x >= b && x < c
        mu = (c - x) / (c - b);
    else
        mu = 0.0;
    end
end

function mu = high_mu(~, x)
    a = 0.7; b = 1.0;
    if x <= a
        mu = 0.0;
    elseif x >= b
        mu = 1.0;
    else
        mu = (x - a) / (b - a);
    end
end

```

## Key Implementation Notes:

**1. Membership Functions:** Defined as subfunctions (`low_mu`, `medium_mu`, `high_mu`) attached to the problem object `obj` for modularity. Called via `obj.low_mu(x)` syntax.

### 2. Fuzzy Variables:

- Computed after primary cost/constraint calculations
- `D_loadDiff` and `D_routeLenDiff` quantify distribution fairness
- `F_val` combines both fairness dimensions
- `O_driver` and `N_driver` address driver workload
- `S_custAvg` and `Q_service` represent customer-centric metrics

### 3. Truth Aggregation:

- Kleene-Dienes implication used for conditional propositions
- min-operator aggregates truth across all propositions
- Penalty =  $10^5 \times (0.6 - \alpha_{\text{global}})$

**4. Efficiency:** Route distances, times, and loads computed during initial cost evaluation to avoid recomputation.

**5. Edge Cases:** Handles zero-route solutions, division by zero, and bounds clipping for fairness index.

This implementation integrates fuzzy logic seamlessly with the existing GCVRPTW objective/constraint structure, enforcing fairness and sustainability through mathematically rigorous fuzzy propositions.

(注：文档部分内容可能由 AI 生成)