

# ### 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} \left( \text{or} \left( D_{\text{loadDiff}} \text{ is Low}, D_{\text{routeLenDiff}} \text{ is Low} \right), F \text{ is High} \right), \\ \quad \text{where } D_{\text{loadDiff}} = (\text{Driver Load Difference}), D_{\text{routeLenDiff}} = (\text{Route Length Difference}), F = (\text{Fairness}) \\ p_2: \text{and} \left( O_{\text{driver}} \text{ is Low}, N_{\text{driver}} \text{ is Medium or Low} \right), \\ \quad \text{where } O_{\text{driver}} = (\text{Driver Overtime Hours}), N_{\text{driver}} = (\text{Driver Count}) \\ p_3: \text{implies} \left( S_{\text{custAvg}} \text{ is High}, Q_{\text{service}} \text{ is High} \right), \\ \quad \text{where } S_{\text{custAvg}} = (\text{Customer Avg Satisfaction}), Q_{\text{service}} = (\text{Service Quality}) \\ p_4: U_{\text{capacity}} \text{ is Medium}(0.5-0.2, 0.5+0.2), \\ \quad \text{where } U_{\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 sa
tisfaction]

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:

## 1. Logical Proposition Set $\mathcal{F}_s$

- **p<sub>1</sub>**: If driver load difference is **Low** OR route length difference is **Low**, then fairness is **High**.
- **p<sub>2</sub>**: Driver overtime is **Low** AND driver count is **Medium or Low**.
- **p<sub>3</sub>**: If average customer satisfaction is **High**, then service quality is **High** (equivalent to satisfaction).
- **p<sub>4</sub>**: Capacity utilization is **Medium** (triangular membership centered at 0.5).

## 2. Universe of Discourse

- **D\_loadDiff**:  $[0, Q]$  (vehicle capacity)
- **D\_routeLenDiff**:  $[0, \infty)$  km (normalized by max route length)
- **O\_driver**:  $[0, \infty)$  minutes (normalized by 480-minute shift)
- **N\_driver**:  $[1, \text{maxVehicles}]$
- **S\_custAvg**:  $[0, 100]$  (normalized to percentage)
- **U\_capacity**:  $[0, 1]$

## 3. Membership Function Definitions

### **p<sub>1</sub>**: Driver Fairness Propositions

- **D\_loadDiff is Low**:  

$$\mu_{\text{Low}}(x) = 1 - x/Q$$

(Decreases linearly from 1 at  $x = 0$  to 0 at  $x = Q$ )
- **D\_routeLenDiff is Low**:  

$$\mu_{\text{Low}}(x) = 1 - \frac{x}{\max(1, \text{maxRouteLen})}$$

(Uses relative difference; maxRouteLen is max length in current solution)
- **Truth Value Calculation**:  
Antecedent:  $\max(\mu_{\text{Low}}(D_{\text{loadDiff}}), \mu_{\text{Low}}(D_{\text{routeLenDiff}}))$   
(Zadeh OR, using max)

### **p<sub>2</sub>**: Driver Overtime and Count

- **O\_driver is Low**:  

$$\mu_{\text{Low}}(x) = 1 - \min\left(1, \frac{x}{480}\right)$$

(0 overtime = 1; overtime  $\geq 480$  minutes = 0)
- **N\_driver is Low**:  

$$\mu_{\text{Low}}(x) = 1 - \frac{x - 1}{\text{maxVehicles} - 1}$$

(Linear from 1 at min vehicles to 0 at max vehicles)

- **N\_driver is Medium:**

$$\mu_{\text{Medium}}(x) = \begin{cases} 2 \cdot \frac{x-1}{\text{maxVehicles}-1}, & \text{if } x \leq \frac{\text{maxVehicles}+1}{2} \\ 2 \cdot \frac{\text{maxVehicles}-x}{\text{maxVehicles}-1}, & \text{otherwise} \end{cases}$$

(Triangular membership centered at mean of vehicle range)

- **Truth Value Calculation:**

$$\min(\mu_{\text{Low}}(O_{\text{driver}}), \max(\mu_{\text{Low}}(N_{\text{driver}}), \mu_{\text{Medium}}(N_{\text{driver}})))$$

(Zadeh AND/OR using min/max)

### **p<sub>3</sub>: Customer Satisfaction**

- **S\_custAvg is High:**

$$\mu_{\text{High}}(x) = x/100$$

(Linear mapping: 0% = 0, 100% = 1)

- **Truth Value:**  $\mu_{\text{High}}(S_{\text{custAvg}})$

(Service quality equivalent to satisfaction)

### **p<sub>4</sub>: Capacity Utilization**

- **U\_capacity is Medium:**

$$\mu_{\text{Medium}}(x) = \begin{cases} 0, & x \leq 0.3 \\ \frac{x-0.3}{0.2}, & 0.3 < x < 0.5 \\ \frac{0.7-x}{0.2}, & 0.5 \leq x < 0.7 \\ 0, & x \geq 0.7 \end{cases}$$

(Triangular membership over [0.3, 0.7], peak at 0.5)

## **Aggregation and Penalty**

- **Overall Truth Degree:**

$$\alpha = \min(\text{truth}_{p1}, \text{truth}_{p2}, \text{truth}_{p3}, \text{truth}_{p4})$$

(Zadeh AND using min)

- **Penalty for Fuzzy Constraint:**

If  $\alpha < 0.6$ , add  $(0.6 - \alpha) \times 10^5$  to **PopCon**.

## **Complete MATLAB Code for **CalObj2****

```
function [PopObj, PopCon, routesList, C1, C2, C3, C4, ALPHA] = CalObj2(obj, PopDec)
    % CalObj2 - Calculate bi-criteria objectives, constraint violations, route lists, and fuzzy
    % truth values
    N = size(PopDec, 1);
    PopObj = zeros(N, 2);
    PopCon = zeros(N, 1);
    routesList = cell(N, 1);
    C1 = cell(N, 1); % Economic cost (€)
    C2 = cell(N, 1); % Environmental cost (€)
    C3 = cell(N, 1); % Social cost (€)
    C4 = cell(N, 1); % Total customer satisfaction
    ALPHA = cell(N, 1); % Overall truth degree for fuzzy propositions
```

```

depotIdxs = obj.depotIdxs;
maxVehicles = obj.car;
Q = obj.capacity;
FC = 67.62; % Fixed vehicle cost (€/trip)
DW = 9.91 / 60; % Driver cost (€/minute)
Cf = 1.58; % Fuel price (€/l)
fuelEfficiency = 0.052;% Fuel efficiency (L/KM)
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)
capPenalty = 10000;
timePenalty = 10000;

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

    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);

    totalEconomic = 0;
    totalEnvironmental = 0;
    totalSocial = 0;
    totalSatisfaction = 0;
    capViolation = 0;
    timeViolation = 0;
    served = zeros(1, length(obj.demand));

    % Initialize arrays to collect per-route metrics for fuzzy evaluation
    routeLoads = zeros(1, numValidRoutes);
    routeLengths = zeros(1, numValidRoutes);
    routeTimes = zeros(1, numValidRoutes);

```

```

..... for r = 1:numValidRoutes
..... route = validRoutes{r};
..... routeWithDepot = [0, route, 0];
..... routeLen = length(routeWithDepot);
..... demandIdxs = route + 1;
..... totalDemand = sum(obj.demand(demandIdxs));
..... routeLoads(r) = totalDemand;
..... currLoad = totalDemand;

..... if totalDemand > Q
..... capViolation = capViolation + (totalDemand - Q);
..... end

..... distTotal = 0;
..... fuelTotal = 0;
..... arrivalTime = 0;

..... for k = 1:routeLen-1
..... from = routeWithDepot(k);
..... to = routeWithDepot(k+1);
..... dist = obj.C(from+1, to+1);
..... v = 60;
..... travelTimeMin = (dist / v) * 60;
..... arrivalTime = arrivalTime + travelTimeMin;
..... fuel = fuelEfficiency * dist;
..... fuelTotal = fuelTotal + fuel;
..... distTotal = distTotal + dist;
..... emissionCost = Ce * fuel * gamma;
..... totalEnvironmental = totalEnvironmental + emissionCost;

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

..... 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

..... prefTime = (ready + due) / 2;
..... timeDiff = abs(arrivalTime - prefTime);

```

```

..... windowWidth = due - ready;
..... if windowHeight > 0
.....     satisfaction = 100 * max(0, 1 - 2 * timeDiff / windowHeight);
..... else
.....     satisfaction = 0;
..... end
..... totalSatisfaction = totalSatisfaction + satisfaction;
..... currLoad = currLoad - obj.demand(custIdx);
..... arrivalTime = arrivalTime + obj.serviceTime(custIdx);
..... end
..... end
..... routeLengths(r) = distTotal;
..... routeTimes(r) = arrivalTime; % Total route time (min)
..... vehicleCost = FC;
..... driverCost = DW * arrivalTime;
..... fuelCost = Cf * fuelTotal;
..... economicCost = vehicleCost + driverCost + fuelCost;
..... totalEconomic = totalEconomic + economicCost;
..... end

..... % --- Calculate fuzzy truth values for propositions in F_s ---
..... % p1: Driver load and route length fairness
..... if numValidRoutes > 0
.....     maxLoad = max(routeLoads);
.....     minLoad = min(routeLoads);
.....     maxLength = max(routeLengths);
.....     minLength = min(routeLengths);
.....     D_loadDiff = maxLoad - minLoad;
.....     D_routeLenDiff = maxLength - minLength;
.....     muLoadLow = 1 - (D_loadDiff / Q);
.....     if maxLength > 0
.....         muRouteLow = 1 - (D_routeLenDiff / maxLength);
.....     else
.....         muRouteLow = 1;
.....     end
.....     truth_p1 = max(muLoadLow, muRouteLow); % Zadeh OR (max)
..... else
.....     truth_p1 = 1; % No routes → vacuously fair
..... end

..... % p2: Driver overtime and count
..... if numValidRoutes > 0
.....     O_driver = max(max(0, routeTimes - 480)); % Max overtime across routes
.....     muOvertimeLow = 1 - min(1, O_driver / 480);
.....     if maxVehicles > 1
.....         muDriverLow = 1 - (numValidRoutes - 1) / (maxVehicles - 1);
.....         midpoint = (maxVehicles + 1) / 2;
.....         if numValidRoutes <= midpoint
.....             muDriverMedium = 2 * (numValidRoutes - 1) / (maxVehicles - 1);
.....         else
.....             muDriverHigh = 1 - (numValidRoutes - 1) / (maxVehicles - 1);
.....         end
.....     end
..... end

```

```

..... muDriverMedium = 2 * (maxVehicles - numValidRoutes) / (maxVehicles - 1);
..... end
..... muDriverMedLow = max(muDriverLow, muDriverMedium); % Zadeh OR (max)
..... else
..... muDriverMedLow = 1; % Only one vehicle
..... end
..... truth_p2 = min(muOvertimeLow, muDriverMedLow); % Zadeh AND (min)
..... else
..... truth_p2 = 1; % No routes → vacuously satisfied
..... end

..... % p3: Customer satisfaction
..... numServed = sum(served);
..... if numServed > 0
..... S_custAvg = totalSatisfaction / numServed;
..... truth_p3 = S_custAvg / 100; % Linear mapping to [0,1]
..... else
..... truth_p3 = 0; % No customers served → satisfaction=0
..... end

..... % p4: Capacity utilization
..... if numValidRoutes > 0
..... totalDemand = sum(routeLoads);
..... U_capacity = totalDemand / (numValidRoutes * Q);
..... if U_capacity < 0.3 || U_capacity > 0.7
..... truth_p4 = 0;
..... else
..... if U_capacity < 0.5
..... truth_p4 = (U_capacity - 0.3) / 0.2;
..... else
..... truth_p4 = (0.7 - U_capacity) / 0.2;
..... end
..... end
..... else
..... truth_p4 = 0; % No routes → utilization=0
..... end

..... % --- Aggregate truth values and apply penalty ---
..... overall_truth = min([truth_p1, truth_p2, truth_p3, truth_p4]); % Zadeh AND
..... ALPHA{i} = overall_truth;

..... fuzzyPenalty = 0;
..... if overall_truth < 0.6
..... fuzzyPenalty = 100000 * (0.6 - overall_truth); % Scale penalty by violation degree
..... end

..... % --- Base constraint penalty ---
..... PopCon(i) = capPenalty * capViolation + timePenalty * timeViolation + fuzzyPenalty;
..... PopObj(i, :) = [totalEconomic + totalEnvironmental + totalSocial, -totalSatisfaction];
.....
```

```
..... C1{i} = totalEconomic;
..... C2{i} = totalEnvironmental;
..... C3{i} = totalSocial;
..... C4{i} = totalSatisfaction;
.... end
end
```

## Key Implementation Notes:

1. **Per-Route Metrics:** For fuzzy evaluation, we collect `routeLoads`, `routeLengths`, and `routeTimes` during route processing.

2. **Membership Functions:** Defined inline using:

- Linear functions for "Low" (e.g., driver load difference).
- Triangular functions for "Medium" (e.g., capacity utilization).
- Relative normalization (e.g., route length difference scaled by max length in solution).

3. **Truth Aggregation:**

- Zadeh logic: AND as `min`, OR as `max`.
- Overall truth is the minimum of all proposition truth values.
- Penalty added to `PopCon` if truth < 0.6 (scaled by  $10^5$ ).

4. **Edge Cases:** Handles no-route/no-customer scenarios with vacuously true/false assignments.

This implementation integrates fuzzy proposition evaluation directly into the objective calculation, providing penalties for solutions violating fairness and sustainability constraints.

(注: 文档部分内容可能由 AI 生成)