

# ### 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 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 sloved.
- **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

The fuzzy proposition set  $\mathcal{F}_s$  remains as specified:

- $p_1$ : implies (or ( $D_{\text{loadDiff}}$  is Low,  $D_{\text{routeLenDiff}}$  is Low),  $F$  is High)
- $p_2$ : and ( $O_{\text{driver}}$  is Low,  $N_{\text{driver}}$  is Medium or Low)
- $p_3$ : implies ( $S_{\text{custAvg}}$  is High,  $Q_{\text{service}}$  is High) (Note:  $S_{\text{custAvg}} = Q_{\text{service}}$ )
- $p_4$ :  $U_{\text{capacity}}$  is Medium(0.5-0.2, 0.5+0.2)

## 2. Universe of Discourse Selection

- $D_{\text{loadDiff}}$  (**Driver Load Difference**):  $[0, Q]$  (normalized to  $[0, 1]$  by  $D_{\text{loadDiff}}/Q$ )
- $D_{\text{routeLenDiff}}$  (**Route Length Difference**):  $[0, \text{maxRouteLength}]$  (normalized to  $[0, 1]$  by  $D_{\text{routeLenDiff}}/\text{maxRouteLength}$ )
- $F$  (**Fairness**):  $[0, 1]$  (defined as  $1 - 0.5D_{\text{loadDiff,norm}} - 0.5D_{\text{routeLenDiff,norm}}$ )
- $O_{\text{driver}}$  (**Driver Overtime**):  $[0, 720]$  minutes (normalized to  $[0, 1]$ )
- $N_{\text{driver}}$  (**Driver Count**):  $[0, \text{maxVehicles}]$  (normalized to  $[0, 1]$ )
- $S_{\text{custAvg}}$  (**Customer Avg Satisfaction**):  $[0, 100]$  (normalized to  $[0, 1]$ )
- $Q_{\text{service}}$  (**Service Quality**): Identical to  $S_{\text{custAvg}}$
- $U_{\text{capacity}}$  (**Capacity Utilization**): Natural ratio in  $[0, \infty)$ , capped to  $[0, 1]$  for membership

## 3. Membership Function Definitions

### (1) Low Membership (For $D_{\text{loadDiff}}$ , $D_{\text{routeLenDiff}}$ , $O_{\text{driver}}$ ):

- **Formula:**  $\max(0, 1 - 2x)$
- **Description:** Linear decay from 1 at  $x = 0$  to 0 at  $x = 0.5$

### (2) High Membership (For $F$ , $S_{\text{custAvg}}$ , $Q_{\text{service}}$ ):

- **Formula:**

$$\max \left( 0, \min \left( 1, \frac{x - 0.5}{0.5} \right) \right)$$

- **Description:** Linear increase from 0 at  $x = 0.5$  to 1 at  $x = 1$

### (3) Medium or Low Membership (For $N_{\text{driver}}$ ):

- **Low for  $N_{\text{driver}}$ :**

$$y = \begin{cases} 1 & x \leq 0.2 \\ \frac{0.6-x}{0.4} & 0.2 < x < 0.6 \\ 0 & x \geq 0.6 \end{cases}$$

- Medium for  $N_{\text{driver}}$ :

$$y = \begin{cases} 0 & x < 0.3 \\ \frac{x-0.3}{0.1} & 0.3 \leq x < 0.4 \\ 1 & 0.4 \leq x \leq 0.6 \\ \frac{0.7-x}{0.1} & 0.6 < x \leq 0.7 \\ 0 & x > 0.7 \end{cases}$$

- Combined ( $N_{\text{driver}}$  is Medium or Low):  $\max(\text{Low}(x), \text{Medium}(x))$

#### (4) Medium Membership (For $U_{\text{capacity}}$ ):

- Formula:

$$y = \begin{cases} 0 & x < 0.3 \\ \frac{x-0.3}{0.2} & 0.3 \leq x < 0.5 \\ \frac{0.7-x}{0.2} & 0.5 \leq x < 0.7 \\ 0 & x \geq 0.7 \end{cases}$$

- Description: Triangular peak at  $x = 0.5$  with base  $[0.3, 0.7]$

#### 4. Complete **CalObj2** MATLAB Code

```

function [PopObj, PopCon, routesList, C1, C2, C3, C4, ALPHA] = CalObj2(obj, PopDec)
    % Calculate objectives and constraints with fuzzy propositions

    % --- Nested Membership Functions ---
    function y = low_mf_1(x)
        y = max(0, 1 - 2*x);
    end

    function y = high_mf_1(x)
        y = max(0, min(1, (x - 0.5)/0.5));
    end

    function y = low_mf_0(x)
        y = 1 - x;
    end

    function y = low_mf_N(x)
        if x <= 0.2
            y = 1;
        elseif x >= 0.6
            y = 0;
        else
            y = (0.6 - x)/0.4;
        end
    end

    function y = medium_mf_N(x)

```

```

..... if x < 0.3
.....     y = 0;
..... elseif x < 0.4
.....     y = (x - 0.3)/0.1;
..... elseif x <= 0.6
.....     y = 1;
..... elseif x < 0.7
.....     y = (0.7 - x)/0.1;
..... else
.....     y = 0;
..... end
..... end

..... function y = medium_mf_U(x)
..... if x < 0.3
.....     y = 0;
..... elseif x < 0.5
.....     y = (x - 0.3)/0.2;
..... elseif x < 0.7
.....     y = (0.7 - x)/0.2;
..... else
.....     y = 0;
..... end
..... end

..... % --- Precompute Parameters ---
..... 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);

..... 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;

..... % Precompute maxRouteLength for normalization
..... maxDistEdge = max(max(obj.C));
..... nTotalNodes = size(obj.C, 1);
..... nCustomers = nTotalNodes - 1; % Depot excluded

```

```

maxRouteLength = (nCustomers + 1) * maxDistEdge;

% --- Process each solution ---
for i = 1:N
    % --- Extract and validate chromosome ---
    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 into non-empty 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;
        route = chrom(startIdx+1:endIdx);
        route = route(~ismember(route, depotIdxs));
        if ~isempty(route)
            validRoutes{end+1} = route;
        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));
    routeLoads = [];
    routeLengths = [];
    workingTimePerRoute = [];

    % --- Process each route ---
    for r = 1:numValidRoutes
        route = validRoutes{r};
        routeWithDepot = [0, route, 0];
        routeLen = length(routeWithDepot);
        demandIdxs = route + 1;
        totalDemand = sum(obj.demand(demandIdxs));
        currLoad = totalDemand;
        if totalDemand > Q
            capViolation = capViolation + (totalDemand - Q);
        end
    end

```

```

..... distTotal = 0; fuelTotal = 0; timeTotal = 0; arrivalTime = 0;
..... for k = 1:routeLen-1
.....     from = routeWithDepot(k);
.....     to = routeWithDepot(k+1);
.....     dist = obj.C(from+1, to+1);
.....     travelTimeMin = (dist / 60) * 60; % Assume v=60 km/h
.....     timeTotal = timeTotal + travelTimeMin;
.....     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 windowWidth > 0
.....         satisfaction = 100 * max(0, 1 - 2*timeDiff/windowWidth);
.....     else
.....         satisfaction = 0;
.....     end
.....     totalSatisfaction = totalSatisfaction + satisfaction;

.....     currLoad = currLoad - obj.demand(custIdx);
.....     arrivalTime = arrivalTime + obj.serviceTime(custIdx);
..... end
..... end

..... vehicleCost = FC;
..... driverCost = DW * timeTotal;
..... fuelCost = Cf * fuelTotal;
..... economicCost = vehicleCost + driverCost + fuelCost;
..... totalEconomic = totalEconomic + economicCost;

```

```

..... routeLoads(end+1) = totalDemand;
..... routeLengths(end+1) = distTotal;
..... workingTimePerRoute(end+1) = arrivalTime; % Time at depot return
..... end

..... % --- Fuzzy Proposition Evaluation ---
..... % p1: Driver Fairness
..... if numValidRoutes > 0
..... D_loadDiff_norm = (max(routeLoads) - min(routeLoads)) / Q;
..... D_routeLenDiff_norm = (max(routeLengths) - min(routeLengths)) / maxRouteLength;
..... F_val = 1 - 0.5*D_loadDiff_norm - 0.5*D_routeLenDiff_norm;
..... t_loadLow = low_mf_1(D_loadDiff_norm);
..... t_routeLow = low_mf_1(D_routeLenDiff_norm);
..... t_A = max(t_loadLow, t_routeLow);
..... t_C = high_mf_1(F_val);
..... t1 = max(1 - t_A, t_C); % Kleene-Dienes
..... else
..... t1 = 1; % No routes: ideal fairness
..... end

..... % p2: Driver Workload
..... if numValidRoutes > 0
..... maxWorkingTime = max(workingTimePerRoute);
..... O_driver_raw = max(0, maxWorkingTime - 480); % Overtime minutes
..... normalized_O_driver = min(O_driver_raw / 720, 1); % Cap at 1
..... normalized_N_driver = numValidRoutes / maxVehicles;
..... t_O_low = low_mf_0(normalized_O_driver);
..... t_low_N = low_mf_N(normalized_N_driver);
..... t_medium_N = medium_mf_N(normalized_N_driver);
..... t_N = max(t_low_N, t_medium_N);
..... t2 = min(t_O_low, t_N);
..... else
..... t2 = 1; % No drivers: ideal condition
..... end

..... % p3: Customer Service (Tautology)
..... if numValidRoutes > 0 && any(served)
..... totalServed = sum(served(2:end)); % Exclude depot
..... S_custAvg = totalSatisfaction / (totalServed * 100);
..... t3 = 1; % S_custAvg high implies Q_service high (always true)
..... else
..... t3 = 1;
..... end

..... % p4: Capacity Utilization
..... if numValidRoutes > 0
..... totalDemandAll = sum(routeLoads);
..... U_capacity = totalDemandAll / (Q * numValidRoutes);
..... t4 = medium_mf_U(min(U_capacity, 1)); % Cap utilization at 1

```

```

        else
            t4 = medium_mf_U(0);
        end

        % --- Overall Truth Value and Penalty ---
        overall_truth = min([t1, t2, t3, t4]);
        ALPHA{i} = overall_truth;
        if overall_truth < 0.6
            fuzzy_penalty = (0.6 - overall_truth) * 1e5;
            PopCon(i) = PopCon(i) + fuzzy_penalty;
        end

        % --- Objective and Constraint Assignment ---
        PopCon(i) = PopCon(i) + capPenalty * capViolation + timePenalty * timeViolation;
        totalCost = totalEconomic + totalEnvironmental + totalSocial;
        PopObj(i, :) = [totalCost, -totalSatisfaction];
        C1{i} = totalEconomic;
        C2{i} = totalEnvironmental;
        C3{i} = totalSocial;
        C4{i} = totalSatisfaction;
    end
end

```

## Key Implementation Notes:

### 1. Fuzzy Variables:

- Driver fairness (`F_val`) defined as weighted average of load/route length differences.
- Overtime (`O_driver_raw`) is max per-driver overtime exceeding 480 minutes.
- `U_capacity` is total demand divided by available capacity.
- `S_custAvg` is average satisfaction per served customer (scaled 0-1).

### 2. Membership Functions:

- Implemented as nested functions for encapsulation.
- Zadeh's `min / max` used for AND/OR; Kleene-Dienes for implication.
- `ALPHA` stores the solution's overall truth value.

### 3. Normalization:

- All fuzzy inputs linearly scaled to [0,1] per universe definitions.
- `maxRouteLength` computed as worst-case single-route distance.

### 4. Efficiency:

- Route metrics (load, distance, time) collected during existing loops.
- Fuzzy penalties added to `PopCon` if overall truth < 0.6.

(注: 文档部分内容可能由 AI 生成)