

# ### 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 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:**

## Final **CalObj2** Function Implementation

```

function [PopObj, PopCon, routesList, C1, C2, C3, C4, ALPHA] = CalObj2(obj, PopDec)
    % Define triangular membership function helper
    function y = tri(x, a, b, c)
        if x <= a || x >= c
            y = 0;
        elseif x <= b
            y = (x - a) / (b - a);
        else
            y = (c - x) / (c - b);
        end
    end

    % Initialize 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);

    % 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;

    % Penalty factors
    capPenalty = 10000;
    timePenalty = 10000;

```

```

.... % Main population loop
.... 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

.... % Extract 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;
....     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));
.... nCustomers = sum(obj.demand > 0);

.... % Per-route metrics storage
.... route_loads = zeros(numValidRoutes, 1);
.... route_distances = zeros(numValidRoutes, 1);
.... route_times = zeros(numValidRoutes, 1);
.... route_idx = 1;

.... v = 60; % Average speed (km/h)

.... % Process each route
.... for r = 1:numValidRoutes
....     route = validRoutes{r};
....     routeWithDepot = [0, route, 0];
.... 
```

```

routeLen = length(routeWithDepot);

% Calculate demand and capacity constraints
demandIdxs = route + 1;
totalDemand = sum(obj.demand(demandIdxs));
route_loads(r) = totalDemand;
currLoad = totalDemand;
if totalDemand > Q
    capViolation = capViolation + (totalDemand - Q);
end

% Initialize route metrics
distTotal = 0;
fuelTotal = 0;
timeTotal = 0;
arrivalTime = 0;

% Traverse route nodes
for k = 1:routeLen-1
    from = routeWithDepot(k);
    to = routeWithDepot(k+1);
    dist = obj.C(from+1, to+1);

    % Calculate time and distance
    travelTimeMin = (dist / v) * 60;
    timeTotal = timeTotal + travelTimeMin;
    distTotal = distTotal + dist;
    fuel = fuelEfficiency * dist;
    fuelTotal = fuelTotal + fuel;
    arrivalTime = arrivalTime + travelTimeMin;

    % Environmental costs
    emissionCost = Ce * fuel * gamma;
    totalEnvironmental = totalEnvironmental + emissionCost;

    % Social costs (accident risk)
    if to > 0
        accidentCost = a * dist * currLoad;
        totalSocial = totalSocial + accidentCost;
    end

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

        % Time window constraints
        if arrivalTime < ready

```

```

..... arrivalTime = ready;
..... elseif arrivalTime > due
.....     lateTime = arrivalTime - due;
.....     timeViolation = timeViolation + lateTime;
..... end

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

..... % Update load
..... currLoad = currLoad - obj.demand(custIdx);

..... % Service time
..... arrivalTime = arrivalTime + obj.serviceTime(custIdx);
..... end
..... end

..... % Final route metrics
..... route_distances(r) = distTotal;
..... route_times(r) = arrivalTime;

..... % Economic costs
..... vehicleCost = FC;
..... driverCost = DW * timeTotal;
..... fuelCost = Cf * fuelTotal;
..... economicCost = vehicleCost + driverCost + fuelCost;
..... totalEconomic = totalEconomic + economicCost;
..... end

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

..... % Store cost dimensions
..... C1{i} = totalEconomic;
..... C2{i} = totalEnvironmental;
..... C3{i} = totalSocial;
..... C4{i} = totalSatisfaction;

..... % Calculate fuzzy constraints
..... truth_vals = zeros(1, 4);
..... alpha = 0.6;
```

```

penaltyFactor = 1e5;

if numValidRoutes > 0
    % Per-route metrics
    max_load = max(route_loads);
    min_load = min(route_loads);
    max_route = max(route_distances);
    min_route = min(route_distances);
    total_distance = sum(route_distances);

    % P1: Driver fairness
    D_loadDiff_norm = (max_load - min_load) / Q;
    D_routeLenDiff_norm = (max_route - min_route) / max(1, total_distance);
    mu_low_load = 1 - D_loadDiff_norm;
    mu_low_route = 1 - D_routeLenDiff_norm;
    A_p1 = max(mu_low_load, mu_low_route);
    F_fairness = min(1 - D_loadDiff_norm, 1 - D_routeLenDiff_norm);
    truth_p1 = min(1, 1 - A_p1 + F_fairness); % Łukasiewicz
    truth_vals(1) = truth_p1;

    % P2: Driver overtime and count
    route_overtimes = max(0, route_times - 480);
    O_driver = max(route_overtimes);
    normalized_O = min(O_driver, 600) / 600;
    mu_low_ovt = 1 - normalized_O;

    normalized_N = numValidRoutes / maxVehicles;
    mu_low_N = tri(normalized_N, 0, 0.2, 0.4);
    mu_med_N = tri(normalized_N, 0.3, 0.5, 0.7);
    part2_p2 = max(mu_med_N, mu_low_N);
    truth_p2 = min(mu_low_ovt, part2_p2);
    truth_vals(2) = truth_p2;

    % P3: Customer satisfaction (tautology)
    truth_vals(3) = 1;

    % P4: Capacity utilization
    U_capacity = mean(route_loads) / Q;
    truth_vals(4) = tri(U_capacity, 0.3, 0.5, 0.7);

    % Overall truth degree
    T_overall = min(truth_vals);

    % Add penalty if below threshold
    if T_overall < alpha
        PopCon(i) = PopCon(i) + penaltyFactor * (alpha - T_overall);
    end
    ALPHA{i} = T_overall;
else
    % Penalize invalid solutions
end

```

```

        PopCon(i) = PopCon(i) + penaltyFactor * alpha;
        ALPHA{i} = 0;
    end
end

```

## Key Implementation Details:

### 1. Fuzzy Variable Definitions

- **D\_loadDiff (Driver Load Difference)**: Normalized difference between max/min route demands  $(\max\_load - \min\_load) / Q$ . Universe:  $[0, 1]$
- **D\_routeLenDiff (Route Length Difference)**: Normalized range of route distances  $(\max\_route - \min\_route) / \text{total\_distance}$ . Universe:  $[0, 1]$
- **O\_driver (Driver Overtime)**: Max overtime across routes normalized to  $[0, 1]$  ( $\min(ovt, 600)/600$ )
- **N\_driver (Driver Count)**: Normalized number of routes  $\text{numRoutes} / \maxVehicles$ . Universe:  $[0, 1]$
- **U\_capacity (Capacity Utilization)**: Average load fraction across routes  $\text{mean}(loads)/Q$ . Universe:  $[0, 1]$

### 2. Membership Functions

- **Low**: Linear decreasing function  $\mu(x) = 1 - x$
- **High**: Linear increasing function  $\mu(x) = x$
- **Medium**: Triangular function with parameters:
  - $\text{tri}(x, 0.3, 0.5, 0.7)$  for Capacity Utilization (p4)
  - $\text{tri}(x, 0.3, 0.5, 0.7)$  for Driver Count "Medium" (p2)
  - $\text{tri}(x, 0, 0.2, 0.4)$  for Driver Count "Low" (p2)

### 3. Proposition Truth Calculations

- **p1 (Fairness)**:

```

Antecedent = max(mu_low(loadDiff), mu_low(routeDiff))
Consequent = min(1-loadDiff, 1-routeDiff) // F_fairness
truth_p1 = min(1, 1 - Antecedent + Consequent) // Łukasiewicz

```

- **p2 (Driver Constraints)**:

```

part1 = mu_low(Overtime)
part2 = max(mu_med(DriverCount), mu_low(DriverCount))
truth_p2 = min(part1, part2) // Zadeh AND

```

- **p3 (Customer Satisfaction)**: Tautology  $\text{truth\_p3} = 1$
- **p4 (Capacity)**:  $\text{truth\_p4} = \text{tri}(U\_capacity, 0.3, 0.5, 0.7)$

### 4. Penalty Mechanism

- Overall truth:  $T_{\text{overall}} = \min(\text{truth\_p1}, \text{truth\_p2}, \text{truth\_p3}, \text{truth\_p4})$
- Penalty added to **PopCon** if  $T_{\text{overall}} < 0.6$ :
 

```
penalty = (0.6 - T_overall) * 100,000
```

## 5. Helper Function

- **Triangular Membership ( `tri()` )**: Defined as nested function using piecewise linear calculations

This implementation integrates fuzzy logic with vehicle routing constraints, balancing driver fairness, customer satisfaction, and sustainability metrics through mathematically grounded membership functions and Łukasiewicz/Zadeh connectives. Penalties enforce solutions to meet minimum fairness/sustainability thresholds.

(注：文档部分内容可能由 AI 生成)