

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 (GCVRPWTW) (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.

$\mathcal{F}_s := \left\{ \right.$

$\begin{array}{l} \{$

$p_1: \text{implies} \left(\left(\text{or} \left(D_{\text{loadDiff}} \text{ is Low}, D_{\text{routeLenDiff}} \text{ is Low} \right), F \text{ is High} \right), \right. \backslash$

$\quad \text{where } D_{\text{loadDiff}} = (\text{Driver Load Difference}), D_{\text{routeLenDiff}} = (\text{Route Length Difference}), F = (\text{Fairness}) \backslash$

$p_2: \text{and} \left(O_{\text{driver}} \text{ is Low}, N_{\text{driver}} \text{ is Medium or Low} \right), \backslash$

$\quad \text{where } O_{\text{driver}} = (\text{Driver Overtime Hours}), N_{\text{driver}} = (\text{Driver Count}) \backslash$

$p_3: \text{implies} \left(S_{\text{custAvg}} \text{ is High}, Q_{\text{service}} \text{ is High} \right), \backslash$

$\quad \text{where } S_{\text{custAvg}} = (\text{Customer Avg Satisfaction}), Q_{\text{service}} = (\text{Service Quality}) \backslash$

$p_4: U_{\text{capacity}} \text{ is Medium}(0.5-0.2, 0.5+0.2), \backslash$

$\quad \text{where } U_{\text{capacity}} = (\text{Capacity Utilization})$

$\end{array} \right.$

$\}$

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, et
c.

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

% Output Parameters:

%   PopObj - Bi-criteria objective matrix [total cost, -total customer satisfac
tion]

%   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 cus
tomer 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 const

```

```
raints (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 con  
sumed 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:en
nd)];

        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
(reserved)

% 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));

```

very

```
currLoad = totalDemand; % Delivery scenario: load decreases after deli
```

```
% 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)
```

```
    v = 60; % km/h average speed (km/h)
```

```

% Calculate driving time (minutes): (distance/speed)×60 (convert ho
urs 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 × di
stance

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

        windowWidth = due - ready; % minutes

        if windowWidth > 0

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

```



```

        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 α , 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 (Don't 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 and Implementation of Fuzzy Propositions in GCVRPTW

1. Logical Proposition Set \mathcal{F}_s

The context \mathcal{F}_s is defined by:

- p_1 : implies (or (D_{loadDiff} is Low, $D_{\text{routeLenDiff}}$ is Low), F is High)
- p_2 : and (O_{driver} is Low, N_{driver} is Medium or Low)
- p_3 : implies (S_{custAvg} is High, Q_{service} is High)
- p_4 : U_{capacity} is Medium_[0.3,0.7]

2. Universe of Discourse

Variables' ranges standardized to [0,1] where applicable:

- D_{loadDiff} (Load Difference): $[0, Q] \rightarrow$ Normalized to $[0, 1]$
- $D_{\text{routeLenDiff}}$ (Route Length Difference): $[0, \infty) \rightarrow$ Normalized by max route distance per solution
- O_{driver} (Driver Overtime): $[0, 480]$ minutes \rightarrow Normalized to $[0, 1]$ for membership
- N_{driver} (Driver Count): $[1, \text{maxVehicles}]$
- S_{custAvg} (Avg. Satisfaction): $[0, 100] \rightarrow$ Direct scalar
- U_{capacity} (Capacity Utilization): $[0, 1]$

3. Membership Functions

Linguistic Terms & Definitions (Zadeh Operators: max/min for OR/AND, Łukasiewicz for Implication)

- **Low (Decreasing):**

```
function mu = Low(x, threshold)
    if x <= 0
        mu = 1;
```

```

elseif x <= threshold
    mu = 1 - x/threshold;
else
    mu = 0;
end
end
end

```

- **Medium (Triangular):**

```

function mu = Medium(x, a, b, c)
    if x <= a || x >= c
        mu = 0;
    elseif x < b
        mu = (x - a) / (b - a);
    else
        mu = (c - x) / (c - b);
    end
end
end

```

- **High (Increasing):**

```

function mu = High(x, threshold)
    if x >= threshold
        mu = 1;
    elseif x >= threshold/2
        mu = 2*(x - threshold/2)/threshold;
    else
        mu = 0;
    end
end
end

```

- **Łukasiewicz Implication:**

$$\text{implies}(A, B) = \min(1, 1 - \mu_A + \mu_B)$$

4. Membership Function Parameters

- D_{loadDiff} is Low: Threshold = $0.5Q$
 - $D_{\text{routeLenDiff}}$ is Low: Threshold = 0.5 (normalized)
 - O_{driver} is Low: Threshold = 60 mins (normalized: 0.125)
 - N_{driver} is Low: $a = 1$, $b = \lceil 0.3 \cdot \text{maxVehicles} \rceil$
 - N_{driver} is Medium: $a = \lceil 0.3 \cdot \text{maxVehicles} \rceil$, $b = \text{midpoint}$, $c = \lceil 0.7 \cdot \text{maxVehicles} \rceil$
 - S_{custAvg} is High: Threshold = 100
 - U_{capacity} is Medium: $a = 0.3$, $b = 0.5$, $c = 0.7$
-

Complete 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 list
    % s
    % with integrated fuzzy proposition evaluation
    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;
    maxWorkingTime = 480; % 8 hours = 480 minutes

    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:en
d)];
        end

        depotPos = find(ismember(chrom, depotIdxs));
        depotPos = [depotPos, length(chrom)+1];
    end
```

```

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));

% Per-route metrics for fuzzy evaluation
working_times = zeros(1, numValidRoutes);
route_loads = zeros(1, numValidRoutes);
route_distances = 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));
    currLoad = totalDemand;
    route_loads(r) = totalDemand;

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

    distTotal = 0;
    timeTotal = 0;
    arrivalTime = 0;
    v = 60; % km/h average speed

    for k = 1:routeLen-1

```

```

        from = routeWithDepot(k);
        to = routeWithDepot(k+1);
        dist = obj.C(from+1, to+1);
        travelTimeMin = (dist / v) * 60;
        timeTotal = timeTotal + travelTimeMin;
        arrivalTime = arrivalTime + travelTimeMin;
        fuel = fuelEfficiency * dist;
        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 * fuel;

```

```

        economicCost = vehicleCost + driverCost + fuelCost;
        totalEconomic = totalEconomic + economicCost;

        working_times(r) = arrivalTime;
        route_distances(r) = distTotal;
    end

    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
    truth_p1 = 1;
    truth_p2 = 1;
    truth_p3 = 1;
    truth_p4 = 1;

    if numValidRoutes > 0
        % Extract metrics
        D_loadDiff = max(route_loads) - min(route_loads);
        D_routeLenDiff = max(route_distances) - min(route_distances);
        overtimes = max(0, working_times - maxWorkingTime);
        O_driver = max(overtimes);
        N_driver = numValidRoutes;
        served_customers_count = sum(served(2:end)); % Exclude depot
        total_demand_served = sum(route_loads);
        U_capacity = total_demand_served / (N_driver * Q);

        if served_customers_count > 0
            S_custAvg = totalSatisfaction / served_customers_count;
        else
            S_custAvg = 0;
        end

        % P1: D_loadDiff is Low OR D_routeLenDiff is Low -> F is High
        mu_load_low = Low(D_loadDiff / Q, 0.5); % Threshold = 0.5 * capacity
        if max(route_distances) > 0
            mu_route_low = Low(D_routeLenDiff / max(route_distances), 0.5);
        else
            mu_route_low = 1; % One route, difference=0
        end
    end

```



```

    A = max(mu_load_low, mu_route_low); % OR (Zadeh: max)
    F = min(mu_load_low, mu_route_low); % F = min load and route fairness
    B = High(F, 1); % F is in [0,1], threshold=1.0
    truth_p1 = min(1, 1 - A + B); % Łukasiewicz

    % P2: O_driver is Low AND (N_driver is Low OR Medium)
    mu_overtime_low = Low(O_driver, 60); % Threshold=60 mins
    low_bound = 1;
    med_low = ceil(0.3 * maxVehicles);
    med_high = ceil(0.7 * maxVehicles);
    mu_low_N = Low(N_driver, med_low);
    mu_med_N = Medium(N_driver, med_low, (med_low + med_high)/2, med_high);
    B = max(mu_low_N, mu_med_N); % OR (Zadeh: max)
    truth_p2 = min(mu_overtime_low, B); % AND (Zadeh: min)

    % P3: S_custAvg is High -> Q_service is High
    mu_high_s = High(S_custAvg, 100); % Threshold=100%
    Q_service = S_custAvg; % Assume service quality = satisfaction
    mu_high_q = High(Q_service, 100);
    truth_p3 = min(1, 1 - mu_high_s + mu_high_q); % Łukasiewicz

    % P4: U_capacity is Medium
    truth_p4 = Medium(U_capacity, 0.3, 0.5, 0.7);
end

% Aggregate truth values using min t-norm
truths = [truth_p1, truth_p2, truth_p3, truth_p4];
overall_truth = min(truths);
ALPHA{i} = overall_truth;

% Apply penalty if below threshold  $\alpha=0.6$ 
if overall_truth < 0.6
    fuzzy_penalty = 1e5 * (0.6 - overall_truth);
    PopCon(i) = PopCon(i) + fuzzy_penalty;
end
end
end

% Helper Functions (Defined as object methods)
function mu = Low(x, threshold)
    if x <= 0
        mu = 1;
    elseif x < threshold
        mu = 1 - x/threshold;
    else

```

```

        mu = 0;
    end
end

function mu = Medium(x, a, b, c)
    if x <= a || x >= c
        mu = 0;
    elseif x < b
        mu = (x - a)/(b - a);
    elseif x <= c
        mu = (c - x)/(c - b);
    else
        mu = 0;
    end
end

function mu = High(x, threshold)
    if x >= threshold
        mu = 1;
    elseif x > threshold/2
        mu = 2*(x - threshold/2)/threshold;
    else
        mu = 0;
    end
end
end

```

Key Implementation Notes:

1. Fuzzy Operators:

- **OR/AND:** Zadeh (\max / \min)
- **Implication:** Łukasiewicz ($\min(1, 1 - \mu(A) + \mu(B))$)

2. Variable Definitions:

- $F = \min(\mu_{\text{loadLow}}, \mu_{\text{routeLow}})$ (fairness proxy)
- $Q_{\text{service}} = S_{\text{custAvg}}$ (no separate measure)

3. Scoped Calculations:

- Per-solution metrics (e.g., D_{loadDiff} , O_{driver}) computed after routing
- Normalization used where required to handle varying scales

4. Efficiency:

- Fuzzy eval and penalty applied after core routing constraints
- Per-route data stored during solution evaluation

5. Thresholds:

- Empirically set (e.g., 60 min overtime threshold) based on domain insight
- Adjustable via parameters in membership functions

This implementation integrates fuzzy constraints into the objective evaluation, balancing fairness, sustainability, and operational efficiency in GCVRPTW.