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 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€/hou
r)
    Cf = 1.58;                      % Fuel price (€/l)
    fuelEfficiency = 0.052;         % Fuel efficiency (L/KM, 0.052 liters consumed pe
r 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)


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

                    % Calculate driving time (minutes): (distance/speed)×60 (convert hours to m
inutes)
                    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 ti
me
                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, -totalSatisfactio
n];
```

```
            % 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 $F_s = \{p_1 : \ldots, p_2 : \ldots, p_3 : \ldots p_4 : \ldots\}$.
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 corretly 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， plaese correctly define and when invoking it, use obj.helperFunction (eg. output = Auxfunctions(obj,...))..

Deepseek-R1:

Here's the complete solution integrating fuzzy propositions into the `CalObj2` function:

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

    % Extract problem 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;
    totalCustomers = length(obj.demand) - 1; % Exclude depot

    % Main population loop
    for i = 1:N
        % Process chromosome and extract routes
        chrom = PopDec(i, PopDec(i, :) ~= -1);
        if isempty(chrom), chrom = depotIdxs(1); end

        % Ensure first node is 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 empties
        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 solution metrics
        totalEconomic = 0;
        totalEnvironmental = 0;
        totalSocial = 0;
        totalSatisfaction = 0;
        capViolation = 0;
        timeViolation = 0;
        served = zeros(1, length(obj.demand));
        driver_times = zeros(numValidRoutes, 1);
        loads = zeros(numValidRoutes, 1);
        route_lengths = zeros(numValidRoutes, 1);
        total_overtime = 0;

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

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

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

```
        % Traverse nodes in route
        for k = 1:routeLen-1
            from = routeWithDepot(k);
            to = routeWithDepot(k+1);
            dist = obj.C(from+1, to+1);
            v = 60; % km/h
            travelTimeMin = (dist / v) * 60;
            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

            % Service customer nodes
            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

        % Update route metrics
        vehicleCost = FC;
        driverCost = DW * timeTotal;
```

```
            fuelCost = Cf * fuelTotal;
            economicCost = vehicleCost + driverCost + fuelCost;
            totalEconomic = totalEconomic + economicCost;
            driver_times(r) = timeTotal;
            route_lengths(r) = distTotal;
            total_overtime = total_overtime + max(0, timeTotal - 480); % Overtime calculation
        end

        % Check unserved customers
        servedCustomers = sum(served(1:totalCustomers));
        unserved = totalCustomers - servedCustomers;
        if unserved > 0
            PopCon(i) = PopCon(i) + capPenalty * unserved;
        end

        % Calculate constraint violation
        PopCon(i) = 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_vals = NaN(1,4); % Track truth values for p1-p4

        % p1: Load and route fairness
        if numValidRoutes > 0
            D_loadDiff_norm = (max(loads) - min(loads)) / Q;
            max_route = max(route_lengths);
            if max_route > 0
                D_routeLenDiff_norm = (max(route_lengths) - min(route_lengths)) / max_route;
            else
                D_routeLenDiff_norm = 0;
            end
            F = 1 - max(D_loadDiff_norm, D_routeLenDiff_norm);
            mu_low_load = 1 - D_loadDiff_norm;
            mu_low_route = 1 - D_routeLenDiff_norm;
            antecedent_p1 = max(mu_low_load, mu_low_route);
            truth_p1 = min(1, 1 - antecedent_p1 + F); % Łukasiewicz implication
        else
            truth_p1 = 1;
        end
        truth_vals(1) = truth_p1;

        % p2: Driver overtime and count
        if numValidRoutes > 0
            % Overtime
            O_driver_norm = total_overtime / (120 * numValidRoutes); % Normalized
            mu_low_ot = max(0, 1 - O_driver_norm);
```

```
            % Driver count
            N_driver_norm = numValidRoutes / maxVehicles;
            % Medium membership (triangular)
            if N_driver_norm < 0.3
                mu_med = 0;
            elseif N_driver_norm < 0.5
                mu_med = (N_driver_norm - 0.3) / 0.2;
            elseif N_driver_norm < 0.7
                mu_med = (0.7 - N_driver_norm) / 0.2;
            else
                mu_med = 0;
            end
            % Low membership for driver count
            mu_low_driver = max(0, 1 - N_driver_norm);
            truth_p2 = min(mu_low_ot, max(mu_med, mu_low_driver)); % Zadeh AND
        else
            truth_p2 = 1;
        end
        truth_vals(2) = truth_p2;

        % p3: Customer satisfaction (always true with equivalence)
        truth_p3 = 1;
        truth_vals(3) = truth_p3;

        % p4: Capacity utilization (triangular medium)
        if numValidRoutes > 0
            utilizations = loads / Q;
            U_capacity = mean(utilizations);
            if U_capacity < 0.3
                truth_p4 = 0;
            elseif U_capacity < 0.5
                truth_p4 = (U_capacity - 0.3) / 0.2;
            elseif U_capacity < 0.7
                truth_p4 = (0.7 - U_capacity) / 0.2;
            else
                truth_p4 = 0;
            end
        else
            truth_p4 = 0;
        end
        truth_vals(4) = truth_p4;

        % Aggregate truth value
        T_overall = min(truth_vals); % Zadeh t-norm

        % Apply penalty if below threshold
        if T_overall < 0.6
            fuzzy_penalty = 100000 * (0.6 - T_overall);
            PopCon(i) = PopCon(i) + fuzzy_penalty;
```

```
        end

        ALPHA{i} = truth_vals;
    end
end
```

Explanation:

1. **Fuzzy Proposition Set ($\mathcal{F}_s$)**

- p_1: Fairness (F) should be high if either load difference (D_{loadDiff}) or route length difference ($D_{\text{routeLenDiff}}$) is low
- p_2: Driver overtime (O_{driver}) should be low AND driver count (N_{driver}) should be medium or low
- p_3: High average customer satisfaction (S_{custAvg}) implies high service quality (Q_{service})
- p_4: Capacity utilization (U_{capacity}) should be medium (0.5±0.2)

2. **Variable Standardization & Universe of Discourse**

- D_{loadDiff}: [0, Q] → normalized to [0,1] by dividing by Q (vehicle capacity)
- $D_{\text{routeLenDiff}}$: [0, max_route] → normalized by maximum route length
- O_{driver}: [0, 120 * drivers] → normalized per-driver max overtime
- N_{driver}: [0, maxVehicles] → normalized by max available vehicles
- U_{capacity}: [0,1] (already normalized by definition)

3. **Membership Functions**

- **Low**: $1 - x$ (linear decreasing, for difference metrics and overtime)
- **High**: x (linear increasing, for fairness/satisfaction)
- **Medium (Triangular)**: Defined for capacity utilization (peaks at 0.5, bounds at 0.3/0.7)
- **Logical Operators**: Łukasiewicz for implications, Zadeh for AND/OR

4. **Key Implementation Notes**

- Overtime calculated as total minutes > 8 hours (480 minutes)
- Fairness (F) modeled as 1 - max(normalized load/route differences)
- Service quality (Q_{service}) considered equivalent to satisfaction
- Unserved customer penalty added to constraints
- Fuzzy penalty applied if overall truth degree < 0.6

The implementation calculates relevant variables during route processing, standardizes them, computes proposition truth values using appropriate membership functions and logical operators, and applies penalties for violating sustainability constraints.