

1. Generate pseudo code for example

Import necessary libraries

Import optimization library, array manipulation library, and data processing library

1. Initialize Parameters:

- total_saturation_min = 0.05
- total_saturation_max = 0.8
- saturation_step = 0.015
- total_flow_min = 600
- total_flow_max = 5600
- flow_step = 100

2. Generate Basic Dataset:

- total_saturation_values = Generate range from total_saturation_min to total_saturation_max with step size saturation_step
- total_flow_values = Generate range from total_flow_min to total_flow_max with step size flow_step
- basic_data_set = Cartesian product of total_saturation_values and total_flow_values

3. Define the Objective Function:

- Function objective_function(phase_flows):
 - Calculate phase_saturations using calculate_phase_saturation(phase_flows)
 - Calculate average_delay using calculate_average_delay(phase_saturations)
 - Return average_delay

4. Generate Random Phase Flows and Constrain Conditions:

- Function generate_random_phase_flows(total_flow, saturation, num_phases):
 - Generate random phase_flows within bounds (0, total_flow / num_phases)
 - If sum(phase_flows) exceeds total_flow, adjust phase_flows to satisfy the total_flow constraint
 - Return phase_flows

5. Traverse the Basic Dataset and Solve the Optimization Problem:

- For each (saturation, total_flow) in basic_data_set:
 - Initialize initial_guess using generate_random_phase_flows(total_flow, saturation, num_phases=4)
 - Define constraints:
 - Total flow constraint: $\text{sum}(\text{phase_flows}) == \text{total_flow}$
 - Phase saturation constraint: $\text{max}(\text{phase_saturations}) \leq \text{saturation}$
 - Solve the optimization problem using `maximize(objective_function, initial_guess, constraints)`
 - If result is successful, append result.x to results
- **End of loop**

2. Feasibility analysis of delay-fairness model pseudo-code

Import necessary libraries

Import optimization library, array manipulation library, and data processing library

1. Set parameters

q = Array of flow proportions for each phase

s = Array of saturation flows for each phase

M = Length of s, representing the number of phases

L = Constant used for cycle length calculation

dmin = Minimum delay target value

2. Define cycle length function

Function C(g):

Calculate and return the cycle length, which is the sum of green times for all phases plus constant L

3. Define the first model's delay objective function

Function D(q, g, s):

Calculate and return the total delay value based on the Webster model

Function fun1():

Return the lambda expression of the objective function D

4. Define the second model's entropy objective function

Function total_vehicle_delay(g, q, s):

Calculate and return the average delay for each phase

Function total_shang(g, q, s):

Initialize the total entropy sum to 0

Iterate over each phase i:

Calculate the delay proportion for the current phase and update sum with its entropy

Return the total entropy sum

Function fun2():

Return the lambda expression of the objective function total_shang

5. Define the third model's combined delay and entropy objective function

Function fun3():

Return the lambda expression of the ratio of D to total_shang

6. Define common constraints for all three models

Function con():

Initialize the set of constraints cons

Add non-negativity constraints for each phase $g[i]$
Add constraints between total green time and cycle length
Add constraints related to phase delay and d_{min}
Return the set of constraints $cons$

7.Main program entry

If the program is executed from the main entry:

Initialize the set of constraints $cons$
Create two data frames $model$ and $model1$ to store the results

Run 100 simulations:

Randomly generate an initial green time array $g0$
Use the SLSQP method to optimize the second model and find the optimal green time allocation

Retrieve the optimized green times $g3$
Calculate the cycle length $c3$, delay $dsmin3$, and entropy $hh3$
Store the above results in the $model1$ data frame

Calculate the average values in $model1$ and store the results in the $model$ data frame

Save the $model$ data frame to an Excel file

End

3.Comparative analysis of model fairness in pseudo-code

Function fun1(args):

Input: Parameters a, b, c, d, e

Output: Function v(g)

Definition: v(g) calculates the value of a complex function

Function fun2(args):

Input: Parameters a, b, c, d, e

Output: Function v(g)

Definition: v(g) calculates the value of a complex function using a different formula

Function fun3(args):

Input: Parameters a, b, c, d, e

Output: Function v(g)

Definition: v(g) calculates the value of a complex function using a different formula

Function con(args1):

Input: Parameters h, x1min, x1max

Output: Constraint conditions cons

Definition: Constraint conditions involve variable g and other input parameters

Main program entry

Main Program:

Initialize parameters args = (1, 2, 1, 2, 1)

Initialize constraint parameters args1 = (5, 15, 220)

Call function con(args1) to generate constraint conditions cons

Create a DataFrame model to store computation results

For saturation from 0.1 to 0.88 with a step of 0.015:

For Q from 600 to 5800 with a step of 100:

Generate a random value p1

Compute f1, k1

Compute p2, k2

Compute p3

If p3 > 0:

Compute p4

If p4 > 0:

Combine p1, p2, p3, p4 into vector p

Generate array q

Use fun1, fun2, fun3 functions to compute p

Generate a new row of DataFrame data

Append the computed result to model

Output: model

End