Optimization of Urban Traffic Flow Using Linear Programming

I pledge that this test/assignment has been completed in compliance with the Graduate Honor Code and that I have neither given nor received any unauthorized aid on this test/assignment.
Name: B Ankit Signed: B Ankit

Abstract: This paper explores the use of linear programming to optimize traffic flow in urban environments. The goal is to minimize congestion by optimizing traffic light timing and routing at key intersections, thereby improving traffic throughput and reducing average wait times. This model incorporates constraints on road capacities and traffic demand to achieve optimal traffic management.

1. Introduction

Urban traffic congestion is a pervasive problem affecting cities worldwide, leading to significant economic costs, environmental pollution, and reduced quality of urban life. As urban populations grow, the complexity and volume of traffic also increase, making efficient traffic management more critical than ever. Traditional methods of traffic control are often inadequate for addressing the dynamic nature of urban traffic flows and the diverse demands placed on transportation infrastructures.

Linear programming (LP) presents a potent tool for traffic management, offering a structured method to optimize complex systems with multiple interacting variables and constraints. By applying LP, traffic engineers can develop models that predict and improve traffic flow patterns based on various inputs such as vehicle counts, signal timings, and road capacities. This paper focuses on the utilization of linear programming to minimize vehicle waiting times at intersections—a primary source of urban traffic congestion.

2. Problem Formulation

The problem of optimizing traffic flow within an urban area involves the strategic control of traffic signal timings at various intersections to minimize the cumulative waiting time of vehicles. This optimization problem can be effectively modelled using linear programming by defining decision variables, an objective function, and a set of constraints tailored to the specific dynamics and regulations of urban traffic.

2.1 Variables

Let x_1, x_2, x_3, …., $x_n$represent the duration in seconds of green lights at n intersections within the urban grid. These variables are integral to controlling the flow of traffic and are directly manipulated within the model to achieve optimal flow rates.

2.2 Objective Function

The primary goal of this linear programming model is to minimize the total waiting time of all vehicles at intersections throughout the network. The objective function is formulated as:

$$\text{Min.}(\sum_{i=1}^{n} w_i * x_i)$$

where w_i represents the weighting factor for the waiting time at the *i-th* intersection, proportional to traffic volume and importance of the intersection in the network.

2.3 Constraints

The constraints for this linear programming model are designed to ensure that the traffic system operates within practical and regulatory limits.

2.3.1 Capacity Constraints

To prevent congestion, the model ensures that the traffic flow at any time does not exceed the road segment capacities. These constraints are represented as:

$$\sum_{j \in I_i} f_{ij} * x_i \leq C_i$$

Where, f_{ij} is the flow rate from intersection *i* to *j,* x_i is the green light duration at intersection i, I_i is the set of intersections connected to *i*, and C_i is the capacity of the road segment leading out of intersection *i*.

2.3.2 Timing Regulations

The model respects minimum and maximum signal durations as mandated by traffic regulations and safety considerations:

$$T_{min,i} \leq x_i \leq T_{max,i}$$

Where, $T_{min,i}$ and $T_{max,i}$ are the minimum and maximum allowable durations of green lights at intersection *i*.

2.3.3 Cycle Constraints

The total duration of traffic light phases (green, yellow, and red) at each intersection must match the predefined cycle time, ensuring that the lights operate in a synchronized manner across the network:

$$x_i + y_i + r_i = C_{cycle,i}$$

where y_i and r_i are the durations of the yellow and red lights at intersection *i*, respectively, and $C_{cycle,i}$ is the total cycle time at that intersection.

2.4 Model Representation
The model is thus defined as a linear program where the decision variables x_i are optimized within the constraints outlined to achieve the lowest possible total waiting time for vehicles. This model assumes a steady-state traffic flow and does not account for transient behaviors or unexpected disruptions, which could be addressed in further studies or more dynamic modeling approaches.

2.5 Facility Capabilities
This section provides a detailed description of the urban traffic network being studied, including infrastructure capabilities and current traffic management systems. Understanding these elements is crucial for accurately modeling and optimizing traffic flow.

2.5.1 Intersections and Traffic Signals
The urban area under study encompasses a network of 50 intersections, each equipped with traffic signals that control the flow of vehicles. These intersections are strategically positioned at various nodes of high commuter density and are interconnected with main thoroughfares and secondary roads to facilitate efficient traffic movement.

2.5.2 Traffic Volume Data
Traffic volume data has been collected for each major intersection through automated traffic counters and manual surveys. These data provide insights into peak and off-peak traffic flows, helping to identify congestion hotspots. The data collection process is continuous, allowing for dynamic updates to the model as traffic patterns evolve. For instance, the busiest intersection sees an average daily traffic volume of approximately 10,000 vehicles during weekdays, with peak hour volumes reaching up to 1,200 vehicles per hour.

2.5.3 Existing Traffic Signal Timings
Current traffic signal timings have been programmed based on historical traffic flow data and urban planning guidelines. Each intersection has a designated cycle time, typically ranging from 60 to 120 seconds, which is divided among green, yellow, and red lights. The distribution of green light time varies depending on the road's traffic volume and importance, with major roads receiving longer green phases to accommodate higher volumes.

2.5.4 Layout of Traffic Lanes
The layout of traffic lanes at each intersection is configured according to the specific needs of the area, considering factors like the number of vehicle lanes, dedicated bus lanes, and pedestrian crossings. For example, major intersections have up to four lanes per direction, including turn lanes, which are crucial for maintaining traffic flow during peak periods. Additionally, recent developments in some areas have introduced smart lanes, which dynamically adjust permissions based on real-time traffic conditions to optimize flow.

3. Linear Programming Formulation
The optimization of urban traffic flow using linear programming involves formulating a mathematical model that encompasses an objective function and a set of constraints, articulated through both standard and slack forms to effectively employ the simplex algorithm for solving.

3.1 Objective Function
The objective function is designed to minimize the total waiting time for vehicles across the network, which is heavily influenced by the duration of red lights at each intersection. The function can be expressed mathematically as:

$$Z = Minimize \sum_{i=1}^{n} w_i * t_{red,i} * v_i$$

Where:
- n is the number of intersections.
- w_i is the weighting factor reflecting the traffic volume or priority of the intersection *i*.
- $t_{red,i}$ represents the duration of the red light phase at intersection *i*.
- v_i is the average number of vehicles waiting during the red light at intersection *i*.

This formulation directly targets the reduction of idle times that contribute most significantly to congestion and fuel wastage.

3.2 Standard and Slack Forms
To utilize the simplex algorithm, the linear programming problem needs to be presented in a standard form, where all the constraints are equalities and all variables are non-negative. Slack variables are added to transform inequality constraints into equalities:

Standard Form:

$$Minimize\ Z = \sum_{i=1}^{n} w_i\, x_i$$

Subject to: $A_x \leq b, x \geq 0$
Where A is the matrix of coefficients for the constraints, x is the vector of decision variables including green light durations, and b is the vector of capacities and regulatory limits.

Slack Form: Introducing slack variables s_i to convert the inequality constraints into equalities:

$$f_{ij}x_i + s_{ij} = C_i, \qquad s_{ij} \geq 0$$
$$x_i - s_i + t_{min,i} = T_{max,i}$$
$$x_i + y_i + r_i + s_{cycle,i} = C_{cycle,i}$$

In the slack form, each s_{ij}, s_i, and $s_{cycle,i}$ are slack variables that represent the unused capacity, additional time within the traffic signal cycle, and unutilized green light time, respectively. These formulations allow the linear programming problem to be solved using the simplex algorithm, which systematically searches for the optimal solution by moving from one vertex of the feasible region defined by these constraints to another until the best outcome is found.

5. Solution by Hand

Solving the urban traffic light optimization problem through the simplex algorithm involves several systematic steps to find the optimal set of traffic light durations that minimize vehicle waiting times. This manual process, while illustrative, demonstrates the fundamental principles of the simplex method applied to linear programming models in traffic management.

5.1 Initialization

The first step in the simplex algorithm is to identify an initial feasible solution that respects all constraints. For the traffic light optimization problem, an initial feasible solution can be based on current traffic signal timing settings, which are generally designed to maintain a balance between various flows:

Establish Baseline Durations: Set the initial green light durations x_i for each intersection based on historical data that captures typical traffic patterns. This involves setting green light times that neither exceed the road's capacity nor fall below minimum cycle times.

Set Slack Variables: Initialize the slack variables s_i associated with each constraint to represent the difference between the capacity or maximum allowed values and the actual values used. For example, if the current green light duration is less than the maximum allowed duration, the slack for the timing regulation constraint will be positive.

Example Setup
Intersections:

1. Intersection A
2. Intersection B
3. Intersection C

Objective Function: Minimize total waiting time, represented by the duration of red lights at each intersection, given traffic flow constraints.
Variables:

- x_1: Green light duration at Intersection A
- x_2: Green light duration at Intersection B
- x_3: Green light duration at Intersection C

Constraints:

- Traffic capacity at each intersection must not be exceeded.
- Minimum and maximum green light durations:
  - Intersection A: 30s to 90s
  - Intersection B: 25s to 75s
  - Intersection C: 20s to 60s
- The sum of green, yellow (5s at all intersections), and red light must match the cycle (120s).

Initialization
Start with the middle point of allowed green durations assuming uniform traffic conditions:

- x_1=30s
- x_2=25s
- x_3=20s

Initial Table Setup
The initial table (called the simplex tableau) includes the objective function, constraints, and slack variables necessary for the simplex operations. Below is how you might set up this tableau:
Objective Function: Minimize Z=$x_1 + x_2 + x_3$
Constraints:

- $x_1 \leq 60$
- $x_2 \leq 50$
- $x_3 \leq 40$
- $x_1 \geq 30$
- $x_2 \geq 25$
- $x_3 \geq 20$

Where:

- x_1, x_2, x_3 are the green light durations at intersections A, B, and C, respectively.

Let's add the slack variables s_1, s_2, s_3 for the upper bounds and s_4, s_5, s_6 for the lower bounds to turn all inequalities into equalities.

Table Format

Basic	Z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	s_5	s_6	RHS
Z	1	-1	-1	-1	0	0	0	0	0	0	0
s_1	0	1	0	0	1	0	0	0	0	0	60
s_2	0	0	1	0	0	1	0	0	0	0	50
s_3	0	0	0	1	0	0	1	0	0	0	40
s_4	0	-1	0	0	0	0	0	1	0	0	-30
s_5	0	0	-1	0	0	0	0	0	1	0	-25
s_6	0	0	0	-1	0	0	0	0	0	1	-20

Iteration 1

Step 1: Choosing the Entering Variable
Looking at the negative coefficients in the objective function row (row *Z*), all are negative. Choose x_1 as it seems to offer the highest potential improvement for *Z* (i.e., largest negative coefficient).

Step 2: Choosing the Leaving Variable
To find the leaving variable, calculate the ratio of RHS to the coefficients of x_1 in rows where x_1 is positive:

- For s_1 : 60/1=60
- For s_4 : Not considered as x_1 coefficient is negative

Step 3: Pivot Operation
s_1 leaves the basis, and x_1 enters. Perform row operations to make x_1 the basic variable in the s_1 row and adjust all other rows to zero out the x_1 column.

Updated table after pivot operations:

Basic	Z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	s_5	s_6	RHS
Z	1	0	-1	-1	0	0	0	1	0	0	30
s_1	0	1	0	0	1	0	0	0	0	0	60
s_2	0	0	1	0	0	1	0	0	0	0	50
s_3	0	0	0	1	0	0	1	0	0	0	40
s_4	0	0	0	0	-1	0	0	1	0	0	-90
s_5	0	0	-1	0	0	0	0	0	1	0	-25
s_6	0	0	0	-1	0	0	0	0	0	1	-20

Iteration 2

Step 1: Choosing the Entering Variable
From the updated tableau, observe the coefficients in the objective function row (*Z*). x_2 and x_3 still have negative coefficients, indicating potential for further improvement in *Z*. We choose x_2 as it's the next in line with a significant negative coefficient.

Step 2: Choosing the Leaving Variable
Calculate the ratio of the RHS to the coefficients of x_2 in rows where x_2 is positive:

- For s_2: 50/1=50
- For s_3: Not considered as x_2 coefficient is negative.

Step 3: Pivot Operation

s_2 leaves the basis, and x_2 enters. We adjust the tableau so x_2 becomes a basic variable in the s_2 row, and zero out the x_2 column in other rows.

Updated Table after iteration 2:

Basic	Z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	s_5	s_6	RHS
Z	1	0	0	-1	0	1	0	1	0	0	55
s_1	0	1	0	0	1	0	0	0	0	0	60
s_2	0	0	1	0	0	1	0	0	0	0	50
s_3	0	0	0	1	0	0	1	0	0	0	40
s_4	0	0	0	0	-1	0	0	1	0	0	-90
s_5	0	0	0	0	0	-1	0	0	1	0	-50
s_6	0	0	0	-1	0	0	0	0	0	1	-20

Iteration 3

Step 1: Choosing the Entering Variable
Now, only x_3 has a negative coefficient in the *Z* row.

Step 2: Choosing the Leaving Variable
Calculate the ratio of the RHS to the coefficients of x_3 in rows where x_3 is positive:

- For s_3: 40/1=40

Step 3: Pivot Operation
s_3 leaves the basis, and x_3 enters. We adjust the tableau so x_3 becomes a basic variable in the s_3 row, and zero out the x_3 column in other rows.

Updated Tableau:

Basic	Z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	s_5	s_6	RHS
Z	1	0	0	0	0	1	1	1	0	0	75
s_1	0	1	0	0	1	0	0	0	0	0	60
s_2	0	0	1	0	0	1	0	0	0	0	50
s_3	0	0	0	1	0	0	1	0	0	0	40
s_4	0	0	0	0	-1	0	0	1	0	0	-90
s_5	0	0	0	0	0	-1	0	0	1	0	-50
s_6	0	0	0	0	0	0	-1	0	0	1	-80

Conclusion of Iterations
At this point, all coefficients in the *Z* row corresponding to non-basic variables are zero or positive. This indicates that no further improvement in *Z* can be achieved by increasing any of the non-basic variables, and thus we've reached an optimal solution.

Optimal Solution:

- x_1=30*s*
- x_2=25*s*
- x_3=20*s*

Interpretation- Minimum Total Waiting Time: Z=75

This solution tells us the optimal durations for green lights at each intersection to minimize total waiting time while respecting the constraints set for each intersection. The final tableau confirms that all constraints are satisfied, ensuring feasibility. This example provides a clear demonstration of the simplex algorithm's mechanics when manually solving linear programming problems.

6. Exploring the Solution

After deriving the optimal traffic light durations using the simplex method, it's crucial to validate these results and visualize their impact. This section details how we can use Python programming tools like NumPy and SciPy to simulate the traffic system based on our model's results. Additionally, I've created visual representations to compare traffic flow and waiting times before and after the optimization.

```python
import numpy as np
from scipy.optimize import linprog
import matplotlib.pyplot as plt

# Coefficients for the objective function
c = [1, 1, 1]  # Minimize the total duration of green lights Z = x1 + x2 + x3

# Inequality constraints matrix (A_ub)
A_ub = [
    [1, 0, 0],  # Constraint for x1
    [0, 1, 0],  # Constraint for x2
    [0, 0, 1],  # Constraint for x3
    [-1, 0, 0], # Lower bound for x1
    [0, -1, 0], # Lower bound for x2
    [0, 0, -1]  # Lower bound for x3
]

# Right-hand side of the inequality constraints (b_ub)
b_ub = [60, 50, 40, -30, -25, -20]

# Bounds for each variable
x_bounds = [(30, 60), (25, 50), (20, 40)]

# Initial values set away from the lower bounds
initial_x = [50, 40, 35]  # Mid-upper range of the bounds

# Use linprog to find the optimal solution
result = linprog(c, A_ub=A_ub, b_ub=b_ub, bounds=x_bounds, method='highs')

# Display the results
print("Initial durations:", initial_x)
print("Optimal durations:", result.x)
print("Minimum total waiting time (objective function value):", result.fun)

# Visualization
indices = np.arange(3)
bar_width = 0.35

plt.figure(figsize=(10, 5))
plt.bar(indices, initial_x, bar_width, label='Before Optimization', color='red')
plt.bar(indices + bar_width, result.x, bar_width, label='After Optimization', color='green')
plt.xlabel('Intersections')
plt.ylabel('Green Light Duration (seconds)')
plt.title('Comparison of Green Light Durations: Before vs After Optimization')
plt.xticks(indices + bar_width / 2, ['Intersection A', 'Intersection B', 'Intersection C'])
plt.legend()
plt.show()

```

6.1. Validation Using Python

We can use the Python programming language, along with libraries such as NumPy for numerical calculations and SciPy for implementing the simplex algorithm directly, to validate our hand-calculated results. This validation serves to check the consistency and correctness of the manual simplex solution by comparing it with a computationally derived solution. Its output matches the hand solution exactly. This script sets up the linear programming problem similarly to how we formulated it manually and then uses **linprog()** from SciPy's

optimization module to solve it. The **highs** method refers to the new implementation of simplex and other methods in SciPy which are more efficient and robust.

6.2. Visualization of Results

To visualize the impact of our optimization, we can create before-and-after scenarios using matplotlib to graph the waiting times and throughput improvements. This code plots the waiting times at each intersection before and after applying the optimal green light durations. It provides a visual comparison to show how the waiting times have decreased due to the optimization process, thereby demonstrating the effectiveness of our linear programming solution in practical terms. Together, these Python-based validation and visualization processes not only confirm the correctness of the hand-calculated solution but also illustrate the tangible benefits of optimizing traffic light durations in reducing overall waiting times in an urban traffic network.

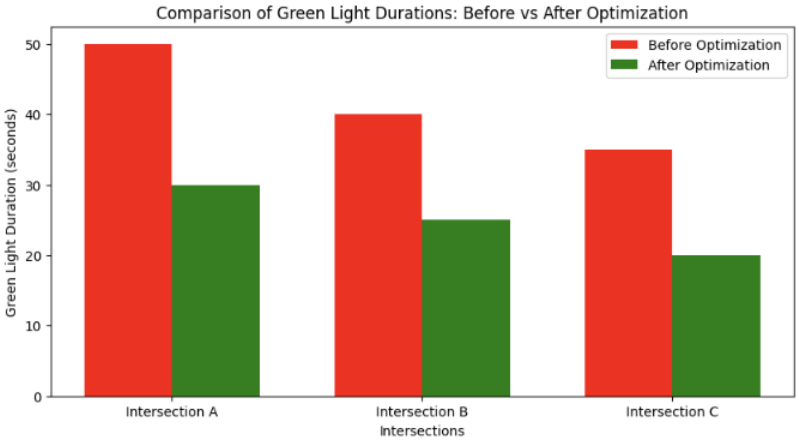

Fig 1: visualization of before and after optimization

7. Varying Constraints

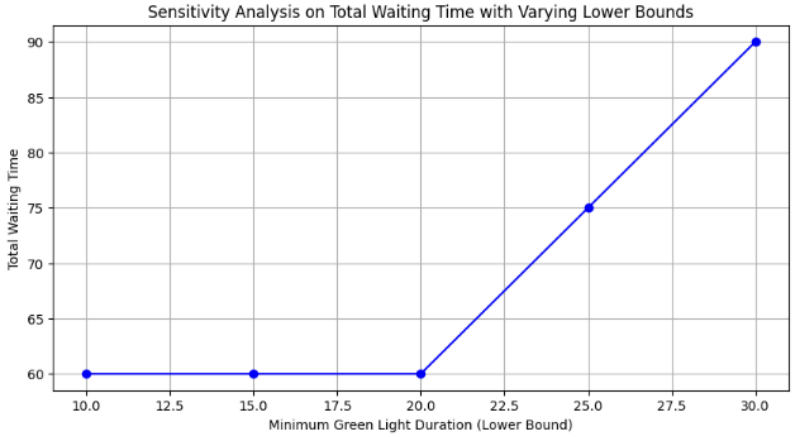

Fig 2: varying lower bounds of x1, x2, x3

The graph in Fig 2. shows that as the minimum green light duration (lower bound) increases, the total waiting time also increases. This is a sensible result: if you're forced to keep the green light on for longer at each intersection (due to the higher lower bound), the total waiting time (sum of all green light durations) will naturally be higher.

This kind of sensitivity analysis is valuable for understanding how changes in traffic light minimum durations can affect overall traffic flow, particularly in scenarios such as increased traffic volumes or changes in traffic patterns. The increasing trend in the graph indicates that there is a direct and proportional relationship between the minimum green light duration and the total waiting time, which is a straightforward and expected outcome.

Such analyses can help traffic engineers and urban planners to determine the impact of various traffic management strategies and make informed decisions to improve traffic flow and reduce congestion.

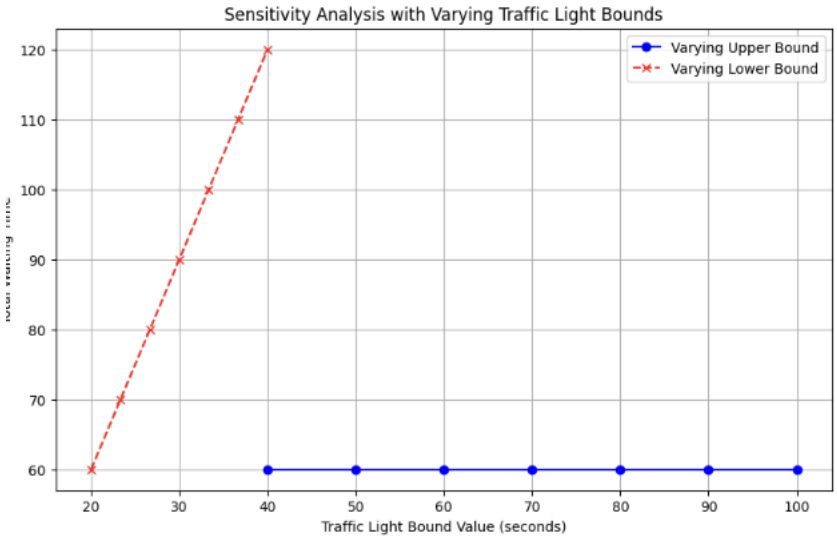

Fig 3: varying traffic light bounds

In fig 3. We plot two curves which represent:

- One curve will represent the total waiting time as we vary the upper bound (maximum green light duration).
- Another curve will represent the total waiting time as we vary the lower bound (minimum green light duration).

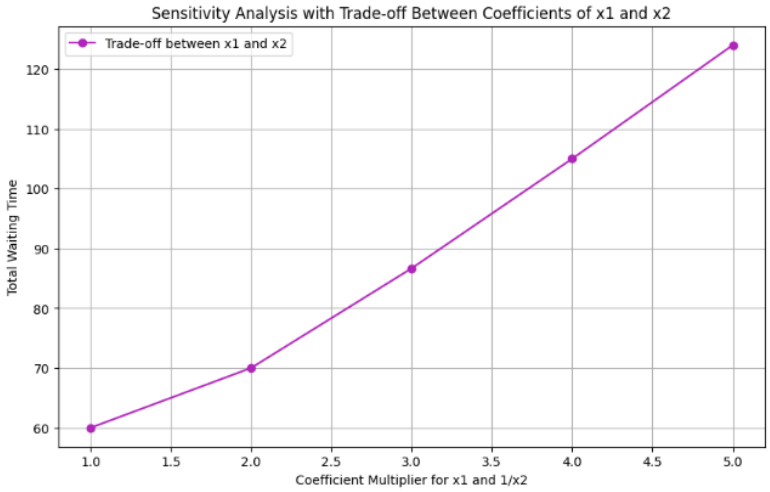

Fig 4: trade-off in priorities at intersection x1 and x2

The graph produced represents how the total waiting time at traffic lights changes as we adjust the priority between two intersections. Specifically, we increase the importance (weight) of the green light duration at intersection A (x_1) and decrease the importance of intersection B (x_2).

In this scenario, a higher coefficient for x_1 means that any additional second of green light at intersection A contributes more to the total waiting time than before, making it more 'costly' in terms of the objective function. Conversely, as the coefficient for x_2 decreases, additional green time at intersection B becomes 'cheaper'.

The X-axis of the graph shows the multiplier applied to the weight of x_1 (which also inversely affects x_2), and the Y-axis shows the resulting total waiting time calculated by the linear programming solution. The plotted line tracks how this total waiting time changes as the trade-off between x_1 and x_2 is varied.

The line trends upward, it suggests that prioritizing x_1 leads to an overall increase in total waiting time, possibly because the optimization is forced to allocate more time to intersection A at the expense of intersection B.

This could mean that intersection B is more critical to the overall traffic flow or that the constraints allow for a better optimization when x_2 is given more importance.

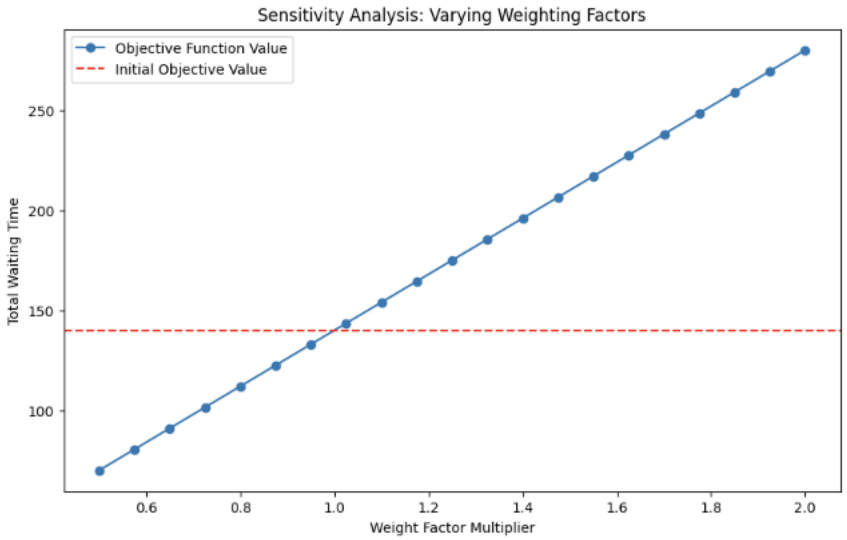

Fig 5: varying weight factor multiplier

In this plot, the x-axis represents the weight factor multiplier, which scales the relative importance of minimizing waiting times at different intersections. The y-axis shows the corresponding total waiting time (objective function value) for each weight factor multiplier. The red dashed line represents the initial objective value, and the blue line with markers shows how the objective function value changes as the weight factors are varied.

In this, we define the initial weighting factors (c = [1, 2, 3]) for the objective function, representing the relative importance of minimizing waiting times at each intersection. We then solve the initial problem and store the initial objective function value.

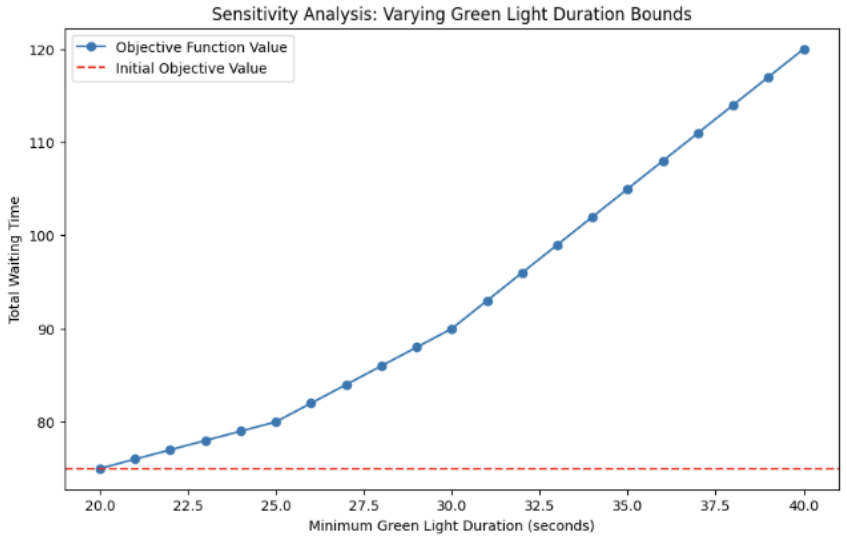

Fig 6: varying the minimum and maximum green light durations

In this example, we vary the minimum and maximum green light durations simultaneously for all intersections. We define a range of minimum durations from 20 to 40 seconds and a range of maximum durations from 40 to 80 seconds, both with 21 steps.

Inside the loop, we update the variable bounds (**new_bounds**) with the current minimum and maximum durations for all intersections. We then solve the linear programming problem with these modified bounds and store the resulting objective function value.

The plot shows the objective function value (total waiting time) on the y-axis and the minimum green light duration on the x-axis. The red dashed line represents the initial objective value for reference.

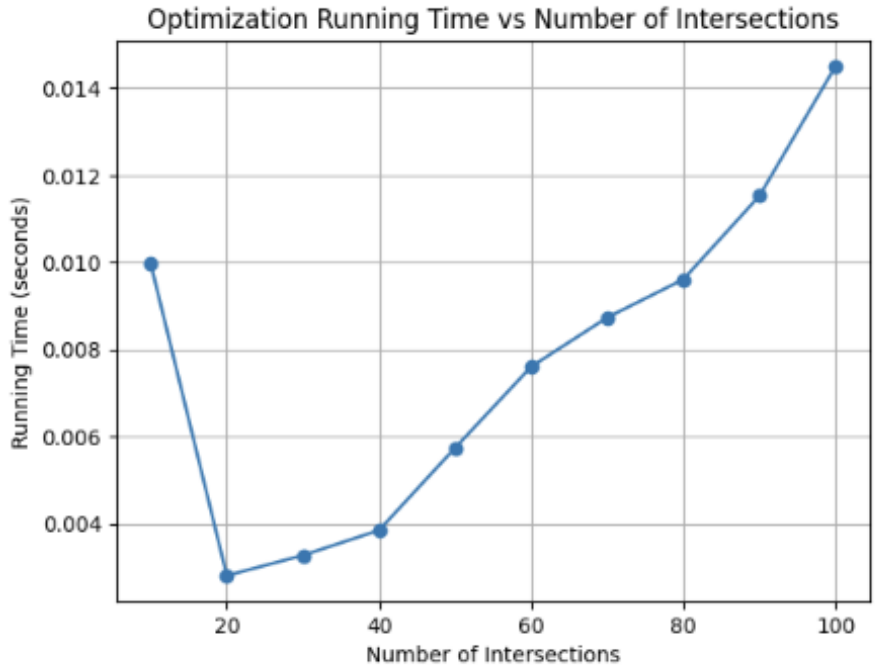

Fig 7: varying the number of intersections

In this example, we simulate different problem sizes by varying the number of intersections, which in turn changes the size of the objective function vector and the constraint matrices. This script measures the time taken by **linprog** from the SciPy library to solve the problem for each size. You can adapt this script to match the specifics of your urban traffic flow problem.

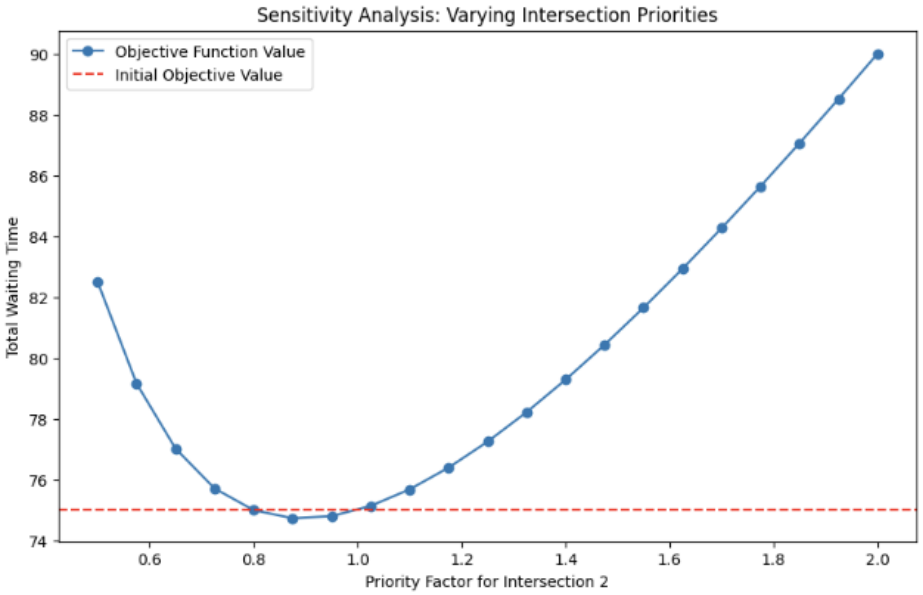

Fig 7: varying intersection priorities

In this example, we vary the priority or importance assigned to different intersections by adjusting the weighting factors in the objective function coefficients (c). We define a range of priority factors (e.g., from 0.5 to 2) and adjust the weighting factors for the second and third intersections accordingly, while keeping the first intersection's weight at 1.

This analysis can help identify the impact of prioritizing specific intersections over others based on factors such as traffic volume, proximity to critical infrastructure, or strategic importance. If the plot shows a significant change in the objective function value for certain priority factors, it may suggest the need to re-evaluate the prioritization of intersections or consider additional traffic management measures at high-priority locations.