Traffic lights timing optimizer poc

introduction

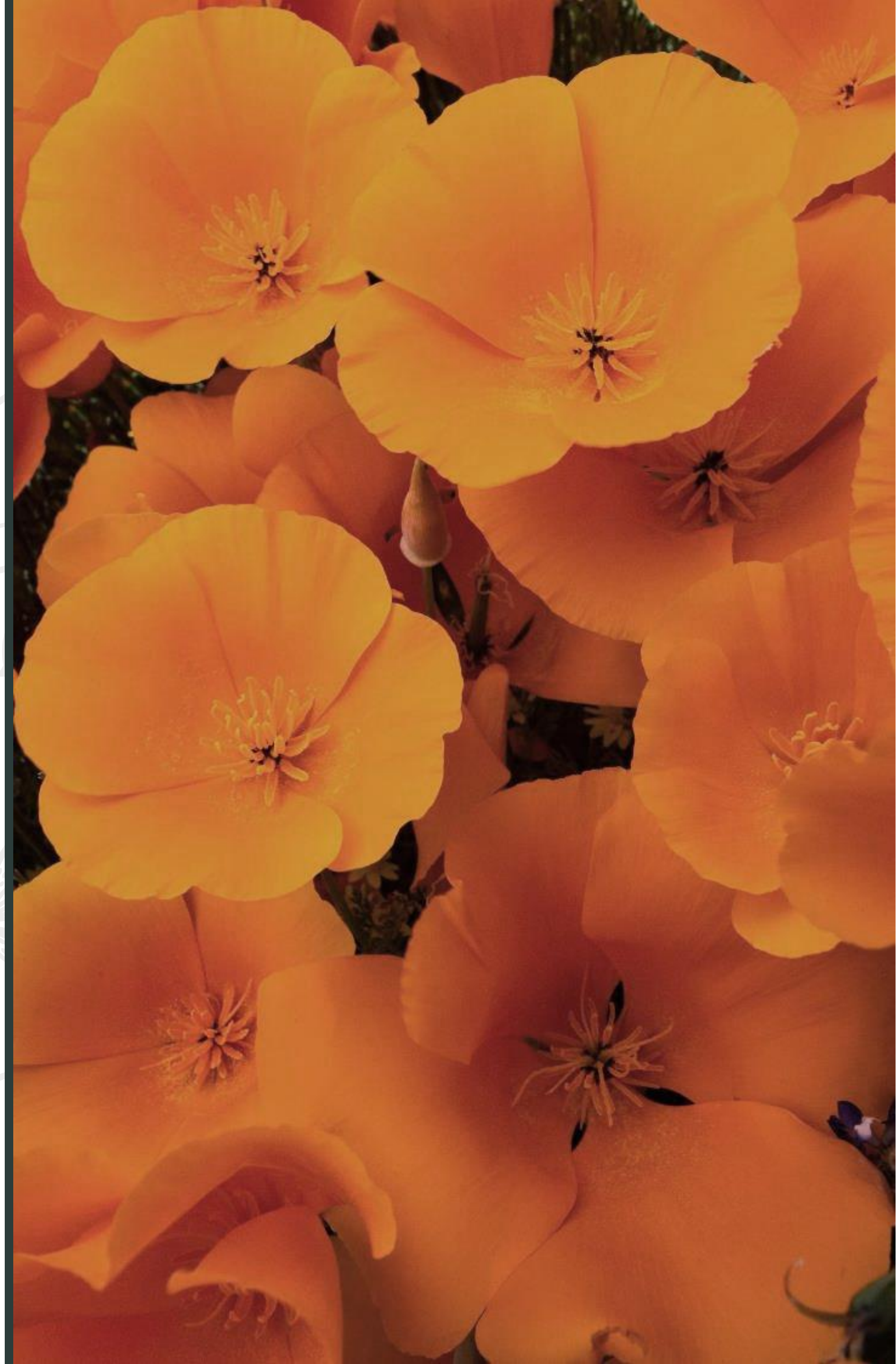
The Traffic Lights Timing Optimizer aims to address these challenges by implementing an adaptive traffic signal control system that adjusts light timings based on real-time traffic conditions. This proof of concept (PoC) seeks to demonstrate the effectiveness of using data-driven algorithms to optimize traffic flow at intersections.

Use of traffic lights timing optimizer poc

- **Reduces Congestion**: Minimize waiting times and traffic buildup.

- **Improve Safety**: Ensure pedestrian and vehicle safety.

- **Enhance Flow**: Optimize traffic flow during peak and off-peak hours.



Software used in traffic lights timing optimizer poc

- **VISSIM**
- **Overview**: A micro simulation tool that models traffic flow and allows for detailed modeling of vehicle interactions and behaviors.
- **Features**:
  - Supports multi-modal traffic (cars, buses, bicycles, pedestrians).
  - Highly customizable traffic signal control systems.
  - Integration with other software (e.g., GIS tools).
- **Use Cases**: Traffic signal optimization, congestion analysis, and pedestrian flow modeling.

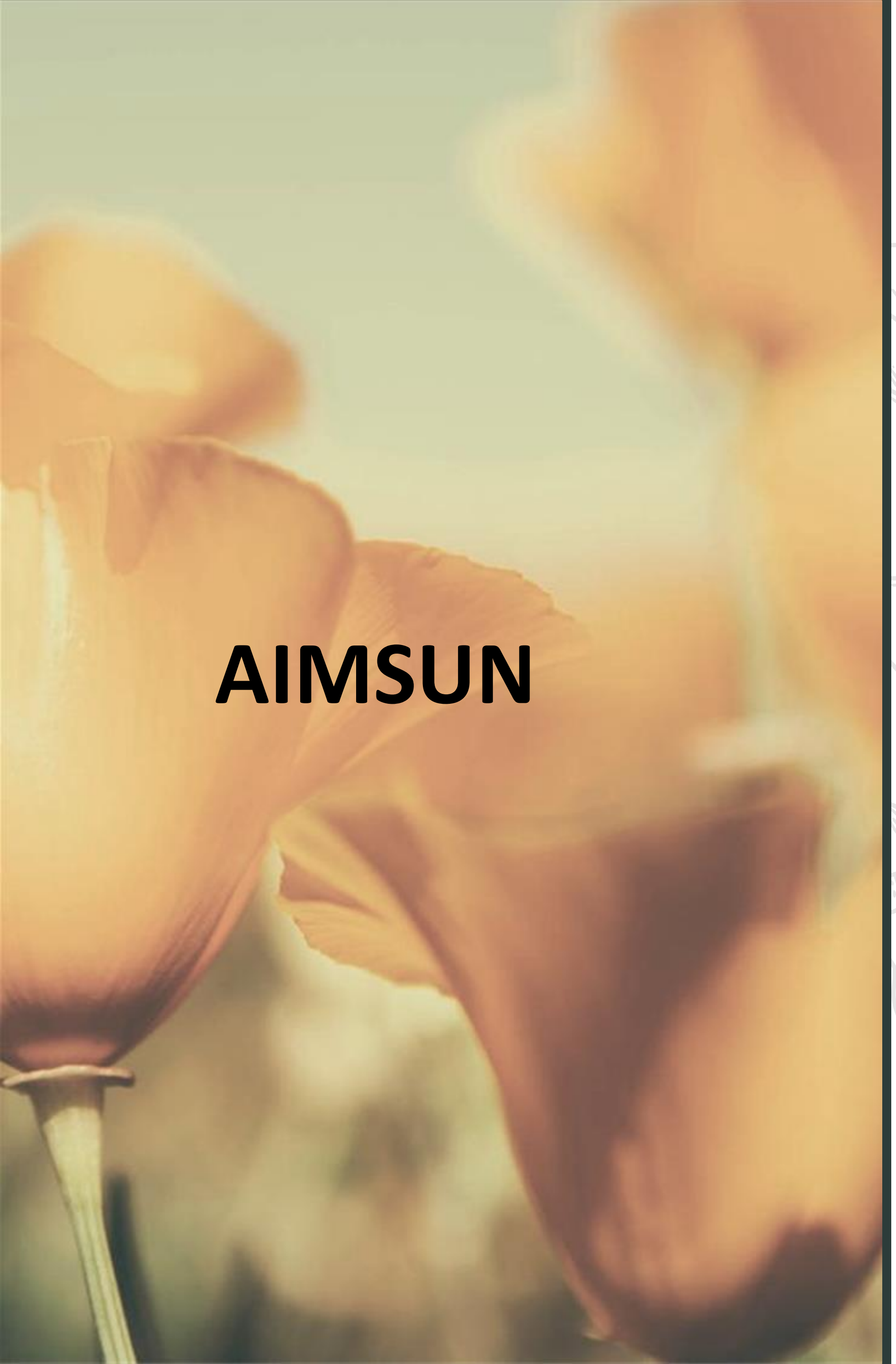
SUMO (Simulation of Urban MObility)

SUMO (Simulation of Urban MObility)

- **Overview**: An open-source, highly portable traffic simulation suite.
- **Features**:
  - Supports large-scale simulations with millions of vehicles.
  - Customizable vehicle types and routing algorithms.
  - Integration with other tools and programming languages (e.g., Python).

Dynamic Adjustment Logic

- **Real-Time Adaptation**:
    - Use real-time traffic data to adjust light timings dynamically.
    - Implement rules like:
        - If traffic volume exceeds a threshold, extend the green light duration.
        - If traffic is low, reduce the green light duration to minimize wait times.



AIMSUN

- **Overview**: A traffic simulation software that supports both microsimulation and macrosimulation.

- **Features**:
  - Real-time traffic modeling and analysis.
  - Advanced algorithms for traffic signal control and optimization.
  - Integration with data from various sources (e.g., GPS, traffic sensors).

Algorithm Development

Algorithm Development

Signal Timing Model: Develop algorithms based on traffic volume and patterns.

Adaptive Timing: Use real-time data to adjust light cycles dynamically.

Priority Systems: Implement bus or emergency vehicle prioritization.

```
class TrafficLight:
def __init__(self, signal_id, green_time, yellow_time, red_time):
self.signal_id = signal_id
self.green_time = green_time
self.yellow_time = yellow_time
self.red_time = red_time

def __repr__(self):
return (f"TrafficLight(signal_id={self.signal_id}, "
f"green_time={self.green_time}, "
f"yellow_time={self.yellow_time}, "
f"red_time={self.red_time})")
```

```
class TrafficLightSystem:
def __init__(self):
self.traffic_lights = {}

def create_signal(self, signal_id, green_time, yellow_time, red_time):
self.traffic_lights[signal_id] = TrafficLight(signal_id, green_time, yellow_time, red_time)

def read_signal(self, signal_id):
return self.traffic_lights.get(signal_id)

def update_signal(self, signal_id, green_time=None, yellow_time=None, red_time=None):
signal = self.traffic_lights.get(signal_id)
if signal:
if green_time is not None:
signal.green_time = green_time
if yellow_time is not None:
signal.yellow_time = yellow_time
if red_time is not None:
signal.red_time = red_time

def delete_signal(self, signal_id):
if signal_id in self.traffic_lights:
del self.traffic_lights[signal_id]

def optimize_traffic_signals(self, signal_id):
# Placeholder for optimization logic
signal = self.traffic_lights.get(signal_id)
if signal:
# Example: Increase green time by 10 seconds
signal.green_time += 10

def analyze_traffic_impact(self, impact_id):
# Placeholder for impact analysis logic
# For simplicity, let's just return the current timings
signal = self.traffic_lights.get(impact_id)
if signal:
return {
"signal_id": signal.signal_id,
"green_time": signal.green_time,
"yellow_time": signal.yellow_time,
"red_time": signal.red_time
}
return None
```

```
import unittest

class TestTrafficLightSystem(unittest.TestCase):
def setUp(self):
self.system = TrafficLightSystem()
self.system.create_signal("TL1", 30, 5, 25)

def test_create_signal(self):
self.assertIsNotNone(self.system.read_signal("TL1"))

def test_update_signal(self):
self.system.update_signal("TL1", green_time=40)
self.assertEqual(self.system.read_signal("TL1").green_time, 40)

def test_delete_signal(self):
self.system.delete_signal("TL1")
self.assertIsNone(self.system.read_signal("TL1"))

def test_optimize_traffic_signals(self):
self.system.optimize_traffic_signals("TL1")  # Optimization will fail as TL1 was deleted
self.system.create_signal("TL1", 30, 5, 25)
self.system.optimize_traffic_signals("TL1")
self.assertEqual(self.system.read_signal("TL1").green_time, 40)  # Initially 30 + 10

def test_analyze_traffic_impact(self):
impact = self.system.analyze_traffic_impact("TL1")
self.assertEqual(impact["signal_id"], "TL1")

if __name__ == "__main__":
unittest.main()
```

- **Algorithm Components**
- **Cycle Time Calculation**:
  - Define the total cycle time based on traffic volume.
- Use formulas like: Cycle Time=Green Time + Yellow Time + Red Time

- **Pedestrian Timing**:
- Allocate specific intervals for pedestrian crossing, which could adjust based on pedestrian volume.

Conclusion

the traffic lights timing optimizer PoC can significantly enhance urban mobility and reduce congestion. By leveraging real-time data and adaptive algorithms, it sets the stage for smarter city infrastructure.

Thank you