GHULAM ISHAQ KHAN INSTITUTE OF ENGINEERING SCIENCES AND TECHNOLOGY – GIKI

CS351 – INTRO TO AI

SEMESTER PROJECT

Project Report

Instructor:

Prof. Hafiz Syed

Ahmed Qasim

Group Members:

Muhammad Bilal - 2022359

Hassaan Shah - 2022385

Traffic Optimization Using Genetic Algorithms

Project Overview

This project, **Traffic Optimization Using Genetic Algorithms**, seeks to optimize the traffic light timings at multiple intersections in a simulated urban environment. The primary goal is to **minimize the total waiting time for vehicles while ensuring smooth traffic flow**. The Genetic Algorithm (GA) serves as the optimization technique, simulating evolutionary principles to find near-optimal solutions for traffic light configurations.

Genetic Algorithm Process in the Project

The Genetic Algorithm employed in this project follows a systematic evolutionary process, consisting of key steps: **initialization, evaluation, selection, crossover, mutation, and iteration through generations**. Below, we detail the process as implemented in the code.

Initialization

The algorithm starts by generating a **random population** of candidate solutions. Each solution (individual) is represented as a list of timing values for traffic lights, which include:

- Green light duration
- Yellow light duration
- Red light duration

For each intersection, six timing values are generated:

1. North-South (NS) green, yellow, red
2. East-West (EW) green, yellow, red

The ***create_individual*** method generates these timing values while adhering to specified constraints:

- Green: 20 to 60 seconds
- Yellow: 3 to 5 seconds
- Red: 20 to 60 seconds

Population Size: The initial population consists of 30 individuals (`population_size`).

Fitness Evaluation

The fitness of each individual is evaluated using the ***simulate_traffic*** method. This method simulates traffic flow over a fixed simulation time (100 seconds) and calculates the **negative total waiting time** (minimizing this is equivalent to maximizing fitness).

Simulation Details:

- Traffic flow is managed by ***TrafficLight*** objects.
- Light states (GREEN, YELLOW, RED) are determined using the provided timing values.
- New cars are randomly added to queues at each timestep.
- Cars are processed based on the state of the lights, and waiting times for vehicles are accumulated.

The fitness of a solution is thus determined by how effectively it minimizes total vehicle waiting time across all intersections.

Selection

Parent selection is performed using **tournament selection**:

- Two separate tournaments are conducted, each selecting the best individual from a randomly chosen subset of 3 individuals.
- The winners of these tournaments are selected as parents for the next generation.

This selection method balances exploration and exploitation by favoring higher-fitness individuals while maintaining diversity.

Crossover

Crossover is the process of combining genetic information from two parents to create offspring. The ***crossover*** method performs **intersection-wise crossover**:

- Each intersection's timings are inherited from either parent, chosen randomly.
- A crossover rate of 80% (***crossover_rate***) ensures that most offspring are produced via crossover, while some are direct copies of parents.

This process introduces variation while preserving good traits from both parents.

Mutation

Mutation introduces random changes to individuals, preventing premature convergence and maintaining population diversity. The ***mutate*** method:

- Mutates each timing value with a small probability (`mutation_rate = 0.1`).
- Ensures that mutated values adhere to constraints:
  - Green: 20 to 60 seconds
  - Yellow: 3 to 5 seconds
  - Red: 20 to 60 seconds

Mutation adds exploratory potential to the algorithm, allowing it to escape local optima.

Elitism

To preserve high-performing solutions, **elitism** is implemented:

- The top 2 individuals (***elite_size***) in each generation are directly carried forward to the next generation.
- This ensures that the best solutions are not lost during the evolutionary process.

Generational Evolution

The GA iterates through 50 generations (***num_generations***), improving the population at each step. For each generation:

1. The fitness of all individuals is evaluated.
2. A new population is created using elitism, selection, crossover, and mutation.

3. The best solution and its fitness are tracked.

At the end of the process, the solution with the highest fitness represents the **optimal traffic light timings.**

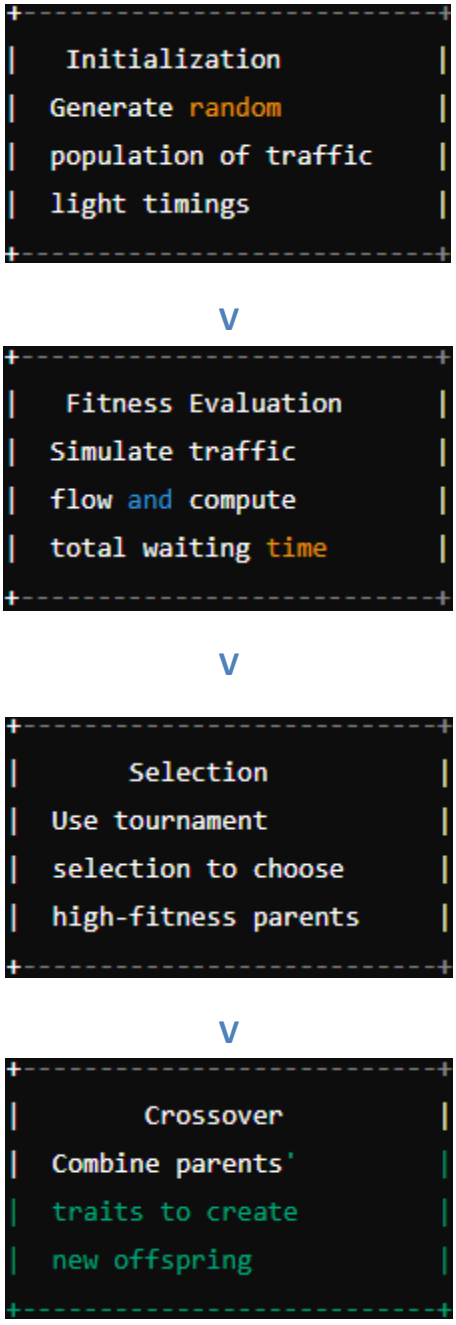
Key Metrics and Parameters

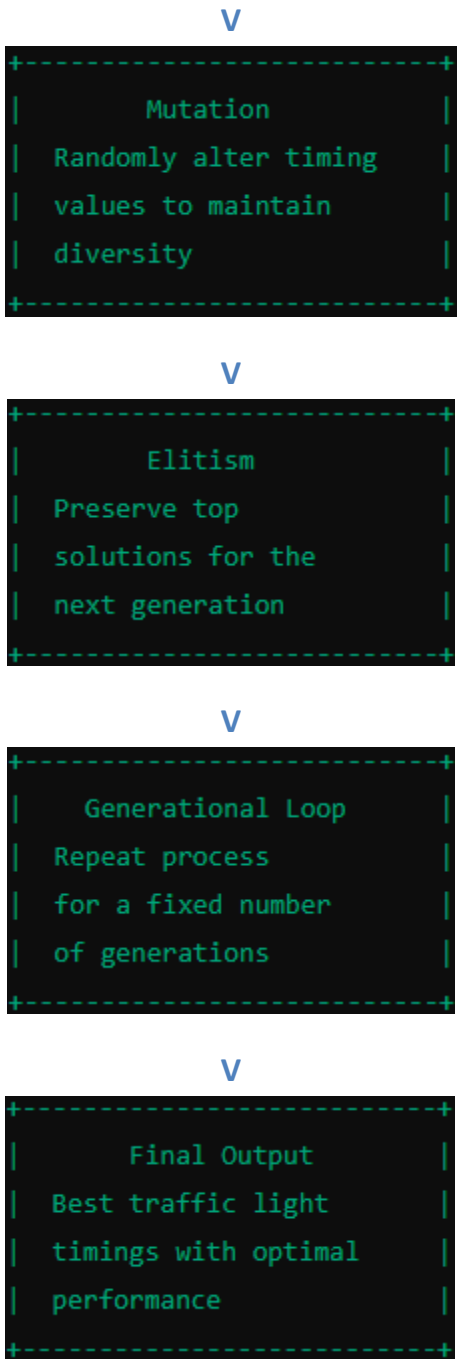
Parameter	Value
Number of intersections	4
Population size	30
Number of generations	50
Mutation rate	0.1
Crossover rate	0.8
Elite size	2
Green light duration range	20–60 seconds
Yellow light duration range	3–5 seconds
Red light duration range	20–60 seconds

Summary of Algorithm Workflow

1. **Initialization**: Generate a random population of traffic light timing configurations.
2. **Fitness Evaluation**: Simulate traffic flow and compute waiting times to evaluate fitness.
3. **Selection**: Use tournament selection to choose parents.
4. **Crossover**: Combine parents' traits to produce offspring.
5. **Mutation**: Introduce random changes to offspring.
6. **Elitism**: Preserve the best individuals.
7. **Iteration**: Repeat the process for 50 generations to converge on an optimal solution.

Diagram Explanation of the Process:




V
Mutation
Randomly alter timing values to maintain diversity
V
Elitism
Preserve top solutions for the next generation
V
Generational Loop
Repeat process for a fixed number of generations
V
Final Output
Best traffic light timings with optimal performance

GUI Development

Implementation Overview

The GUI for this project was developed using **Tkinter**, a standard Python library for GUI applications. Additional functionalities and enhancements were implemented using:

- **PIL (Pillow)**: For image handling and rendering.
- **Threading**: To ensure smooth execution of simulations without freezing the GUI.

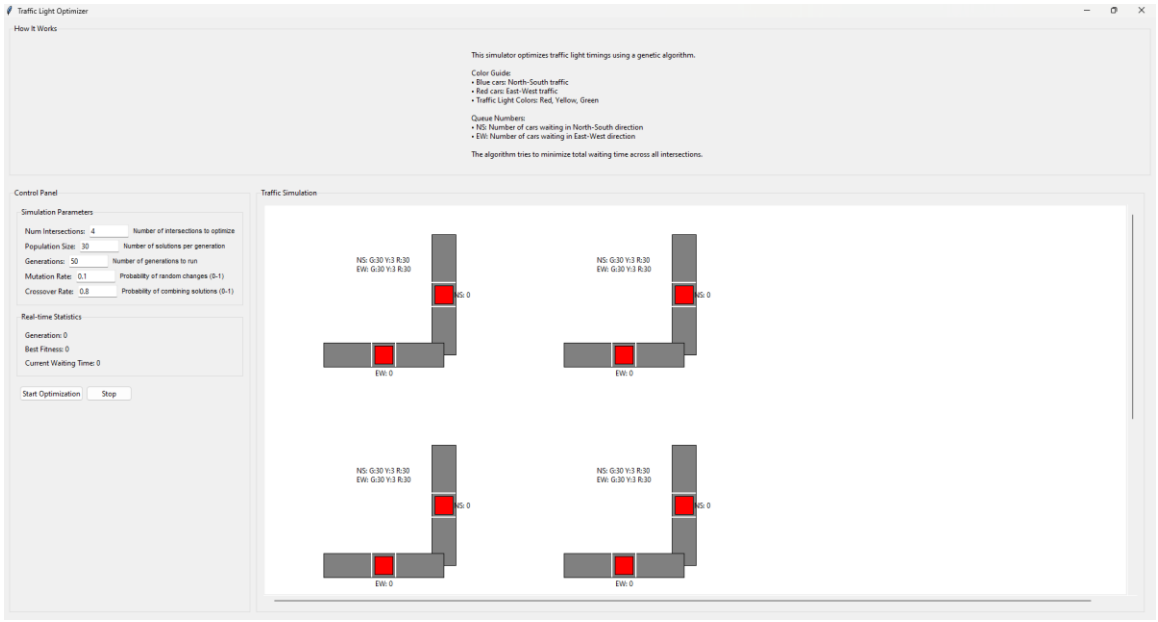
The following modules and techniques were employed:

- ***tkinter* Widgets**: To create the main application window, labels, buttons, and other interactive components.
- **Traffic Simulation Display**: Dynamic updates of traffic light states and vehicle queues.
- **Integration with Backend**: The GUI interacts seamlessly with the genetic algorithm via the ***SimpleTrafficOptimizer*** class.
- **Threaded Execution**: To allow simulations to run concurrently with GUI updates, enhancing user experience.

Features

- **Real-Time Simulation**: Displays changing traffic light states and vehicle queues in real time.
- **Start/Stop Controls**: Allows users to start or stop the simulation.
- **Visualization**: Provides a graphical representation of intersections and traffic flow.
- **Performance Metrics**: Displays the total waiting time and other key metrics dynamically.

GUI Preview:


Traffic Light Optimizer
How It Works
This simulator optimizes traffic light timings using a genetic algorithm.
Color Guide:
• Blue cars: North-South traffic
• Red cars: East-West traffic
• Traffic Light Colors: Red, Yellow, Green
Queue Numbers:
• NS: Number of cars waiting in North-South direction
• EW: Number of cars waiting in East-West direction
The algorithm tries to minimize total waiting time across all intersections.
Control Panel
Simulation Parameters
Num Intersections: 4
Number of intersections to optimize
Population Size: 30
Number of solutions per generation
Generations: 50
Number of generations to run
Mutation Rate: 0.1
Probability of random changes (0-1)
Crossover Rate: 0.8
Probability of combining solutions (0-1)
Real-time Statistics
Generation: 0
Best Fitness: 0
Current Waiting Time: 0
Start Optimization
Stop
Traffic Simulation
NS: G:30 Y:3 R:30
EW: G:30 Y:3 R:30
NS: 0
EW: 0