GREEN WAVES - OPTIMIZING TRAFFIC LIGHT CONTROL

1. INTRODUCERE

Congestionarea traficului a generat numeroase provocări în orașele aglomerate. Congestionarea înseamnă pierdere de timp, combustibil și frustrare. Problema este rezumată de creșterea semnificativă a numărului de vehicule în fiecare an în toate părțile lumii, în special în marile orașe, ceea ce duce la apariția unor blocaje de trafic acute. Prin urmare, necesitatea de a crea o nouă infrastructură, cum ar fi poduri, tunele sau semafoare la intersecții, este necesară pentru a facilita fluxul de vehicule. Podurile și tunelele nu sunt întotdeauna soluția perfectă din cauza costurilor ridicate de construcție, întreținere și lipsei de spațiu. Pe de altă parte, utilizarea semafoarelor cu cicluri fixe normale va crește blocajele de trafic și emisiile de dioxid de carbon. Semafoarele cu "cicluri fixe normale" se referă la modul tradițional de funcționare al acestora, în care durata fiecărei faze a semaforului este predefinită și nu se ajustează în funcție de condițiile de trafic în timp real. Prin urmare, cercetătorii au lucrat recent la propunerea mai multor tehnici care să gestioneze intersecțiile într-un mod inteligent, adecvat fluxului actual de vehicule.

2. TEHNICI DE OPTIMIZARE A TRAFICULUI

Tehnicile de optimizare a traficului la intersecții pot fi împărțite în două categorii principale:

- Sisteme inteligente de semafoare
- Sisteme virtuale de semafoare.

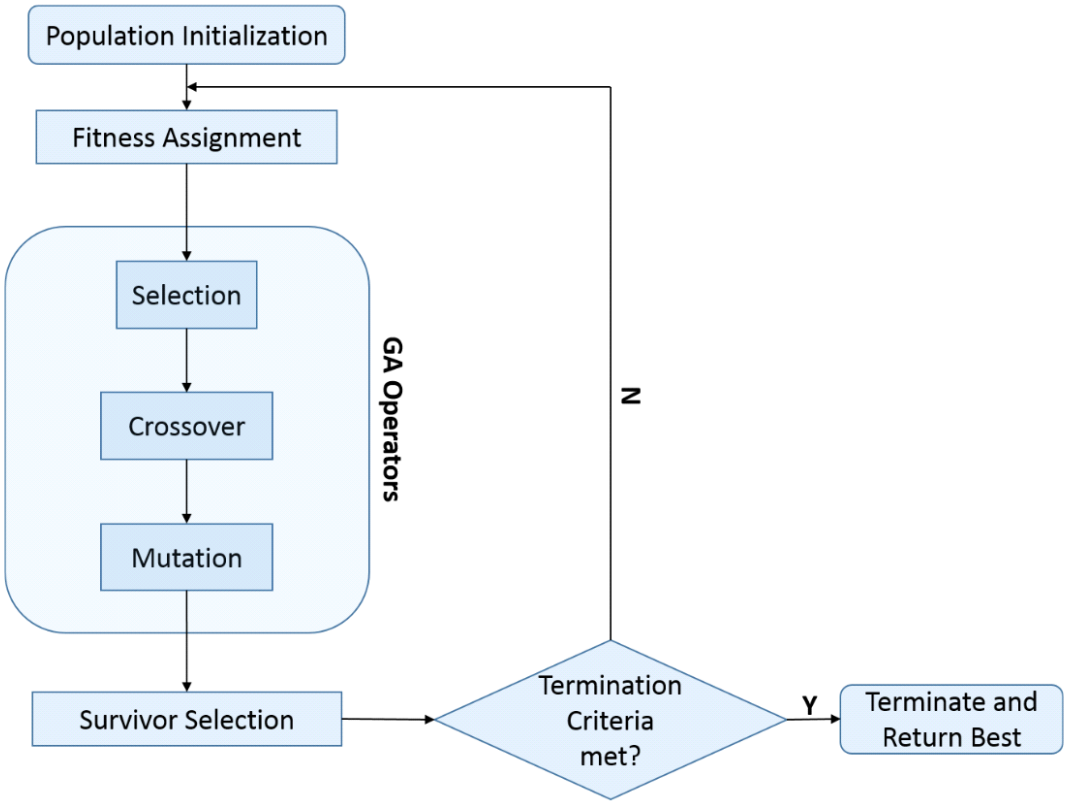
Iată o prezentare generală a fiecărei tehnici:

1. Sisteme inteligente de semafoare:

Aceste sisteme utilizează tehnologii avansate pentru a monitoriza și controla semafoarele în timp real, în funcție de condițiile actuale de trafic. Principalele tehnici utilizate în sistemele inteligente de semafoare includ:

- Algoritmi genetici: Algoritmul genetic este o tehnică de optimizare inspirată din procesele biologice de selecție naturală și moștenire genetică. Este utilizat pentru a găsi configurații optime ale semafoarelor și pentru a îmbunătăți fluxul de

trafic la intersecții.



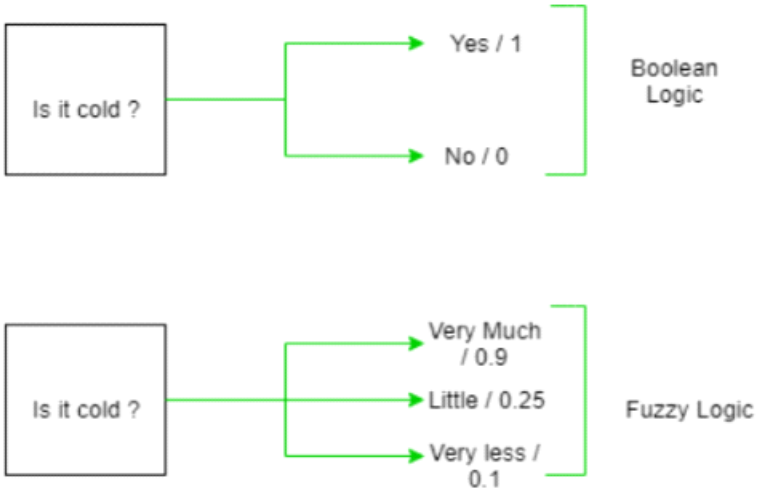
Acesta are capacitatea de a reînnoi constant deciziile și de a crea soluții. Algoritmul genetic este implementat în cinci etape, începând cu:

1. populația inițială,
2. funcția de adecvare,
3. selecțiile,
4. încrucișările,
5. mutațiile.

O propunere a fost utilizarea algoritmului genetic pentru îmbunătățirea fluxului de trafic al vehiculelor la intersecții, concentrându-se pe creșterea duratei luminii verzi până la cinci secunde, ceea ce a dus la o îmbunătățire a rezultatelor cu 21.9%. Altii au propus, de asemenea, utilizarea algoritmului genetic aplicat la o intersecție și patru semafoare. În spusele lor, ei au sugerat o durată minimă a luminii verzi de 10 secunde și o durată

maximă a luminii verzi de 60 de secunde. Rezultatele au arătat, după simularea dezvoltată în Matlab, o îmbunătățire cu 28.4% în numărul de vehicule care au părăsit intersecția în comparație cu alte lucrări care au folosit același algoritm.

- Rețele neurale: Rețelele neurale sunt modele matematice inspirate din funcționarea creierului uman, care pot fi antrenate să învețe și să recunoască modele în date. În contextul traficului rutier, rețelele neurale pot fi utilizate pentru a prezice modele de trafic și pentru a adapta semafoarele în consecință.
- Controlori inteligenți: Acestea sunt sisteme de control bazate pe logica și algoritmi complexi, care iau decizii în timp real în funcție de datele de intrare primite de la senzorii de trafic și alte surse de informații.
- Logica 'Fuzzy': Logica 'fuzzy' este o metodă de modelare matematică care tratează gradul de adevăr ca o valoare continuă între "adevărat" și "fals". În contextul semafoarelor, logica 'fuzzy' poate fi utilizată pentru a lua decizii de control în funcție de parametrii de trafic și de condițiile de mediu.



Folosirea logicii fuzzy a fost propusă în mai multe lucrări pentru îmbunătățirea fluxului de vehicule, cum ar fi sugestia lui Omina (James Adunya Omina), care a adoptat un studiu transversal cu scopul de a monitoriza traficul în centrul de afaceri al orașului Nairobi și în zonele înconjurătoare. Datele au fost colectate prin observații asupra comportamentului traficului la trei intersecții, respectiv Railways, Hili Salice și Oficiul Poștal General. Datele au fost analizate și prezentate utilizând statistici descriptive, tabele și grafice folosind Excel 2003. Pentru a testa controlul adaptiv al semafoarelor, s-a folosit o simulare utilizând instrumentele MATLAB, cu unele dezvoltări folosind software-ul C++ și Qt pentru validare. Rezultatele au arătat că utilizarea logicii fuzzy a

fost de succes, deoarece a redus timpul mediu de așteptare al vehiculelor în comparație cu unitățile de control fixe.

2. Sisteme virtuale de semafoare:

Aceste sisteme utilizează tehnologii de comunicații și informații pentru a coordona fluxul de trafic fără a fi nevoie de semafoare fizice. Principalele tehnici utilizate în sistemele virtuale de semafoare includ:

- Comunicare Vehicul-la-Infrastructură (V2I): Această tehnologie permite vehiculelor să comunice cu infrastructura rutieră, cum ar fi semafoarele și sistemele de gestionare a traficului. Prin transmiterea informațiilor despre poziție, viteză și direcție, vehiculele pot primi instrucțiuni pentru a optimiza deplasarea lor prin intersecții.

- Comunicare Vehicul-la-Vehicul (V2V): Această tehnologie permite vehiculelor să comunice între ele în timp real. Prin partajarea informațiilor despre mișcare și intențiile de deplasare, vehiculele pot coopera pentru a evita coliziunile și pentru a optimiza fluxul de trafic la intersecții.

- Sisteme de navigație și rutare inteligentă: Aceste sisteme utilizează date despre trafic, condiții meteorologice și alte factori pentru a calcula rutele optime pentru vehiculele în tranzit. Prin dirijarea vehiculelor pe rutele cele mai eficiente, aceste sisteme pot reduce congestionarea la intersecții și în zonele urbane aglomerate.

3.IMPLEMENTAREA ALGORITMULUI GENETIC

```
import random

class Intersection:
    def __init__(self, id, green_duration, red_duration):
        self.id = id
        self.green_duration = green_duration
        self.red_duration = red_duration
        self.state = 0   # 0 pentru roșu, 1 pentru verde
        self.time_elapsed = 0
        self.queue = []

    def update(self):
        self.time_elapsed += 1
        if self.state == 1:
            self.queue = [vehicle for vehicle in self.queue if vehicle.wait_time < self.green_duration]
            for vehicle in self.queue:
                vehicle.wait_time += 1

class Vehicle:
```

```
    def __init__(self, id, intersection_id):
        self.id = id
        self.intersection_id = intersection_id
        self.wait_time = 0

class TrafficLightConfiguration:
    def __init__(self, intersections):
        self.intersections = intersections

    def apply(self):
        total_wait_time = sum(intersection.time_elapsed for intersection in self.intersections)
        return total_wait_time / len(self.intersections)   # Returnăm media timpului total de așteptare

def fitness_function(configuration):
    total_wait_time = 0
    total_travel_time = 0
    vehicles = [Vehicle(i, random.randint(0, len(configuration.intersections) - 1)) for i in range(100)]

    for vehicle in vehicles:
        current_intersection = configuration.intersections[vehicle.intersection_id]
        total_travel_time += current_intersection.green_duration
        vehicle.wait_time = current_intersection.time_elapsed
        current_intersection.queue.append(vehicle)

    for intersection in configuration.intersections:
        intersection.update()

    total_wait_time = sum(vehicle.wait_time for vehicle in vehicles)
    return total_wait_time + total_travel_time

def selection(population, fitness_values):
    idx = random.choices(range(len(population)), weights=[1 / (fitness + 1) for fitness in
fitness_values])[0]
    return population[idx]

def crossover(parent1, parent2):
    crossover_point = random.randint(1, len(parent1.intersections) - 1)
    child_intersections = parent1.intersections[:crossover_point] + parent2.intersections[crossover_point:]
    return TrafficLightConfiguration(child_intersections)

def mutation(configuration, mutation_rate):
    for intersection in configuration.intersections:
        if random.random() < mutation_rate:
            intersection.green_duration = random.randint(5, 15)
            intersection.red_duration = random.randint(5, 15)
    return configuration

def generate_initial_population(population_size, num_intersections):
    return [TrafficLightConfiguration([Intersection(i, random.randint(5, 15), random.randint(5, 15)) for i in
range(num_intersections)]) for _ in range(population_size)]

def genetic_algorithm(population_size, generations, mutation_rate, num_intersections):
    population = generate_initial_population(population_size, num_intersections)
    for _ in range(generations):
        fitness_values = [fitness_function(configuration) for configuration in population]
        new_population = []
```

```
        for _ in range(population_size):
            parent1 = selection(population, fitness_values)
            parent2 = selection(population, fitness_values)
            child = crossover(parent1, parent2)
            child = mutation(child, mutation_rate)
            new_population.append(child)
        population = new_population
    best_solution_idx = min(range(len(population)), key=lambda x: fitness_function(population[x]))
    best_solution = population[best_solution_idx]
    return best_solution

population_size = 100
generations = 50
mutation_rate = 0.1
num_intersections = 5

best_solution = genetic_algorithm(population_size, generations, mutation_rate, num_intersections)
print("Configurația optimă a semafoarelor:")
for intersection in best_solution.intersections:
    print(f"Intersecția {intersection.id}: Durata verde = {intersection.green_duration} min, Durata roșu =
{intersection.red_duration} min")
print("Fitness:", fitness_function(best_solution))
```

- Intersection:

__init__: Această funcție este constructorul clasei Intersection. Atribuie fiecărei intersecții un id, durata, pentru starea verde și starea roșie a semaforului, setează starea inițială a semaforului la roșu, inițializează timpul scurs la 0 și creează o coadă vidă pentru vehiculele care așteaptă la intersecție.

update: Această metodă actualizează timpul scurs la intersecție și gestionează comportamentul semaforului în funcție de starea sa actuală. Dacă semaforul este în starea verde, elimină vehiculele care au așteptat prea mult timp și crește timpul de așteptare pentru celelalte vehicule din coadă.

- Vehicle:

__init__: Constructorul clasei Vehicle. Atribuie fiecărui vehicul un id și id-ul intersecției la care se află. Inițializează timpul de așteptare al vehiculului la 0.

- TrafficLightConfiguration:

__init__: Constructorul clasei TrafficLightConfiguration. Primește o listă de intersecții și le stochează într-un atribut al obiectului.

apply: Această metodă calculează un scor de fitness pentru configurația semafoarelor. În acest caz, se calculează media timpului total petrecut la semafoare pentru toate intersecțiile din configurație.

fitness_function:

Calculează un scor de fitness pentru o anumită configurație de semafoare. Se adună timpul total de așteptare al vehiculelor și timpul total de călătorie al vehiculelor prin intersecții. Scopul este de a minimiza acest scor de fitness.

selection:

Realizează selecția în cadrul algoritmului genetic, selectând configurațiile de semafoare pe baza scorurilor de fitness.

crossover:

Realizează operația de încrucișare între două configurații de semafoare pentru a genera un copil. Acesta efectuează un punct de încrucișare, luând jumătate din intersecțiile primului părinte și jumătate din intersecțiile celui de-al doilea părinte.

mutation:

Realizează operația de mutație asupra unei configurații de semafoare cu o anumită rată de mutație. Mutațiile consta în modificarea aleatorie a duratei verzi și roșii a intersecțiilor cu o probabilitate dată.

generate_initial_population:

Generează o populație inițială de configurații aleatorii de semafoare.

genetic_algorithm:

Implementează algoritmul genetic pentru a optimiza configurațiile semafoarelor. Realizează selecția, încrucișarea și mutația pentru a genera o nouă generație de configurații, iar apoi determină cea mai bună configurație din această generație.

Am realizat testari, pe rezultatele obtinute in 10 rulari diferite cu urmatoarele valori:

population_size = 100

generations = 50

mutation_rate = 0.1

num_intersections = 5

Nr rulare	**Calitate (fitness value)**
1	46254
2	44223
3	60203
4	48814
5	51667
6	48482
7	47066
8	61966
9	47489
10	61532

Analiza calității	
Best value	44223
Average value	51979.6
Worst value	61966

- **Exemplu de afisare dupa rulare:**

Configurația optimă a semafoarelor:

Intersecția 0: Durata verde = 9 min, Durata roșu = 12 min

Intersecția 1: Durata verde = 12 min, Durata roșu = 14 min

Intersecția 2: Durata verde = 12 min, Durata roșu = 9 min

Intersecția 3: Durata verde = 14 min, Durata roșu = 11 min

Intersecția 4: Durata verde = 7 min, Durata roșu = 7 min

Fitness: 61532