МІНІСТЕРСТВО ОСВІТИ і НАУКИ УКРАЇНИ

ПрАТ «ПВНЗ «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»

Кафедра Економічної кібернетики та інженерії програмного забезпечення

ДО ЗАХИСТУ ДОПУЩЕНА
Зав.кафедри _____
д.е.н., доцент Левицький С.І.

БАКАЛАВРСЬКА ДИПЛОМНА РОБОТА

РОЗРОБКА СИСТЕМИ ОПТИМІЗАЦІЇ СПОЖИВАННЯ ПАЛИВА НА ОСНОВІ ГЕНЕТИЧНИХ АЛГОРИТМІВ

Виконав
ст. гр. ІПЗ-217 _____ В.В. Волошин
(підпис)

Керівник _____ О.А. Хараджян
к.т.н. (підпис)

Запоріжжя
2022

ПРАТ «ПВНЗ «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ
ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»

Кафедра Економічної кібернетики та інженерії програмного забезпечення

ЗАТВЕРДЖУЮ
Зав.кафедри _____
д.е.н., доцент Левицький С.І.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ

Студенту гр. *ІПЗ – 217*, спеціальності *«Інженерія програмного забезпечення»*

Волошину Владиславу Віталійовичу

1.Тема: *Розробка системи оптимізації споживання палива на основі генетичних алгоритмів.*

затверджена наказом по інституту № 06.2-14-2 від 18.02.2022 р.

2. Термін здачі студентом закінченої роботи: 17.06.2022 р.

3. Перелік питань, що підлягають розробці:

1. Аналіз роботи двигунів внутрішнього згоряння.
2. Аналіз методів оптимізації роботи двигунів внутрішнього згоряння.
3. Застосування генетичних алгоритмів для оптимізації.
4. Обгрунтування вибору програмних засобів.
5. Розробка алгоритму оптимізації споживання палива.
6. Опис даних програми.
7. Опис програмних модулів.
8. Тестування системи оптимізації споживання палива.

Дата видачі завдання: 17.01.2022 р.

Студент _____ В.В. Волошин
(підпис) (прізвище та ініціали)

Керівник роботи _____ О.А. Хараджян
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Дипломний робота містить 81 сторінок пояснюючої записки, 18 рисунків, 3 додатки.

Об'єкт дослідження: системи оптимізації на основі генетичних алгоритмів.

Предмет дослідження: системи оптимізації споживання палива на основі генетичних алгоритмів.

Мета роботи: розробити алгоритми та програмну реалізацію системи оптимізації споживання палива на основі генетичних алгоритмів.

Завдання роботи: аналіз роботи двигунів внутрішнього згоряння; аналіз методів оптимізації роботи двигунів внутрішнього згоряння; застосування генетичних алгоритмів для оптимізації; обгрунтування вибору програмних засобів; розробка алгоритму оптимізації споживання палива; тестування системи оптимізації споживання палива.

В роботі розглянуто принцип роботи автомобільного двигуна внутрішнього згорання, та створено спрощену модель легкового автомобіля, котра дозволяє розрахувати миттєву витрату палива в залежності від режиму роботи двигуна та положенні органів керування.

Для пошуку оптимальної стратегії керування автомобілем використано генетичний алгоритм.

Розроблено програму моделювання та оптимізації роботи автомобільного двигуна внутрішнього згорання.

ГЕНЕТИЧНІ АЛГОРИТМИ, ДВИГУН ВНУТРІШНЬОГО ЗГОРЯННЯ, МОДЕЛЮВАННЯ, ОПТИМІЗАЦІЯ, PYTHON

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Слово / словосполучення	Скорочення	Умови використання
Г		
Генетичний алгоритм	ГА	
Д		
Двигун внутрішнього згоряння	ДВЗ	
К		
Коефіцієнт корисної дії	ККД	
С		
Constant speed	CS	
G		
Genetic algorithm	GA	

ВСТУП

Клімат нашої планети постійно змінювався протягом усієї геологічної історії Землі, і ці зміни супроводжувалися значними коливаннями середньосвітових температур.

Однак зараз потепління відбувається набагато швидше, ніж будь-коли в минулому. Зміна клімату виражається не тільки у підвищенні температури, а й у екстремальних погодних явищах, підвищенні рівня моря, зміні популяцій диких тварин та місць їх проживання, і багатьох інших явищах.

Наукове співтовариство досягло консенсусу з приводу того, що глобальне потепління в основному викликане діяльністю людей – такого висновку дійшли 97% кліматологів.

Одним з основних факторів зміни клімату є спалювання викопних видів палива, вугілля, газу та нафти, що призвело до підвищення концентрації парникових газів, наприклад, вуглекислого газу, в нашій атмосфері.

Для багатьох галузей діяльності людства побічним ефектом є викид парникових газів. Серед них виробництво електроенергії (25%), промисловість (24%), сільске господарство (11%). Але найбільшим джерелом викидів вуглекислого газу є транспортна галузь (27%). Чимала частина викидів транспортної галузі припадає на автомобілі з двигунами внутрішнього згорання.

На даний момент неможливо замінити автомобілі з двигунами внутрішнього згорання. Галузь електроавтомобілів знаходиться на стадії швидкого росту, але вона не здатна сьогодні задовольнити попит на транспортування. Крім того вони використовують електроенергію, яка також більшістю виробляється за рахунок спалювання викопного палива, хоч і з більшою ефективністю.

Тому єдиним виходом є підвищення паливної ефективності при використання автомобілів з двигунами внутрішнього згоряння. Одним зі шляхів, якими це досягається – збільшення ефективності ДВЗ: створення двигунів, які

виконують свою задачу, використовуючи менше палива, підтримання ДВЗ в справному стані, тощо.

Інший шлях – це оптимізація стратегії керування автомобілем з точки зору мінімізації витрат пального. Інтенсивність відкриття акселератора, інтенсивність гальмування, оберти двигуна для перемикання передач – все це параметри, змінюючи які можна впливати на витрату пального автомобілем.

В цій роботі розроблено моделі двигуна внутрішнього згорання та автомобіля. Базуючись на цих моделях та використовуючи генетичний алгоритм, розроблено програму для пошуку оптимальної з точки зору витрат пального стратегії керування автомобілем.

Об'єкт дослідження: системи оптимізації на основі генетичних алгоритмів.

Предмет дослідження: системи оптимізації споживання палива на основі генетичних алгоритмів.

Мета роботи: розробити алгоритми та програмну реалізацію системи оптимізації споживання палива на основі генетичних алгоритмів.

Завдання роботи:

- аналіз роботи двигунів внутрішнього згоряння;
- аналіз методів оптимізації роботи двигунів внутрішнього згоряння;
- застосування генетичних алгоритмів для оптимізації;
- обгрунтування вибору програмних засобів;
- розробка алгоритму оптимізації споживання палива;
- тестування системи оптимізації споживання палива.

РОЗДІЛ 1
АНАЛІЗ СПОЖИВАННЯ ПАЛИВА ДВИГУНАМИ ВНУТРІШНЬОГО ЗГОРАННЯ

1.1. Аналіз роботи двигунів внутрішнього згорання

Двигун внутрішнього згорання – це найбільш поширений тип теплового двигуна. Найчастіше використовуються чотирьохтактні двигуни внутрішнього згорання. Їх робота розділяється на чотири такти. Спочатку відкривається вхідний клапан, поршень відходить, збільшуючи об’єм камери згорання та в неї заходить суміш палива і повітря. Після цього поршень рухається всередину камери, збільшуючи тиск і температуру суміші. Відбувається згорання суміші, за рахунок чого значно збільшується тиск і під дією сили тиску поршень виштовхується.

Трансмісія складається з зчеплення, коробки передач, карданного валу, приводного валу, кінцевої передачі та колеса. Зчеплення передає крутний момент двигуна на коробку передач. Коробка передач вважається автоматизованою ручною і складається з набору передач з різними коефіцієнтами перетворення. Карданний вал передає крутний момент від коробки передач до кінцевої передачі, яка, в свою чергу, передає крутний момент на колеса через карданний вал, змушуючи автомобіль рухатися.

Двигуна створює крутний момент, необхідний для руху автомобіля. Двигун передає корисний крутний момент T_e, на трансмісію, що є різницею між крутним моментом двигуна T_{ind}, і втратами на тертя в двигуні T_{fric}:

$$T_e = T_{ind} - T_{fric}(\omega_e) \tag{1.1}$$

Передбачається, що $T_{fric}(\omega_e)$ залежить приблизно лінійний від швидкості обертання валу двигуна ω:

$$T_{fric}(\omega_e) = a_1 \omega_e + a_2. \quad (1.2)$$

де a_1 і a_2 – константи.

Коли включена передача, двигун передає крутний момент на зчеплення T_c:

$$J_e \omega_e = T_e - T_c \quad (1.3)$$

де J_e – момент інерції двигуна.

Коли включена нейтральна передача, зчеплення розмикається, і крутний момент не передається на трансмісію. Маючи на увазі, що $T_c = 0$, і рівняння вище перетворюється:

$$J_e \omega_e = T_e. \quad (1.4)$$

Передбачається, що трансмісія жорстка і, отже, в ній немає коливань. Моменти інерції зчеплення, карданних валів разом з інерцією колеса, позначимо J_w. Отриманий коефіцієнт перетворення коробки передач і кінцевої передачі визначається як $i(g)$. Втрати енергії в коробці передач і в кінцевій передачі моделюються за допомогою ККД $\eta(g)$. $i(g)$ і $\eta(g)$ залежать від поточної включеної передачі g.

Коли включена передача, залежність між швидкістю обертання двигуна та кутовою швидкістю колеса:

$$\omega_e = i(g)\omega_w \quad (1.5)$$

Рівняння виконується лише тоді, коли включена передача, відмінна від нейтральної. Крутний момент, що передається на колесо T_w від зчеплення становить

$$T_w = i(g)\eta(g)T_c \tag{1.6}$$

Динаміка колеса виражається рівнянням:

$$J_w \omega_w = T_w - T_b - r_w F_w \tag{1.7}$$

де T_b — крутний момент від гальма;

r_w – ефективний радіус колеса;

F_w – результуюча сила тертя колеса.

Коли включена нейтральна передача, крутний момент T_c = 0, оскільки крутний момент не передається на трансмісію. Отже, крутний момент на колесі дорівнює T_w = 0. Динаміку колеса на нейтральній передачі можна виразити так:

$$J_w \omega_w = -T_b - r_w F_w \tag{1.8}$$

Зовнішніми силами на транспортний засіб є зовнішній опір повітря $F_a(v)$, опір коченню $F_r(\alpha)$ і гравітаційні сили $F_N(\alpha)$. Опір повітря $F_a(v)$ можна оцінити за допомогою такого виразу

$$F_a(v) = \frac{1}{2} c_w A_a \rho_a v^2 \tag{1.9}$$

де c_w - коефіцієнт опору повітря;

A_a - площу поперечного перерізу передньої частини транспортного засобу;

ρ_a - щільність повітря;

v - позначає швидкість транспортного засобу.

Силу опору коченню $F_r(\alpha)$ можна моделювати з різним ступенем точності. Вона залежить від нормальної сили автомобіля, коефіцієнту опору коченню, сила опору, спричинена тиском, температурою та швидкістю шини. Однак останні три фактори вважаються малими порівняно з нормальною силою і ними нехтуватимемо. Тоді $F_r(\alpha)$ буде мати вид

$$F_r(\alpha)=c_r m g_0 \cos(\alpha) \tag{1.10}$$

де c_r – коефіцієнт опору коченню,

m — маса транспортного засобу,

g_0 — прискорення сили тяжіння,

α — ухил дороги.

Сила тяжіння $F_g(\alpha)$ виражається через

$$F_g(\alpha)=m g_0 \cos(\alpha) \tag{1.11}$$

Рух транспортного засобу моделюється в загальному виді

$$m\frac{dv}{dt}=F_w-F_a(v^2)-F_r(\alpha)-F_N(\alpha) \tag{1.12}$$

Якщо ефективний радіус колеса r_w відомий, то співвідношення між швидкістю транспортного засобу та його кутовою швидкістю можна виразити як

$$v=r_w \omega_w \tag{1.13}$$

Зведення рівнянь разом дозволяє виразити поздовжню динаміку транспортного засобу за допомогою наступного рівняння

$$\frac{dv}{dt}=\frac{r_w}{J_w+mr_w^2-\eta(g)i(g)J_e}\left[\eta(g)i(g)T_e-T_b-r_w\left(F_a\left(v^2\right)+F_r(\alpha)+F_N(\alpha)\right)\right] \tag{1.14}$$

Інерція двигуна і колеса набагато менші за масу транспортного засобу, тому зробимо наближення $J_w \approx 0$ і $J_e \approx 0$, та спростимо вираз

$$\frac{dv}{dt}=\frac{1}{mr_w}\left[\eta(g)i(g)T_e-T_b-r_w\left(F_a\left(v^2\right)+F_r(\alpha)+F_N(\alpha)\right)\right] \tag{1.15}$$

Наведена вище динаміка залежить від часу, тобто *dv*/*dt* , однак більш бажано сформулювати це так, щоб воно залежало від поточної позиції *s*. Іншими словами, більш бажано моделювати динаміку за допомогою *dv/ds* з лівого боку. Це спрощує реалізацію моделі, оскільки нахил дороги залежить від пройденої відстані, а не від пройденого часу.

При оптимізації в функції дистанції як незалежної змінної краще моделювати динаміку транспортного засобу енергією, а не швидкістю.

Зміну кінетичної енергії транспортного засобу можна виразити так

$$\frac{m}{2}d\,v^2=F_{tot}ds \tag{1.16}$$

Перегрупування вищевказаних рівнянь дає наступний вираз для поздовжньої динаміки транспортного засобу

$$\frac{d\,v^2}{st}=2\frac{dv}{dt}=2\frac{1}{mr_w}\left(\eta(g)i(g)T_e-T_b-r_w\left(F_a(v^2)+F_r(\alpha)+F_N(\alpha)\right)\right)=\frac{2}{m}F_{tot} \quad (1.17)$$

Навіть якщо рівняння спрощує реалізацію моделі, динаміка автомобіля в рівнянні використовується щоразу, коли транспортний засіб перемикає передачу. Це пояснюється тим, що динаміка автомобіля під час перемикання передач все ще залежить від часу.

У двигуні масова витрата палива за час залежить від необхідного корисного крутного моменту двигуна T_e і поточної швидкості двигуна w_e.

Простий спосіб розрахунку витрати палива полягає в припущенні постійної ефективності згоряння в двигуні. Не враховується той факт, що двигун працює з різним ККД при різних оборотах двигуна і для різних необхідних крутних моментів двигуна.

Якщо відомі крутний момент і частота обертання двигуна, то потужність, що виробляється двигуном, є:

$$P=T_e\omega_e \quad (1.18)$$

Припустимо, що двигун має постійну ефективність згоряння $\eta_{дизель}$, а вміст енергії на масу дизельного палива дорівнює $E_{дизель}$. Згенерована потужність може потім бути перетворена в масу за час, використовуючи наступне співвідношення:

$$m_f=\frac{1}{\eta_{дизель}E_{дизель}}P=\frac{1}{\eta_{дизель}E_{дизель}}T_e\omega_e \quad (1.19)$$

Ця масова витрата за одиницю часу є швидкістю використання палива.

Карта палива містить виміряні дані про швидкість використання палива в різних робочих точках двигуна. Її можна виразити як функцію, що залежить від значень T_e та ω_e, наступним чином:

$$m_f = f_m(T_e, \omega_e) \tag{1.20}$$

Наступний графік є типовою паливною картою з виміряних даних (рис.1.1). Швидкість використання палива (масовий витрата за одиницю часу) виражається на осі *z* і залежить від T_e та ω_e.

Можна помітити, що присутні нелінійності і максимальний крутний момент двигуна T_e від двигуна змінюється в залежності від частоти обертання двигуна. Використання цього методу для розрахунку швидкості витрати враховує, що двигун не має однакової швидкості згоряння для всіх швидкостей двигуна ω_e і бажаного крутного моменту двигуна T_e, і що деякі робочі точки в двигуні кращі за інші.

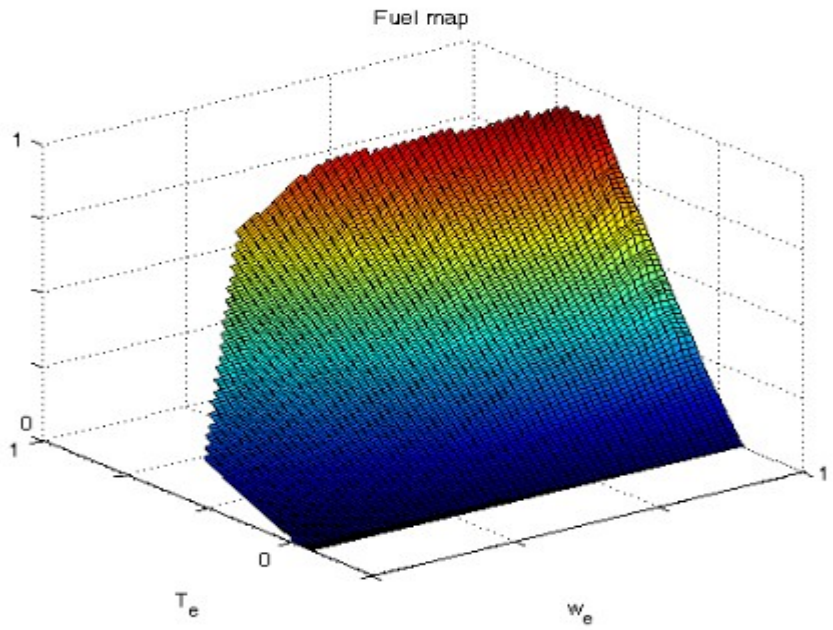


Рис.1.1 - Типова паливна карта.

Перемикання передач моделюється як постійний інтервал часу τ_{shift}, на якому включена нейтральна передача. Під час τ_{shift} двигун прискорюється, якщо попередньо сформовано перемикання на нижню передачу, і сповільнюється, якщо

попередньо виконується перемикання вгору. Нова передача включається в кінці інтервалу часу. Для простоти динаміка прискорення та уповільнення двигуна не моделюється, і передбачається, що бажана швидкість двигуна завжди буде досягнута в кінці процесу.

Припустимо, що перемикання передач з g_1 на g_2 вимагається контролером у момент часу t=0, перемикання передач моделюється як наступний процес

$$g(t)=\begin{cases} g_1, t<0 \\ 0, 0\le t\le \tau_{shift} \\ g_2, t\ge\tau_{shift} \end{cases} \tag{1.21}$$

Оскільки перемикання передач відбувається протягом інтервалу часу τ_{shift}, воно не залежить від пройденої відстані, тому динаміка автомобіля під час перемикання передач моделюється як функція часу, що дає

$$\frac{dv}{dt}=\frac{1}{mr_w}\left(-T_b-r_w\left(F_a(v^2)+F_r(\alpha)+F_N(\alpha)\right)\right) \tag{1.22}$$

Коли включена нейтральна передача, зчеплення розмикається, і крутний момент не передається на трансмісію. Це означає, що сила до колеса F_w=0 і поздовжня динаміка стає

$$\frac{dv^2}{ds}=2\frac{1}{mr_w}\left(-T_b-r_w\left(F_a+F_r+F_N\right)\right) \tag{1.23}$$

Автомобіль котиться легше, ніж котиться з включеною передачею, немає тертя двигуна. Загальний опір коченню транспортного засобу обумовлений лише зовнішніми силами.

Завдання коробки передач полягає в тому, щоб дозволити двигуну працювати в бажаному діапазоні кутових швидкостей, коли швидкість транспортного засобу змінюється. За допомогою набору зубчастих коліс і валів крутний момент може бути посилений або зменшений через коробку передач, в той час як кутова швидкість відповідно зменшується або посилюється. Кожна окрема передача має співвідношення між вхідним і вихідним крутним моментом або кутовою швидкістю, яке визначається передавальним числом вибраної передачі. Використовуючи передавальне відношення $i(g)$, де g — включена передача, вхідний крутний момент T_{in} і кутова швидкість ω_{in}, і вихідний крутний момент T_{out} і кутова швидкість, ω_{out}, можна отримати наступні рівняння для коробки передач

$$T_{¿}\, i(g) = T_{out}, \frac{w_{¿}}{i(g)} = \omega_{out} \tag{1.24}$$

З цих рівнянь можна вивести що:

$$P_{¿} = T_{¿}\, \omega_{¿} = T_{out} \omega_{out} = P_{out} \tag{1.25}$$

де P_{in} і P_{out} - це вхідна і вихідна потужність коробки передач, відповідно.

Ці рівняння справедливі для ідеальної коробки передач без втрат. Однак реалістична коробка передач матиме втрати через тертя та інші ефекти в коробці передач. Простим способом моделювання цих втрат є припущення постійної ефективності $\eta(g)$ для кожної передачі. Ефективність різна для кожної передачі через різні налаштування зубчатого колеса. Зазвичай є також шестерня, яка є лише валом через коробку передач, яка не змінює крутний момент або швидкість двигуна. Ця передача називається прямим приводом і має набагато кращу ефективність, ніж інші шестерні. Наведені вище рівняння тепер виглядають так:

$$P_{¿}\eta(g)=T_{¿}\omega_{¿}\eta(g)=T_{out}\omega_{out}=P_{out} \quad (1.26)$$

Хоча постійний ККД для кожної передачі фіксує деякі втрати в коробці передач, вимірювання показують, що втрати в коробці передач сильно залежать від крутного моменту та кутової швидкості. Наприклад, коли крутний момент в коробці передач дорівнює нулю, ефективність не може бути визначена, хоча відомо, що завжди є втрати, коли включена передача і транспортний засіб рухається. Інший приклад - коли крутний момент негативний через тертя в двигуні. У цьому випадку ефективність не може бути використана, оскільки робота, яку виконує тертя коробки передач, буде додана в напрямку руху автомобіля. Для визначення залежності від кутової швидкості та крутного моменту використовується карта ефективності коробки передач, де крутний момент і частота обертання двигуна переводять у ефективність коробки передач. Для подолання проблем, пов'язаних з використанням ККД, значення карти перераховують у терміни втрати крутного моменту. Потім значення для нульового крутного моменту двигуна екстраполюються та додаються до карти. Щоб мати можливість моделювати поведінку негативного крутного моменту двигуна, робиться припущення, що негативний крутний момент дає такі самі втрати, як і позитивний. Таким чином, абсолютне значення крутного моменту можна використовувати при інтерполяції значення з карти коробки передач. Інша проблема, яка виникає, полягає в тому, що крутний момент, необхідний для виконання заданої зміни швидкості, залежить від втрат в коробці передач. Тим часом втрати в коробці передач залежать від крутного моменту від двигуна. Оскільки втрати в коробці передач отримані з карти, а не з аналітичного виразу, немає рівняння, яке можна було б вирішити для необхідного крутного моменту двигуна. Для подолання цієї проблеми виконується наступний алгоритм оцінки втрат редуктора:

- апроксимація T_a необхідного крутного моменту двигуна розраховується за допомогою постійного значення ефективності передачі;
- наближений крутний момент T_a використовується разом із відомою кутовою швидкістю ω для інтерполяції значення втраченого крутного моменту T_{loss} в коробці передач;
- значення T_{ideal} необхідного крутного моменту двигуна розраховується за умови відсутності втрат у коробці передач;
- необхідний крутний момент двигуна T_n розраховується за формулою: $T_n=T_{ideal} + T_{loss}$.

Цей метод можна виконати протягом кількох ітерацій, щоб отримати все більш точне значення необхідного крутного моменту, перейшовши від останнього кроку до першого, використовуючи T_n замість T_a. Щоб не сповільнювати алгоритм бажано щоб кількість інтерполяції була мінімальною.

На графіку (рис.1.2) зображені втрати коробки передач на 11-й передачі при 1500 об/хв як функція необхідного крутного моменту двигуна, розрахованого з постійним ККД (синій) і з карти коробки передач з виконанням однієї (червоної) та десяти (зеленої) ітерацій оцінки.

На рис.1.2 показано втрату крутного моменту, розраховану шляхом виконання однієї та десяти ітерацій для фіксованого значення кутової швидкості 1500 об/хв. Втрачений крутний момент, отриманий при використанні постійного значення ККД, також показаний на рисунку.

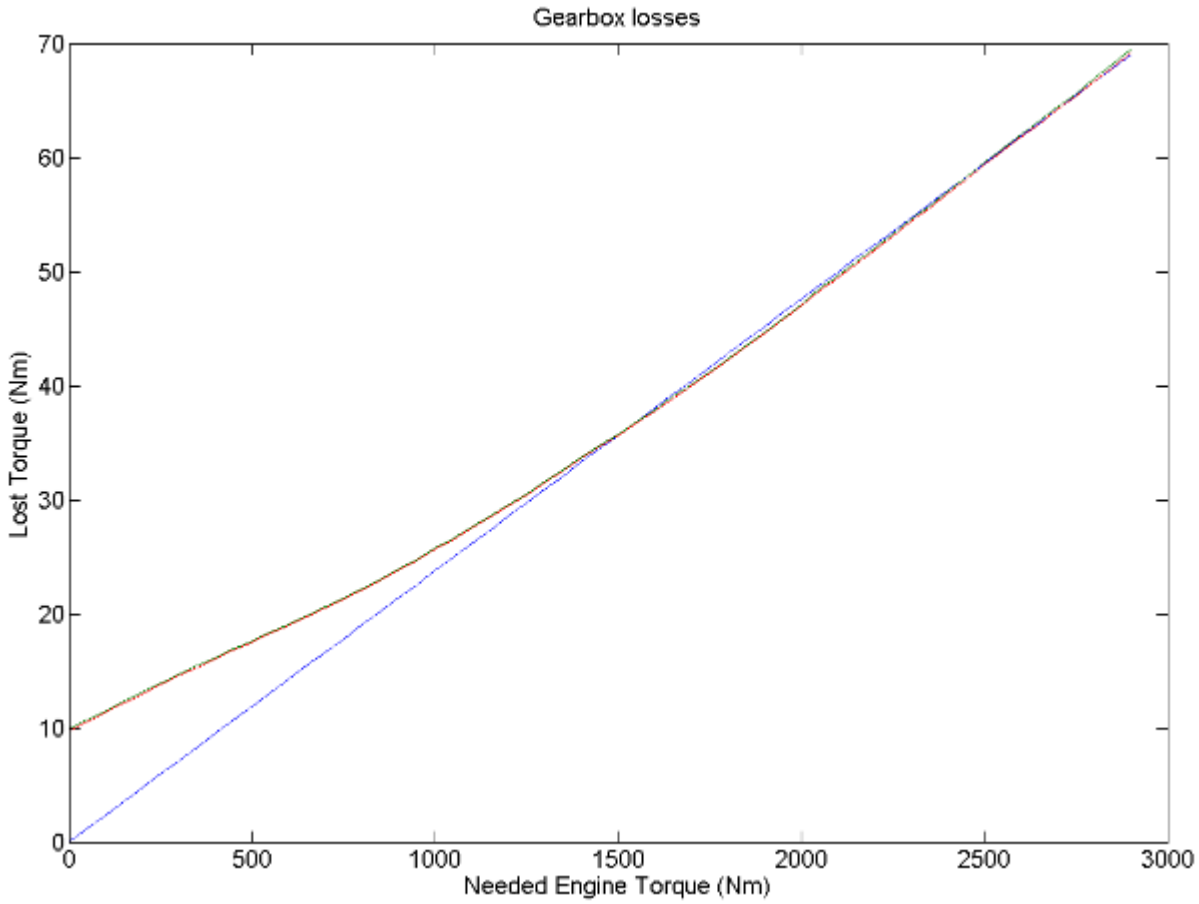


Рис.1.2 - Втрати крутного моменту в трансмісії

Як видно, різниця між першою та десятою ітераціями ледь помітна. Крім того, втрати, розраховані на основі постійної ефективності, занижують втрачений крутний момент приблизно до 1500 Нм. При більш високих крутних моментах лінії дуже близькі, і постійна ефективність дуже добре оцінює втрати в коробці передач.

Для візуалізації різниці між однією і десятою ітераціями на малюнку показано відносну різницю між двома випадками для всіх кутових швидкостей і необхідного крутного моменту двигуна. Відносна різниця найбільша при дуже низьких крутних моментах, де вона становить приблизно 1-1,5%. При більших крутних моментах відносна різниця становить близько 0,5%. Оскільки різниця між

однією і десятьма ітераціями така мала, то в алгоритмі розв’язування виконується лише одна ітерація.

1.2. Аналіз методів оптимізації роботи двигунів внутрішнього згорання

Сучасні наземні транспортні засоби покладаються майже виключно на паливо на основі нафти і виділяють велику кількість парникових газів. Занепокоєння щодо енергетичної безпеки та зміни клімату призводять до суворих правил викидів вуглецю в багатьох частинах світу. Тому технології економії палива, такі як гібридні двигуни та легкі конструкції, вивчалися протягом останнього десятиліття. Іншим фактором, який впливає на економію палива, є те, як керують транспортним засобом. Було продемонстровано, що стиль водіння може змінити паливну ефективність до 10 відсотків у звичайному русі. Виявлено, що деякі водії змогли заощадити паливо, не збільшуючи час поїздки, маніпулюючи лише швидкістю автомобіля.

Дві спроби вивчення цих навичок водіння були зроблені в проектах екологічного водіння та за допомогою економно-орієнтованих методів поздовжнього контролю. Цілі першою полягали в тому, щоб навчити навичкам економії палива шляхом навчання водіїв, зворотного зв’язку з показником палива та моніторингу. Інший реалізує навички економії палива в системах автоматичного керування, щоб досягти більш ефективних операцій, ніж це може зробити пересічний водій. В обох проектах важливою вимогою є чітке розуміння стратегій водіння, що економлять паливо. Один із способів досягти цього – зібрати велику кількість даних про водіння та узагальнити риси економних водіїв. Багато проектів з екологічного водіння використовували цей метод і генерували різні поради щодо екологічного водіння, такі як плавне прискорення, підтримка темпу водіння та усунення надмірного холостого ходу. Крім того, відомо, що так

звана стратегія пульсу і ковзання (PnG) досягає більшої економії палива в умовах невеликого руху, і вона була прийнята багатьма командами в змаганнях з паливної ефективності, таких як серія змагань SAE Supermileage. Основна ідея PnG полягає в тому, щоб прискорюватися, запускаючи двигун на високій потужності, зберігати кінетичну енергію в інерції транспортного засобу, а потім рухатися до нижчої швидкості, коли двигун вимкнений. Слід зазначити, що описані вище «підказки щодо екологічного водіння» та стратегія PnG є якісними. Ці правила можна використовувати для навчання водіїв, але вони не готові до застосування в системах керування транспортними засобами. Крім того, деякі з набутих уроків, здається, суперечать один одному. Наприклад, стратегія PnG перемикається між прискоренням і вибігом, тоді як багато порад щодо економії палива пропонують стабільну та плавну роботу. Тому незрозуміло, як і коли ці стратегії економії палива можуть бути реалізовані в системах автоматичного керування. Інший спосіб отримати реалізовані алгоритми економії палива — прийняти підхід на основі моделі стратегії водіння безпосередньо з динаміки автомобіля та характеристик споживання палива. Для умов вільного потоку Чанг і Морлок дослідили різні профілі швидкості для даного шляху і дійшли висновку, що постійна швидкість є оптимальною, якщо швидкість витрати палива пропорційна потужності руху. Хукер оптимізував витрату палива в процесі прискорення. Він виявив, що більші транспортні засоби повинні мати вищу оптимальну швидкість, а швидкість спуску повинна бути вищою, ніж швидкість підйому. Метою цієї роботи є розробка підходу, заснованого на моделі, та визначення оптимальних стратегій водіння легкового автомобіля за сценарієм, що рухається за автомобілем (коли присутній провідний транспортний засіб), що є більш практичним, ніж умови вільного потоку, які зазвичай вивчаються. Основна відмінність між сценарієм руху автомобіля та ситуацією вільного потоку полягає в необхідності накладення обмежень на дальність транспортного засобу з міркувань безпеки та

потоку. На основі визначених стратегій можна виділити правила економії палива, які реалізуються в системах керування, таких як адаптивний круїз-контроль.

Виконуючи алгоритм динамічного програмування з різними значеннями параметра штрафу β, можна отримати різні стратегії для однієї і тієї ж дороги. При зниженні β час буде менш важливим, і буде знайдена повільніша стратегія, яка використовує менше палива. Якщо замість цього збільшити β, час буде важливішим і швидшим, але результатом буде більше стратегій споживання палива. На рис. 1.3-1.7 показано три стратегії з різними β відрізку дороги довжиною 5000 метрів з ухилом 5% на 5% приблизно на 1500 метрів. Усі стратегії досягають максимально дозволеної швидкості 89 км/год безпосередньо перед тим, як пагорб спричинить уповільнення автомобіля. Це показує, що збільшення швидкості перед підйомом буде оптимальним як для повільних, але менших стратегій споживання палива, так і для більш швидких, але більших стратегій споживання палива. Усі стратегії також використовують крутний момент, набагато нижчий, ніж максимальний, щоб розганятися до 89 км/год. Той факт, що опір повітря має квадратичну залежність від швидкості, говорить про те, що краще почати прискорення пізніше і використовувати максимальний крутний момент. Однак максимальний крутний момент не використовується в оптимальному рішенні, що говорить про те, що двигун може використовувати паливо більш ефективно при менших крутних моментах. На початку пагорба стратегії починають відрізнятися. Червона стратегія зміщується вниз безпосередньо перед пагорбом і, таким чином, втрачає менше швидкості, коли починається пагорб. Зелена стратегія, яка визначає час менше, ніж червона, але більше, ніж синя, перемикається незабаром після початку пагорба, а синя підтримує дванадцяту передачу приблизно до 700 метрів у пагорб, а потім виконує 2 послідовні перемикання передач. Синя стратегія економить паливо, але втрачає час, залишаючись довше на 12-й передачі, яка має вищу ефективність, ніж інші передачі.

Коли пагорб вирівнюється, знову видно, що через підвищення ефективності двигуна для прискорення вибирається крутний момент, набагато нижчий за максимальний. Для подальшої оцінки результатів оптимізації досліджується відносна різниця в часі та витраті палива між усіма стратегіями. Це робиться шляхом вибору однієї стратегії та порівняння отриманої витрати палива та часу з іншими стратегіями. Крім того, виконується традиційне моделювання CC та додаються його результати для порівняння.

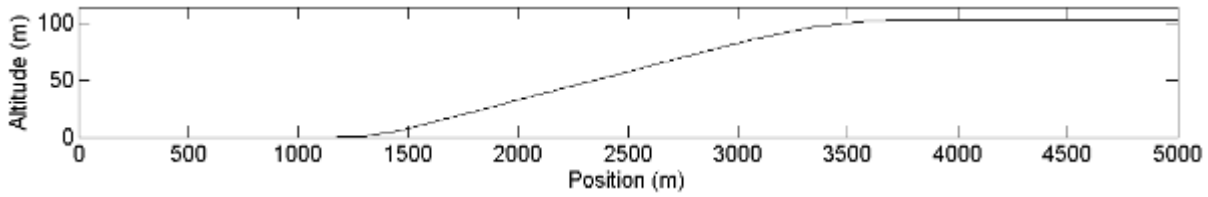


Рис. 1.3 - Рельєф дороги

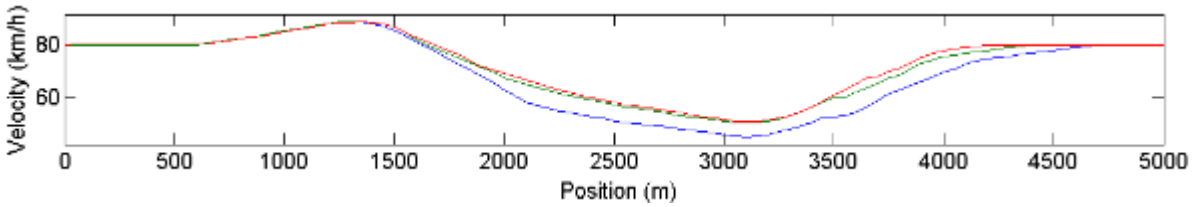


Рис. 1.4 - Швидкість автомобіля на кожній ділянці дороги

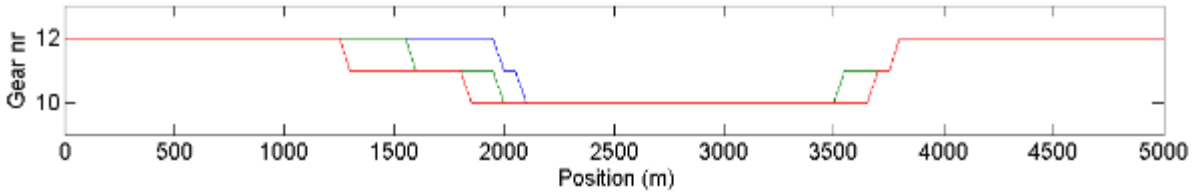


Рис. 1.5 - Передача на кожній ділянці дороги

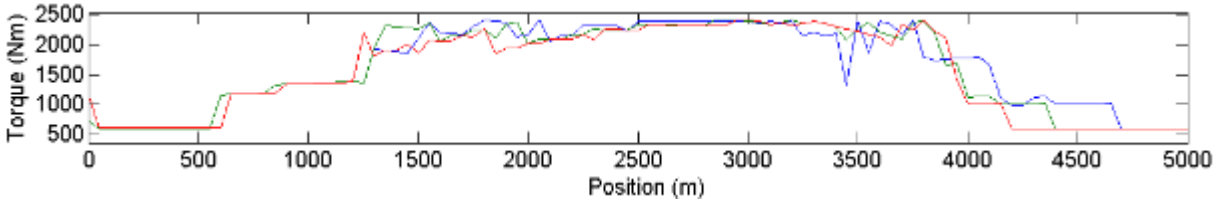


Рис. 1.6 - Тяга двигуна на кожній ділянці дороги

Результати для стратегій на рисунку та моделювання круїз-контролю порівняно з зеленою стратегією наведені на графіку. Традиційна симуляція CC споживає більше палива, ніж усі оптимізовані стратегії, але працює значно повільніше. Це показує, що потенційна економія оптимізованої стратегії є значною і що можна заощадити і час, і паливо одночасно. Загалом здається, що для такого типу доріг потенційна економія часу є більшою, ніж економія палива. Це можна побачити більш чітко, запустивши більше оптимізацій з меншою різницею між кожним значенням β. Результати на малюнку показують, що різні стратегії впливатимуть на час і паливо, необхідні для підйому на пагорби, де потрібне переміщення.

Різні стратегії призведуть до різного загального часу та обсягу палива, використаного також під час спуску з схилу. Коли дорога опускається вниз, з'являється можливість використовувати нейтральну передачу для руху з відключеним двигуном від коліс. Завдяки цьому двигун може працювати на холостому ходу, щоб зменшити втрати на тертя та насос, а також можна значно заощадити паливо.

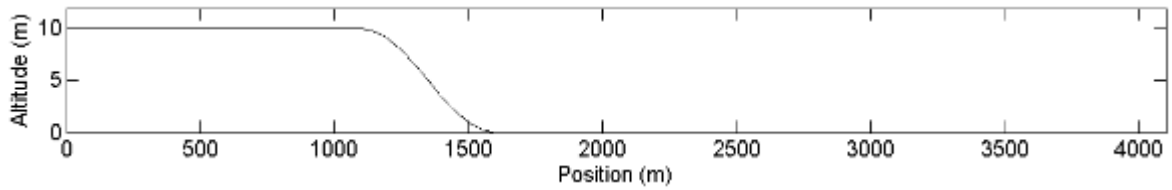


Рис. 1.7 - Рельєф дороги

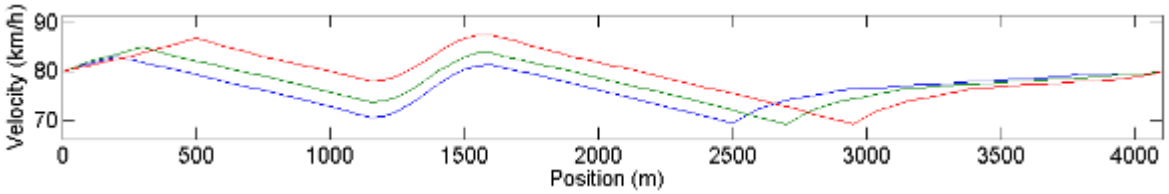


Рис. 1.8 - Швидкість автомобіля на кожній ділянці дороги

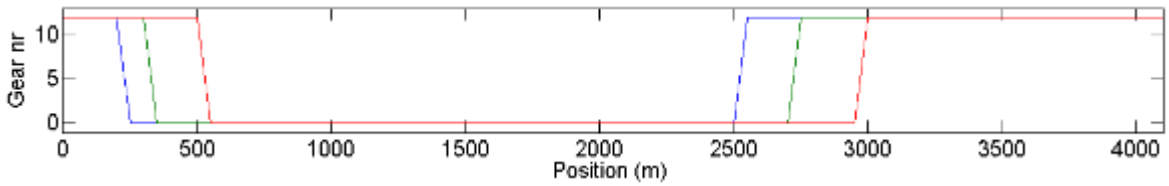


Рис. 1.9 - Передача на кожній ділянці дороги

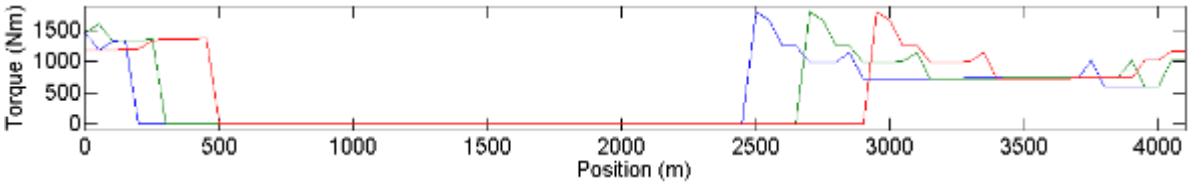


Рис. 1.10 - Тяга двигуна на кожній ділянці дороги

На рис.1.7-1.10 показано три стратегії з різними β для відрізку дороги довжиною 4100 метрів зі схилом 2% приблизно на 500 метрів. Усі стратегії розганяються до швидкості, вищої за встановлену швидкість, а потім вмикають нейтральну передачу. Це дозволяє транспортному засобу рухатися на дуже велику відстань і говорить про те, що катання на нейтральній передачі є дуже ефективним способом економії палива. Після включення нейтральної передачі стратегії скочуються на швидкості до пагорба. Залишаючись на нейтральній передачі, він розганяється силою тяжіння до максимальної швидкості на спуску. Нарешті, 12-та передача знову включається після зниження швидкості приблизно до 70 км/год. Червоне рішення є найшвидшим і оскільки воно має найвищу швидкість при включенні нейтральної передачі, воно має найдовшу дистанцію прокатки 2450 метрів. Синє рішення є найповільнішим і котиться на 2300 метрів. На графіку відносні витрати на оптимізовані стратегії та традиційне моделювання CC показані порівняно з зеленим рішенням. На відміну від випадку в гору, де можна заощадити набагато більше часу, ніж паливо, рішення у випадку спуску означають, що можна трохи більше заощадити з точки зору палива. споживання, ніж час. Червона стратегія на 1,4% швидше і споживає на 1,6% менше палива, тоді як синя на 1,8% повільніше, але споживає на 2,5% менше палива в порівнянні з

зеленою стратегією. Оскільки традиційний CC не знижує свою швидкість до або після пагорба, це швидше, ніж оптимізовані стратегії. Його нездатність передбачити майбутній ухил дороги та використовувати нейтральну передачу, однак, призводить до використання палива на 14% більше.

Проблема зі стратегіями, показана на малюнку, полягає в тому, що вони не показують інтуїтивно зрозумілий спосіб водіння. Причину збільшення швидкості перед спуском легко зрозуміти, але водіям вантажівок може бути важко прийняти, коли автомобіль спочатку збільшує, а потім зменшує швидкість перед спуском. Щоб отримати більш інтуїтивно зрозумілі рішення, алгоритм динамічного програмування обмежено, щоб він не міг включити нейтральну передачу зі швидкістю вище встановленої. Результати запуску алгоритму для тієї самої дороги, як на малюнку, показані нижче на малюнку. Оскільки всі стратегії включають нейтральну передачу із заданої швидкості замість збільшеної швидкості, відстань, на якій стратегії використовує нейтральну передачу, тепер значно нижча. Крім того, стратегії поводяться так само, як на малюнку. Щоб порівняти витрати на стратегії, що дозволяють збільшити швидкість перед підйомом, зі стратегіями, які не дозволяють збільшити швидкість, кілька оптимізацій, як на малюнку було зроблено з різними β. Усі результати порівнюються із зеленою стратегією на малюнку.

До цього моменту алгоритм динамічного програмування використовувався для типів доріг з ухилом або вгору, або вниз. Щоб розширити кількість типів доріг, тепер буде досліджено поєднання перших двох. Якщо відстань між підйомом і спуском досить велика, автомобіль зможе відновити задану швидкість після підйому до того, як буде досягнуто спуску. Тоді рішення буде таким же, як якщо б обидва пагорби запускалися окремо. Але якщо за підйомом одразу слід спуск, стратегію доведеться відкоригувати, оскільки змінилися кінцеві умови для підйому та умови початку спуску.

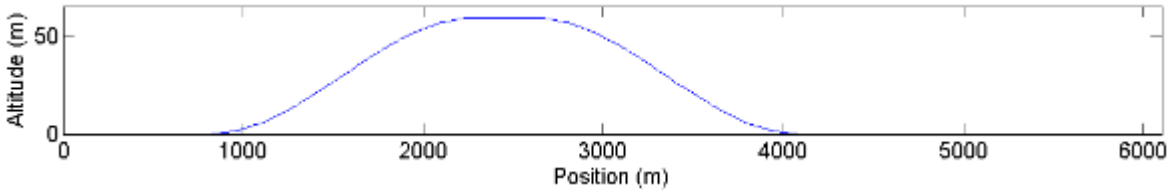


Рис. 1.11 - Рельєф дороги

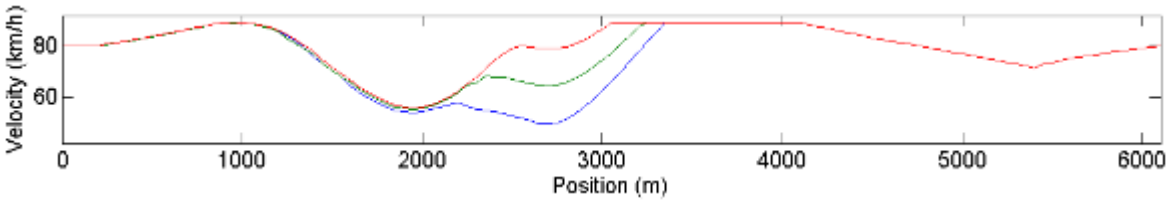


Рис. 1.12 - Швидкість автомобіля на кожній ділянці дороги

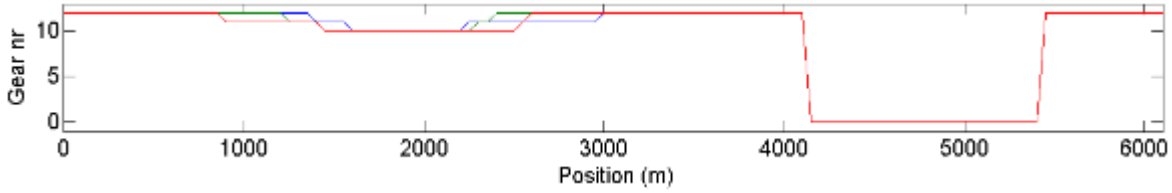


Рис.1.13 - Передача на кожній ділянці дороги

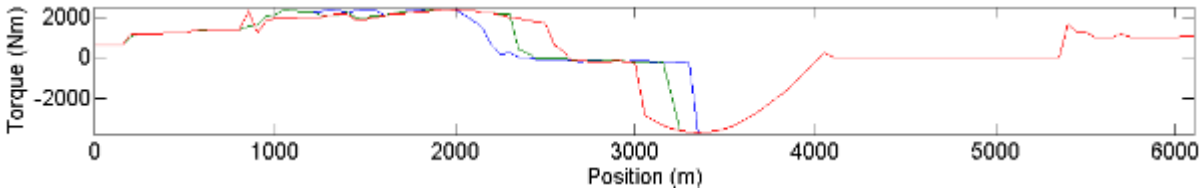


Рис.1.14 - Тяга двигуна на кожній ділянці дороги

На малюнку пагорб, який спочатку схиляється вгору приблизно на 4% протягом 1500 метрів, а потім спускається на таку ж відстань, був пропущений за допомогою алгоритму динамічного програмування для 3 різних значень штрафного параметра β. Подібно до результатів підйому раніше, стратегії здійснюють збільшення швидкості, яка досягає піку безпосередньо перед початком пагорба. Червоне рішення виконує перемикання передач безпосередньо перед початком пагорба, а дві інші залишаються на 12-й передачі майже до середини пагорба. На вершині пагорба стратегії перемикаються з максимального на мінімальний крутний момент у різних точках, що призводить до серйозних

відмінностей у профілях швидкості на цій частині дороги. Синій розчин має найнижчу швидкість перед спуском, таким чином, він може чекати найдовше, перш ніж почати гальмувати. Червоний розчин має набагато більшу швидкість, і йому доводиться гальмувати раніше, і витрачається більше енергії. Після того, як всі стратегії почали гальмувати, усі вони перебувають у точному стані. Оскільки параметр штрафу β однаковий для всіх стратегій після пагорба, стратегії для решти дороги ідентичні. Відразу після підйому включається нейтральна передача, і автомобіль котиться з максимальної швидкості вниз до швидкості, нижчої за встановлену швидкість, перш ніж знову включити 12-ту передачу.

РОЗДІЛ 2
РОЗРОБКА АЛГОРИТМУ ОПТИМІЗАЦІЇ СПОЖИВАННЯ ПАЛИВА

2.1. Застосування генетичних алгоритмів для оптимізації

Генетичні алгоритми це алгоритми пошуку, засновані на принципах природного відбору та природної генетики. Ці алгоритми зашифровують потенційний розв’язок задачі в систему хромосом. Генетичні алгоритми імітують процес еволюції. Вони оперують багатьма потенційними розв’язками, так званою популяцією, застосовуючи до них принцип виживання найпридатнішого щоб знайти кращі наближення до розв’язку. Генетичні алгоритми часто розглядають як оптимізатор функції, проте використовують їх для широкого кола задач. Робота алгоритму починається з популяції, зазвичай, випадковим чином генерованих розв’язків. Принцип виживання найпридатніших використовується для того, щоб обрати найкращі наближення до рішення, на основі яких створюється наступне покоління популяції.

В генетичних алгоритмах кожний агент представляє потенційний розв’язок даної задачі. Продуктивність кожного агенту вимірюється щоб надати йому міру пристосованості. Деякі агенти проходять через генетичні трансформації. Ці трансформації бувають двох типів.

Схрещення, котре створює нових агентів, об’єднуючи частини попередніх мутацій. Створення нового агенту, методом модифікації вже існуючого.

Загалом генетичні алгоритми складаються з наступних етапів:

- ініціалізація - під час цього етапу випадковим чином створюється перше покоління популяції;
- оцінка - під час оцінки знаходиться міра пристосованості кожного агента;

- відбір - відбираються агенти з найбільшою пристосованістю, на основі яких буде створене наступне покоління;
- відновлення - на основі відібраних агентів за допомогою схрещення та мутацій створюється наступне покоління популяції;
- перевірка на припинення - перевірити, чи алгоритм має закінчити свою роботу. При закінченні повернути найкраще рішення, інакше повернутись на етап оцінки.

На етапі ініціалізації створюємо популяцію, генеруючи q випадкових перестановок. На етапі оцінки використовується невід'ємна функція пристосованості F(S). На етапі відтворення створюється парний пул розміром q/2. Щоб застосувати операцію схрещення на етапі регенерації, розчини в пулі сполучень випадковим чином розподіляються на пари. З імовірністю p_{cross} кожна пара проходить схрещення; в іншому випадку пара не змінюється. Під час операції перехресного переходу два рішення, які ми називаємо батьками, об'єднуються для отримання двох нащадків, кожне з яких містить деякі характеристики кожного з батьків. Сподіваємося, що один із нащадків успадкує бажані риси кожного з батьків, щоб створити якісне рішення. Операція мутації застосовується до рішень перед тим, як помістити їх у нову популяцію, кожен елемент кожного рядка (кожне завдання в перестановці) вибирається з імовірністю p_{mut}. Наприклад, якщо завдання рядка вибрано для мутації, то воно замінюється іншим випадково вибраним завданням у тому самому рядку (що дає сусіда в околиці підкачки). В якості тесту завершення встановлюється ліміт часу, і алгоритм припиняє роботу, коли цей ліміт перевищено. Таким чином, основними кроками є створення сукупності рішень, пошук цільової функції та функції пристосованості та застосування генетичних операторів.

Для основних операцій ГА: одне покоління розбивається на фазу відбору та фазу рекомбінації. Під час вибору рядки призначаються в сусідні слоти. Важливою характеристикою генетичного алгоритму є кодування змінних, що

описують проблему. Найпоширенішим методом кодування є перетворення змінних у двійковий рядок або вектор; ГА найкраще працюють, коли вектори рішення є двійковими. Якщо в задачі є більше однієї змінної, кодування з багатьма змінними створюється шляхом конкатенації такої кількості кодування окремих змінних, скільки змінних у задачі. Генетичний алгоритм обробляє декілька рішень одночасно. У кожному поколінні створюється новий набір апроксимацій шляхом відбору індивідуумів відповідно до їхнього рівня пристосованості до проблемної області та розведення їх разом за допомогою операторів, запозичених з природної генетики. Цей процес призводить до еволюції популяцій особин, які краще пристосовані до свого оточення, ніж особини, з яких вони були створені, так само, як і при природному адаптації. Генетичний алгоритм відрізняється від інших методів пошуку тим, що він шукає серед сукупності точок і працює з кодуванням набору параметрів, а не з самими значеннями параметрів. Схема переходу генетичного алгоритму є ймовірнісною, тоді як традиційні методи використовують градієнтну інформацію. Через ці особливості генетичного алгоритму вони використовуються як алгоритм оптимізації загального призначення. Вони також забезпечують засоби пошуку нерегулярного простору і, отже, застосовуються до різноманітних програм оптимізації функцій, оцінки параметрів і машинного навчання. Виходячи з припущення, що початкова популяція генерується випадковою вибіркою із заміною (що в даному контексті є консервативним припущенням), можна знайти ймовірність того, що принаймні один алель присутній у кожному локусі. Для двійкових рядків це легко побачити, з чого ми можемо обчислити, що, наприклад, популяції розміру 17 достатньо, щоб гарантувати, що необхідна ймовірність перевищує 99,9% для рядків довжиною в 50 символів.

Генетичні алгоритми працюють на двох типах просторів альтернативно: простір кодування і простір розв’язків, або іншими словами, простір генотипів і простір фенотипу. Генетичні оператори (схрещення і мутація) працюють на

просторі генотипів, а еволюція та відбір – на просторі фенотипу. Відбір є зв’язком між хромосомами та продуктивністю декодованих розчинів. Відображення з простору генотипу в простір фенотипу має значний вплив на продуктивність генетичних алгоритмів. Генетичні алгоритми забезпечують спрямований випадковий пошук у складних ландшафтах. Існують дві важливі проблеми щодо стратегій пошуку: дослідження (дослідження нових і невідомих областей у пошуковому просторі) та використання (використання знань про рішення, раніше знайдені в просторі пошуку, щоб допомогти знайти кращі рішення). Це можна зробити, змусивши генетичних операторів виконувати по суті сліпий пошук; з надією, що оператори відбору спрямовують генетичний пошук до бажаної області простору рішень. Один із загальних принципів розробки реалізації генетичних алгоритмів для конкретної задачі реального слова полягає в тому, щоб створити гарний баланс між дослідженням і використанням простору пошуку. Для цього необхідно ретельно вивчити всі оператори та параметри генетичних алгоритмів. Для підвищення продуктивності в алгоритм слід включити евристики додавання. На відміну від простих методів пошуку по сусідству, які припиняються, коли досягається локальний оптимум, ГА є стохастичними методами пошуку, які в принципі можуть працювати вічно. На практиці необхідний критерій припинення; Загальні підходи полягають у тому, щоб встановити обмеження на кількість оцінок фітнесу або час комп’ютерного годинника, або відстежити різноманітність популяції та зупинитися, коли це падає нижче встановленого порогу. Значення різноманітності в останньому випадку не завжди є очевидним, і воно може стосуватися або генотипу, або фенотипу, або навіть, можливо, придатності, але найпоширенішим способом його вимірювання є статистика генотипу. Наприклад, ми могли б вирішити припинити запуск, якщо в кожному локусі частка одного конкретного алеля піднялася вище 90%. Після кількох поколінь генетичний алгоритм зближується до найкращого індивіда, який, сподіваємося, представляє оптимальне близьке до оптимального рішення проблеми.

Як і для будь-якого методу пошуку та навчання, спосіб кодування потенційних рішень є центральним, якщо не центральним, фактором успіху генетичного алгоритму. Кодування – це процес, що виконується за допомогою бітів, масивів, дерев, чисел або списку для представлення окремих генів. Більшість програм ГА використовують бітові рядки фіксованої довжини фіксованого порядку для кодування варіантів рішень. Як закодувати рішення проблеми в хромосомах є ключовим питанням при використанні генетичних алгоритмів. Однією з визначних проблем, пов’язаних із кодуванням, є те, що деякі люди відповідають нездійсненним або незаконним рішенням даної проблеми. Це може стати дуже серйозним для проблем оптимізації з обмеженими можливостями та проблем комбінаторної оптимізації. Необхідно розрізняти два поняття: нездійсненність і незаконність. Нездійсненність відноситься до явища, що рішення, розшифроване з хромосоми, лежить за межами можливої області даної проблеми. Штрафні методи можна використовувати для обробки нездійсненних хромосом. Один із цих методів полягає в тому, щоб примусово наближати генетичні алгоритми до оптимальної форми по обидва боки можливих і нездійсненних областей. Незаконність відноситься до явища, що хромосома не є рішенням даної проблеми. Методи відновлення зазвичай використовуються для перетворення нелегальної хромосоми в легальну.

Функціонування пропорційного відтворення відповідно до парадигми генетичного програмування є основним двигуном дарвінівського відтворення та виживання найбільш пристосованих. Кожній особини в популяції присвоюється значення придатності в результаті її взаємодії з навколишнім середовищем. Функція пристосованості — це рівняння, яке є функцією включення властивостей (генів) у кожен рядок (хромосоми). Загальний вигляд цієї функції залежить від досліджуваної проблеми і зазначає кінцеву мету задачі. Придатність є рушійною силою дарвінівського природного відбору і, так само, генетичних алгоритмів. Зауважте, що батьки залишаються в популяції, поки виконується ця операція, і

тому потенційно можуть неодноразово брати участь у цій операції (та інших операціях) протягом поточного покоління. Тобто відбір батьків здійснюється з дозволеною заміною (тобто перевибором). Наприклад, у задачі оптимізації ця функція може бути цільовою функцією, а потім максимізація або мінімалізація цілі може бути кінцевою метою задачі. Розшифрувавши представлення хромосоми в домен змінної рішення, можна оцінити продуктивність або придатність окремих членів популяції. Це здійснюється за допомогою цільової функції, яка характеризує роботу індивіда в проблемній області. У світі природи це була б здатність людини виживати в нинішньому середовищі. Таким чином, цільова функція створює основу для відбору пар особин, які будуть спаровуватися разом під час розмноження. Під час фази відтворення кожній людині призначається значення придатності, отримане на основі його первинного показника продуктивності, заданого цільовою функцією. Це значення використовується при відборі для упередження щодо більш підготовлених осіб. Високопридатні особини відносно всієї популяції мають високу ймовірність бути відібраними для спарювання, тоді як менш придатні особини мають відповідно низьку ймовірність бути відібраними. Після того, як особинам буде присвоєно значення придатності, їх можна вибрати з популяції з ймовірністю відповідно до їх відносної придатності та рекомбінувати для отримання наступного покоління. Цільова функція використовується для вимірювання того, як люди працювали в проблемній області. У випадку проблеми мінімізації найбільш придатні особи матимуть найнижче числове значення відповідної цільової функції. Цей необроблений показник придатності зазвичай використовується лише як проміжний етап у визначенні відносної продуктивності осіб у GA. Інша функція, функція пристосованості, зазвичай використовується для перетворення значення цільової функції в міру відносної придатності таким чином: де f - цільова функція, g перетворює значення цільової функції в невід'ємне число, а F - результат відносна придатність. Таке відображення завжди необхідне, коли цільова функція має бути

мінімізована, оскільки нижчі значення цільової функції відповідають людям, які краще підходять. У багатьох випадках значення функції придатності відповідає кількості потомства, яке індивід може розраховувати на народження в наступному поколінні. Часто використовуваним перетворенням є пропорційне пристосування.

Функція втрат є допоміжною функцією, метою якої є включення втрат, що виникають внаслідок порушення обмежень, у цільову функцію. Зазвичай обмеження поділяються на дві групи: 1- жорсткі обмеження, 2 - м'які обмеження. Жорстке обмеження завершує деякі обмеження, які необхідно дотримуватись під час процесу. М’які обмеження – це інший тип обмеження, який може дотримуватись чи ні, але перше має призвести до певних втрат або витрат. Розробка функції втрат дуже важлива в багатьох задачах генетичного алгоритму і зменшує час збіжності. Функція пристосованості також може враховувати вторинні фактори (наприклад, ефективність S-виразу, економність S-виразу, відповідність початковим умовам диференціального рівняння тощо). Коли кількість екологічних випадків велика, іноді оперативніше обчислити функцію придатності, використовуючи лише вибірку можливих екологічних випадків (включаючи, можливо, вибірку, яка змінюється від покоління до покоління, щоб мінімізувати можливе упередження в результаті такої вибірки).

Відтворення — це оператор, який створює більше копій кращих рядків у новій популяції. Відтворення зазвичай є першим оператором, який застосовується до популяції. Розмноження відбирає хороші рядки в популяції і формує пул для спарювання. Таким чином, в процесі відтворення процес природного відбору змушує тих особин, які кодують успішні структури, частіше виробляти копії. Для підтримки покоління нової популяції необхідно відтворення особин у нинішній популяції. Для кращих особин вони повинні бути з найпридатніших особин попередньої популяції. GAs імітує природний принцип виживання найпридатнішого, щоб зробити процес пошуку. У генетичному алгоритмі придатність використовується для виділення репродуктивних ознак індивідам у

популяції і, таким чином, діє як деяка міра доброти, яку необхідно максимізувати. Це означає, що особи з вищим рівнем придатності матимуть більшу ймовірність бути обраними кандидатами для подальшого обстеження. На цьому етапі особини вибираються випадковим чином із популяції в порядку придатної особини. Це показує, що особи з поганим статусом фізичної підготовки вилучаються з наступної популяції. Тому цей етап GA допомагає вибрати хороших і більш пристосованих особин з популяції для рекомбінації, виробляючи нове та добре підігнане потомство для наступного покоління. Потім певні оператори GA використовуються для відібраних кандидатів для рекомбінації їхніх хромосом. У літературі GA існує ряд операторів відтворення, але основна ідея в усіх полягає в тому, що рядки вище середнього вибираються з поточної сукупності, а їх численні копії вставляють у пул парування ймовірно. Придатність у біологічному сенсі — це цінність якості, яка є мірою репродуктивної ефективності хромосом.

Виділення зазвичай є першим оператором, який застосовується до сукупності після кодування. Вибір — це оператор, який створює більше копій кращих рядків у новій сукупності. Відбір — це процес визначення кількості разів або випробувань, коли конкретна особина вибирається для відтворення, і, таким чином, кількість потомства, яке дасть індивідуум. Це компонент, який направляє алгоритм до рішення, віддаючи перевагу людям з високою фізичною підготовкою, а не з низькою. Це може бути детермінована операція, але в більшості реалізацій вона має випадкові компоненти. Мета відбору, звичайно, полягає в тому, щоб наголосити на більш придатних особинах у популяції в надії, що їх потомство, у свою чергу, матиме ще вищу придатність. У процесі відбору хромосома вибирається відповідно до вимірювання її цільової функції (біолог назвав її функцією придатності). Ця функція може показувати деякі конкретні вимірювання, такі як корисність, вигода або будь-які інші цілі, які слід максимізувати або мінімізувати. Корисна хромосома для копіювання визначається за тими ж функціями. Функція відбору визначає перспективний генетичний

матеріал і визначає, скільки його має бути в наступному поколінні. Генетичний матеріал, який складає генотипи вищої якості, як правило, має більшу ймовірність опинитися повторно використаним у тій чи іншій формі в наступному поколінні. У більшості систем GP відбір застосовується на рівні організму, а термін «парний пул» означає сховище, що містить особин, відібраних для розмноження. Хоча системи GP зазвичай вибирають генотипи, є й інші можливості. Зокрема, можливий вибір цілих наборів генотипів (наприклад, схем) або комбінацій наборів та окремих генотипів. Схема відбору генотипу визначає ймовірність того, що генотип буде відібрано для отримання потомства шляхом схрещування або мутації. З метою пошуку фенотипів, які все більш підходять, вищі ймовірності відбору призначаються генотипам з кращими фенотипами. Оператор відбору вибирає генотипи, виходячи із загального принципу, що чим більш пристосована особина, тим вищою має бути її ймовірність бути відібраною для відтворення. Було запропоновано, досліджено та порівняно багато методів відбору. Як і у випадку з кодуванням, ці описи не дають чітких вказівок щодо того, який метод слід використовувати для вирішення якої проблеми; це все ще відкрите питання для ГА.

Вибір може працювати як із заміною, так і без неї. При заміні рішення, яке додається до нової батьківської сукупності, зберігається в об’єднаній сукупності, і для нової батьківської сукупності можна вибрати хороше рішення кілька разів. Використовується термін із заміною, оскільки вихідний генотип доступний для подальшого відбору в міру відбору додаткового генетичного матеріалу. Система не зберігає пам'ять про попередній вибір. Без заміни вибране рішення поміщається в батьківську сукупність і видаляється з комбінованої сукупності, і кожне рішення можна вибрати лише один раз для нової батьківської сукупності. Відбір забезпечує рушійну силу в генетичних алгоритмах. З надто великою силою генетичний пошук закінчиться передчасно. Таким чином, відбір спрямовує

генетичний пошук до перспективних регіонів у просторі пошуку, що покращить продуктивність генетичних алгоритмів.

У більшості GA індивіди представлені рядками фіксованої довжини, а схрещення діє на парах індивідуумів (батьків), щоб створити нові рядки (нащадки) шляхом обміну сегментами з батьківських рядків. Традиційно кількість точок перетину (яка визначає, скільки сегментів буде обмінюватися) була зафіксована на дуже низькому постійному значенні 1 або 2. Підтвердженням цьому рішення стала рання робота як теоретичного, так і емпіричного характеру Голланда. Незважаючи на це, інші задачі GA були реалізовані за допомогою інших типів схрещення. Операція схрещування (рекомбінації) для парадигми генетичного програмування створює варіації в популяції, виробляючи потомство, яке поєднує ознаки від двох батьків. Схрещення — це просто заміна деяких генів одного з батьків відповідними генами іншого. При статевому розмноженні, як це виглядає в реальному світі, генетичний матеріал двох батьків змішується, коли гамети батьків зливаються. Зазвичай хромосоми випадковим чином розщеплюються та зливаються, внаслідок чого одні гени дитини походять від одного з батьків, а інші — від інших батьків. Цей механізм називається схрещенням. Одним з унікальних аспектів роботи, пов’язаної з генетичними алгоритмами, є важлива роль, яку схрещення відіграє у розробці та реалізації надійних еволюційних систем. Оператор схрещення використовується для рекомбінації двох рядків, щоб отримати кращий рядок. Під час роботи схрещення процес рекомбінації створює різних особин у наступних поколіннях шляхом поєднання матеріалу двох особин попереднього покоління. . Схрещення є дуже потужним інструментом для введення нового генетичного матеріалу та підтримки генетичного різноманіття, але з тим видатним властивістю, що хороші батьки також народжують дітей, які успішно працюють, або навіть кращих. Декілька досліджень прийшли до висновку, що схрещування є причиною того, що види, що розмножуються статевим шляхом, адаптувалися швидше, ніж ті, що розмножуються безстатево. В

основному, схрещення - це обмін генами між хромосомами двох батьків. У найпростішому випадку ми можемо реалізувати цей процес, перерізавши дві струни у випадково вибраному положенні та помінявши місцями два хвоста. При відтворенні хорошим рядкам у популяції імовірнісно призначається більша кількість копій, і формується пул спарювання. Важливо відзначити, що на етапі відтворення нових струн не утворюються. В операторі схрещення нові рядки створюються шляхом обміну інформацією між рядками суміжного пулу. Дві рядки, які беруть участь в операції схрещення, відомі як батьківські рядки, а отримані рядки відомі як дочірні рядки. З цієї конструкції інтуїтивно зрозуміло, що хороші підрядки з батьківських рядків можна об’єднати, щоб утворити кращий дочірній рядок, якщо вибрано відповідний сайт. З випадковим сайтом створені дочірні рядки можуть мати чи не мати комбінацію хороших підрядків із батьківських рядків, залежно від того, чи потрапить сайт перетину у відповідне місце. Але це не викликає серйозного занепокоєння, тому що якщо хороші струни створюються схрещенням, їх копій буде більше в наступному пулі сполучень, створеного схрещенням. З цієї дискусії зрозуміло, що ефект перехресного переходу може бути згубним або корисним. Таким чином, щоб зберегти деякі з хороших струн, які вже присутні в сполучуваному пулі, усі струни в пулі сполучень не використовуються в схрещенні. Оператор схрещення в основному відповідає за пошук нових рядків, хоча оператор мутації також використовується для цієї мети помірно. Швидкість перехресного переходу визначає ймовірність того, що відбудеться схрещення. Схрещення породить нових особин у популяції шляхом об’єднання частин існуючих особин. Багато дослідників вважають, що коефіцієнт перехресного переходу становить від 0,6 до 0,95. Основна мета схрещення полягає в обміні інформацією між рядками, щоб отримати рядок, який, можливо, є кращим за батьківський. Після процесу відбору основний оператор для створення нових хромосом, який є оператором кросинговеру, обмінює основну

частину кожної з двох обраних хромосом, щоб зберегти найкращі властивості цих двох рядків.

2.2. Обґрунтування вибору програмних засобів

Бібліотека буде реалізована засобами мови програмування Python.

Можна виділити наступні переваги Python:

Python — це мова програмування високого рівня. Завдяки простоті Python розробники можуть зосередитися на вирішенні проблеми.

Python є інтерпретованою мовою, що означає, що Python безпосередньо виконує код рядок за рядком. У разі будь-якої помилки він зупиняє подальше виконання та повідомляє про помилку, яка сталася. Python показує лише одну помилку, навіть якщо програма містить кілька помилок.

Python автоматично призначає тип даних під час виконання.

Python поширюється на ліцензію відкритого коду. Це робить його безкоштовним у використанні та розповсюдженні. Ви можете завантажити вихідний код, змінити його і навіть поширити свою версію Python. Це корисно для організацій, які хочуть змінити певну поведінку та використовувати свою версію для розробки.

При цьому необхідно зауважити, що Python не є ідеальною мовою програмування. Під час вибору також приймались до уваги і його недоліки.

Вище ми обговорювали, що Python є інтерпретованою мовою і динамічно типізованою мовою. Порядкове виконання коду часто призводить до повільного виконання.

Динамічна природа Python також відповідає за повільну швидкість Python, оскільки він повинен виконувати додаткову роботу під час виконання коду. Отже, Python не використовується для цілей, де швидкість є важливим аспектом проекту.

Щоб забезпечити простоту для розробника, Python має зробити невеликий компроміс. Мова програмування Python використовує великий обсяг пам'яті. Це може бути недоліком під час створення програм, коли ми віддаємо перевагу оптимізації пам’яті.

Python - це мова з динамічним типом, тому тип даних змінної може змінюватися в будь-який час. Змінна, що містить ціле число, може містити рядок у майбутньому, що може призвести до помилок під час виконання. Тому програмістам Python необхідно провести ретельне тестування програм.

В даній роботі не є основними питання швидкості роботи та оптимальності використання пам’яті, що дозволяє нам не звертати увагу на перші два з зазначених недоліків. Також програма має відносно не високу складність, тому третій з зазначених недоліків також не має великого значення.

Можна зробити висновок що незважаючи на свої недоліки, мова програмування Python добре підходить для поставленої задачі.

2.3. Розробка алгоритму оптимізації споживання палива

В першу чергу потрібно створити модель автомобіля, яка буде використовуватись для оцінки агентів. Для цього створимо клас `Car`. Його атрибути будуть зберігати стан автомобіля – швидкість, швидкість двигуна, положення педаль газу, передача, пробіг та об’єм палива, що залишилось. Для забезпечення керування автомобілем створені методи `throttle` та `changeGear`, які дозволяють змінити позицію педалі газу та передачу відповідно (існує можливість переключитись на одну передачу вниз, але вона не використовується). Для забезпечення зворотного зв’язку метод `getState` повертає стан автомобіля. На випадок, якщо автомобіль заглохне, створено метод

`die`, який обнуляє кількість палива в бензобаку, чим завершує оцінювання даного агента. Також метод reset обнулює стан автомобіля та поповнює паливо.

Далі потрібно створити методи, для оновлення стану автомобіля. Першим таким методом є `fuelConsumption`. Цей метод за формулою

$$a + b*x + c*n + d*n*x + e*n^2 + f*x^2,$$

де n –швидкість двигуна в обертах в хвилину;

x – позиція педалі газу;

a, b, c, d, e, f – коефіцієнти, знайдені експерементальним шляхом.

Наступний метод – `findForce`. В ньому проводяться обчислення сумарної сили, що діє на автомобіль. З миттєвої витрати палива за допомогою `fuelConsumption` знаходиться потужність, яка переводиться в крутний момент двигуна, крутний момент коліс і, в силу, що діє на автомобіль.

Наступний метод `accelerate` викликає метод findForce який розраховує силу, що діє на автомобіль, прискорення та оновлює швидкість. Метод move за допомогою оновленої швидкості обчислює пробіг.

Метод `update` послідовно викликає всі методи, для оновлення миттєвого стану автомобіля.

Після цього потрібно створити всі методи, що будуть реалізовувати операції генетичного алгоритму. Метод `generate_starting_population` генерує початкову популяцію. У кожного гена є 1% вірогідності стати командою переключення на передачу вгору та 99% - змінити положення педалі газу. Для другого типу також випадковим чином генерується позиція педалі.

Метод `select_mating_pool` обирає серед всієї популяцію агентів, які пройшли найбільшу дистанцію.

Метод `crossover` обирає пари агентів та створює їх «потомство».

Метод mutation змінює в кожному агенті 5% генів. У гена є 5% вірогідність змінити команду (переключити передачу або змінити положення педалі газу). Якщо команда гена – змінити положення педалі газу, то змінюється нове положення.

Наступний модуль `simulation`. Спочатку створюється об’єкт класу `Car`, який буде використовуватись для оцінки. Також з стандартного входу отримується режим роботи програми – відобразити агента з найбільшим пробігом, або провести навчання.

Для того, щоб оцінити агента використовується наступний алгоритм (рис.2.1) – створюється лічильник time з початковим значенням 0. Якщо time ділиться на 10 без залишку, то виконується команда, закодована в гені під номером `time/10`. Після цього виконується метод `Car update`, значення time збільшується на одиницю. Далі перевіряється стан автомобіля і, якщо кількість палива більше нуля, то алгоритм повертається к виконанню гена, а інакше цикл переривається і виконується метод `Car restart` (рис.2.2).

Рис.2.1 - Блок схема алгоритму оцінки агента

В режимі відображення найкращого агента зчитується геном з файлу bestAgent.txt та проходить його оцінка. Також в кожний момент фіксується стан автомобіля, що дозволяє після оцінки відобразити їх у вигляді графіку.

Рис.2.2 - Блок-схема генетичного алгоритму

В режимі симуляції створюється популяція. Перевіряється чи існує файл `currentPop.txt`. Якщо він існує – з нього зчитується популяція і продовжується її навчання, інакше популяція створюється методом

`generate_starting_population`. Далі проводиться оцінка кожного агента, при чому результат кожного зберігається. Також популяція записується в файл `currentPop.txt` для можливості відновити навчання у випадку його екстреного переривання. З популяції методом `select_mating_pool` обирається половина агентів з найкращим результатом, після чого на їх основі за допомогою методів `crossover` та `mutation` створюється наступне покоління, яким замінюється гірша половина популяції. Після чого цей цикл повторюється.

Після навчання зчитується геном з файлу `bestAgent.txt` та порівнюється з найкращим агентом цієї сесії. Якщо новий агент досяг кращого результату, то його геном записується в цей файл. Якщо файлу не існує, то він створюється і в нього записується геном кращого агента цієї сесії.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

3.1. Опис даних програми

Основна структура даних – клас `Car`. Він представляє собою модель автомобіля. Цей клас зберігає миттєвий стан автомобіля – швидкість, швидкість двигуна, положення педалі газу, передача, пробіг та кількість палива в бензобаку. Його функціональність включає в себе зміну позиції педалі газу і зміну передачі, обчислення витрати палива, загальної сили, що діє на автомобіль, прискорення, оновлення миттєвого стану та повернення його.

Також важливою структурою даних є numpy array population, в якому зберігається поточне покоління. Він є трьохвимірним масивом. Перший вимір відповідає номерам агентів. Геном кожного агента представляє собою послідовність команд. Кожна команда складається з двох частин – число 0 або 1, що відповідає командам змінити передачу або положення педалі газу і специфікації: в випадку зміни передачі також 0 або 1, що відповідає зміні передачі на один вгору або вниз, в випадку педалі газу – число в діапазоні від 0 до 100, наскільки педаль вижата, де 0 означає що на педаль немає жодного тиску, а 100 – педаль повністю вижата.

Також програма включає списки `speedPlot, rpsPlot, gearPlot, throttlePlot` та `rangePlot`, які зберігають відповідно швидкість, швидкість двигуна, передачу та положення педалі газу для кожного моменту часу та найбільшу дальність на кожному поколінні. Перші чотири використовуються щоб відстежити поведінку кращого агента в кожному поколінні або загалом, а останній для відстеження росту ефективності з поколіннями.

3.2. Опис програмних модулів

Програма складається з трьох модулів: `model.py`, `ga.py`, `simulation.py`.

Модуль `model.py` включає в себе клас car та його параметри.

Перша частина включає об’явлення та надання значення параметрам, які будуть використовуватись під час моделювання автомобіля. Серед них фізичні константи та специфікації автомобіля.

```
from math import pi
#Передавальні числа трансмісії
gearRatios = [3.091, 1.864, 1.321, 1.029, 0.794]
#Передаточне число кінцевої передачі
finalDrive = 4.067
#Радіус колеса
wheelRadius = 0.2413
#Окружність колеса
wheelCircumference = wheelRadius * 2 * pi
#Максимальний об'єм палива
maxFuel = 53
#Маса автомобіля
mass = 1710
#Коєфіцієнт гальмування двигуном
engineBrakingCoefficient = 0.5
#Сопротивление воздуха
aerodeynamicResistance = 0.33
#Площадь поверхности воздушного потока
airflowSurface = 3
#Густота повітря
```

```
airDensity = 1.25
#Сопротивление кочения
rollingResistance = 0.01
#Прискорення свободного падіння
g = 9.8
#Коєффіціент корисної дії двигуна
efficiency = 0.5
#Енергоємність палива
energyIntensity = 1700000
```

Коефіцієнти для обчислення витрат палива від положення педалі газу та швидкості двигуна

```
a = 0.76712 / 3600
b = -0.25558 / 3600
c = 1.004*10**-3 / 3600
d = 1.95*10**-5 / 3600
e = -1.1*10**-7 / 3600
f = 0.0175 / 3600
```

Після цього створюється клас автомобіль.

```
class Car():
```

В методі `init` створюються атрибути, які використовуються для зберігання миттєвого стану автомобіля та ініціалізується його початковий стан.

```
    def __init__(self, startRps = 0):
        self.fuel = maxFuel
        self.rps = startRps
        self.throttlePosition = 0
```

```
            self.gear = 0
            self.mileage = 0
                        self.speed = self.rps / finalDrive /
gearRatios[self.gear] * wheelCircumference
```

Наступні методи призначені для забезпечення керування.

```
    def throttle(self, throttlePosition):
        self.throttlePosition = throttlePosition

    def ChangeGear(self, a):
        if a == 1:
            if self.gear < 4:
                self.gear += 1
        else:
            if self.gear > 0:
                self.gear -= 1
```

Метод `getState` повертає стан автомобіля.

```
    def getState(self):
        state = {
            "speed":self.speed,
            "rps":self.rps,
            "gear":self.gear,
            "fuel":self.fuel,
            "mileage":self.mileage,
            "throttle":self.throttlePosition}
        return state
```

Методи призначені для моделювання автомобіля.

```
    def fuelConsumption(self, timeStep):
            n = self.rps * 60
            x = self.throttlePosition
            dfuel = a+b*x+c*n+d*n*x+e*n*n+f*x*x
            if self.fuel < dfuel:
                dfuel = self.fuel
            return dfuel

        def update(self, timeStep):
            self.fuel -= self.fuelConsumption(timeStep)
            self.move(timeStep)
            self.accelerate(timeStep)

        def findForce(self, timeStep):
            if self.rps < 1:
                self.die()
                return
            power = energyIntensity * self.fuelConsumption(timeStep)
* efficiency / timeStep
            engineTorque = power * 0.16 / self.rps
            engineTorque -= engineBrakingCoefficient * self.rps
               wheelForce = engineTorque * gearRatios[self.gear] *
finalDrive / 2 / wheelRadius

                 airDrag = airDensity * aerodeynamicResistance *
airflowSurface * self.speed*self.speed / 2

            rollingDrag = rollingResistance * mass * g * 4

            force = wheelForce - airDrag - rollingDrag
            return force
```

```
        def accelerate(self, timeStep):
            a = self.findForce(timeStep) / mass
            self.speed += a * timeStep
             self.rps = self.speed / wheelCircumference * finalDrive
* gearRatios[self.gear]

        def move(self, timeStep):
            self.mileage += self.speed * timeStep
```

Метод `restart` призначений для того, щоб повернути автомобіль в його початковий стан.

```
        def restart(self, startRps = 0):
            self.fuel = maxFuel
            self.rps = startRps
            self.throttlePosition = 0
            self.gear = 0
            self.mileage = 0
                        self.speed = self.rps / finalDrive /
gearRatios[self.gear] * wheelCircumference
```

Останній метод використовується в одній ситуації – коли двигун глохне. Кількість палива обнулюється для того, щоб показати що оцінка даного агенту закінчена.

```
        def die(self):
            self.fuel = 0
```

Модуль ga.py містить функції, що забезпечують роботу генетичного алгоритму.

```
import numpy
```

Перший метод цього модуля створює початкову популяцію. Агенти мають наступну структуру – кожний ген складається з двох частин – з команди і специфікації.

```
def generate_starting_population(num, size):
    #Випадковим чином створює перше покоління популяції
    pop = numpy.ndarray((num, size, 2))
    val = numpy.random.uniform(0, 100, (num, size))
    for i in range(num):
        for j in range(size):
            if val[i][j] <= 1:
                pop[i][j][0] = 0
                pop[i][j][1] = 1
            else:
                pop[i][j][0] = 1
                pop[i][j][1] = numpy.random.uniform(0, 100)
    return pop
```

Наступний метод з популяції відбирає num_parents найкращих агентів, на основі котрих буде створене наступне покоління

```
def select_mating_pool(pop, fitness, num_parents):
    parents = numpy.empty((num_parents, pop.shape[1], 2))
    for parent_num in range(num_parents):
                    max_fitness_idx = numpy.where(fitness == numpy.max(fitness))
```

```
            max_fitness_idx = max_fitness_idx[0][0]
            parents[parent_num, :] = pop[max_fitness_idx, :]
            fitness[max_fitness_idx] = -99999999999
        return parents
```

Цей метод реалізує операцію схрещення.

```
    def crossover(parents, offspring_size):
        #Схрещення
        offspring = numpy.empty(offspring_size)
        crossover_point = numpy.uint8(offspring_size[1]/2)

        for k in range(offspring_size[0]):
            parent1_idx = k%parents.shape[0]
            parent2_idx = (k+1)%parents.shape[0]
              offspring[k, 0:crossover_point] = parents[parent1_idx,
0:crossover_point]
               offspring[k, crossover_point:] = parents[parent2_idx,
crossover_point:]
        return offspring
```

В цьому методі реалізується операція мутації

```
    def mutation(offspring_crossover):
        #Мутація
        for idx in range(offspring_crossover.shape[0]):
            for i in range(offspring_crossover.shape[1] // 20):
                            gene_idx  =  int(numpy.random.uniform(0,
offspring_crossover.shape[1]))
                if offspring_crossover[idx][gene_idx][0] == 1:
                    change = numpy.random.uniform(0, 20)
                    if change == 1:
```

```
                        offspring_crossover[idx][gene_idx][0] = 0
                        offspring_crossover[idx][gene_idx][1] = 1
                    else:
                        random_value = numpy.random.uniform(-50, 50,
1)
                            offspring_crossover[idx][gene_idx][1] +=
random_value
                          if offspring_crossover[idx][gene_idx][1] >
100:
                            offspring_crossover[idx][gene_idx][1] =
100
                        elif offspring_crossover[idx][gene_idx][1] <
0:
                            offspring_crossover[idx][gene_idx][1] =
0
                else:
                    change = numpy.random.uniform(0, 20)
                    if change == 1:
                        offspring_crossover[idx][gene_idx][0] = 1
                            offspring_crossover[idx][gene_idx][1] =
numpy.random.uniform(0, 100)
        return offspring_crossover
```

Модуль simulation.py.

```
import matplotlib.pyplot as plt
import numpy
import os
from model import Car
import ga
```

Спочатку створюються змінні, які будуть використовуватись для симуляції.

Списки speedPlot, rpsPlot, gearPlot та throtlePlot призначені для того, щоб відобразити стратегію керування, яку використовує агент.

```
speedPlot = []
rpsPlot = []
gearPlot = []
throttlePlot = []
```

Наступні змінні використовуються для симуляції.

```
startRps = 18
car = Car(startRps=startRps)
timeStep = 0.1
comands = [car.ChangeGear, car.throttle]
mode = input()

generationsNumber = 2000
populationSize = 100
genomeSize = 1000
bestGenome = None
fitness = [0] * populationSize
parentsNumber = 50
```

Якщо обрано режим showBest, то програма відобразить стратегію, яку використовує кращий агент.

```
if mode == "showBest":
    time = 0
    script = numpy.ndarray((genomeSize, 2))
    with open("bestAgent.txt", "r") as f:
```

```
                        genomeCommands = [float(x) for x in
f.readline().split()]
                    genomeSpecifications = [float(x) for x in
f.readline().split()]
            for j in range(genomeSize):
                script[j][0] = genomeCommands[j]
                script[j][1] = genomeSpecifications[j]
        while car.getState()["fuel"]:
            if time//10 < len(script) and time % 10 == 0:
                comands[int(script[time//10][0])](int(script[time//1
0][1]))
            car.update(timeStep)
            time += 1
            state = car.getState()
            speedPlot.append(state["speed"])
            rpsPlot.append(state["rps"])
            gearPlot.append(state["gear"])
            throttlePlot.append(state["throttle"])

        plt.subplot(221)
        plt.plot(range(time), rpsPlot, linewidth=2.0)
        plt.ylabel("Revjlutions per Second")
        plt.subplot(222)
        plt.plot(range(time), speedPlot, linewidth=2.0)
        plt.ylabel("Speed")
        plt.subplot(223)
        plt.plot(range(time), throttlePlot, linewidth=2.0)
        plt.ylabel("Throttle Position")
        plt.subplot(224)
        plt.plot(range(time), gearPlot, linewidth=2.0)
        plt.ylabel("Gear")
        plt.show()
```

Інакше буде створено нову популяцію та проводить навчання.

```
else:
    rangePlot = []
    population = numpy.array((populationSize, genomeSize, 2))
    g = 0
```

В процесі навчання кожне нове покоління зберігається в файлі currentPop щоб у випадку якщо навчання було екстрено зупинена, його можна було продовжити з останнього створеного покоління.

```
        try:
            with open("currentPop.txt", "r") as f:
                g = int(f.readline().split()[0]) + 1
                for i in range(populationSize):
                                genomeCommands = [int(x) for x in
f.readline().split]
                        genomeSpecifications = [float(x) for x in
f.readline().split]
                    for j in range(genomeSize):
                        population[i][j][0] = genomeCommands[j]
                                            population[i][j][1] =
genomeSpecifications[j]
```

Якщо збереженого покоління немає, то створюється нове.

```
        except:
                                                    population     =
ga.generate_starting_population(populationSize, genomeSize)
        while g < generationsNumber:
    Оцінка
```

```
            for s in range(population.shape[0]):
                script = population[s]
                time = 0
                while car.getState()["fuel"]:
                    if time//10 < len(script) and time % 10 == 0:
                                    comands[int(script[time//10][0])]
(int(script[time//10][1]))
                    car.update(timeStep)
                    time += 1
                    fitness[s] = car.getState()["mileage"]
                car.restart(startRps=startRps)
```

Створюється нове покоління та зберігається в файл currentPop

```
              parents = ga.select_mating_pool(population, fitness,
parentsNumber)
             bestGenome = ga.select_mating_pool(population, fitness,
1)[0]
              population = ga.crossover(parents, (populationSize//2,
genomeSize, 2))
            population = ga.mutation(population)
            population = numpy.concatenate((population, parents))
            g += 1
                 with open("currentPop.txt", "w") as f:
                log = []
                log.append(str(int(g))+"\n")
                for i in range(populationSize):
                    genomeCommands = ""
                    genomeSpecifications = ""
                    for j in range(genomeSize):
                          genomeCommands += str(int(population[i][j]
[0])) + ' '
```

```
                        genomeSpecifications += str(population[i][j]
[1]) + ' '
                    genomeCommands += "\n"
                    genomeSpecifications += "\n"
                    log.append(genomeCommands)
                    log.append(genomeSpecifications)
                f.writelines(log)
```

Ще раз проводиться оцінка найкращого агента в поколінні для того, щоб відобразити його стратегію керування.

```
            rangePlot.append(max(fitness))
            script = bestGenome
            time = 0
            while car.getState()["fuel"]:
                if time//10 < len(script) and time % 10 == 0:
                                comands[int(script[time//10][0])]
(int(script[time//10][1]))
                car.update(timeStep)
                time += 1
                state = car.getState()
                speedPlot.append(state["speed"])
                rpsPlot.append(state["rps"])
                gearPlot.append(state["gear"])
                throttlePlot.append(state["throttle"])

            plt.subplot(321)
            plt.plot(range(time), rpsPlot, linewidth=2.0)
            plt.subplot(322)
            plt.plot(range(time), speedPlot, linewidth=2.0)
            plt.subplot(323)
            plt.plot(range(time), throttlePlot, linewidth=2.0)
```

```
            plt.subplot(324)
            plt.plot(range(time), gearPlot, linewidth=2.0)
            plt.subplot(325)
            try:
                plt.plot(range(g), rangePlot, linewidth=2.0)
            except:
                pass
            speedPlot = []
            rpsPlot = []
            gearPlot = []
            throttlePlot = []
            plt.show()
```

Після навчання найкращий агент даної сесії порівнюється з найкращим агентом з файлу `bestAgent.txt`. Після цього той з них, хто показав кращий результат, зберігається в цей файл.

```
        try:
            f = open("bestAgent.txt", "r")
            script = bestGenome
            time = 0
            car.restart()
            while car.getState()["fuel"]:
                if time//10 < len(script) and time % 10 == 0:
                                    comands[int(script[time//10][0])]
(int(script[time//10][1]))
                car.update(timeStep)
                time += 1
                state = car.getState()
                distance = state["mileage"]

            genomeCommands = [int(x) for x in f.readline().split]
```

```
                    genomeSpecifications = [float(x) for x in
f.readline().split]
            for j in range(genomeSize):
                script[j][0] = genomeCommands[j]
                script[j][1] = genomeSpecifications[j]
            f.close()

            car.restart()
            time = 0
            while car.getState()["fuel"]:
                if time//10 < len(script) and time % 10 == 0:
                                    comands[int(script[time//10][0])]
(int(script[time//10][1]))
                car.update(timeStep)
                time += 1
                state = car.getState()
                speedPlot.append(state["speed"])
                rpsPlot.append(state["rps"])
                gearPlot.append(state["gear"])
                throttlePlot.append(state["throttle"])
            maxDistance = state["mileage"]

            if distance > maxDistance:
                with open("bestAgent.txt", "w") as f:
                    genomeCommands = ""
                    genomeSpecifications = ""
                    for j in range(genomeSize):
                        genomeCommands += str(int(population[i][j]
[0])) + ' '
                        genomeSpecifications += str(population[i][j]
[1]) + ' '
                    genomeCommands += "\n"
                    genomeSpecifications += "\n"
```

```
                                        f.writelines(genomeCommands,
genomeSpecifications)
        except:
            with open("bestAgent.txt", "w") as f:
                genomeCommands = ""
                genomeSpecifications = ""
                for j in range(genomeSize):
                  genomeCommands+=str(int(population[i][j][0]))+' '
                  genomeSpecifications+=str(population[i][j][1])+' '
                genomeCommands += "\n"
                genomeSpecifications += "\n"
                f.writelines([genomeCommands, genomeSpecifications])

        os.remove("currentPop.txt")
```

В кінці програми видаляється файл `currentPop.txt`. Щоб показати що навчання закінчено, програма закінчена в штатному режимі і немає необхідності продовжувати сесію. Під час наступного запуску програми буде створено нову популяцію і навчання буде проводитись з початку.

3.3. Тестування системи оптимізації споживання палива

Тестування програми розпочинаємо з режиму навчання. Після п’ятисот поколінь ми маємо наступний графік (рис.3.1).

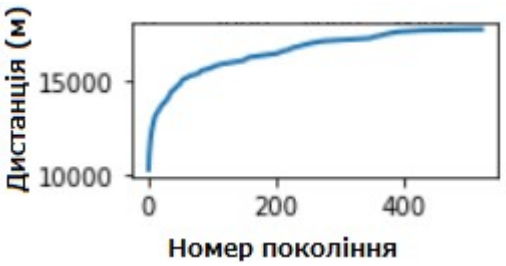
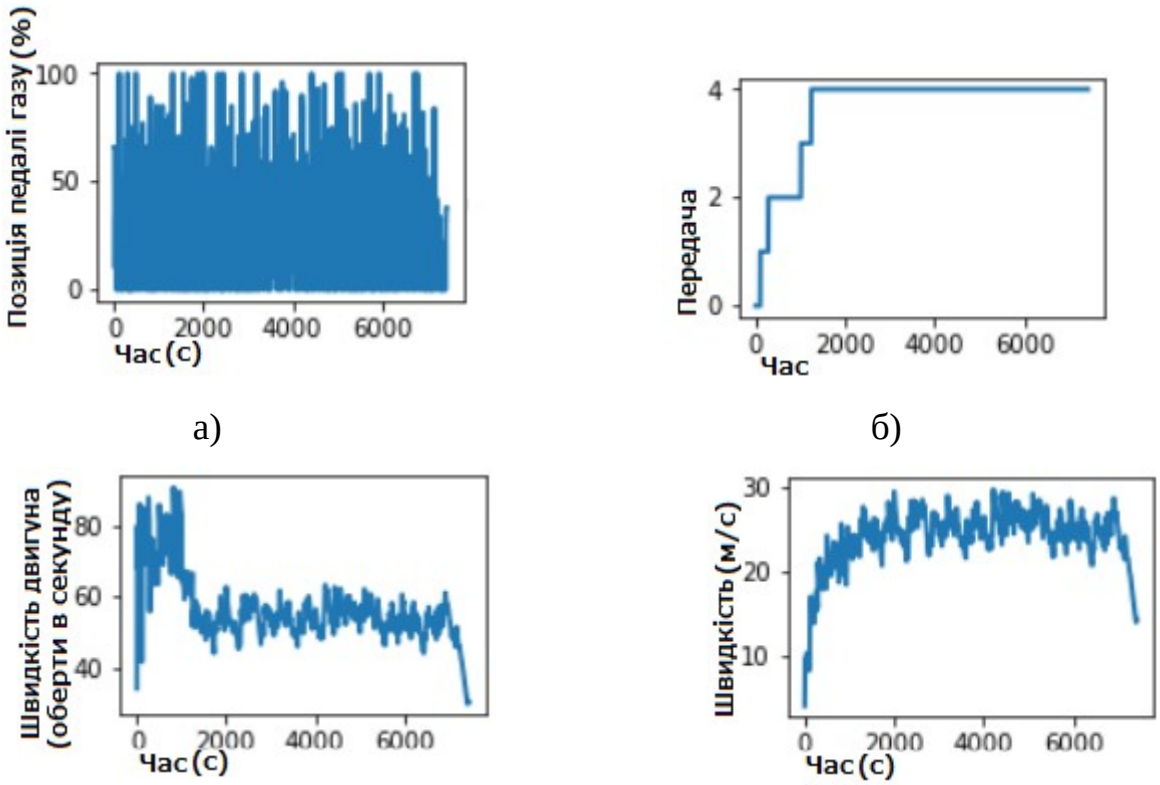


Рис.3.1 - Ріст ефективності агентів з поколіннями

На рис.3.1 відображається результат найкращого агенту в кожному поколінні. Видно, що з часом відбувається ріст ефективності стратегії керування, яка використовується агентами.

Якщо запустити програму в режимі відображення найкращого агенту, то формуються графіки позиція педалі газу, ввімкненої передачі, обертів двигуна та швидкість автомобіля (рис.3.2).



а) б)

в) г)

Рис. 3.2 – Результати моделювання роботи алгоритму:

а) позиція педалі газу;

б) ввімкнена передача;

в) оберти двигуна;

г) швидкість автомобіля.

З графіків (рис.3.2) можна побачити стратегію, яку використовує агент. Спочатку він підвищує оберти автомобіля, за рахунок чого йде прискорення, і швидко переключається на найвищу передачу, після цього зменшує оберти і тримає постійну швидкість. Ця стратегія співпадає з вищенаведеною теоретичною стратегією керування.

У зв’язку з природою агенту, створеного низкою випадкових змін, агент не тримає постійні швидкість двигуна, швидкість автомобіля та позицію педалі газу, але можна побачити що швидкість двигуна після розгону коливається навколо 50 об./с, а швидкість автомобіля – 25 м/с. Можна зробити припущення, що для конкретного автомобіля, який використовується для навчання агентів ці значення близькі до оптимальних.

ВИСНОВКИ

Для виконання величезних об’ємів перевезень різних вантажів, що необхідні для функціонування сучасної економіки, людство використовує безліч найрізноманітніших транспортних засобів. На сьогодні переважна більшість транспортних задач виконуються транспортними засобами, котрі в якості джерела механічної енергії використовують двигуни внутрішнього згорання. Зростання населення, ріст економіки тягнуть за собою необхідність збільшувати обсяги перевезень, і, як наслідок, збільшення витрат палива. Що, в свою чергу, окрім економічних втрат, тягне за собою ще й екологічні наслідки.

Найбільш кількісним видом транспорту з двигуном внутрішнього згорання наразі є автомобіль. Тому бажане зменшення сукупних витрат палива прямо залежить від можливості зменшення витрат палива автомобілями.

Існує кілька підходів для забезпечення зменшення витрат палива. В першу чергу це постійне вдосконалення конструкції двигунів для досягнення найвищого ККД. Наступний підхід – побудова оптимальної логістики, яка забезпечить зменшення необхідності кількості автомобілів та зменшення сукупних пробігів. Ще одним підходом для покращення показників використання палива, може бути розробка стратегії керування автомобілем з урахуванням особливостей конструкції двигуна внутрішнього згорання.

В роботі розглянуто принцип роботи автомобільного двигуна внутрішнього згорання, та створено спрощену модель легкового автомобіля, котра дозволяє розрахувати миттєву витрату палива в залежності від режиму роботи двигуна та положенні органів керування.

Використання такої моделі дозволяє обчислити сукупну витрату пального двигуном автомобіля при проходженні певного відрізка шляху з урахуванням стратегій керування ним, а саме – інтенсивності натиснення на педаль

акселератора, обертів двигуна, на яких відбувається перемикання передач, інтенсивності гальмування, тощо.

Для пошуку оптимальної стратегії керування автомобілем використано генетичний алгоритм.

В отриманій стратегії можна виділити основні тенденції – агент на початку шляху переключається на найвищу передачу та продовжує рух з відносно постійною швидкістю. Ця стратегія відповідає представленій в першому розділі роботи теоретичній оптимальній стратегії.

При збільшенні виділеного для навчання алгоритму часу ефективність отриманої стратегії керування збільшується.

РЕКОМЕНДАЦІЇ

Розроблений метод та програмне забезпечення дозволяє моделювати миттєву витрату палива двигуном автомобіля, та, використовуючи ці дані, знаходити оптимальну з точки зору мінімізації витрат палива стратегію керування автомобілем.

Програма може використовуватись як частина комплексу програмного забезпечення для оптимізації витрат пального, так і окремо, для проведення обчислювальних експериментів для двигунів з різними характеристиками.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Andrew Owusu-Hemeng, Working Principles of Genetic Algorithm, IPMP Journal of Management & Science, 2018
2. Branislav Šarkan, Vehicle fuel consumption prediction based on the data record obtained from an engine control unit, ResearchGate, 2019
3. Carpenter M.H., Mathematical modelling of spark-ignition engines, Butterworth & Co., 1985
4. Greenhouse Gas Inventory Data Explorer[Електронний ресурс] – Режим доступу: https://cfpub.epa.gov/ghgdata/inventoryexplorer/
5. Hadi Adibi-Asl, Math-based Spark Ignition Engine Modelling Including Emission Prediction for Control Applications, Inderscience, 2015
6. Iztok Ciglaric, Vehicle Dynamics Simulation, Part 1: Mathematical Model, International Design Conference – Design, 2002
7. Mengxi Wu, Optimal Driving Strategies for Minimizing Fuel Consumption and Travelling Time, KTH SCI, 2013
8. Shunan Lan, Genetic algorithm optimization research based on simulated annealing, IEEE, 2016
9. Yao Zhou, Study On Genetic Algorithm Improvement And Application, Worcester Polytechnic Institute, 2006
10. Вирсански Э., Генетические алгоритмы на Python, издательство ДМК Пресс, 2020
11. Грицишин Я.М., Корпильов Д.В., Кривий Р.З., Свірідова Т.В., Ткаченко С.П. Генетичні алгоритми для розв’язання задач розміщення // Національний університет “Львівська політехніка”, 2009
12. Гулаєва Н.М. Еволюційні алгоритми // Вісник Київського національного університету імені Тараса Шевченка Серія фізико-математичні науки, 2013

13. Гуляницький Л.Ф., Мулеса О.Ю. Прикладні методи комбінаторної оптимізації // Київський Національний Університет Імені Тараса Шевченка, 2016
14. Лата В.Н. Основы моделирования управляемого движения автомобиля // Издательство ТГУ, 2012
15. Максимов Н.М., Корнякова О.Ю., Головань И.Н. Математическая модель механической части двигателя внутреннего сгорания // Всероссийская научная конференция «Достижения науки и технологий-ДНиТ-2021», 2021
16. Мужиков І.О. Реалізація паралельного генетичного алгоритму // Житомирський державний технологічний університет
17. Щукина В.Н. Анализ методов определения механических потерь для их последующего применения в процессе эксплуатации, 2016

ДОДАТКИ

Додаток А. Вихідний файл ga.py

```
import numpy

def generate_starting_population(num, size):
    #Випадковим чином створює перше покоління популяції
    pop = numpy.ndarray((num, size, 2))
    val = numpy.random.uniform(0, 100, (num, size))
    for i in range(num):
        for j in range(size):
            if val[i][j] <= 1:
                pop[i][j][0] = 0
                pop[i][j][1] = 1
            else:
                pop[i][j][0] = 1
                pop[i][j][1] = numpy.random.uniform(0, 100)
    return pop

def select_mating_pool(pop, fitness, num_parents):
     #3 усієї популяції відбирає num_parents найкращих агентів,
на основі котрих буде створене наступне покоління
    parents = numpy.empty((num_parents, pop.shape[1], 2))
    for parent_num in range(num_parents):
                    max_fitness_idx  =  numpy.where(fitness  ==
numpy.max(fitness))
        max_fitness_idx = max_fitness_idx[0][0]
        parents[parent_num, :] = pop[max_fitness_idx, :]
        fitness[max_fitness_idx] = -99999999999
    return parents

def crossover(parents, offspring_size):
    #Схрещення
    offspring = numpy.empty(offspring_size)
    crossover_point = numpy.uint8(offspring_size[1]/2)

    for k in range(offspring_size[0]):
        parent1_idx = k%parents.shape[0]
        parent2_idx = (k+1)%parents.shape[0]
          offspring[k, 0:crossover_point] = parents[parent1_idx,
0:crossover_point]
           offspring[k, crossover_point:] = parents[parent2_idx,
crossover_point:]
    return offspring
```

```
    def mutation(offspring_crossover):
        #Мутація
        for idx in range(offspring_crossover.shape[0]):
            for i in range(offspring_crossover.shape[1] // 20):
                            gene_idx  =  int(numpy.random.uniform(0,
offspring_crossover.shape[1]))
                if offspring_crossover[idx][gene_idx][0] == 1:
                    change = numpy.random.uniform(0, 20)
                    if change == 1:
                        offspring_crossover[idx][gene_idx][0] = 0
                        offspring_crossover[idx][gene_idx][1] = 1
                    else:
                        random_value = numpy.random.uniform(-50, 50,
1)
                           offspring_crossover[idx][gene_idx][1] +=
random_value
                          if offspring_crossover[idx][gene_idx][1] >
100:
                            offspring_crossover[idx][gene_idx][1] =
100
                        elif offspring_crossover[idx][gene_idx][1] <
0:
                            offspring_crossover[idx][gene_idx][1] =
0
                else:
                    change = numpy.random.uniform(0, 20)
                    if change == 1:
                        offspring_crossover[idx][gene_idx][0] = 1
                            offspring_crossover[idx][gene_idx][1] =
numpy.random.uniform(0, 100)
        return offspring_crossover
```

Додаток Б. Вихідний файл model.py

```
from math import pi

#Передавальні числа трансмісії
gearRatios = [3.091, 1.864, 1.321, 1.029, 0.794]
#Передаточне число кінцевої передачі
finalDrive = 4.067
#Радіус колеса
wheelRadius = 0.2413
#Окружність колеса
wheelCircumference = wheelRadius * 2 * pi
#Максимальний об'єм палива
maxFuel = 53
#Маса автомобіля
mass = 1710
#Коєфіцієнт гальмування двигуном
engineBrakingCoefficient = 0.5
#Сопротивление воздуха
aerodeynamicResistance = 0.33
#Площадь поверхности воздушного потока
airflowSurface = 3
#Густота повітря
airDensity = 1.25
#Сопротивление кочения
rollingResistance = 0.01
#Прискорення свободного падіння
g = 9.8
#Коєффіцієнт корисної дії двигуна
efficiency = 0.5
#Енергоємність палива
energyIntensity = 1700000

#Коефіцієнти для підрахунку витрат палива від положення педалі газу та швидкості двигуна
a = 0.76712 / 3600
b = -0.25558 / 3600
c = 1.004*10**-3 / 3600
d = 1.95*10**-5 / 3600
e = -1.1*10**-7 / 3600
f = 0.0175 / 3600

class Car():
    def __init__(self, startRps = 0):
        self.fuel = maxFuel
        self.rps = startRps
        self.throttlePosition = 0
        self.gear = 0
```

```
            self.mileage = 0
                        self.speed = self.rps / finalDrive /
gearRatios[self.gear] * wheelCircumference

        def throttle(self, throttlePosition):
            self.throttlePosition = throttlePosition

        def fuelConsumption(self, timeStep):
            n = self.rps * 60
            x = self.throttlePosition
            dfuel = a+b*x+c*n+d*n*x+e*n*n+f*x*x
            if self.fuel < dfuel:
                dfuel = self.fuel
            return dfuel

        def ChangeGear(self, a):
            if a == 1:
                if self.gear < 4:
                    self.gear += 1
            else:
                if self.gear > 0:
                    self.gear -= 1

        def update(self, timeStep):
            self.fuel -= self.fuelConsumption(timeStep)
            self.move(timeStep)
            self.accelerate(timeStep)

        def getState(self):
            state = {
                "speed":self.speed,
                "rps":self.rps,
                "gear":self.gear,
                "fuel":self.fuel,
                "mileage":self.mileage,
                "throttle":self.throttlePosition}
            return state

        def findForce(self, timeStep):
            if self.rps < 1:
                self.die()
                return
            power = energyIntensity * self.fuelConsumption(timeStep)
* efficiency / timeStep
            engineTorque = power * 0.16 / self.rps
            engineTorque -= engineBrakingCoefficient * self.rps
               wheelForce = engineTorque * gearRatios[self.gear] *
finalDrive / 2 / wheelRadius
```

```
                    airDrag = airDensity * aerodeynamicResistance *
airflowSurface * self.speed*self.speed / 2

                rollingDrag = rollingResistance * mass * g * 4

                force = wheelForce - airDrag - rollingDrag
                return force

            def accelerate(self, timeStep):
                a = self.findForce(timeStep) / mass
                self.speed += a * timeStep
                 self.rps = self.speed / wheelCircumference * finalDrive
* gearRatios[self.gear]

            def move(self, timeStep):
                self.mileage += self.speed * timeStep

            def restart(self, startRps = 0):
                self.fuel = maxFuel
                self.rps = startRps
                self.throttlePosition = 0
                self.gear = 0
                self.mileage = 0
                            self.speed = self.rps / finalDrive /
gearRatios[self.gear] * wheelCircumference

            def die(self):
                self.fuel = 0
```

Додаток В. Вихідний файл simulation.py

```
import matplotlib.pyplot as plt
import numpy
import os
from model import Car
import ga


speedPlot = []
rpsPlot = []
gearPlot = []
throttlePlot = []
startRps = 18
car = Car(startRps=startRps)
timeStep = 0.1
comands = [car.ChangeGear, car.throttle]
mode = input()

generationsNumber = 2000
populationSize = 1000
genomeSize = 1000
bestGenome = None
fitness = [0] * populationSize
parentsNumber = 500

if mode == "showBest":
    time = 0
    script = numpy.ndarray((genomeSize, 2))
    with open("bestAgent.txt", "r") as f:
                    genomeCommands = [float(x) for x in
f.readline().split()]
                genomeSpecifications = [float(x) for x in
f.readline().split()]
        for j in range(genomeSize):
            script[j][0] = genomeCommands[j]
            script[j][1] = genomeSpecifications[j]
    while car.getState()["fuel"]:
        if time//10 < len(script) and time % 10 == 0:
            comands[int(script[time//10][0])](int(script[time//1
0][1]))
        car.update(timeStep)
        time += 1
        state = car.getState()
        speedPlot.append(state["speed"])
        rpsPlot.append(state["rps"])
        gearPlot.append(state["gear"])
        throttlePlot.append(state["throttle"])
```

```
            plt.subplot(221)
            plt.plot(range(time), rpsPlot, linewidth=2.0)
            plt.ylabel("Revolutions per Second")
            plt.subplot(222)
            plt.plot(range(time), speedPlot, linewidth=2.0)
            plt.ylabel("Speed")
            plt.subplot(223)
            plt.plot(range(time), throttlePlot, linewidth=2.0)
            plt.ylabel("Throttle Position")
            plt.subplot(224)
            plt.plot(range(time), gearPlot, linewidth=2.0)
            plt.ylabel("Gear")
            plt.show()
        else:
            rangePlot = []
            population = numpy.array((populationSize, genomeSize, 2))
            g = 0
            try:
                with open("currentPop.txt", "r") as f:
                    g = int(f.readline().split()[0]) + 1
                    for i in range(populationSize):
                                    genomeCommands = [int(x) for x in
f.readline().split]
                                genomeSpecifications = [float(x) for x in
f.readline().split]
                            for j in range(genomeSize):
                                population[i][j][0] = genomeCommands[j]
                                                    population[i][j][1] =
genomeSpecifications[j]
            except:
                                                            population     =
ga.generate_starting_population(populationSize, genomeSize)

            while g < generationsNumber:
                for s in range(population.shape[0]):
                    script = population[s]
                    time = 0
                    while car.getState()["fuel"]:
                        if time//10 < len(script) and time % 10 == 0:
                                        comands[int(script[time//10][0])]
(int(script[time//10][1]))
                        car.update(timeStep)
                        time += 1
                        fitness[s] = car.getState()["mileage"]
                    car.restart(startRps=startRps)

                with open("currentPop.txt", "w") as f:
                    log = []
```

```
                log.append(str(int(g))+"\n")
                for i in range(populationSize):
                    genomeCommands = ""
                    genomeSpecifications = ""
                    for j in range(genomeSize):
                          genomeCommands += str(int(population[i][j]
[0])) + ' '
                        genomeSpecifications += str(population[i][j]
[1]) + ' '
                    genomeCommands += "\n"
                    genomeSpecifications += "\n"
                    log.append(genomeCommands)
                    log.append(genomeSpecifications)
                f.writelines(log)

               parents = ga.select_mating_pool(population, fitness,
parentsNumber)
            bestGenome = parents[0]
              population = ga.crossover(parents, (populationSize//2,
genomeSize, 2))
            population = ga.mutation(population)
            population = numpy.concatenate((population, parents))
            g += 1
            rangePlot.append(max(fitness))

            script = bestGenome
            time = 0
            while car.getState()["fuel"]:
                if time//10 < len(script) and time % 10 == 0:
                                    comands[int(script[time//10][0])]
(int(script[time//10][1]))
                car.update(timeStep)
                time += 1
                state = car.getState()
                speedPlot.append(state["speed"])
                rpsPlot.append(state["rps"])
                gearPlot.append(state["gear"])
                throttlePlot.append(state["throttle"])

            plt.subplot(321)
            plt.plot(range(time), rpsPlot, linewidth=2.0)
            plt.subplot(322)
            plt.plot(range(time), speedPlot, linewidth=2.0)
            plt.subplot(323)
            plt.plot(range(time), throttlePlot, linewidth=2.0)
            plt.subplot(324)
            plt.plot(range(time), gearPlot, linewidth=2.0)
            plt.subplot(325)
            try:
```

```
                plt.plot(range(g), rangePlot, linewidth=2.0)
            except:
                pass
            speedPlot = []
            rpsPlot = []
            gearPlot = []
            throttlePlot = []
            plt.show()

        try:
            f = open("bestAgent.txt", "r")
            script = bestGenome
            time = 0
            car.restart()
            while car.getState()["fuel"]:
                if time//10 < len(script) and time % 10 == 0:
                                comands[int(script[time//10][0])]
(int(script[time//10][1]))
                car.update(timeStep)
                time += 1
                state = car.getState()
                distance = state["mileage"]

            genomeCommands = [int(x) for x in f.readline().split]
                      genomeSpecifications  =  [float(x)  for  x  in
f.readline().split]
            for j in range(genomeSize):
                script[j][0] = genomeCommands[j]
                script[j][1] = genomeSpecifications[j]
            f.close()

            car.restart()
            time = 0
            while car.getState()["fuel"]:
                if time//10 < len(script) and time % 10 == 0:
                                comands[int(script[time//10][0])]
(int(script[time//10][1]))
                car.update(timeStep)
                time += 1
                state = car.getState()
                speedPlot.append(state["speed"])
                rpsPlot.append(state["rps"])
                gearPlot.append(state["gear"])
                throttlePlot.append(state["throttle"])
            maxDistance = state["mileage"]

            if distance > maxDistance:
                with open("bestAgent.txt", "w") as f:
                    genomeCommands = ""
```

```
                    genomeSpecifications = ""
                    for j in range(genomeSize):
                          genomeCommands += str(int(population[i][j]
[0])) + ' '
                        genomeSpecifications += str(population[i][j]
[1]) + ' '
                    genomeCommands += "\n"
                    genomeSpecifications += "\n"
                                        f.writelines(genomeCommands,
genomeSpecifications)
        except:
            with open("bestAgent.txt", "w") as f:
                genomeCommands = ""
                genomeSpecifications = ""
                for j in range(genomeSize):
                     genomeCommands += str(int(population[i][j][0]))
+ ' '
                    genomeSpecifications += str(population[i][j][1])
+ ' '
                genomeCommands += "\n"
                genomeSpecifications += "\n"
                f.writelines([genomeCommands, genomeSpecifications])

        os.remove("currentPop.txt")
```