ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Пояснювальна записка

до магістерської роботи
на ступінь вищої освіти магістр

на тему: «**МОДЕЛЮВАННЯ ПРОЦЕСУ АВТОМАТИЗОВАНОГО ПАРКУВАННЯ АВТОМОБІЛЯ НА ОСНОВІ ГЕНЕТИЧНОГО АЛГОРИТМУ»**

Виконав: студент 6 курсу, групи ПДМ–61
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
Даценко.М.А.
(прізвище та ініціали)
Керівник Негоденко О.В.
(прізвище та ініціали)
Рецензент
(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут інформаційнихї технологій

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти - «Магістр»
Напрям підготовки -121 «Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри
Інженерії програмного забезпечення

Негоденко.О.В

____‖ _____2022 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Даценко Михайло Андрійович

(прізвище, ім‘я, по батькові)

1. Тема роботи: «Моделювання процесу автоматизованого паркування автомобіля на основі генетичного алгоритму»

Керівник роботи Негоденко.О.В
(прізвище, ім‘я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від «12» жовтня 2022 року № 122.

2. Строк подання студентом роботи 31.12.2022

3. Вхідні дані до роботи:
   3.1.Програми для розробки: Visual Studio Code, Figma, Fiddler, Chrome, Trello
   3.2.Вхідні поняття: Генетичний алгоритм, Сигмоїда, Геном
   3.3.Інформаційні ресурси: Wikipedia, Typescript.com, Npm.com

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).
   4.1 Вибір алгоритму для розвитку мозку автомобіля.
   4.2 Програмування органів управління та сенсорів автомобіля.
   4.3 Розробка логіки генерування та корекції генному автомобіля.
   4.4 Запуск та аналіз еволюції автомобіля.

5. Перелік графічного матеріалу

1. Мета, об’єкт та предмет дослідження
2. Огляд аналогів: згорткові нейронні мережі
3. Огляд аналогів: Q-lerning
4. 4 кроки роботи генетичного алгоритму
5. Генетичний алгоритм у деталях: лінійні многочлени
6. Генетичний алгоритм у деталях: сигмоїдна функція
7. Генетичний алгоритм у деталях: коефіцієнти управління автомобілем
8. Практичний результат: початок еволюції
9. Практичний результат: еволюція плинула довше доби
10. Порівняння продуктивності алгоритмів
11. Висновки
12. Апробація

6. Дата видачі завдання 14.10.2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	19.07-20.08	Виконано
2	Вибір алгоритму для навчання моделі автомобіля	21.08-23.08	Виконано
3	Розробка 3D світу для автомобіля	23.08-25.09	Виконано
4	Програмування логіки генерації генному автомобіля	26.09-28.10	Виконано
5	Корекція логіки та тестування еволюції автомобіля	1.11-30.11	Виконано
6	Висновки	5.12	Виконано
7	Розробка демонстраційних матеріалів	5.12	Виконано
8	Попередній захист роботи	15.12	Виконано
9	Здача роботи	31.12	Виконано

Студент_____ _____
(підпис) (прізвище та ініціали)

Керівник роботи_____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина магістерської роботи: 75 с., 2 табл., 12 рис., 1 дод.,
30 джерел.

Ключові слова: Автоматизація паркування автомобіля, генетичний алгоритм, геном, еволюція, покоління, сигмоїдна функція.

Об‘єкт дослідження: паркування автомобіля.

Предмет дослідження: генерація геному автомобіля за допомогою генетичного алгоритму та сигмоїдної функції.

Мета роботи: підвищення точності паркування автомобіля за рахунок використання в атоматизованій системі генетичного алгоритму

Методи дослідження: експеримент, мозковий штурм, спостереження тестування зі зміною об’єму даних.

Обрано оптимальний алгоритм для навчання моделі автомобіля самостійному пакуванню

Реалізовано органи керування та орієнтації 3D моделі автомобіля в просторі.

Було впроваджено генетичний алгоритм як основний елемент розвитку та інтелекту автомобіля. З використанням алгоритму було сформовано геном як носія поведінки автомобіля.

Визначено необхідну тривалість навчання автомобіля у розмірі доби, для паркування у визначене місце з точністю до одного метра

Упровадження алгоритму дозволяє спросити(замість складних та об’ємних нейронних мереж) логіку навчання автомобіля паркуванню, що також зменшує час еволюції автомобіля до однієї доби.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

GA або **ГА** (англ. Genetic algorithm) – генетичний алгоритм це алгоритм пошуку що використовується для вирішення задач оптимізаці

DE (англ. Differential evolution) – диференціальна еволюція, метод багатовимірної математичної оптимізації.

PSO (англ. Particle Swarm Optimization) – метод чисельної оптимізації.

ВСТУП

Актуальність дослідження. Враховуючи те, що зараз дуже активно розвивається сфера автомобільних асистентів і майже в кожній сучасній машині існує декілька допоміжних систем паркування, було взято за *проблему* поліпшення та спрощення алгоритмів використаних у подібних системах. У цій роботі планується дослідити та визначити, чи можливо замінити масивні нейронні мережі, звичайним алгоритмом оптимізації, і скільки часу займе навчити автомобіль пакуватися з точністю до одного метра в статичне місце.

Значущість цієї проблеми в збільшені кількості автомобілів у містах та складності умов паркування, між щільно припаркованим транспортом. А також в тому що водії можуть неохоче звертатися до страхових компаній у разі якщо їх автомобіль подряпав інший водій, бо це складний доказовий процес, якщо не має запису з відео регістратора.

Об'єкт дослідження: паркування автомобіля.

Предмет дослідження: генерація геному автомобіля за допомогою генетичного алгоритму та сигмоїдної функції.

Мета роботи: підвищення точності паркування автомобіля за рахунок використання в атоматизованій системі генетичного алгоритму

Методи дослідження: експеримент, мозковий штурм, спостереження тестування зі зміною об'єму даних.

*Завдання розробки:*в процесі автоматизування вирішувалися такі завдання:

1. Розробка 3D моделі автомобіля з органами керування
2. Довести можливість використання лінійних рівнянь замість нейронних мереж у задачах оптимізації
3. Згенерувати оптимальний геном автомобіля, який паркується на місце з точністю до метра
4. Встановити чи генетичний алгоритм більш ефективний ніж диференціальна

еволюція та оптимізація рою частинок

Наукова новизна: цього дослідження полягає в наступному:

1. Використання еволюційних алгоритмів в задачах в яких зазвичай потрібні нейронні мережі
2. Використання генному як основного носія поведінки автомобіля.
3. Впровадження етапу мутації у еволюцію, щоб отримати більш різноманітний діапазон генів.

Практична значущість: ця візуалізація навчання автомобіля, є показовим прикладом ефективного використання генетичного алгоритму. Це може стати платформою, для подальшого вивчення поведінки еволюційних алгоритмів, у задачах пошуку оптимального рішення.

1. ОГЛЯД НАУКОВОЇ ЛІТЕРАТУРИ

1.1 Огляд посібника з генетичних алгоритмів

У книзі[19] з основ генетичного програмування автори є експертами з давнім і видатним досвідом. Ріккардо Полі з університету Ессексі має на своєму рахунку понад 493 публікацій на різних наукових інтернет ресурсах. Та разом з Nicholas Mcphee мають більш як 50 років спільного досвіду в теорії та практиці генетичного програмування.

Ріккардо Полі — професор кафедри обчислювальної техніки та електронних систем в Ессексі. Він розпочав свою академічну кар'єру як інженер-електронник, захистивши докторську дисертацію на тему аналізу біомедичних зображень, щоб згодом стати експертом у сфері комп'ютерних обчислень.

Окрім статей він ще має одну завершену книгу[8] про теорію та застосування генетичного програмування на практиці.

Також Ріккардо має знання в таких областях як:

- біомедична інженерія;
- інтерфейси мозок-комп'ютер;
- нейронні мережі та обробка зображень;
- біологія та психологія.

Він є членом Міжнародного товариства генетичних і еволюційних обчислень, лауреатом нагороди EvoStar за видатний внесок у цю сферу який відбувся у 2007 році, а також Член правління ACM SIGEVO.

Частина перша книги охоплює основи генетичного програмування. Починається з легкого вступу, який дає опис тому, як зберігається сукупність програм в комп'ютері, як вони можуть розвиватися з часом. Пояснюється, як програми розкривають інформацію про самих себе. Та як генетичне програмування створює нове покоління мозку програми, шляхом мутації теперішнього розвитку або

комбінування пар кращих показників для створення нащадків. Це супроводжується простим поясненням того, як застосовувати ГП та ілюстративними прикладами використання ГП.

Частина два містить опис різних альтернативних представлень для мозку програм і деякі передові методи ГП. До них відносяться: еволюція машинного коду і паралельні програми, використання граматики мов програмування та ймовірності дистрибуції для генерації логіки програм, варіанти ГП яких дозволяють розв'язувати проблеми із кількома цілями, багато прийомів прискорення і деякі корисні теоретичні інструменти.

Частина три містить цінну інформацію для всіх, хто зацікавлений у використанні ГП, а саме практичних застосуваннях з прикладами. Щоб проілюструвати масштаби генетичного програмування, ця частина містить огляд багатьох реальних програм з вмістом генетичних алгоритмів. До них належать: підгонка кривої, моделювання даних, символьна регресія, аналіз зображень, обробка електронних сигналів, фінансова торгівля, прогнозування часових рядів, економічне моделювання, управління промисловими процесами, медицина, біологія, біоінформатика, гіперевристика, мистецькі програми, комп’ютерні ігри, розваги, стиснення зображень та результати конкуренції між людьми. Далі слідує серія рекомендацій і пропозицій, щоб отримати максимум від використання ГП. У кінці цієї частини автори роблять деякі висновки.

Частина чотири завершує книгу. Окрім бібліографії та покажчика, ця частина містить два додатки, які містять багато вказівок на ресурси якими можна доповнити знання. А також та проста реалізація алгоритмів у мові програмування Java.

1.2 Сфери використання генетичного алгоритму

Генетичний алгоритм (GA) і його варіанти ефективно використовувалися вченими в багатьох галузях для вирішення проблем. З пулу еволюційних алгоритмів дослідники вибрали GA, який був популярним як корисний і ефективний оптимізатор. Він має ряд переваг і недоліків. Однак він пропонує рішення для різноманітних проблем, таких як космічні дослідження, економіка, вивчення ринку, географія, дистанційне зондування, сільське господарство, аналіз даних, виявлення раку та багато іншого. У цьому розділі обговорюється використання GA в деяких із цих сфер із кількома експериментальними результатами, такими як кластеризація даних, ідентифікація шаблонів і зіставлення, а також виявлення форми. Результати проілюстровано та пояснено причинами для кращого розуміння застосування GA у наукових галузях**.**

У процесі проектування наукових проблем основною метою дослідників є розробка системи для забезпечення ефективного недорогого рішення. У цій спробі їм потрібен хороший оптимізатор, щоб досягти своєї мети. Доступно кілька оптимізаторів, які ефективно працюють для різних проблем. Серед них генетичний алгоритм (GA), його вибрано як функціональний інструмент для хорошого та корисного оптимізатора. Алгоритм виявився класом ефективних методів оптимізації для багатьох застосувань у інженерії, економіці, виробництві, штучному інтелекті, дослідженні операцій, космічній науці, сільському господарстві, фізиці, хімії, біоінформатиці, медицині та багатьох інших. Проте є переваги та недоліки використання генетичних алгоритмів як оптимізаторів. Можна застосувати готовий генетичний алгоритм до задачі оптимізації, зіставляючи всі вхідні дані функції з попередньо заданим представленням, таким як двійкові рядки. Однак часто генетичний алгоритм спеціалізується на конкретній проблемній області, використовуючи додаткові евристики, спеціалізоване представлення та/або генетичні оператори. Хтось може вирішити використовувати генетичний алгоритм, не

докладаючи зусиль для налаштування його під наявну проблему, компроміс зазвичай полягає в тому, що неспеціалізований генетичний алгоритм може дати погані результати, які можуть включати обчислювальну неефективну оптимізацію, що призводить до зближення до субоптимального рішення. Вчені включили налаштування в звичайну підготовку GA, відповідно до потреб і специфікації своїх дослідницьких завдань.

У літературі виділяють три основні методи пошуку, а саме:

- на основі числення;
- перелічувальний;
- випадковий пошук.

На основі числення далі методи поділяються на два класи: непрямий і прямий. Метод непрямого пошуку шукає локальні екстремуми шляхом розв’язування набору рівнянь, що є результатом похідної цільової функції. З іншого боку, метод прямого пошуку шукає локальні оптимуми, переходячи на функцію та рухаючись у напрямку, пов’язаному з локальним градієнтом. Перелічувальна схема розглядалася в багатьох формах і розмірах. Тут алгоритм пошуку знаходить значення цільової функції в кожній точці в межах кінцевого або дискретного нескінченного простору. Випадковий пошук є найбільш популярним серед дослідників, оскільки тут усуваються недоліки двох інших методик. Прикладом процедури пошуку цієї категорії є генетичний алгоритм рис. 1.1.

Рисунок 1.1 – Задача-вісімка з початковим і кінцевим станом.

Протягом кількох років, починаючи з 1950 року, багато комп’ютерних дослідників незалежно вивчали еволюційні системи з ідеєю, що в майбутньому еволюція може бути використана як важливий і суттєвий інструмент оптимізації для вирішення різноманітних проблем у інженерії. Ідея полягала в тому, щоб розробити систему з сукупністю варіантів вирішення даної проблеми з операторами, на які впливають як природний відбір, так і природні генетичні варіації. Рехенберг представив[14] «стратегії еволюції». Далі цю ідею розвинули[30] Швефель та Фогель та ін. розробивши техніку під назвою «еволюційне програмування», де можливі рішення для заданих завдань були представлені у вигляді скінченних автоматів, які еволюціонували шляхом випадкової зміни їхніх діаграм переходів між станами та вибору найпристосованіших серед них індивідуумів. Стратегії еволюції, еволюційне програмування та генетичні алгоритми разом складають основу еволюційних обчислень. Кілька інших дослідників розробили еволюційні алгоритми для оптимізації та машинного навчання. Бокс, Фрідман, Бремерманн і Reed et al працювали[28] в цій галузі, хоча їхні дослідження мало впливали на поле. Біологи-еволюціоністи також використовували комп’ютери для моделювання еволюції, та для контрольованих експериментів.

Нарешті, в 1960-х і 1970-х роках в Університеті штату Мічиган, Холланд вперше винайшов генетичні алгоритми, а пізніше вдосконалив їх разом зі своїми колегами. На відміну від попередніх стратегій еволюції, намір Холланда полягав у тому, щоб формально вивчити явище адаптації, яке відбувається в природі, замість розробки алгоритмів для вирішення конкретних проблем. Він також розробив способи імпорту природних механізмів адаптації в комп'ютерні системи. Генетичний алгоритм був представлений Холландом як абстракція біологічної еволюції, а також дав теоретичну основу для адаптації. Його GA був методом переходу від однієї популяції «хромосом» (ланцюжок одиниць і нулів) до нової популяції за допомогою свого роду «природного відбору» разом із генетичними операторами, а саме кросинговером, мутацією та інверсією. Кожна хромосома була комбінацією «генів» (бітів), і кожен ген був певним

«алелем» (або 0, або 1). У популяції оператор відбирав корисні для розмноження хромосоми. Таким чином, придатніші хромосоми дали більше нащадків порівняно з менш придатними. У процесі кросинговеру відбувся обмін ділянками двох хромосом. Це в основному імітувало біологічну рекомбінацію між двома однохромосомними (гаплоїдними) організмами.

Протягом останніх кількох років дослідники активно вивчали та взаємодіяли з різними методами еволюційного обчислення, і врешті вдалося певною мірою зруйнувати межі між GA, еволюційним програмуванням, стратегіями еволюції та іншими еволюційними підходами. Сьогодні термін «генетичний алгоритм» використовується дослідниками для опису чогось дуже далекого від початкової концепції Холланда.

Пошук збережених даних– Тут проблема полягає в тому, щоб ефективно отримати інформацію, що зберігається в пам’яті комп’ютера. Припустімо, що у великій базі даних імен та адрес, збережених певним чином, ми хочемо знайти запис, що відповідає заданому прізвищу, використовуючи двійковий пошук. Ця форма пошуку є центральною для багатьох методів штучного інтелекту. Рисунок 1.1 ілюструє простий приклад 8-пазл. Набір плиток, пронумерованих 1–8, розміщують у квадраті, залишаючи одне місце порожнім. Переміщення однієї із сусідніх плиток у порожній простір називається рухом. Типовими алгоритмами є пошук гілок, пошук у глибину, тощо. Пошук рішень– це скоріше загальний клас пошуку порівняно з шукати шляхи до мети. Тут концепція полягає в ефективному пошуку остаточного рішення проблеми у величезному просторі варіантів вирішення. Серед цих рішень одне може бути або не бути ціллю, і рішення може або не може бути покращено з ходом пошуку. Це види задач пошуку, де використовуються генетичні алгоритми. Хоча генетичний алгоритм спрямований на “шукати шляхи до мети” або пошук рішень, не можна гарантувати, що він завжди досягне мети. Це пояснюється його стохастичним або випадковим характером пошуку. З іншого боку, ця характеристика дозволяє швидко визначити рішення. З удосконаленням процесу пошуку, генетичні алгоритми

виконують певні конкретні кроки, які не схожі на традиційні процедури пошуку.

Генетичні алгоритми використовувалися для вирішення багатьох проблем. Деякі дослідження, пов'язані з реєстрацією медичних зображень, сегментацією зображення та розпізнаванням контурів. Крім того, прикладами деяких практичних застосувань є класифікація ендотеліальних клітин у зрізі тканини, нормалізація китайського почерку та оцінка ризику землетрусу для геологічних структур. GA також використовувалися для оптимізації ланцюга виділення ознак, ізоморфізму графа з виправленням помилок і зіставлення точкового шаблону. Більше того, методи генерації нечітких правил для відстеження цілей, проблем обмеженого розміщення, планування процесу для відповідності вакансій та ідентифікації сліпих каналів із кумуляційною підгонкою вищого порядку використовують GA. У сфері пошуку інформації GA використовується для отримання динамічного веб-контенту. Також вчені працювали із зміщеною мутацією в еволюційному алгоритмі для розв'язання задач вибору підмножини на повних графах.

Незважаючи на те, що простий генетичний алгоритм використовувався для вирішення різних проблем, дослідники завжди намагалися підвищити продуктивність GA шляхом модифікації алгоритму. Серед кількох підходів до покращення продуктивності в 1970 році Cavicchio розробив техніку збереження найкращих особин шляхом заміни нижчих батьків, якщо пристосованість нащадків перевищувала пристосованість неповноцінного батька. Схема[21] скупченості Де Йонга спрямована на збереження різноманітності та найкращих особин у популяції шляхом обміну максимально подібними рядками. Фогель [18], а також Бек і Швефель[1] протестували деякі оптимізуючі функції для своїх алгоритмів і показали, що еволюційний метод із самоадаптивною мутацією працює краще, ніж метод без самоадаптивної мутації. Крім простих генетичних підходів, Девіс припустив, що гібридизація генетичних алгоритмів з найбільш успішними методами оптимізації для конкретних проблем забезпечує найкращу продуктивність[22]. Гібридна генетична схема спрямована на підйом на гору з кількох точок, розкиданих у просторі пошуку.

У випадку мультимодальної цільової функції очевидно, що деякі хромосоми/струни (нащадки) будуть у басейні тяжіння глобального рішення, де підйом на пагорб є ефективною та швидкою формою пошуку. Однак гібридний підхід псує гіперплощинну дискретизацією, але не руйнує її повністю.

У літературі також зустрічаються ще деякі варіанти ГА. Однак реалізація модифікованого генетичного алгоритму часто збільшує час обчислення для вирішення різних складних задач, і дослідники намагалися збільшити швидкість алгоритму за допомогою паралельного/розподіленого GA, коли час обчислення великий для конкретної проблеми. Різні паралельні реалізації GA обговорюються[2] вченими. Задача багатокритерійної оптимізації також вирішується паралельним генетичним алгоритмом. Основні паралельні багатоцільові генетичні алгоритми обговорюються, а деякі спостереження включені в дослідження[15]. Крім того, два підходи паралельних GA, а саме внутрішня модель і дифузія моделі, обговорюються в [23,24] відповідно. У внутрішній моделі популяція поділяється на кілька менших популяцій. Серед субпопуляцій міграція особин відбувається час від часу у процесі пошуку. Проте відбір особин для міграції та частота міграції є суттєвими дискусійними проблемами[21]. Навпаки, кожна особина в дифузійній моделі пов’язана з просторовим розташуванням на сітці низької розмірності. Вся популяція розглядається як група активних особин, які взаємодіють лише з сусідами. Інше важливе використання розподіленого GA помічено в задачі маршрутизації, яка може обробляти як електричні, так і топологічні обмеження проблеми [12].

Дослідники також успішно використовували GA для вирішення різних проблем в інших наукових галузях. Notredome і Higgins[9] розробили популярне програмне забезпечення для вирівнювання послідовностей з двома цільовими функціями. Було розроблено кілька підходів на основі GA для вирішення багатопослідовного вирівнювання. У хімії ГА використовується для вивчення окислення води[16], магнітного зберігання [17], каталізу [24], стабільності борних коліс[5]. Також вчені розробили генетичний алгоритм для класифікації генів для ідентифікації генів-

біомейкерів, щоб запропонувати індивідуальне лікування. Як надійний евристичний метод пошуку GA допоміг отримати інформацію з великих наборів даних. Його було застосовано для виявлення цікавих і корисних зв’язків між елементами даних з абстракцією в області правил асоціації.

Інший метод класифікації на основі GA був використаний для покращення вибору ознак за допомогою класифікатора опорних векторних машин. Вибір ознак використовувався для класифікації набору даних раку молочної залози з 699 випадками на два класи, а саме доброякісні та злоякісні, кожен з 11 атрибутами. Генетичний підхід також застосовувався в машинному навчанні для вибору правильного та найкращого ходу в грі в шахи. Техніка допомогла визначити ефективний хід шляхом класифікації набору правил для конкретного рішення.

Крім того, лікарі-практики, а також вчені запропонували декілька GA та гібридних GA для виявлення раку за допомогою різних підходів, таких як вибір ознак, категоризація та класифікація, реєстрація різниці температур тощо. Розроблено гібридний генетичний підхід для ранньої діагностики раку молочної залози за допомогою термографії. Цей метод вимірював різницю температур між раковими та здоровими тканинами з високим рівнем успіху.

Також генетичний алгоритм використовувався у обробці точкових візерунків або наборів точок на площині, що є корисним та важливим в задачах розпізнавання образів. Точкові моделі зустрічаються в різних проблемах, включаючи точки в просторі ознак, пікселі в цифровому зображенні, фізичні об’єкти, такі як зірки в галактиці, або просторові дані. Атрибутом точкових візерунків є створена ними візуальна форма.

Ще одним прикладом використання ГА є вирівнювання на рівні послідовності, що представляє первинну структуру біологічних макромолекул, яка представлена у вигляді лінійного послідовного ланцюга.

ГА використовуються в різних областях серцево-судинної медицини. Атеросклеротичні бляшки є ознаками більшості інфарктів міокарда та інсультів.

Визначення механічних властивостей бляшок, таких як еластичність, дасть змогу лікарям краще визначати місцезнаходження та картувати вразливі або нестабільні бляшки. Вчені використовували систему, що включає GA для оцінки параметрів, необхідних для точного кількісного визначення еластичності тканини. Ця система є кращою за градієнтні методи, які використовуються для оцінки параметрів для неоднорідних просторів, які містять кілька локальних мінімумів і вимоги до значного обчислювального часу обмежують їх застосування.

Сфера відкриття біомаркерів і клінічної протеоміки швидко розвивається в медичній діагностиці, прогнозі та подальшому спостереженні за захворюваннями. Передові технології, такі як мас-спектрометрія, можуть генерувати тисячі білків із зразків пацієнтів; однак вартість і складність таких методів, з одного боку, і обчислювальних і статистичних методів аналізу, з іншого боку, зумовлюють необхідність вибору кількох відповідних маркерів для розробки клінічного аналізу. Вчені використовували вдосконалену версію GA, що підтримується рекурсивною локальною плаваючою технікою посилення, щоб передбачити ризик великої несприятливої серцевої події. Ця методика дозволила вибрати панель із семи білків, включаючи мієлопероксидазу, щоб передбачити ризик з точністю 77%, що перевершило кілька сучасних методів.

Моделі логістичної регресії часто використовуються для діагностики захворювань. Завдяки своїм видатним характеристикам GA було використано для вибору найкращих змінних для системи логістичної регресії, спрямованої на моделювання наявності інфаркту міокарда у пацієнтів із болем у грудях. Метод, заснований на GA, перевершував інші традиційні методи у виборі змінних.

Одним із ключових елементів автоматичної інтерпретації електрокардіограми (ЕКГ) є виявлення комплексів QRS, що дозволить оцінити варіабельність серцевого ритму та інші відповідні діагностичні параметри. Вчені представили простий та ефективний підхід з використанням GA для виявлення комплексів.

Належне управління грошовими ресурсами та персоналом є невід’ємною

частиною систем охорони здоров'я в усьому світі. Одним із важливих елементів управління лікарнею, який може підвищити рівень обслуговування пацієнтів, задоволеність і економічну ефективність, є ефективне планування прийому пацієнтів. Було розроблено та оптимізовано математичну модель за допомогою GA для покращення планування пацієнтів в офтальмологічній лікарні. Новий алгоритм був кращим від традиційної моделі «першим прийшов, перший обслужений» у тому, що він скоротив список очікування, знизив рівень вакантності лікарняних ліжок, зменшив передопераційний час очікування пацієнтів і збільшив кількість пацієнтів, які виписуються з лікарні. Інший звіт показав, що поєднання GA та оптимізації роїв частинок, іншого потужного метаевристичного алгоритму, змогло покращити планування пацієнтів, зменшити втрату часу та підвищити задоволеність пацієнтів.

У клінічних лабораторіях регулярна ротація персоналу на основі їхніх навичок у різних приміщеннях має основне значення для підтримки робочих навичок і компетентності. GA були застосовані для покращення планування ротації персоналу в клінічній лабораторії. В одному звіті програмне забезпечення на основі GA було здатне ефективно планувати ротацію персоналу, забезпечуючи підтримку техніки та навичок, заощаджуючи час і кошти, необхідні для процесу планування, і це було підтверджено задоволенням відповідального наглядового персоналу.

Біомедична інженерія запропонувала чудові рішення в галузі ортопедичної хірургії. Тотальне ендопротезування суглобів покращило лікування різних захворювань, що призводять до інвалідності. Тим не менш, неспроможність стегнової кістки може поставити під загрозу успіх лікування. Вчені повідомили про використання GA у розробці оптимізованої геометрії компонента стегнової кістки. GA також використовувалися для вибору найкращої конструкції фіксуючих гвинтів великогомілкової кістки, щоб зменшити ймовірність поломки або ослаблення гвинта.

Променева терапія з модульованою інтенсивністю була розроблена для передачі точної дози випромінювання до мішені. Застосування GA могло б покращити вибір кутів порталу за розумний час.

2. ОБҐРУНТУВАННЯ ВИБОРУ ГЕНЕТИЧНОГО АЛГОРИТМА

2.1 Огляд аналогів генетичного алгоритму

2.1.1 Огляд диференціальної еволюції

Диференціальна еволюція (DE) — це метаевристичний алгоритм пошуку на основі сукупності, який оптимізує проблему шляхом ітераційного вдосконалення кандидата на рішення на основі еволюційного процесу. Такі алгоритми роблять мало або взагалі не роблять припущень щодо основної проблеми оптимізації та можуть швидко досліджувати дуже великі простори проектування. DE, мабуть, є одним із найбільш універсальних і стабільних алгоритмів пошуку на основі сукупності, який виявляє стійкість до мультимодальних проблем. У сфері будівельної інженерії більшість практичних проблем оптимізації пов’язані з одним або декількома поведінковими обмеженнями. Задачі обмеженої оптимізації є досить складними для вирішення через їх складність і високу нелінійність. У цьому підрозділі ми досліджуємо продуктивність кількох варіантів DE, а саме стандартного DE, композитного DE (CODE), адаптивний DE з додатковим зовнішнім архівом (JADE) і самоадаптивний DE (JDE і SADE) для обробки проблем структурної оптимізації з обмеженнями, пов’язаних із фермовими конструкціями. Продуктивність кожного варіанту DE оцінюється за допомогою п’яти добре відомих еталонних структур у 2D та 3D. Оцінка проводиться на основі остаточного оптимального результату та швидкості конвергенції. Цінні висновки отримані в результаті статистичного аналізу, який може допомогти інженеру-будівельнику на практиці вибрати відповідний алгоритм для такого роду задач. Оцінка проводиться на основі остаточного оптимального результату та швидкості конвергенції.

Оптимізація конструкцій була темою, яка викликає великий інтерес як для науковців, так і для інженерів, особливо в останні роки. Метаевристичні алгоритми пошуку широко визнані як ефективні підходи до вирішення складних задач

оптимізації. Такі алгоритми розроблені для розв’язання широкого спектру задач оптимізації наближеним способом, без необхідності явного пристосування до кожної окремої проблеми. Крім того, вони можуть бути застосовані до задач, для яких не існує задовільного алгоритму для конкретної проблеми.

Останніми роками[25] диференціальна еволюція (DE) набула все більшого інтересу для вирішення задач оптимізації в багатьох наукових та інженерних областях. Сьогодні він вважається одним із найпопулярніших алгоритмів оптимізації для задач безперервної оптимізації. Метод був спочатку запропонований[26] для мінімізації недиференційованих і, можливо, нелінійних безперервних функцій. Варто зазначити, що незважаючи на те, що DE є еволюційним алгоритмом, він не має природної парадигми і не є біологічно натхненним, як більшість інших еволюційних алгоритмів. DE продемонстрував дуже хорошу продуктивність у різноманітних задачах оптимізації з різних галузей науки. Він належить до стохастичних еволюційних методів на основі популяцій і, як і інші еволюційні алгоритми, використовує популяцію варіантів рішень, а пошук здійснюється стохастичним способом шляхом застосування операторів мутації, кросинговеру та відбору, щоб спрямувати популяцію до кращих рішень

Основною перевагою стандартного DE є те, що він має лише три контрольні параметри, які потрібно налаштувати. Продуктивність DE в конкретній оптимізаційній задачі значною мірою залежить як від схеми генерації пробного вектора, так і від вибору контрольних параметрів. Спочатку потрібно вибрати схему генерації пробного вектора, а потім налаштувати параметри керування для задачі оптимізації, щоб отримати хороші результати оптимізації. Пошук правильних значень контрольних параметрів не завжди легкий і може стати трудомістким і складним, особливо для складних проблем оптимізації. Це спонукало дослідників до вивчення та розробки нових вдосконалених варіантів DE, які демонструють адаптивні та самоадаптивні параметри керування. У разі адаптивного керування параметрами[4] параметри адаптуються на основі відгуків, отриманих під час процесу пошуку.

У 2017 представили[6] модифіковану схему генерації вектора мутації для базового DE для вирішення проблеми стагнації. Було запропоновано новий варіант DE, і його продуктивність перевірена на 24 еталонних функціях. Аббас та ін. у 2015 запропонував варіант батьківського вибору алгоритму DE на основі турнірів, намагаючись покращити можливості пошуку та покращити швидкість конвергенції DE.

У статті також описується статистичне порівняння існуючих варіантів мутації DE, класифікуючи ці варіанти на основі їх загальної продуктивності. Харалампакіс і Ціатас порівнюють[3] варіанти генетичних алгоритмів з оптимізацію роєм частинок та моделюванням відпалу в проблемах структурної оптимізації розмірів ферм. Автори стверджують, що для досліджуваних задач DE є найнадійнішим алгоритмом, демонструючи надійність, відмінну продуктивність і масштабованість. Мезура-Монтес та ін. представили[29] емпіричне порівняння кількох варіантів DE у вирішенні задач глобальної оптимізації, де було досліджено 13 еталонних проблем із літератури та реалізовано вісім різних варіантів.

У цьому розділі ми досліджуємо продуктивність п’яти популярних варіантів DE у вирішенні проблем структурної оптимізації з обмеженнями. Більш конкретно розглядаються такі п'ять проблем:

- стандартна диференціальна еволюція;
- композитна диференціальна еволюція;
- диференціальна еволюція самоадаптивних параметрів керування;
- адаптивна диференціальна еволюція з додатковим зовнішнім архівом;
- самоадаптивна диференціальна еволюція.

Ми перевіряємо продуктивність кожного алгоритму в п’яти задачах структурної оптимізації, трьох площинних і двох просторових еталонних конструкцій, де метою є мінімізація конструктивної ваги з урахуванням обмежень щодо напружень і переміщень.

Багато проблем у структурній інженерії часто включають роботу з великою

кількістю проектних параметрів, які можуть вплинути на продуктивність системи. Проектування та випробування інженерних рішень вимагає часто ітераційного процесу з належним налаштуванням параметрів, що може бути важким і трудомістким. Оптимізація дизайну пропонує вирішення цієї проблеми шляхом організованої та автоматизованої зміни параметрів дизайну з метою досягнення оптимального рішення. Метою оптимізації зазвичай є мінімізація функції витрат.

У задачах структурної оптимізації мета пов'язана з мінімізацією ваги конструкції за деяких поведінкових обмежень, які зазвичай пов'язані з переміщеннями та напругами. Параметри конструкції пов'язані з розмірами поперечних перерізів елементів конструкції. Дане дослідження зосереджено на двовимірних і тривимірних конструкціях, а розрахункові змінні є безперервними, що представляють площі поперечного перерізу елементів конструкції. Для таких задач цільовою функцією зазвичай є вага (або маса) конструкції, а задача формулюється як на рис. 2.1:

$$\begin{aligned} \min_{x_i} \quad & W(\mathbf{x}) = \sum_{i=1}^{N_e} L_i x_i \rho_i \\ \text{s.t.} \quad & g_k(\mathbf{x}) \leq 0, \quad k = \{1, \ldots, K\} \end{aligned}$$

Рисунок 2.1 – Задача оптимізації інженерних конструкцій.

де W (x) – загальна структурна вага, N e – кількість структурних елементів, x = { x 1 , …, x N e } – вектор, який містить площі поперечних перерізів x і усіх елементів, L i – довжина, а ρ i – щільність матеріалу елемента i . Крім того, g (x) є поведінковими обмеженнями. Поведінкові обмеження - це зв'язки, що включають зазвичай напруги та прогини різних елементів і вузлів конструкції, наприклад, максимальне напруження, максимальний прогин або мінімальну навантажувальну здатність для задоволення вимог. Багато разів, для практичних цілей і для одноманітності, площі

окремих елементів групуються разом, так що кількість проектних змінних може стати значно меншою, ніж загальна кількість елементів конструкції. Це дуже зручно, особливо для великих конструкцій з великою кількістю елементів. Групування в оптимізації також відповідає фактичному групуванню, яке виконується на практиці завдяки симетрії та простоті.

Загальною практикою для вирішення проблеми оптимізації, яка включає обмеження нерівності, є використання функції штрафу. Такі функції можна використовувати для перетворення початкової проблеми з обмеженнями в необмежену. Це був популярний підхід для вирішення таких проблем, оскільки він простий і досить легкий у застосуванні. У цьому дослідженні для обробки обмежень задачі оптимізації використовується наступне формулювання як на рис. 2.2

$$F(\mathbf{x}) = f(\mathbf{x}) + P(\mathbf{x}) = f(\mathbf{x}) + \mu \sum_{k=1}^{N} H_k(\mathbf{x})\, g_k^2(\mathbf{x})$$

Рисунок 2.2 – Функція штрафу

де f(x) — цільова функція, яку потрібно мінімізувати, P (x) — штрафний термін, gk(x) — k -та функція обмеження у формі gk(x) ≤ 0, μ ≥ 0 — коефіцієнт штрафу, який має бути достатньо великим, а функція Hk(x) визначається наступним чином, як на рис. 2.3:

$$X_i(\mathbf{x}) = \begin{cases} 1, & \text{якщо } g_i(\mathbf{x}) \geq 0 \\ 0, & \text{інакше} \end{cases} \quad (3)$$

Рисунок 2.3 – Hk(x) функція

DE — це популярний метод оптимізації, що використовується для багатовимірних дійсних функцій, який використовує сукупність індивідуальних

рішень. Метод не вимагає градієнтної інформації, що означає, що задача оптимізації не повинна бути диференційованою. Алгоритм шукає простір проектування, зберігаючи популяцію рішень-кандидатів (окремих осіб) і створюючи нові рішення, об'єднуючи існуючі відповідно до певного процесу. Кандидати з найкращими об'єктивними значеннями зберігаються в наступній ітерації алгоритму таким чином, що нове об'єктивне значення індивідуума покращується, формуючи в результаті частину популяції, інакше нове об'єктивне значення відкидається. Процес повторюється, доки не буде задоволено заданий критерій завершення.

Основною перевагою DE є те, що він має лише три контрольні параметри, які користувач алгоритму повинен налаштувати. До них належать розмір популяції NP , де $NP \geq 4$, фактор мутації (або диференціальна вага, або коефіцієнт масштабування) $F \in [0, 2]$ та ймовірність кросинговеру (або контрольний параметр кросинговеру) $CR \in [0, 1]$. У стандарті DE ці контрольні параметри були фіксованими для всього процесу оптимізації. Розмір населення має значний вплив на здатність алгоритму досліджувати. У разі проблем із великою кількістю вимірів розмір популяції також має бути великим, щоб зробити алгоритм здатним здійснювати пошук у багатовимірному просторі проектування. Розмір популяції 30–50 зазвичай достатній для більшості проблем, що представляють інженерний інтерес. Коефіцієнт мутації F є позитивним контрольним параметром для масштабування та керування ампліфікацією різницевого вектора. Малі значення F призведуть до менших розмірів кроку мутації, і, як наслідок, для збіжності алгоритму знадобиться більше часу. Великі значення F полегшують дослідження, але можуть призвести до виходу алгоритму за хороші оптимуми. Таким чином, значення має бути достатньо малим, щоб пришвидшити дослідження, але також достатньо великим, щоб підтримувати різноманітність. Імовірність кросинговеру впливає на різноманітність DE, оскільки вона контролює кількість елементів, які будуть змінюватися. Більші значення CR призведуть до появи більшої кількості варіацій у новій популяції, отже, її збільшення збільшує дослідження. Але знову ж таки потрібно знайти компромісне значення, щоб

забезпечити як локальні, так і глобальні можливості пошуку.

Дослідження показали, що як керуючі параметри, так і схеми генерації пробного вектора можуть суттєво впливати на продуктивність алгоритму. Таким чином, різні схеми генерації пробного вектора та контрольні параметри можна комбінувати для покращення продуктивності процесу оптимізації для різних типів проблем. Mallipeddi та ін. у 2011 був першим, хто спробував[7] поєднати різні схеми генерації пробного вектора з різними налаштуваннями контрольних параметрів.

Стандартний алгоритм DE включає набір параметрів, які залишаються фіксованими протягом усього процесу оптимізації. Ці параметри потрібно було б налаштувати для кожної окремої задачі оптимізації, щоб забезпечити оптимальну продуктивність. Деякі дослідники стверджують, що параметри DE не так вже й важко встановити вручну. Проте інші стверджують[10], що процес може бути досить складним, особливо для конкретних задач оптимізації. Ефективність, результативність і стійкість алгоритму DE дуже чутливі до значень контрольних параметрів, тоді як певні параметри можуть добре працювати в одній задачі, але не дуже добре в інших проблемах, що робить оптимальний вибір параметрів проблемою.

JDE має самоадаптивні налаштування параметрів керування та показав[27] прийнятну продуктивність у задачах тестування у 2006 році. Він використовує ідею еволюції, тобто використовує процес еволюції для точного налаштування параметрів алгоритму оптимізації. Хоча ідея звучить багатообіцяюче, доказ збіжності алгоритмів самоадаптації в цілому є складним завданням. У JDE техніка керування параметрами базується на самоадаптації параметрів F і CR еволюційного процесу DE, створюючи гнучку DE, яка налаштовується сама для досягнення найкращого результату оптимізації.

2.1.2 Порівняння продуктивності генетичного алгоритму, диференціальної еволюції та оптимізації рою частинок

Генетичний алгоритм (GA), диференціальна еволюція (DE) і оптимізація роєм

частинок (PSO) завжди реалізуються для вирішення різних типів складних задач оптимізації. Кожен метод має свої переваги, а продуктивність залежить від різних умов . Існує багато методів програмного обчислення, які можуть генерувати різні результати для одних і тих самих задач оптимізації. Однак точного результату не виходить, оскільки в методах зазвичай використовується випадкова функція. На продуктивність може впливати налаштування параметрів або операції всередині кожного методу. Таким чином, мотивація цього розділу полягає в тому, щоб порівняти продуктивність GA, DE та PSO за допомогою тих самих параметрів налаштування та задач оптимізації. Експерименти можуть підтвердити, що, незважаючи на те саме налаштування параметрів, можна отримати різну придатність і час. На основі результатів було доведено, що GA має кращі результати порівняно з DE та PSO в отриманні найбільшої кількості найкращих мінімальних показників придатності та зробивши це швидше, ніж обидва інші методи.

Генетичний алгоритм (GA), диференціальна еволюція (DE) і оптимізація роєм частинок (PSO) є популярними методами, які завжди застосовуються для вирішення різних типів складних задач оптимізації. Кожен метод має свої переваги під час його використання, і продуктивність залежить від різних умов. Іноді переваги містять недоліки, оскільки кожен метод працює по-різному в кожній задачі оптимізації. Деякі методи містять складні операції, але все ж можуть давати чудові результати. Хоча деякі методи є простими, але через налаштування параметрів дають погані результати.

Одні й ті самі проблеми оптимізації перевірялися за допомогою різних методів, оскільки дослідники хочуть перевірити ефективність кожного методу, наприклад, отримати найоптимальніший результат або перевірити швидкість методу. Якщо отриманий результат ближчий до фактичного оптимального або швидкість вища, отже, цей метод вважається таким, що має кращу продуктивність порівняно з іншими. Є багато методів, які можуть генерувати різні результати для одних і тих самих задач оптимізації. Отже, продуктивність методів оптимізації є хорошим полем для досліджень.

Результати можуть підтвердити, що, незважаючи на те саме налаштування параметрів, можна отримати різні результати (мін., макс. і середнє) і швидкість (час), що може чітко продемонструвати ефективність кожного методу.

Застосовується стандартне налаштування параметрів, оскільки в цій роботі порівнюються різні методи. Таким чином, результати можуть показати та підтвердити продуктивність кожного методу точно для отримання мінімального значення. Табл. 2.1 показує налаштування параметрів, задіяних у кожному методі. Для DE немає стандартних налаштувань параметрів. Через те, що мутація та кросинговер для DE розглядаються як один процес, отже, параметр 6 застосовується з діапазоном стандартної ймовірності кросинговеру GA. Параметру 7 призначається менше значення, оскільки воно значно вплине на мутантний вектор.

Таблиця 2.1 – Налаштування параметрів у методах

Номер	Параметр	Значення
1.	Номер покоління	2000
2.	Чисельність населення, чол	40
3.	Номер розміру, d	30
4.	GA Імовірність кросинговеру	0.7
5.	GA Ймовірність мутації	0.01
6.	DE Імовірність кросинговеру	0.8
7.	DE Диференціальна вага	0.3
8.	PSO Інерційна вага	0.7
9.	PSO Константа прискорення	2

0,8 є стандартним значенням для параметра 8, оскільки воно впливатиме на попереднє значення швидкості, тому призначається менше значення. Максимальна генерація використовується як критерій завершення.

У табл. 2.2 наведено еталонні математичні функції з діапазоном набору даних,

які використовуються як функція придатності для перевірки ефективності кожного методу. Кожна функція бенчмарку містить різні набори даних, які генеруються випадковим чином, і немає стандартного набору даних для тестування функцій бенчмарку.

Таблиця 2.2 – Еталонні математичні функції

No	Benchmark Function	Range [min,max]	Act. Opt.
1.	Сфера, f1(x)	[-5.12,5.12]	0
2.	Еклі, f2(x)	[-30,30]	0
3.	Растригін, f3(x)	[-5.12,5.12]	0
4.	Захаров, f4(x)	[-5,10]	0
5.	Гіпереліпсоїд, паралельний осі, f5(x)	[-5.12,5.12]	0
6.	Гриванк, f6(x)	[-600,600]	0
7.	Сума різної потужності, f7(x)	[-1,1]	0
8.	Подвійна сума Швефеля, f8(x)	[-65.536,65.536]	0
9.	Квартик із шумом, f9(x)	[-1.28,1.28]	0
10.	Міхалевіц, f10(x)	[0,π]	-0.966d

Кожну еталонну функцію перевіряли 10 разів і записували мінімальне та максимальне оптимальні значення. Це означає, що різні оптимальні значення можна отримати за допомогою одного методу. Середнє 10-кратне тестування для всіх функцій розраховується та використовується як ще одне вимірювання продуктивності. Крім того, середній час отримання результату було взято як інше порівняння, щоб довести швидкість кожного методу. Таким чином, метод, який здатний виробляти найбільшу кількість мінімальних і середніх значень з найвищою

швидкістю, вважається найкращим методом із відмінною продуктивністю. Усі експерименти проводяться за допомогою Dev C++.

Немає математичних доказів у тестуванні конвергенції GA, і точних результатів не буде отримано від GA. Це твердження також справедливе для інших методів через невпевненість, коли алгоритм буде припинено, і кожен раз під час виконання експерименту буде отримано інший результат. Отже, цей експеримент лише для перевірки ефективності кожного методу шляхом порівняння мінімального значення та середнього часу, витраченого на ‘отримання результату.

На основі огляду літератури та результатів див. Додаток А, аналіз показує, що GA містить 5/10 найкращої мінімальної придатності, 3/10 найкращої середньої придатності та 10/10 швидше, ніж DE і PSO. Було доведено, що GA є найкращим методом, який виконує мету цієї роботи шляхом отримання найбільшої кількості найкращих мінімальних показників придатності та найшвидшого методу порівняно з DE та PSO. Хоча GA може добре працювати в задачах оптимізації, але це сильно залежить від методів, реалізованих в операціях. З точки зору швидкості, DE на 10/10 швидше, ніж PSO. Однак, виходячи з аспекту найкращої мінімальної та середньої придатності, PSO містить 3/10 найкращої мінімальної придатності та 6/10 найкращої середньої придатності, що краще, ніж DE. Результат показує, що PSO є лише трохи повільнішим, ніж DE, що робить висновок, що PSO має кращу продуктивність, ніж DE.

GA працює швидше, ніж інші методи, оскільки реалізовано заміну в стаціонарному стані. Найгірша хромосома замінюється в кожному поколінні на кращу хромосому. DE іноді працює краще в отриманні мінімального значення, а оновлення поколінь для DE добре з точки зору узагальненої хромосоми попереднього покоління. Однак DE повільніше порівняно з GA з точки зору швидкості. Кожна DE-хромосома порівнюється з пробним вектором для отримання нових рішень. PSO також має оновити розташування всіх частинок. Це можна показати результатом, згідно з яким використовувалася така сама максимальна генерація, але DE та PSO потребували

більшого часу порівняно з GA. Таким чином, GA може працювати навіть краще, якщо стандартне використання часу використовується як критерій завершення. Однак це все ще залежить від техніки операції, яка використовується в GA, що може вплинути на загальний результат.

Нові генетичні значення будуть введені DE під час селекційної операції, тоді як нове розташування частинок буде оновлено PSO за допомогою процесу оновлення. Простір пошуку можна покращити та отримати кращий результат. Однак недоліком є те, що для отримання результату буде використано більше часу. Два методи GA, а саме GA1 і GA2 з різними техніками роботи, були застосовані[11] для перевірки продуктивності GA. Автори показали, що GA1 не вносить жодних нових генетичних значень у структуру, тому що була здійснена взаємозаміна, але була сформована нова структура хромосоми. Коли GA2 вводить нові генетичні значення під час операцій мутації, методу потрібно ще більше часу для пошуку оптимального значення.

Насправді, операція мутації DE залежить від імовірності кросинговеру, завдяки чому результат буде застосовано лише під час операції кросинговеру. Отже, можна вважати, що мутація мала місце лише тоді, коли було виконано кросинговер. Як стверджує автор [13], DE не має окремого оператора мутації. PSO занадто сильно залежить від використання параметрів. Інші частинки потраплять у локальний оптимум. Крім того, в PSO використовувалося багато параметрів. Усі параметри використовуються для зменшення результату частинок із збільшенням ітерації.

Хоча параметри лінійно зменшуються, коли ітерація збільшується, це все одно може сильно вплинути на отриманий результат. Попередня швидкість використовуються для оновлення поточного розташування частинки, що дозволить частинці відійти від оптимального результату. Диференційна вага для DE використовується для зменшення результату під час операції мутації, тоді як вага інерції, константа прискорення та випадкове число використовуються PSO для зменшення значення швидкості під час процесу оновлення. Але для GA він не містив параметрів, які безпосередньо впливатимуть на результат під час виконання операцій.

Це твердження показує, що GA містить чистий результат і не використовує жодних параметрів для зменшення його результату.

Якщо згенероване число менше ймовірності кросинговеру DE, отже, значення мутованого вектора не застосовуються до нового пробного вектора. Тому кросинговер і мутація розглядаються як один процес. Обмеження можна вирішити, ввівши нову операцію, яка дозволяє хромосомі навчатися до глобального оптимуму. Якщо звичайні селекційні операції не виконуються, нові генетичні значення можуть бути введені в результаті нової операції. PSO занадто сильно залежить від використання параметрів. Параметри керуватимуть відходом від глобального оптимуму. Крім того, надто багато параметрів було використано для оновлення розташування частинок, що збільшило час, необхідний для отримання результату. Слід зменшити використання параметрів, щоб уникнути впливу параметрів на результат.

Як висновок, наразі немає жодних дослідницьких робіт щодо впровадження GA та DE для вивчення та наближення до глобального оптимуму, що є хорошою характеристикою PSO. Крім оновлення локальної хромосоми, вони можуть наближатися до глобального оптимуму, який є ближчим до оптимального результату. Таким чином, пропозиції можна застосувати до нових гібридних методів, перевіряючи їх продуктивність за допомогою тих самих функцій порівняння.

2.1.3 Висновки після порівняння продуктивності трьох алгоритмів

Доведено, що GA працює краще порівняно з DE та PSO. Він може отримати найбільшу кількість найкращих мінімальних придатностей швидше, ніж обидва методи. Однак методи роботи GA все ще відіграють важливу роль, що може вплинути на ефективність GA. Хоча DE швидше, ніж PSO, але на основі мінімальної та середньої придатності PSO має кращу продуктивність, ніж DE. Операція по оновленню DE-хромосом і PSO-частинок є однією з проблем, яка впливає на продуктивність обох методів. Хоча були введені нові параметри, але для отримання результату потрібно було більше часу. Крім того, недоліками обох методів є дві

операції DE на основі одного параметра та використання занадто великої кількості параметрів для PSO.

Для майбутньої роботи, як пропонують автори [13] щодо покращення продуктивності GA, можна протестувати інші методи роботи GA і, можливо, можна отримати кращий результат. Крім того, один і той самий набір даних можна використовувати для тестування різними методами, і можна отримати найбільш оптимальні значення. Це також може підтвердити ефективність методів і технік.

Для покращення загального результату можна комбінувати різні методи. Крім того, до методів можна застосувати нову операцію або вдосконалення, що може покращити результат. Крім того, для кожного методу можна реалізувати гібрид між GA, DE або PSO. Метою цієї гібридизації є подальше вдосконалення дослідницьких і експлуатаційних можливостей пошуку за допомогою простих понять. Гібрид використовує комбіновану та неперервну схему розподілу оптимізації мурашиної колонії, щоб спеціально допомогти генетичному алгоритму в аспекті пошуку. Запропонований метод перевірено при розв’язанні тестових функцій і відомих задач інженерного проектування. Він демонструє чудові можливості глобального пошуку навіть за наявності нелінійності, мультимодальності та обмежень, включаючи велику кількість вимірів, а також великі області пошуку. На відміну від стандартного PSO, PSO-GA більш надійний у наданні кращої якості рішень за розумний час обчислення, оскільки гібридна стратегія дозволяє уникнути передчасної конвергенції процесу пошуку, долокального оптимуму і забезпечує краще дослідження процесу пошуку. Можливо, мутацію DE та кросовер можна розділити, а продуктивність можна покращити. Можливо, поточну концепцію розташування частинок для PSO можна видалити, щоб уникнути віддалення частинок від фактичного оптимуму. Ця ідея може скоротити час на пошук оптимального результату, та дати більш точний результат. Виконуючи такого типу комбінації алгоритмів оптимізації, ми беремо вже перевірені та ефективні етапи євристичних алгоритмів. Але також такі комбінації можуть давати більш точний результат, який супроводжується більшим часом розрахунків.

2.2 Вступ до мета евристичних алгоритмів

У цьому розділі обговорюється аналіз останніх досягнень генетичних алгоритмів. Для аналізу відібрано генетичні алгоритми, які викликають значний інтерес дослідницького співтовариства. Цей огляд допоможе новим і вимогливим дослідникам надати ширше бачення генетичних алгоритмів. Представлено відомі алгоритми та їх реалізацію з їх перевагами та недоліками. Охоплено різні галузі досліджень, пов'язані з генетичними алгоритмами. Обговорюються майбутні напрямки досліджень у галузі генетичних операторів, функції пристосованості та гібридні алгоритми. Цей структурований огляд буде корисним для наукових досліджень і викладання в аспірантурі.

В останні роки метаевристичні алгоритми використовуються для вирішення реальних складних проблем, що виникають у різних сферах, таких як економіка, інженерія, політика, управління та інженерія. Інтенсифікація та диверсифікація є ключовими елементами метаевристичного алгоритму. Належний баланс між цими елементами необхідний для ефективного вирішення проблеми реального життя.

Більшість метаевристичних алгоритмів натхненно процесом біологічної еволюції, поведінкою рою та законами фізики. Ці алгоритми загалом класифікуються на дві категорії, а саме: одиночне рішення та метаевристичний алгоритм на основі сукупності. Метаевристичні алгоритми на основі одного рішення використовують єдине рішення-кандидат і покращують це рішення за допомогою локального пошуку. Однак рішення, отримане за допомогою метаевристик на основі одного рішення, може застрягти в локальних оптимумах.

Добре відомі метаевристики на основі єдиного рішення – імітований відпал, пошук табу, мікроканонічний відпал і керований локальний пошук. Метаевристика на основі популяції використовує кілька варіантів рішень під час процесу пошуку. Ці метаевристики зберігають різноманітність популяції та уникають того, щоб рішення застрягли в локальних оптимумах.

Деякі з добре відомих популяційних метаевристичних алгоритмів — це:

- генетичний алгоритм;
- оптимізація рою частинок;
- оптимізація колонії мурашок;
- оптимізація плямистої гієни;
- оптимізація імператорського пінгвіна;
- оптимізація чайки.

Серед метаевристичних алгоритмів генетичний алгоритм є добре відомим алгоритмом, який базується на процесі біологічної еволюції. Він імітує дарвінівську теорію виживання найпристосованіших у природі. GA був запропонований Джоном Холандом у 1992 році. Основними елементами GA є представлення хромосом, вибір придатності та біологічні оператори. Голланд також представив новий елемент, а саме інверсію, яка зазвичай використовується в реалізаціях GA.

Як правило, хромосоми мають двійковий рядковий формат. У хромосомах кожен локус (конкретне положення на хромосомі) має дві можливі алелі (варіантні форми генів) - 0 і 1. Хромосоми розглядаються як точки в просторі рішення. Вони обробляються за допомогою генетичних операторів шляхом ітеративної заміни популяції.

Функція відповідності використовується для визначення значення для всіх хромосом у популяції. Біологічно інспіровані оператори - це відбір, мутація та кросинговер. При відборі хромосоми відбираються на основі їх придатності для подальшої обробки. В операторі кросинговеру вибирається випадковий локус, який змінює підпослідовності між хромосомами для створення нащадків. Під час мутації деякі біти хромосом будуть перевернуті випадковим чином на основі ймовірності.

2.3 Генетичний алгоритм

2.3.1 Вступ до генетичного алгоритму

Генетичний алгоритм – це адаптивний евристичний алгоритм пошуку рішення. Він використовується для вирішення проблем оптимізації в машинному навчанні. Це один із важливих алгоритмів, оскільки він допомагає вирішувати складні задачі, робота над якими потребує багато часу. Генетичні алгоритми широко використовуються в різних додатках у реальному світі, наприклад, проектування електронних схем, обробка зображень і малювання за допомогою штучного інтелекту.

У цьому розділі буде пояснено що таке генетичний алгоритм, включаючи основні терміни, які використовуються в ньому, як він працює, переваги та обмеження тощо.

Для повного розуміння алгоритму, треба переглянути основні терміни які входять до теоретичної частини:

- популяція це підмножина всіх можливих або ймовірних рішень, які можуть вирішити дану проблему;
- хромосоми є одним із рішень у популяції для даної проблеми, а сукупність генів створює хромосому;
- ген хромосома розділена на інший ген або є елементом хромосоми;
- алель це значення, яке надається гену в конкретній хромосомі;
- функція доречності - використовується для визначення індивідуального рівня підготовки індивіда в популяції. Це означає здатність конкурувати з іншими індивідами.

Генетичний алгоритм це евристичний алгоритм для вирішення задач оптимізації. Це підмножина еволюційних алгоритмів, які використовуються в обчисленнях. Генетичний алгоритм використовує концепції генетичного та природного відбору для вирішення задач за допомогою методів оптимізації.

Методи оптимізації — це методи, які використовуються для пошуку найкращого рішення з усіх можливих та доступних за наявних обмежень. Генетичний алгоритм є одним із таких алгоритмів оптимізації, побудованих на основі природного еволюційного процесу нашої природи. Тут використовується ідея природного відбору та генетичної спадковості. На відміну від інших алгоритмів, він використовує керований випадковий пошук, тобто пошук оптимального рішення, починаючи з випадкової функції початкової вартості, а потім шукаючи лише в просторі з найменшою вартістю (у керованому напрямку). Підходить, коли ви працюєте з величезними та складними наборами даних.

2.3.2 Схеми кодування у генетичному алгоритмі

Для більшості обчислювальних задач важливу роль відіграє схема кодування (тобто перетворення в певну форму). Надана інформація про геном має бути закодована в певному бітовому рядку. Схеми кодування розрізняються відповідно до предметної області. Добре відомі схеми кодування: двійкова, вісімкова, шістнадцяткова, перестановка, на основі значень і дерево.

Двійкове кодування є загальновживаною схемою кодування. Кожен ген або хромосома представлена у вигляді рядка з 1 або 0.

У двійковому кодуванні кожен біт представляє характеристики рішення. Він забезпечує швидшу реалізацію операторів кросинговеру та мутації. Однак для перетворення в двійкову форму потрібні додаткові зусилля, і точність алгоритму залежить від двійкового перетворення. Двійкова схема кодування не підходить для деяких проблем інженерного проектування через епістаз і природне представлення.

У вісімковій схемі кодування ген або хромосома представлена у вигляді вісімкових чисел (0–7). У шістнадцятковій схемі кодування ген або хромосома представлені у вигляді шістнадцяткових чисел (0–9, AF). Схема перестановочного кодування зазвичай використовується в задачах упорядкування. У цій схемі кодування ген або хромосома представлена рядком чисел, що представляє позицію в послідовності.

У схемі кодування значень ген або хромосома представлена за допомогою рядка деяких значень. Ці значення можуть бути дійсними, цілими числами або символами. Ця схема кодування може бути корисною для вирішення проблем за допомогою GA, у яких використовуються складніші значення. Оскільки двійкове кодування може дати збій у таких проблемах. В основному двійкова система використовується в нейронних мережах для знаходження оптимальних ваг.

У кодуванні дерева ген або хромосома представлена деревом функцій або команд. Ці функції та команди можна віднести до будь-якої мови програмування. Це дуже схоже на представлення процесу архівування у форматі дерева.

2.3.3 Як працює генетичний алгоритм

Це алгоритм пошуку на основі популяції, який використовує концепцію виживання найбільш пристосованого. Нові популяції створюються шляхом повторного використання генетичних операторів на особинах, присутніх у популяції. Представлення хромосом, відбір, кросинговер, мутація та розрахунок функції відповідності є ключовими елементами GA.

Процедура обчислення полягає в наступному. Популяція (Y) з n хромосомами ініціалізується випадковим чином. Обчислюється придатність кожної хромосоми в Y популяції. Дві хромосоми кажуть, що C1 і C2 вибрані з популяції Y відповідно до значення придатності. Одноточковий оператор кросинговеру з імовірністю кросинговеру (C p) застосовується до C1 і C2 для створення нащадка, наприклад O.

Після цього єдиний оператор мутації застосовується до отриманого потомства (O) з імовірністю мутації (M p), щоб створити O′ . Нове потомство O′ поміщається в нову популяцію. Операції відбору, кросинговеру та мутації повторюватимуться для поточної популяції, доки нова популяція не буде завершена.

Алгоритм динамічно змінює процес пошуку через ймовірності кросинговеру та мутації та досягає оптимального рішення. Алгоритм може модифікувати закодовані

гени. А також може оцінити кількох осіб і створити кілька оптимальних рішень. Отже, він має кращі можливості глобального пошуку.

Формула кросинговеру представлена нижче (1.1):

$$R=(G+2g\surd)/3G \tag{1.1}$$

Де g – кількість поколінь, а G – загальна кількість еволюційних поколінь, встановлених популяцією. З рівняння видно, що R динамічно змінюється і збільшується зі збільшенням числа еволюційних поколінь.

На початковій стадії розрахунків схожість між особами дуже низька. Значення R має бути низьким, щоб гарантувати, що нова популяція не зруйнує генетичну схему особин. Наприкінці еволюції подібність між особинами дуже висока, а також значення R має бути високим.

Згідно з теоремою про схему, оригінальну схему потрібно замінити модифікованою схемою. Щоб зберегти різноманітність популяції, нова схема зберігає початкову популяцію на ранній стадії еволюції. Знизу можна побачити псевдокод класичного генетичного алгоритму:

Початок

Згенерувати початкову популяцію з N хромосом Y (i=1,2,3,4…,n)

Встановити лічильник ітерації i = 0

Вирахувати значення придатності кожної хромосоми

Поки (t < MAX)

Обрати пару хромосом з першопочаткової популяції базуючись на придатності

Використати операцію кросоверу на вибраній парі з вірогідністю кросоверу

Використати мутацію з вірогідністю мутації

Замінити стару популяцію на нову згенеровану

Наростити поточну ітерацію і на 1

Кінец циклу

Повернути найкраще рішення Y

Кінец

Генетичний алгоритм працює на основі еволюційного циклу поколінь для створення правильних рішень. Ці алгоритми використовують різні операції, які або покращують, або замінюють популяцію, щоб отримати покращене рішення відповідності.

В основному він включає п’ять етапів вирішення складних задач оптимізації, які наведені нижче:

- ініціалізація;
- перевірка на придатність;
- вибір;
- помноження успішних рішень;
- припинення.

Ініціалізація популяції є першим кроком у процесі генетичного алгоритму. Ініціалізація це процес генетичного алгоритму який починається з генерації набору особин, який називається популяцією. Тут кожна особистість є рішенням даної проблеми. Особа містить або характеризується набором параметрів, які називаються генами. Гени об'єднуються в рядок і генерують хромосоми, що є вирішенням проблеми. Було помічено, що всю популяцію не слід ініціалізувати за допомогою евристики, оскільки це може призвести до того, що популяція матиме подібні рішення та дуже малу різноманітність. Було експериментально помічено, що випадкові рішення – це ті, які приводять популяцію до оптимальності. Тому за допомогою евристичної ініціалізації ми просто заповнюємо популяцію парою хороших рішень, та заповнюючи решту випадковими рішеннями,

Було також помічено, що евристична ініціалізація в деяких випадках впливає лише на початкову придатність популяції, але в кінцевому підсумку саме

різноманітність рішень призводить до оптимальності. Одним із найпопулярніших методів ініціалізації є використання випадкових двійкових рядків як на рисунку 1.1.

Рисунок 1.1 Ініціалізація за допомогою двійкових рядків

Перевірка на придатність використовується для визначення того, наскільки придатне рішення Це означає здатність індивіда конкурувати з іншими індивідами. У кожній ітерації індивідууми оцінюються на основі їх функцій придатності. Функція придатності надає оцінку кожному рішенню. Цей бал додатково визначає ймовірність бути обраним для відтворення нових рішень. Чим вищий показник придатності, тим більше шансів потрапити на відбір.

Відбір є важливим кроком у генетичних алгоритмах, який визначає, чи братиме певний рядок участь у процесі відтворення чи ні. Крок вибору іноді також відомий як оператор відтворення. Добре відомими методами відбору є колесо рулетки, ранг, турнір, больцманівська та стохастична універсальна вибірка.

Вибір за допомогою колеса рулетки відображає всі можливі рядки на колесі, які розподіляється на це колесо відповідно до їх значення придатності. Потім це колесо обертається випадковим чином для вибору конкретних рішень, які братимуть участь у формуванні наступного покоління. Однак цей підхід страждає від багатьох проблем, таких як помилки, внесені його стохастичною природою. Де Йонг і Бріндл модифікували метод вибору колеса рулетки, щоб усунути помилки, ввівши концепцію детермінізму в процедуру відбору. Вибір рангу є модифікованою формою вибору колеса рулетки. Він використовує ранги замість значення придатності. Ранги надаються їм відповідно до рівня підготовки індивідууми, щоб кожна особа отримала шанс бути обраною відповідно до своїх рангів. Метод рангового вибору зменшує шанси передчасного збіжності розв'язку до локальних мінімумів.

Методику турнірного відбору вперше запропонував Бріндл у 1983 році. Індивідууми вибираються відповідно до їхніх показників підготовки за допомогою стохастичного колеса рулетки в парах. Після відбору особини з вищим показником придатності додаються до пулу наступного покоління. У цьому методі відбору кожна особина порівнюється з усіма n-1 іншими особами, якщо вона досягає кінцевої популяції рішень.

Стохастична універсальна вибірка (SUS) є розширенням існуючого методу вибору колеса рулетки. Вона використовує випадкову початкову точку в списку індивідів із покоління та вибирає новий індивідуум через рівномірні інтервали. Це дає всім особам рівні шанси бути обраними для участі в кросовері для наступного покоління. Хоча у випадку проблеми комівояжера SUS працює добре, але зі збільшенням розміру проблеми, традиційний вибір колеса рулетки працює краще.

Вибір Больцмана базується на методах ентропії та вибірки, які використовуються в моделюванні Монте-Карло. Це допомагає у вирішенні проблеми передчасної конвергенції. Імовірність вибору найкращого рядка дуже висока, хоча він виконується за дуже короткий час. Однак є ймовірність втрати інформації. Нею можна керувати через елітарність. Елітний вибір був запропонований KD Jong у 1975 році,

для покращення продуктивності вибору колеса рулетки. Це гарантує, що елітарна особистість покоління завжди поширюється в наступному поколінні. Якщо особина з найвищим показником пристосованості не присутня в наступному поколінні після звичайної процедури відбору, то елітарна включається в наступне покоління автоматично.

Селекційна фаза передбачає відбір найкращих елементів для відтворення наступних поколінь рішень. Потім усі відібрані рішення об'єднуються в пари по дві особини для збільшення відтворення. Потім ці особини передають свої гени наступному поколінню.

Існує три типи методів вибору, а саме:

- вибір колеса рулетки;
- вибір турніру;
- відбір на основі рангу.

Після процесу відбору, помноження успішних рішень відбувається на етапі відтворення. На цьому кроці[20] генетичний алгоритм використовує два оператори варіації, які застосовуються до батьківської популяції. На рисунку 1.2 наведено два оператори, які беруть участь у фазі відтворення:

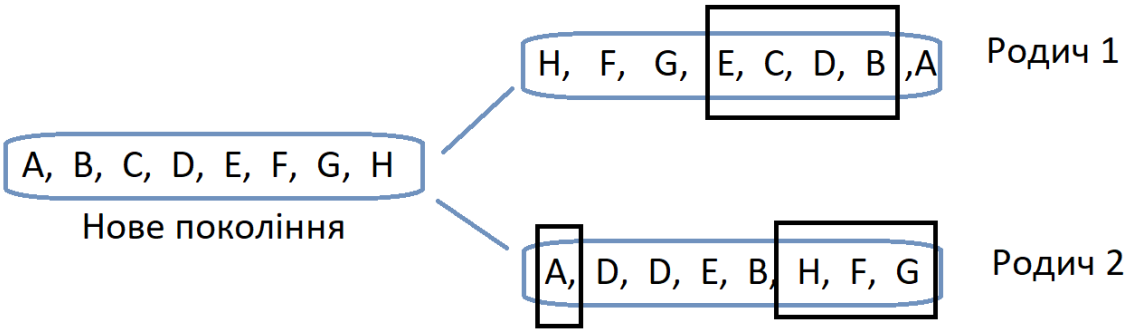


Рисунок 1.2 Фаза відтворення

Перехід до нового покоління відіграє найважливішу роль у фазі відтворення генетичного алгоритму. У цьому процесі точка переходу вибирається випадковим

чином у межах генів. Потім оператор замінює генетичну інформацію двох батьків із поточного покоління, щоб створити нову особину, яка представляє успішне рішення задачі. Гени батьків обмінюються між собою, поки не буде досягнута точка перехрещення. Потім щойно згенеровані нащадки додаються до популяції. Цей процес також називають кросовером. Доступні типи кросоверу:

- кросовер в одну точку;
- двоточковий кросовер;
- ліврея кросовера;
- кросовер успадкованих алгоритмів.

Оператор мутації — це оператор, який підтримує генетичну різноманітність від однієї популяції до наступної. Добре відомими операторами мутації є зсув, проста інверсія та скрембл-мутація. Оператор мутації зміщення (DM) зміщує підрядок окремого рішення всередині себе. Місце вибирається випадковим чином із заданого підрядка для зміщення, щоб отримане рішення було дійсним, а також мутація випадкового зміщення. Існують такі варіанти DM, як обмінна мутація та інсерційна мутація. В операторах мутації обміну та мутації вставки частина окремого рішення або обмінюється іншою частиною, або вставляється в інше місце відповідно.

Оператор простої інверсійної мутації (SIM) змінює підрядок між будь-якими двома вказаними місцями в окремому рішенні. SIM — це оператор інверсії, який перевертає випадково вибраний рядок і розміщує його у випадковому місці. Оператор scramble mutation (SM) розміщує елементи у визначеному діапазоні індивідуального рішення у випадковому порядку та перевіряє, чи покращилось значення придатності створеного рішення чи ні. Оператор мутації вставляє випадкові гени в потомство (новий виток рішення), щоб підтримувати різноманітність популяції. Це можна зробити, перевернувши деякі біти в хромосомах. Мутація допомагає вирішити проблему передчасної конвергенції та посилює диверсифікацію. Типи доступних стилів мутації:

- фліп-бітова мутація;
- мутація гауса;

- мутація обміну.

На рисунку1.3 показано процес мутації.

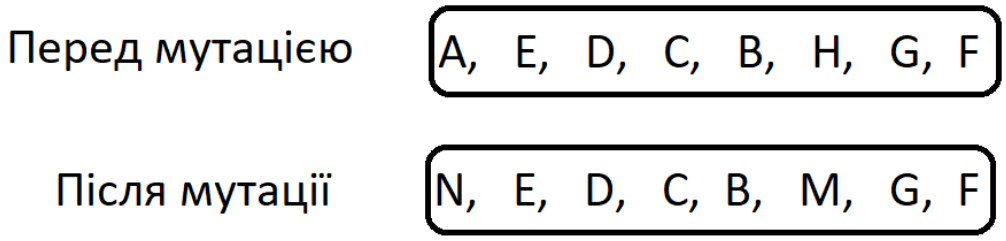


Рисунок 1.3 Процес мутації

Після фази відтворення критерії зупинки застосовуються як основа для припинення. Алгоритм припиняє роботу після досягнення порогового рішення відповідності. Він визначить остаточне рішення як найкраще.

Операція мутації є важливою в генетичному алгоритмі. Вона змінює значення деяких генів, щоб підвищити якість нових. Щоб визначити, мутований ген чи ні, використовується ймовірність мутації. У традиційному генетичному алгоритмі існує лише одне постійне значення ймовірності мутації. Таким чином, незалежно від значення придатності рішення, однакова кількість генів мутується. Оскільки ймовірність мутації є постійною, для кожної проблеми необхідно використовувати розумне значення. Якщо ймовірність велика, то багато генів будуть мутовані. Якщо занадто багато генів мутовано у високоякісному розчині, випадкові зміни можуть погіршити це рішення, порушивши занадто багато його генів. Для низькоякісного рішення мутація великої кількості генів корисна для підвищення його якості, оскільки це змінює багато поганих генів. З іншого боку, невелика ймовірність мутації призводить до мутації лише кількох генів. Для високоякісного рішення випадкова зміна лише деяких його генів збереже його якість високою з можливістю її підвищення. Для неякісного рішення змінюється невелика кількість його генів, тому якість все одно може бути низькою.

2.4 Висновки щодо використання генетичного алгоритму

Переваги генетичного алгоритму:

- паралельні можливості генетичних алгоритмів є найкращими;
- він допомагає в оптимізації різних проблем, таких як дискретні функції, багатоцільові задачі та неперервні функції;
- він забезпечує вирішення проблеми, яке з часом покращується;
- генетичний алгоритм не потребує похідної інформації.

Обмеження генетичних алгоритмів:

- генетичні алгоритми не є ефективними для вирішення простих задач;
- вони не гарантують якість остаточного вирішення проблеми;
- повторне обчислення значень придатності може спричинити певні обчислювальні проблеми.

Різниця між генетичними алгоритмами та традиційними алгоритмами:

- простір пошуку - це сукупність усіх можливих рішень проблеми. у традиційному алгоритмі підтримується лише один набір рішень, тоді як у генетичному алгоритмі можна використовувати кілька наборів рішень у просторі пошуку;
- традиційні алгоритми потребують більше інформації для здійснення пошуку, тоді як генетичні алгоритми потребують лише однієї цільової функції для обчислення придатності рішення;
- традиційні алгоритми не можуть працювати паралельно, тоді як генетичні алгоритми можуть працювати паралельно (обчислення придатності рішення є незалежним);
- однією великою відмінністю генетичних алгоритмів є те, що успадковані алгоритми працюють не безпосередньо з результатами, а з їхніми представленнями (або візуалізаціями), які часто називають хромосомами;

- одна з великих відмінностей між традиційним алгоритмом і генетичним алгоритмом полягає в тому, що він безпосередньо не працює з потенційними рішеннями;
- традиційні алгоритми можуть зрештою генерувати лише один результат, тоді як генетичні алгоритми можуть генерувати кілька оптимальних результатів із різних поколінь;
- традиційні алгоритми є детермінованими за своєю природою, тоді як генетичні алгоритми є ймовірнісними та стохастичними.

GA – це евристична техніка пошуку рішень, натхненна природною еволюцією. Він є надійним і гнучким підходом, який можна застосувати до широкого кола проблем навчання та оптимізації. Вони особливо підходять для вирішення проблем, коли традиційні методи оптимізації ламаються або через нерегулярну структуру простору пошуку (наприклад, відсутність інформації про градієнт), або через те, що пошук стає обчислювально складним. Прості реалізації GA можна зробити з невеликими попередніми зусиллями. У міру накопичення досвіду та виявлення «правильних» проблем, які мають вирішувати GA, мають з’явитися більш якісніші програми для обчислення.

У таких обчисленнях є сильна паралель зі спробами математичних імунологів змоделювати імунні системи. GA є значно спрощеними абстракціями природних процесів, які вони представляють. Однак комп’ютерні вчені, які намагаються зрозуміти та використовувати ці підходи, зараз стикаються з проблемами, дуже схожими на ті, з якими стикаються біологи, щоб зрозуміти, як складні системи працюють у різних масштабах. Хочеться сподіватися, що знання та досвід обох дослідницьких спільнот можна плідно поєднати.

Успішне застосування GA у багатьох сферах свідчить про корисність GA у обробці великомасштабного проектування та проблем оптимізації. Також GA можна застосовувати для вирішення складних проблем у інших областях дослідження, таких як дизайн матеріалів, планування, аналіз зображень тощо.

3. РЕАЛІЗАЦІЯ ГЕНЕТИЧНОГО АЛГОРИТМА

3.1 Як машина буде паркуватись за допомогою генетичного алгоритму

У цьому розділі ми “навчимо” автомобіль виконувати самостійне паркування за допомогою генетичного алгоритму . У 1-му поколінні автомобілі матимуть випадковий геном і поводитимуться дуже хаотично, не маючи можливості точно паркуватись як на рис. 3.1

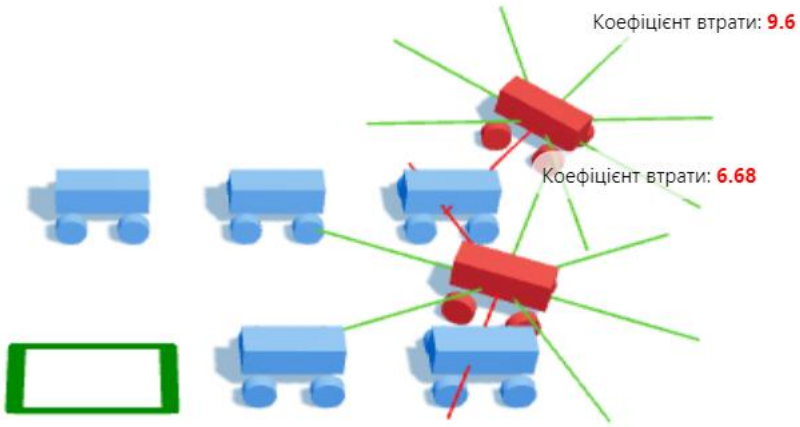


Рисунок 3.1 – Хаотичне паркування у першому поколінні

У 55-му поколінні автомобілі потроху почнуть вчитися парковці і будуть все ближче і ближче підбиратися до місця паркування. Так, на початку машини будуть врізатися в інші машини по дорозі, і неточно будуть ставати на паркувальне місце, але для них це лише перше покоління з моменту створення світу, так що треба буде почекати і дати машинам підрости, покоління за поколінням. Ми можемо запустити процес еволюції у браузері локально. У симуляторі паркування присутні такі об’єкти:

- статичні автомобілі;
- динамічні автомобілі;
- сенсори навколо динамічних автомобілів;

- паркувальне місце.

Генетичний алгоритм для цього проекту реалізовуватимемо на TypeScript. У цьому розділі буде показано скорочений код алгоритму. Генетичний алгоритм буде використано для вирішення конкретної задачі навчання машин самостійного паркування. Однак цей розділ торкнеться тільки основ алгоритму і в жодному разі не буде повним керівництвом до його впровадження. Крок за кроком ми зведемо високорівневу задачу створення автомобіля здатного паркуватися автоматично, до простого завдання знаходження оптимальної комбінації нулів і одиниць, тобто до знаходження оптимального генному. План цієї роботи вкладено у 4 кроки:

- реалізувати можливість автомобіля бачити за допомогою сенсорів, щоб машина могла бачити перешкоди навколо, та подавати сигнали керму та двигуну щоб ці перешкоди оминути;
- додати до автомобіля кермо та двигун, щоб він міг рухатися у бік місця для паркування при цьому дозуючи швидкість, та обираючи оптимальний кут повороту керма. Ці два елементи управління повинні працювати в парі, обмінюючись сигналами між собою;
- дамо автомобілю інтелект, щоб машина могла контролювати рух на підставі того, що машина бачить. Тут допоможуть сенсори які точно будуть знати відстань до перешкоди. Мозок буде простою функцією, яка отримує як параметр данні із сенсорів, які в свою чергу є просто масивом чисел;
- останніх етап це ітеративний розвиток цього мозку, щоб він міг ініціювати правильні рухи на підставі сигналів сенсорів. Тут ми застосуємо генетичний алгоритм. Покоління за поколінням функція мозку буде вчитися як наближати авто до місця паркування. Враховуючи в кожній ітерації, данні з попередньої запуску автомобіля у світ, та акумулюючи їх у локальне сховище користувача. Щоб була можливість у будь-який момент, зберегти результати навчання, тобто геном у вигляді набору нулів і одиниць.

3.2 Програмування органів управління та мозку автомобіля

3.2.1 Сенсори відстані автомобіля

Перш ніж наша машина навчиться самостійно паркуватися, використовуючи свої набуті знання, вона має бачити довкілля. Дамо машині додаткой орган чуття у вигляді датчиків відстані:

- кожен датчик повідомляє останні дані про перешкоди, які він бачить у блок управління автомобіля;
- коли датчик не бачить жодних перешкод, він повідомляє значення “Ok”. Якщо значення датчика невелике, але не нульове, це означатиме, що перешкода близька і промінь датчика змінює колір на червоний;
- кожен датчик може виявляти перешкоду на відстані від нуля до сіми метрів.

На рис. 3.2 бачимо 8 датчиків, це поширене число сенсорів для тиких завдань воно повністю покриває потреби в орієнтації автомобіля і цих даних достатньо щоб формувати якісний геном швидко.

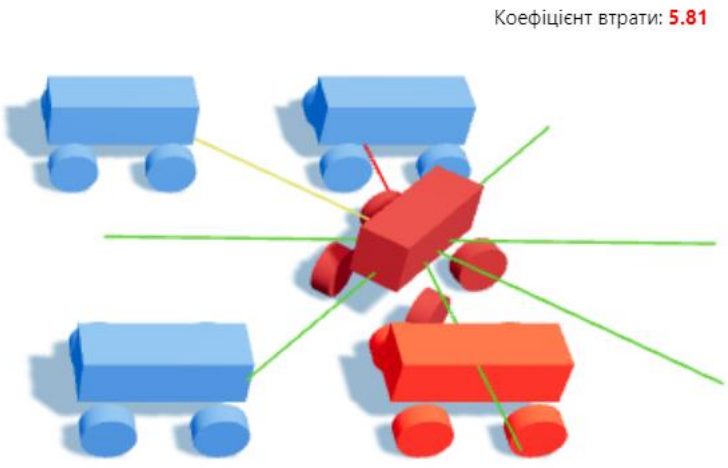


Рисунок 3.2 – Вісім датчиків автомобіля у дії

Якщо порівняти цифрову модель самокерованого автомобіля зі справжньою, то у справжній використовуються дещо інакший підхід. Як і люди, справжні безпілотні автомобілі повинні відчувати оточення, щоб безпечно рухатися. Люди використовують такі органи чуття, як слух, зір, смак, нюх і дотик, щоб взаємодіяти з навколишнім середовищем. Розробники технологій для автономних автомобілів забезпечують безпілотні автомобілі високотехнологічними сенсорними системами для аналогічного відчуття. Серед сенсорів ми вже знаємо:

- лідар;
- радар;
- сонар;
- зйомка зображень за допомогою камер;
- розпізнавання рухів за допомогою інерціальних навігаційних систем;
- відстеження позицій за допомогою глобальної системи позиціонування.

Доволі широкого використання зазнав лідар (виявлення світла та визначення дальності), також відоме як 3D-лазерне сканування, — це інструмент, який використовують безпілотні автомобілі для сканування середовища за допомогою лазерів. Типовий лідарний датчик направляє тисячі променів інфрачервоного лазерного світла в навколишнє середовище та чекає, поки промені відіб’ються від об’єктів навколишнього середовища. Багато імпульсів створюють хмари точок (набори точок, що представляють тривимірні форми в просторі) світла. Системи лідарів вимірюють час, необхідний для випромінювання лазерного сигналу та сприйняття того самого світлового променя, відбитого від фізичної поверхні, на фотодетектори. Лідар використовує швидкість світла для обчислення відстані до об’єктів. Чим довше лідарний фотодетектор приймає зворотний світловий сигнал, тим далі знаходиться об’єкт. Лідарні системи дозволяють безпілотним автомобілям

виявляти маленькі об’єкти з високою точністю. Однак лідар часто ненадійний вночі або в негоду.

Радар (радіовиявлення та визначення дальності) корисний у багатьох контекстах, таких як прогноз погоди, астрономія, зв’язок, океанська навігація, військові дії та автономне водіння. Автономні автомобілі можуть випромінювати радіохвилі у різних напрямках за допомогою радарних передавачів. Відбиті хвилі, які повертаються до радіолокаційного приймача автомобіля, допомагають автомобілю отримувати інформацію про об’єкти навколишнього середовища, як-от кути, діапазони та швидкості об’єктів. Радар зазвичай добре працює на великих відстанях і за більшості погодних умов. Однак це не дуже корисно для ідентифікації об’єктів і він може хибно ідентифікувати об’єкти. Радар, як і лідар, — це технологія, за допомогою якої безпілотні автомобілі можуть визначати місцезнаходження, кут і швидкість об’єктів. Радар старіший за лідар — але він має кілька переваг перед лідаром і технологією оптичного бачення, завдяки яким його варто використовувати в автономних транспортних засобах.

Найважливіше те, що радіохвилі можуть виявляти об’єкти навіть вночі або в хмарних умовах або в умовах слабкого освітлення. На відміну від систем оптичного бачення, радар також може дозволити безпілотному транспортному засобу «бачити» в темряві, навіть якщо його фари не працюють. Будучи старішим, радар також зазвичай дешевший у використанні, ніж лідар. Оскільки для роботи безпілотних автомобілів потрібна велика кількість датчиків, різниця у вартості між радаром і лідаром може істотно вплинути на кінцеву ціну безпілотного автомобіля.

Безпілотні автомобілі можуть використовувати ехолот (звукова навігація та визначення дальності) для виявлення об’єктів і зв’язку з ними, а також для навігації. Сонар може бути пасивним і активним. Пасивні гідролокаційні системи пасивно слухають звуки, які видають сусідні об’єкти. Активні гідролокаційні системи випромінюють звукові імпульси та зчитують відлуння від фізичних поверхонь.

Безпілотні автомобілі можуть використовувати ехолот для виявлення великих об’єктів із твердих матеріалів (наприклад, металу, кераміки) на короткій відстані. Сонарним датчикам для роботи не потрібне світло. Однак датчики гідролокатора обмежені швидкістю звуку (яка менша за швидкість світла) і іноді помилково виявляють неіснуючі об’єкти.

Автономні транспортні засоби можуть візуалізувати своє оточення за допомогою зображень цифрової камери високої роздільної здатності. Автомобілі можуть використовувати зображення камери, щоб «бачити» та інтерпретувати деталі навколишнього середовища (наприклад, знаки, світлофори, тварин) у спосіб, який наближається до людського зору (він же комп’ютерний зір). Автомобілі можуть використовувати багато типів вхідних даних для комп’ютерного зору.

Приклади:

- багатовимірні дані з пристроїв 3d-сканування;
- відео сегменти;
- зображення камери, зроблені під різними кутами огляду.

Автомобілі можуть розпізнавати об’єкти, керувати рухом автомобіля та моделювати 3D-сцени за допомогою даних зображень. Як і інші сенсорні системи, камери мають сильні сторони та обмеження. Камери пропонують переваги, пов’язані із зображеннями з високою роздільною здатністю, але не працюють добре за будь-якої погоди. Крім того, камери знімають лише видимі візуальні дані.

Інерціальні навігаційні системи, такі як інерціальні вимірювальні пристрої (IMU) (наприклад, акселерометри, гіроскопи), виявляють фізичні рухи автомобіля. Ці навігаційні пристрої допомагають самокерованим автомобілям стабілізуватися, а також допомагають автомобілям визначити, чи потрібно їм вживати будь-яких

захисних дій (наприклад, розгорнути подушку безпеки, запобігти перекиданню автомобіля).

Наприклад США володіють 24-супутниковою радіонавігаційною системою під назвою Глобальна система позиціонування (GPS). Користувачі з GPS-приймачем можуть отримати інформацію про геолокацію та час. Безпілотні автомобілі можуть використовувати GPS для геолокації з числовими координатами (наприклад, широтою, довготою), що представляють їх фізичне місцезнаходження в просторі. Вони також можуть здійснювати навігацію, поєднуючи GPS-координати в реальному часі з іншими цифровими картографічними даними (наприклад, через Google Maps). Дані GPS часто змінюються в радіусі п’яти метрів. Щоб компенсувати неточні дані GPS, безпілотні автомобілі можуть використовувати унікальні методи обробки даних, наприклад фільтрацію частинок зображення, щоб підвищити точність визначення місцезнаходження.

Безпілотні автомобілі зазвичай мають багато датчиків із дубльованими функціями. Це робиться для того, щоб у них була резервна система датчиків (якщо один датчик вийде з ладу, інший запрацює) і вони можуть скористатися перевагами різних типів датчиків. Розробники автономних транспортних засобів використовують новітні методи обробки даних, такі як злиття датчиків , для обробки інформації з кількох датчиків даних одночасно в режимі реального часу. Об’єднання датчиків може покращити способи, як безпілотні автомобілі інтерпретують і реагують на зміни навколишнього середовища, а отже, може зробити автомобілі безпечнішими.

Будучи технологією штучного інтелекту, безпілотні автомобілі працюють як люди, щоб дістатися з пункту А в пункт Б. Таким чином, як і люди, автономні транспортні засоби використовують базові навігаційні навички:

- складання та читання карт. Автомобілі, що керують самі собою, поєднують інформацію зі своїх сенсорних систем з іншими даними (наприклад, цифровими картами), щоб створювати та читати карти свого середовища;

- планування шляху. Автономні транспортні засоби зі штучним інтелектом використовують свої сенсорні системи для планування маршрутів через своє віртуальне середовище;
- уникнення перешкод. Безпілотні автомобілі використовують свої сенсорні системи в режимі реального часу для безпечної навігації. Щоб керувати собою, вони повинні точно виявляти, інтерпретувати та реагувати на сигнали навколишнього середовища, щоб уникати таких перешкод, як пішоходи, велосипедисти, будівлі та інші автомобілі.

3.2.2 Розробка логіки двигуна та керма

Щоб рухатися, машині потрібні двигун та кермо. Дамо машині два елементи руху:

- кермо - дозволяє машині повертати вліво , вправо або їхати прямо, тобто мати нейтральне положення;
- двигун - дозволяє машині рухатися назад , вперед або залишатися на місці тобто мати нейтральну передачу.

У нашому випадку елементи керування є приймачами сигналів, які від інтелекту будуть приходити окремо до керма то двигуна. Залежно від значення сигналу мозку, елементи керування діють по-різному. Ми розглянемо інтелектуальну частину нижче, а поки припустимо, що наш мозок може посилати лише 3 можливі сигнали кожному елементу керування: +1, -1 або нуль.

Наприклад, мозок може надіслати сигнал зі значенням -1 у двигун, і він почне рух назад. Сигнал нуль двигуну залишить машину у нейтральній позиції. У той же час, якщо мозок надішле сигнал 1 у кермове управління, воно поверне машину вправо і так далі з іншими варіаціями сигналів

Що до порівняння з реальними(не цифровими) автомобілями то кермо на автоматизованих автомобілях обертається так само, як інші роботизовані суглоби.

блоку обробки інформації (який об’єднує дані з різних датчиків) для обертання на певну відстань рульової рейки, таким чином повертаючи колеса автомобіля, так само, як якщо б людина повертала кермо. У цифрових системах зазвичай існує ланцюг імпульсів, який надходить до розподільної системи, яка, у свою чергу, посилає кожен імпульс у ланці до конкретного двигуна, для якого він призначений (рульове керування, гальмо, дросель). Ширина цього імпульсу визначає, чи змінює положення дросельної заслінки двигун і наскільки сильно. Зазвичай існує номінальна ширина імпульсу, яка встановлює шарнір/кермо в нейтральне положення, і якщо ширина імпульсу змінюється, то комп’ютер аналізує те, який тип оберту потрібно зробити, та наскільки сильно. Датчики на зовнішній стороні автомобіля зчитують, наскільки далеко він знаходиться від ліній та іншого автомобіля, регулюючи швидкість і відповідно змінюючи напрямок коліс за допомогою електричних імпульсів, які надсилаються від датчиків.

Іноді виникає питання що до стабілізації керма на автономних автомобілях, і перечитуючи відповіді експертів, було зроблено висновок. Що практично кожен автомобіль у світі має підвіску, сконструйовану таким чином, що машина стабілізується, щоб їхати прямо, коли ви відпускаєте кермо. Однак, якщо автомобіль не котиться вперед, цей ефект не відбувається. І треба зауважити, що типова конструкція підвіски, яка забезпечує самостабілізацію під час руху вперед, діє з точністю до навпаки, коли автомобіль їде заднім ходом. З повернутим вбік кермом автомобіль буде тягнути і намагатися зробити поворот ще сильніше, якщо ви відпустите кермо.

3.2.3 Даємо машині інтелект

Справжні безпілотні автомобілі стали можливими насамперед завдяки комп’ютерному зору та глибокому навчанню. Вони використовують камери високої роздільної здатності та лідари, які виявляють, що відбувається в безпосередньому оточенні автомобіля. Завдяки цьому системи автомобіля можуть реагувати на можливі

перешкоди та уникати аварій. Звичайно, цього автомобілям недостатньо. Треба також навчити системи автомобіля їздити за правилами дорожнього руху. І тут у гру вступає машинне навчання, підкріплене глибоким навчанням

Глибоке навчання є однією з найпередовіших технологій ШІ, яка працює подібно до людського мозку. Кожна частина даних (щодо безпілотних автомобілів, ми говоримо про дані, отримані датчиками транспортного засобу) проходить через багаторівневу нейронну мережу, що дає змогу аналізувати зображення набагато більш комплексним способом. Це рішення дозволяє автовиробникам досягти значно вищого рівня складності та точності. По суті, безпілотні автомобілі дійсно розумні та можуть працювати навіть у містах із заторами. Також треба враховувати що автономні автомобілі повинні навчитися розпізнавати незліченну кількість об’єктів на шляху транспортного засобу, від гілок і сміття до тварин і людей.

Інші виклики на дорозі – це тунелі, які заважають роботі глобальної системи позиціонування (GPS), будівельні проекти, які спричиняють зміну смуги руху, або складні рішення, наприклад, де зупинитися, щоб пропустити машини екстреної допомоги. Системи повинні миттєво приймати рішення про те, коли уповільнити, відхилити або продовжити нормальне прискорення. Це постійний виклик для розробників, і є повідомлення про те, що безпілотні автомобілі вагаються та повертають без потреби, коли на дорозі чи поблизу неї виявляються предмети. З аваріями також виникає питання відповідальності, і законодавці ще не визначили, хто несе відповідальність, якщо автономний автомобіль потрапив у аварію. Існують також серйозні побоювання, що програмне забезпечення, яке використовується для роботи автономних транспортних засобів, може бути зламано, і автомобільні компанії працюють над усуненням ризиків кібербезпеки. Але в нашій роботі ці виклики можна опустити, тому що ми працюємо в доволі спрощеному світі, де є лише декілька машин і одне паркувальне місце.

На даний момент наша машина може рухатися і орієнтуватися у просторі але вона не має координатора цих дій, який перетворював би сигнали від сенсорів у

правильні рухи двигуна та керма. Нам потрібно дати машині інтелект. Як вхідні дані від датчиків мозок отримуватиме 10 чисел з плаваючою комою кожне з яких знаходиться в діапазоні [-1..5].

Кожну 1/10 секунди мозок повинен видавати на виході два цілих числа:

- перше число це сигнал для двигуна;
- друге число це сигнал для керма.

Враховуючи згадані вище вхідний та вихідний сигнали мозку, ми можемо сказати, що мозок – це доволі проста функція. Щоб зробити машину розумнішою, а її рухи - складнішими, ми могли б використовувати багатошаровий перцептрон . Назва трохи лякає, але це проста нейронна мережа з базовою архітектурою її можна сприймати як велику формулу з безліччю параметрів чи коефіцієнтів. Ця нейронна мережа може виконати задачу по розпізнаванню намальованих чисел. Тим не менш, щоб уникнути введення абсолютно нової концепції нейронних мереж у цій роботі, ми скористаємося набагато більш простим підходом і будемо використовувати два лінійні багаточлени з декількома змінними. Щоб бути більш точним, кожен багаточлен буде мати рівно 10 змінних.

Які будуть виглядати приблизно так:

wheelInput = mindToAction (

*(i1 * i1) + (i2 * i2) + ... + (i9 * i9) + i10)*

engineInput = mindToAction (

*(j1 * j1) + (j2 * j2) + ... + (j9 * j9) + j10)*

c1, c2, c3, c4, c5, c6, c7, c8, c9, c10 – вхідні данні з сенсорів

Де:

- {i1, i2, ..., i10} - коефіцієнти полінома рульового управління. Ці коефіцієнти автомобіль повинен буде знайти та запам’ятати як один виток свого навчання. Після закінчення навчання вони будуть статичними.
- {j1, j2, ..., j10} - коефіцієнти полінома двигуна. Ці коефіцієнти автомобіль повинен буде знайти та вивчити. Після закінчення навчання вони будуть

статичними та збережуться як результат для майбутніх поколінь. Ми матимемо не мало копій таких даних, тому що процес еволюції потребує багато доволі подібних ітерацій

- 10 змінних отриманих з сенсорів, які є значеннями датчиків і два останніх є показниками положення кузова автомобіля по вертикалі і горизонталі. Вони будуть змінюватись динамічно. Щоб ми могли змогу вчасно приймати рішення, та коректувати рух автомобіля поступово. Чим частіше будуть надходити дані з датчиків, тим точніше будуть рухи автомобіля. Але у цій роботі буде використана частота сигналів він 200 до 300 мілісекунд, щоб досягти оптимальної швидкості навчання. Також щоб ще пришвидшити еволюцію, можна скоротити кількість змінних до шести наприклад, тоді мозок буде швидше віддавати сигнали органам управління, але для навчання буде потрібно більше поколінь. Тобто зміна у кількості даних які надходять до мозку, прямо-пропорціонально впливають на швидкість та якість навчання геному автомобіля. У реальних самокерованих автомобілях цей коефіцієнт прямо-пропорційності зовсім інший, тому що використовуються різні комбінації сенсорів.

Ціною використання простішої функції для інтелекту стане те, що автомобіль не зможе навчитися деяким складним рухам, а також не зможе добре узагальнювати та адаптуватися до невідомого оточення. Але для нашого конкретного парко місця, та для демонстрації роботи генетичного алгоритму цього має вистачити повністю. Ми можемо реалізувати універсальну поліноміальну функцію таким чином:

```
function polynomial(vars: number[], coeff: number[]): number {
  // Розраховує значення лінійного поліному базуючись на коефіцієнтах
 let finalSum = 0;
 coeff.forEach((c: number, index: number) => {
  if (index < vars.length) {
   finalSum += vars[index] * c;
```

```
    } else {
      finalSum += c
    }
  });
  return finalSum;
};
```

В даній функції ми перемножмо значення змінних та коефіцієнтів лінійно, проходячи по всьому масиву щоб вирахувати лінійний поліном. Більш детальніше визначення виглядає наступним чином. лінійний поліном — це різновид полінома, де найвищий ступінь змінної дорівнює 1. Іншими словами, старший показник степеня змінної дорівнює 1. Поліноми — це алгебраїчні вирази, у яких змінні мають цілі невід'ємні степені. Вираз складається з одного або кількох членів, таких як змінна, константа та змінна з відмінним від нуля коефіцієнтом. Лінійні поліноми є найпростішою формою поліномів. Лінійний поліном визначається як будь-який поліном, виражений у формі рівняння $p(x) = ax + b$, де a і b — дійсні числа , а $a \neq 0$. У лінійному поліномі ступінь змінної дорівнює 1, тобто старший показник степеня змінної дорівнює одиниці. Лінійний поліном від однієї змінної може мати не більше двох доданків. Обмеження, що а не повинно дорівнювати 0, є обов'язковим, оскільки якщо а дорівнює 0, то це стає постійним поліномом.

Поліном класифікується на 2 різні типи поліномів, на основі ступеня полінома, тобто ступеня провідного члена або найвищого ступеня змінної.

Мозок автомобіля в цьому випадку складатиметься з двох багаточленів і виглядатиме так:

```
let wheelInput = mindToAction(
    polynomial(sensors, wheelCoef));
let engineinput = mindToAction(
    polynomial(sensors, engineCoef));
```

Результатом функції polynomial() є число з плаваючою комою. Функція

mindToAction() повинна перетворити широкий діапазон чисел з плаваючою комою в три конкретні цілих числа, і вона зробить це в декілька етапів:

- перетворення числа з плаваючою комою в одне з трьох цілих значень -1, 0, або 1. Наприклад, число з плаваючою комою, близькі до 0, будуть перетворені на -1, числа з плаваючою комою, близькі до 0.5, будуть перетворені на 0, а числа з плаваючою комою, близькі до 1, будуть перетворені на +1. Це дасть змогу однозначно приймати рішення по сигналам руху, тому що ми не можемо прийняти рішення базуючись на нецілих числах. Цей підхід є оптимальним з точки зору розрахунків, тому що ми оперуємо вже відомими функціями для перетворення чисел;
- перетворення числа з плаваючою точкою широкого діапазону, на число з плаваючою точкою обмеженого діапазону. Усі ці перетворення потрібні для уніфікації результатів та їх підготовки до передачі в органи управління та двигун. Щоб виконати першу частину перетворення, нам потрібно ввести функцію сигмоїду , яка реалізує наступну формулу як на рис. 3.3

$$f(x) = \frac{1}{1 + e^{-x}}$$

Рисунок 3.3 – Формула сигмоїди

Сигмоїдні функції стали популярними в глибокому навчанні , оскільки їх можна використовувати як функцію активації в штучній нейронній мережі . Їх надихнув потенціал активації в біологічних нейронних мережах.

Сигмоїдні функції також корисні для багатьох програм машинного навчання, де дійсне число потрібно перетворити на ймовірність. Сигмоїдна функція, розміщена як останній рівень моделі машинного навчання, може служити для перетворення

вихідних даних моделі в оцінку ймовірності, з якою легше працювати та інтерпретувати.

Також сигмоїдні функції є важливою частиною моделі логістичної регресії. Логістична регресія є модифікацією лінійної регресії для двокласової класифікації та перетворює один або більше вхідних даних із реальним значенням у ймовірність, наприклад ймовірність того, що клієнт придбає продукт. На завершальному етапі моделі логістичної регресії часто встановлюється логістична функція, яка дозволяє моделі виводити ймовірність.- Сігмоїда представлена на рис. 3.4, вона перетворює широкий діапазон чисел з плаваючою комою тобто вісь x, у числа з плаваючою комою з обмеженим діапазоном від одного до нуля. Це перетворення відбувається за один рядок коду, тому що ми вже маємо значення експоненти як константу, і весь розрахунок зводиться до базових арифметичних операцій. Ця функція ще може нам знадобитись, для нормалізації інших чисел з плаваючою комою тому ми її виділили окремо.

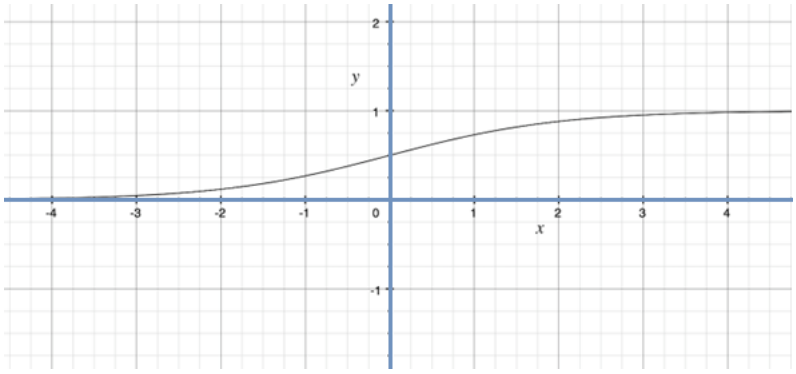


Рисунок 3.4 – Графік сигмоїди.

Ця функція це саме те, що нам потрібне для перетворення, реалізація двох згаданих вище кроків перетворення виглядатиме так:

```
const sigmoidFunction = (coeff: number): number => {
  // Вираховує значення сигмоїди для даного числа
  return 1 / (1 + Math.E ** -coeff); };
```

```
function sigmoidToAction(margin: number = 0.4, sigmoidVal: number) {
    if (sigmoidVal < (0.5 - margin)) {
        return -1;
    }
    if (sigmoidVal > (0.5 + margin)) {
        return 1;
    }
    return 0;
};
// Перетворює сигнал мозку в сигнал дії
function brainToActionInput (brainInput: number) {
    const normalizedInput = sigmoidFunction(brainInput);
    return sigmoidToAction(normalizedInput);
}
```

Головний висновок з компонентів які оцінюють навколишнє середовище, має бути наступний – коефіцієнти з двигуна та керма у вигляді масивів чисел, визначають поведінку машини. Ці числа разом утворюють унікальний геном автомобіля. Якщо піти далі на рівень генів і перевести десяткові числа геному автомобіля у двійковий формат у прості одиниці та нулі. У нашому випадку, щоб зменшити довжину геному, ми перетворимо кожен плаваючий коефіцієнт на нестандартне двійкове число. Щоб більш детально розібратись з перетвореннями ось маємо вихідний код, який виконує перетворення з двійкового формату на десятковий формат чисел з плаваючою комою. Мозку це перетворення знадобиться для декодування геному та створення сигналів на основі даних геному, ці перетворення з десяткової в двійкову і назад вимушені, тому що таким чином формується геном, це є ефективний і перевірний метод на практиці, отже код з перетвореннями:

```
function convertBitsToFraction(bits: Bit[]): number {
    const { bitsNumber, exponentBitsNumber } = precisionConstants;
```

```
    // Розрахунок значення експоненти
    const expBias = 2 ** (exponentBitsNumber - 1) - 1;
    const expBits = bits.slice(bitsNumber, bitsNumber + exponentBitsNumber);
    const expUnbiased = expBits.map(
      (currentExponent: number, currentBit: Bit, bitIndex: number) => {
        const bitPowerOfTwo = 2 ** (exponentBitsNumber - bitIndex - 1);
        return currentExponent + currentBit * bitPowerOfTwo;
      },
      0,
    );
    const exp = expUnbiased - expBias;
    // Вираховуемо значення частки.
    const frBits = bits.slice(bitsNumber + exponentBitsNumber);
    const fr = frBits.map(
      (currentFraction: number, currBit: Bit, index: number) => {
        const bitPowerOfTwo = 2 ** -(index + 1);
        return currentFraction + currBit * bitPowerOfTwo;
      },
      0,
    );
    // Визначаємо знак
    const numSign = (-1) ** bits[0];
    return numSign * (2 ** exp) * (fr+1);
  }
```

Раніше функція нашого мозку безпосередньо працювала з десятковою формою поліноміальних коефіцієнтів. Однак тепер ці коефіцієнти кодуються у двійковій формі геному. Щоб завершити набір функцій які нам потрібні для першого запуску навчання автомобіля, нам ще потрібна функція, яка буде розшифровувати наш геном наприклад

така як знизу:

```
// Конвертує геном від двійкової форми в десяткову
function genToDecimal(genesQuantity: number, gen: Gen): number[] {
   const nums: number[] = [];
   for (let i = 0; i < gen.length; i += genesQuantity) {
      const n: number = bitsToDecimal(gen.slice(i, i + genesQuantity));
      nums.push(n);
   }
   return nums;
};
```

Частину коду що робить другорядні розрахунки було опущено.

Двійкове перетворення в десяткове виконується для перетворення числа, поданого в двійковій системі числення, на його еквівалент у десятковій системі числення. Система числення — це формат для представлення чисел певним чином. Двійкова система числення використовується в комп’ютерах і електронних системах для представлення даних і складається лише з двох цифр, які є 0 і 1. Десяткова система числення є найпоширенішою системою числення в усьому світі, яка легко зрозуміла людям. Перетворення двійкової системи в десяткову можна здійснити двома методами:

- методом позиційної нотації;
- методом подвоєння.

Метод позиційного запису — це метод, у якому значення цифри в числі визначається вагою на основі її положення. Це досягається шляхом множення кожної цифри на основу (2), зведену у відповідний ступінь залежно від положення цієї цифри в числі. Сума всіх цих значень, отриманих для кожної цифри, дає еквівалент даного двійкового числа в десятковій системі. Робота з різними системами числення, дозволяє гнучко працювати з даними на різних етапах алгоритму.

3.3 Роль генетичного алгоритму у вирішені проблеми автоматизованого паркування

Отже, зараз ми підійшли до моменту, коли високорівнева проблема навчання автомобіля самостійному паркуванню зводиться до простої математичної оптимізаційної задачі пошуку оптимальної комбінації одиниць і нулів, тобто знаходження оптимального геному машини. Нам потрібен швидший алгоритм, щоб знайти оптимальне значення геному. Тут на допомогу приходить генетичний алгоритм. Можливо, ми не знайдемо найкраще значення геному, але є шанс, що ми зможемо знайти оптимальне значення. І, що важливіше, нам не потрібно довго чекати.

Генетичні алгоритми натхненні процесом природного відбору, які зазвичай використовуються для вирішення оптимізаційних завдань. Вони покладаються на біологічні оператори, такі як кросовер, мутація та відбір.

Проблема пошуку досить вдалої комбінації генів для автомобіля, виглядає як проблема оптимізації, так що є велика ймовірність, що генетичний алгоритм допоможе нам у цьому. Під оптимізацію тут підпадає наш геном який на початку є не оптимальним набором нулів і одиниць. Тут не буде докладно розписано генетичний алгоритм як це було у розділі 2, але загалом ось основні кроки, які нам потрібно буде зробити вже з практичної точки зору:

- фаза створення - перше покоління автомобілів не може з'явитися з нічого, тому ми генеруємо набір випадкових геномів які представляють собою набір двійкових масивів. Наприклад, ми можемо створити кілька сотень машин. І тут чим більша кількість машин, тобто учасників еволюції, тим вищі шанси знайти оптимальне рішення і зробити це можна швидше;
- фаза вибору - нам потрібно буде вибрати найбільш пристосованих особин із поточного покоління для подальшого об’єднання, тобто те що буде відбуватись у наступному кроці. Придатність кожного індивіда буде визначатися на основі функції придатності.

до цільового місця паркування. Чим ближче машина до місця паркування, тим вона більш придатна з точки зору популяції. Це критерій буде основним для визначення оптимальності конкретного рішення. Якщо б ми вирішували цю задачу на папері, то це б було просто порівняння чисел між собою. Які в свою чергу є результатом роботи функції-інтелекту нашого автомобіля;

- фаза відтворення - тут ідея полягає в тому, що відтворені діти-автомобілі можуть стати кращими або гіршими у паркуванні, якщо успадкують краще або гірше у батьків. А ось оцей критерій успішності можна буде визначити ітераційно у процесі паркування;
- фаза мутації - у процесі відтворення деякі гени можуть випадково мутувати, це значить що одиниці і нулі в геномі дитини можуть змінюватися. Це може призвести до більш широкого розмаїття геномів дітей та, таким чином, до більш широкого розмаїття поведінки автомобілів. У той же час, сильні мутації можуть зруйнувати здорові геноми, тому для майбутнього розвитку цієї програми, можливо є сенс розробити процес відстежування цих аномальних мутації. Це повинно зекономити процесорний час та попередні успішні ітерації навчання.

Далі нам потрібно розробити перевірку, якщо кількість поколінь не досягла межі. Наприклад, не пройшло 50 поколінь, або поки що найуспішніші індивідууми не досягли очікуваного значення функції придатності (якщо кращий автомобіль ще не наблизився до місця стоянки ближче, ніж на 1 метр). Інакше виходимо із циклу навчання автомобіля. Ця перевірка дасть можливість відсіяти непридатні рішення на ранніх стадіях, що також вплине на ефективність навчання. Давайте розвинемо мозок автомобіля за допомогою генетичного алгоритму, реалізувавши чотири кроки зверху у коді.

Метод для фази створення буде відповідальний за побудову покоління , він буде створювати масив випадкових геномів тобто популяцію, та буде приймати декілька параметрів що свідчать про конфігурацію майбутнього покоління:

- genomeLength- Визначає довжину геному кожного індивідуума в популяції

автомобілів. У нашому випадку довжина геному буде 180;

- другий параметр буде визначати розмір покоління. Цей обсяг покоління буде незмінним протягом усього еволюційного процесу, тобто константою. Це потрібно для того щоб усі покоління були однакового розміру, тоді навчальний процес автомобіля буде йти ітераційно та результати, які є набором нулів та одиниць будуть взаємозамінні. Імовірність того, що кожен ген буде дорівнювати нуль чи один є 0.5 код виглядає наступним чином.

```
// Створює випадкове покоління з нулів та одиниць
function produceGeneration(params: Generation) {
  const { genSize, genomeLength: genLength } = params;
  return new Array(genSize)
    .fill(null)
    .map(() => createGen(genLength));
function createGen(length: number): (0|1) {
  return new Array(length)
    .fill(null)
    .map(() => (Math.random() < 0.5 ? 0 : 1)); }
```

Метод для фази мутація змінюватиме деякі гени випадковим чином на основі значення вірогідності мутації.

Наприклад, якщо значення вірогідності мутації = 0.3, то ймовірність мутації кожного геному становить 30 відсотків. Скажімо, якби у нас був геном довжиною 30 генів, який виглядав би як 30 нулів, то після мутації буде ймовірність, що 3 гени будуть мутованими, і ми можемо отримати геном, який може виглядати як набір нулів з одиницею в будь-якому місці, функція знизу показує більш детально процес мутації генів.

```
// Мутує ген випадковим чином
function mutateGen(gen, mutationProbability) {
  for (let index = 0; index < gen.length; index += 1) {
```

```
    const gene = gen[index];
    const mutated = gene === 0 ? 1 : 0;
    gen[index] = Math.random() < mutationProbability ? mutated : gene;
  }
  return gen;
}
```

Мутація дає нам можливість швидше навчитись за допомогою випадковості, яка видозмінює результуючий ген, додаючи у випадкові місця один із варіантів гену. У функції зверху ми інверсійним шляхом змінюємо ген протилежний. Але тут треба підкреслити що такого роду мутації краще використовувати у парі з валідаційними функціями. Це дасть змогу уникнути аномальних мутацій, які можуть поставити під питання доцільність подальших ітерацій навчання. Під аномальною мутацією мається на увазі геном автомобіля, який має дуже високий коефіцієнт втрати або промаху у парко місце. Мутації часто використовуються у різних алгоритмах машинного навчання, щоб отримати унікальні значення за більш короткий час. І у більшості з цих функцій працює функція вірогідності, яка порівнюйтеся зі значеннями взятими з вибірки.

Метод для фази відтворення приймає геноми у якості аргумента, це буде масив елементів. Ми також зробимо додатковий виток мутації під час цієї фази відтворення. Щоб подвоїти вірогідність появи унікальних геномів. У цьому випадку перевірка цієї мутації менш доцільна, тому що після фази відтворення вихідний ген вже має достатній рівень якості даних, тобто самий розвинений геном за весь період існування цього автомобіля на світі.

Кожен біт дитячого геному визначатиметься на основі значень відповідного біта геному батька чи матері. Імовірність того, що дитина успадкує біт батька чи матері, становить 50%. Реалізація функції може виглядати таким чином:

```
function creationPipeline(
  solution1,
```

```
  solution2,
  mutationProb,
){
  const firstChild = [];
  const secondChild = [];
  for (let index = 0; index < solution1.length; index += 1) {
    firstChild.push(
      Math.random() < 0.5 ? solution1[index] : solution2[index]
    );
    secondChild.push(
      Math.random() < 0.5 ? solution1[index] : solution2[index]
    );
  }
  return [
    mutateGen(firstChild, mutationProb),
    mutateGen(secondChild, mutationProb),
  ]; }
```

Метод для фази вібірки буде заключним, тому на ньому зосередимо більше уваги. Щоб вибрати найбільш пристосованих особин для подальшого відтворення, нам потрібен спосіб дізнатися про придатність кожного геному. Для цього скористаємося так званою фітнес-функцією. Функція придатності завжди пов'язана з конкретним завданням. У нашому випадку фітнес-функція вимірюватиме відстань між автомобілем та місцем паркування. Чим ближче машина до місця стоянки, тим вона придатніша для подальшого розвитку геному автомобіля. Цей критерій є основним для переходу у наступне покоління навчання.

Тепер припустимо, що ми маємо значення придатності для кожного індивідуума популяції. Припустимо також, що ми відсортували всіх індивідуумів за значеннями пристосованості, отже перші особини є найсильнішими. Постає питання як нам

вибрати батьків та матерів із цього масиву ?. Нам необхідно зробити відбір таким чином, щоб чим вище значення пристосованості індивіда, тим вище шанси, що цей індивід буде обраний для відтворення покращеного рішення. Так як це математична задача оптимізації, то тут прослідковуєтеся ітеративне покращення саме поточного рішення, а не пошук нового. І для того щоб зрозуміти як нам обирати краще рішення, нам допоможе функція знизу:

```
function weightBasedRandom<T>(population: T[], weights: number {
    // Підготовка масиву з вагами для індивідів
    const weights: number[] = [];
    for (let index = 0; index < weights.length; index += 1) {
    weights[index] = weights[index] + (weights[index - 1] || 0);
}
// Отримання випадкового числа
const randomNumb = maxWeight * Math.random();
const maxWeight = weights[weights.length - 1];
// Вибираємо випадкового індивіда основуючись на його
// значенні придатності. Індивід з найвищим значенням буде
// вибраним у більшості випадків
for (let index = 0; index < population.length; index += 1) {
    if (weights[index] >= randomNumb) {
        return {
            item: population[index],
            index: index,
        };
    }
}
return {
    item: population[population.length - 1],
```

```
    ind: population.length - 1,
};
};
```

Придатність автомобіля визначатиметься відстанню від автомобіля до місця паркування. Чим вища дистанція, тим нижча придатність. Ми будемо розраховувати середню відстань від усіх коліс автомобіля до відповідних чотирьох кутів місця для паркування. Цю відстань ми називатимемо промахом, яка обернено пропорційна пристосованості.

Обчислення відстані між кожним колесом та кожним кутом окремо дозволить автомобілю зберігати правильну орієнтацію щодо паркувального місця, та бути більш точним у своїх рухах. Це також повинно додати плавності у переміщення автомобіля. У нас також є інша альтернатива, це простий розрахунок відстані від центру автомобіля до центру паркувального місця, але цей підхід є більш неточним. Тепер нам залишилось зібрати разом всі функції еволюції і створити світ для автомобіля. Потім запустити цикл еволюції, дати часу йти, покоління – еволюціонувати та йти вперед, а автомобілі – вчитися паркуватись у режимі реального часу. Звісно, увесь період навчання буде складно передивитись, тому що це може відбуватись навіть більше доби. По дорозі машини будуть стикатися з іншими машинами, а також неточно займати місце для паркування, але це всього лише початок світу, та перші покоління з моменту створення геному для них. Так що нам потрібно дати машинам трохи часу для навчання, від пів доби до цілої доби. В залежності від бажання що до якості геному.

Якщо порівняти цей процес навчання з реальними автоматизованими автомобілями, то наразі маємо декілька інноваційних підходів з Бостонського університету. Безпілотні автомобілі працюють на основі алгоритмів машинного навчання, яким для безпечної роботи потрібна величезна кількість даних про водіння. Але якби безпілотні автомобілі могли навчитися керувати так само, як немовлята вчаться ходити, — спостерігаючи та імітуючи оточуючих, — їм було б потрібно

набагато менше збираних даних про водіння. Ця ідея спонукала інженера Бостонського університету Ешеда Он-Бара розробити абсолютно новий спосіб для автономних транспортних засобів навчатися технікам безпечного водіння — спостерігаючи за іншими автомобілями на дорозі, прогнозуючи, як вони реагуватимуть на навколишнє середовище, і використовуючи цю інформацію, щоб зробити свій власні рішення щодо водіння.

Он-Бар , доцент кафедри електротехніки та комп'ютерної інженерії Інженерного коледжу BU та молодший науковий співробітник Інституту обчислювальної техніки та обчислювальної науки та інженерії імені Рафіка Б. Харірі BU, а також Джимуян Чжан, аспірант BU з електротехніки та комп'ютерної інженерії, нещодавно представили своє дослідження на конференції 2021 року з комп'ютерного бачення та розпізнавання образів. Їхня ідея щодо парадигми навчання виникла з бажання збільшити обмін даними та співпрацю між дослідниками у своїй галузі. Наразі автономні транспортні засоби потребують багатогодинного зберігання даних, щоб навчитися безпечно керувати, але деякі з найбільших у світі автомобільних компаній зберігають свої величезні обсяги даних приватні, щоб запобігти конкуренції.

Кожна компанія проходить той самий процес: бере автомобілі, встановлює на них датчики, платить водіям за керування транспортними засобами, збирає дані та навчає автомобілі керувати — каже Он-Бар. Обмін даними про водіння може допомогти компаніям швидше створювати безпечні автономні транспортні засоби, дозволяючи кожному в суспільстві отримати вигоду від співпраці. Он-Бар каже, що системи водіння зі штучним інтелектом вимагають стільки даних для належної роботи, що жодна компанія не зможе вирішити цю проблему самостійно.

Мільярди миль даних, зібраних у дорозі — це лише крапля в океані реальних подій і різноманітності, — каже Он-Бар. Проте відсутність вибірки даних може призвести до небезпечної поведінки та потенційного збою.

Запропонований дослідниками алгоритм машинного навчання працює, оцінюючи точки огляду та сліпі зони інших автомобілів поблизу, щоб створити карту

навколишнього середовища з висоти пташиного польоту. Ці карти допомагають безпілотним автомобілям виявляти перешкоди, як інші автомобілі чи пішоходи, і розуміти, як інші автомобілі повертають, проїжджають і поступаються, не зіткнувшись ні з чим.

Завдяки цьому методу безпілотні автомобілі навчаються, переводячи дії навколишніх транспортних засобів у власні системи відліку — нейронні мережі на основі алгоритму машинного навчання. Ці інші автомобілі можуть бути транспортними засобами, якими керує людина, без будь-яких датчиків, або транспортними засобами з автопілотом іншої компанії. Оскільки спостереження за всіма навколишніми автомобілями в сцені є ключовими для навчання алгоритму, ця парадигма «навчання за допомогою спостереження» заохочує обмін даними та, як наслідок, безпечніші автономні транспортні засоби.

Он-Бар і Чжан випробували свій алгоритм «дивись і навчайся», запропонувавши автономним автомобілям, керованим ним, переміщатися у двох віртуальних містах — одне з прямими поворотами та перешкодами, схожими на їх тренувальну середу, а інше — з несподіваними поворотами, як-от п’ятисторонні перехрестя. В обох сценаріях дослідники виявили, що їх автономна нейронна мережа потрапляє в дуже мало аварій. Маючи лише одну годину даних про водіння для навчання алгоритму машинного навчання, автономні транспортні засоби безпечно прибували до місця призначення в 92 відсотках випадків.

У той час як для попередніх найкращих методів потрібні були години, ми були здивовані тим, що завдяки нашому методу можна було навчитися безпечно керувати автомобілем лише за 10 хвилин даних про водіння, — каже Он-Бар.

Ці результати є багатообіцяючими, каже він, але все ще є кілька відкритих проблем у роботі зі складними міськими умовами. Врахувати суттєву різницю ракурсів у транспортних засобах, які спостерігаються, шум і оклюзію під час вимірювань датчиків, а також різних водіїв дуже важко — каже він.

Заглядаючи вперед, команда каже, що їхній метод навчання автономних

транспортних засобів самостійному керуванню можна також використовувати в інших технологіях. Роботи-доставники або навіть дрони можуть навчатися, спостерігаючи за іншими системами штучного інтелекту в їхньому середовищі — каже Он-Бар.

Взагалі розуміння справжніх водіїв є дуже складним завданням. Це скоріше емоції, ніж логіка, і все це підживлюється реакціями . Стає дуже невизначеним, якими будуть наступні дії водіїв або пішоходів поблизу, тому система, яка може передбачити дії інших учасників дорожнього руху, може бути дуже важливою для безпеки дорожнього руху.

Автомобіль має 360-градусний огляд навколишнього середовища, що дозволяє йому сприймати та фіксувати всю інформацію та обробляти її. Після введення в алгоритм глибокого навчання він може придумати всі можливі рухи, які можуть зробити інші учасники дорожнього руху. Це схоже на гру, де гравець має кінцеву кількість ходів і намагається знайти найкращий хід, щоб перемогти суперника.

Датчики в безпілотних автомобілях дозволяють їм виконувати такі завдання, як класифікація зображень, виявлення об’єктів, сегментація та локалізація. За допомогою різних форм представлення даних автомобіль може прогнозувати об’єкт навколо нього.

Алгоритм глибокого навчання може моделювати таку інформацію як зображення та точки даних з LiDAR і RADAR під час навчання. Ця ж модель, але під час висновку, може допомогти машині підготуватися до всіх можливих рухів, які включають гальмування, зупинку, уповільнення, зміну смуги руху тощо.

Роль глибокого навчання полягає в інтерпретації складних завдань зору, локалізації в навколишньому середовищі, покращенні сприйняття та активації кінематичних маневрів у безпілотних автомобілях. Це забезпечує безпеку на дорозі та легкість поїздок. Але складна частина полягає в тому, щоб вибрати правильну дію з кінцевої кількості дій.

3.4 Як працює мозок у справжнього самокерованого автомобіля

У безпілотних автомобілях життєво важливо приймати рішення. Їм потрібна система, яка є динамічною та точною в невизначеному середовищі. Необхідно враховувати, що не всі показання датчиків будуть правдивими, і що люди можуть робити непередбачувані рішення під час водіння. Ці речі не можна виміряти безпосередньо. Навіть якби ми могли їх виміряти, ми не можемо передбачити їх з достатньою точністю. Для прийняття рішення автомобіль повинен мати достатньо інформації, щоб він міг вибрати необхідний набір дій. Ми дізналися, що датчики допомагають автомобілю збирати інформацію, а алгоритми глибокого навчання можна використовувати для локалізації та прогнозування.

Підсумовуючи, локалізація дозволяє автомобілю знати своє початкове положення, а передбачення створює n кількість можливих дій або рухів залежно від середовища. Залишається питання: який варіант найкращий із багатьох прогнозованих дій?

Коли справа доходить до прийняття рішень, ми використовуємо глибоке навчання з підкріпленням (DRL). Якщо говорити точніше, то в основі DRL лежить алгоритм прийняття рішень, який називається марковським процесом прийняття рішень (MDP) (ми дізнаємось більше про MDP у наступному розділі, де ми говоримо про навчання з підкріпленням).

На рис. 3.5 нижче зображено самокерований автомобіль, який рухається до перехрестя. Інша машина, також рухається до перехрестя. У цьому сценарії безпілотний автомобіль має передбачити, чи поїде інший автомобіль прямо, ліворуч чи праворуч. У кожному випадку автомобіль повинен вирішити, який маневр він повинен виконати, щоб запобігти зіткненню.

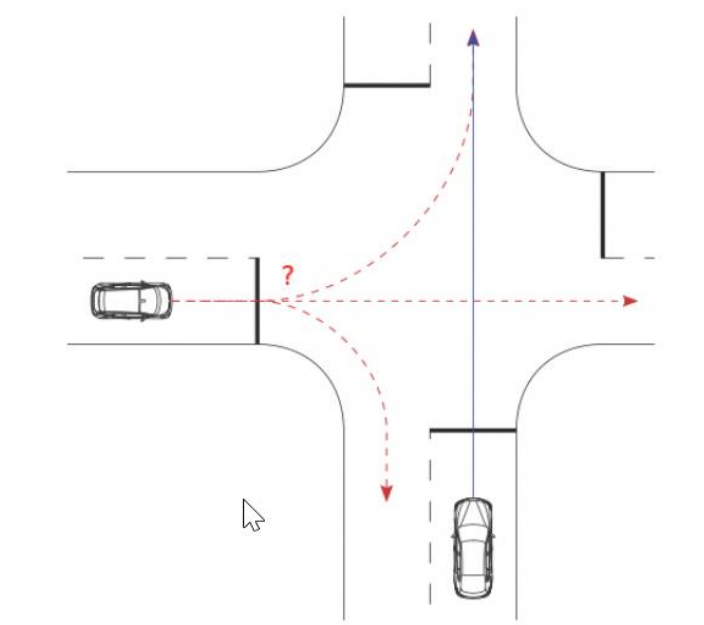


Рисунок 3.5 – Самокерований автомобіль рухається до перехрестя.

Зазвичай MDP використовується для прогнозування майбутньої поведінки учасників дорожнього руху. Слід мати на увазі, що сценарій може стати дуже складним, якщо кількість об'єктів, особливо рухомих, збільшується. Це в кінцевому підсумку збільшує кількість можливих ходів для самого безпілотного автомобіля.

Щоб вирішити проблему пошуку найкращого для себе ходу, модель глибокого навчання оптимізовано за допомогою байєсівської оптимізації . Існують також ситуації, коли для прийняття рішень використовується структура, що складається як з прихованої марковської моделі, так і з байєсівської оптимізації. Загалом, прийняття рішень у безпілотних автомобілях є ієрархічним процесом. Цей процес складається з чотирьох компонентів:

- планування шляху або маршруту: По суті, планування маршруту – це перше з чотирьох рішень, які повинен прийняти автомобіль. Увійшовши в середовище, автомобіль повинен спланувати найкращий можливий маршрут від свого поточного положення до потрібного пункту призначення. Ідея полягає в тому, щоб знайти оптимальне рішення серед усіх інших рішень;
- арбітраж поведінки: після того, як маршрут сплановано, автомобіль повинен

самостійно пройти маршрут. Автомобіль знає про статичні елементи, такі як дороги, перехрестя, середня завантаженість доріг тощо, але він не може точно знати, що інші учасники дорожнього руху збираються робити під час подорожі. Ця невизначеність у поведінці інших учасників дорожнього руху вирішується за допомогою ймовірнісних алгоритмів планування, таких як MDP;

- планування руху: коли рівень поведінки вирішує, як рухатися певним маршрутом, система планування руху керує рухом автомобіля. Рух автомобіля повинен бути посильним і комфортним для пасажира. Планування руху включає швидкість транспортного засобу, зміну смуги руху тощо, і все це має відповідати навколишньому середовищу, в якому перебуває автомобіль;
- управління транспортним засобом: керування транспортним засобом використовується для виконання опорного шляху з системи планування руху.

Одним із способів навчання самокерованих автомобілів також є ChauffeurNet: тренування самостійного автомобіля за допомогою імітаційного навчання. ChauffeurNet — це нейронна мережа на основі RNN, яку використовує Google Waymo, однак CNN насправді є одним із основних компонентів тут і використовується для вилучення функцій із системи сприйняття. CNN у ChauffeurNet описується як мережа згорткових функцій, або FeatureNet, яка витягує контекстне представлення функцій, спільне для інших мереж. Потім ці представлення передаються в рекурентну мережу агентів (AgentRNN), яка ітеративно дає прогноз послідовних точок на траєкторії руху.

Ідея цієї мережі полягає в тому, щоб навчити самокерований автомобіль за допомогою імітаційного навчання. У статті, опублікованій Бансалом та іншими « ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst », вони стверджують, що навчання самокерованого автомобіля навіть з 30 мільйонами прикладів недостатньо. Щоб подолати це обмеження, автори навчили автомобіль

синтетичним даним. Ці синтетичні дані внесли такі відхилення, як внесення збурень на траєкторію, додавання перешкод, введення неприродних сцен тощо. Вони виявили, що такі синтетичні дані можуть тренувати автомобіль набагато ефективніше, ніж звичайні дані.

Як ми бачили раніше, процес самостійного водіння зазвичай має наскрізний процес, де система сприйняття є частиною алгоритму глибокого навчання разом із плануванням і контролем. У випадку з ChauffeurNet система сприйняття не є частиною наскрізного процесу; натомість це система середнього рівня, де мережа може мати різні варіанти введення від системи сприйняття. ChauffeurNet визначає траєкторію руху, спостерігаючи за зображенням сцени на середньому рівні від датчиків, використовуючи вхідні дані разом із синтетичними даними для імітації досвідченого водія.

По суті, представлення середнього рівня безпосередньо не використовує необроблені дані сенсора як вхідні дані, виключаючи завдання сприйняття, тому ми можемо поєднувати реальні та змодельовані дані для легшого навчання передачі. Таким чином, мережа може створити високорівневий погляд на навколишнє середовище з висоти пташиного польоту, що зрештою дає кращі рішення. Краща продуктивність є результатом самооптимізації внутрішніх компонентів для максимізації загальної продуктивності системи замість оптимізації вибраних людиною проміжних критеріїв, таких як визначення смуги руху. Зрозуміло, що такі критерії вибрано для простоти людської інтерпретації.

ВИСНОВКИ

В ході дослідження процесу паркування автомобіля та вибору алгоритму для інтелекту автомобіля, було виконано та задокументовано такі завдання:

1. Розроблено 3D модель автомобіля яка включає кермо та двигун. Ці два елементи керуються за допомогою мозку або інтелекту автомобіля через сигнали у вигляді цілих чисел. Орієнтація автомобіля у просторі відбувається за допомогою сенсорів, данні з яких отримує мозок і вже за допомогою генетичного алгоритму виконується обрахування траєкторії заїзду автомобіля у паркомісце. Це обрахування еволюційне, тобто проходять покоління з якими автомобіль навчається точніше під'їжджати до паркомісця. Було експериментально доведено що достатньо трохи більше доби щоб автомобіль паркувався з точністю до одного метра. Це зафіксовано на відео зі слайду 10, де автомобіль впевнено, не задягяючи сусідній транспорт, встає на своє місце.
2. Доведено можливість використання лінійних рівнянь замість нейронних мереж у задачах пошуку оптимального рішення. Це видно з того наскільки швидко(10 сек.) може проходити одне покоління. Тобто розрахунок за допомогою цих рівнянь доволі ефективно дає в результаті коефіцієнти для двигуна та керма і це тому що базові арифметичні операції виконується миттєво у сучасних мовах програмування. Але є і висновок з цього використання, лінійні рівняння доволі прості щоб навчити автомобіль до нового оточення, тобто якщо змінити місце розташування прямокутника для паркування, наш автомобіль вже не зможе зорієнтуватись у новому просторі. Або навіть новий тип паркувань де автомобілі пакуються двері до дверей вже буде не можливим. А це через те що отримані коефіцієнти не універсальні, тобто вони враховують тільки одну траєкторію і тільки одне положення парко місця.
3. Згенеровано оптимальний геном автомобіля, який паркується на місце з точністю до метра. Цей набір з нулів та одиниць тримає в собі поведінку

автомобіля, тобто напрямок руху у даний конкретний момент або секунду, швидкість руху та бік у який повернути колеса. Це як інструкція того як потрапити у місце призначення. І ця інструкція може бути використана повторно якщо умови паркування такі ж самі. І так як у кожному поколінні ми мутуємо геном, може вийти так що дві різні еволюції будуть мати різну поведінку паркування. І тут з'являється подальша перспектива розвитку цієї платформи у вигляді порівняння геномів, та обрання того який пакується за більш коротший шлях.

4. Встановлено що генетичний алгоритм більш ефективний ніж диференціальна еволюція та оптимізація рою частинок. Це видно з Додатку А. Генетичний алгоритм помітно швидший ніж інші два і це прослідковується і при збільшені об'єму даних. З цього можна зробити висновок, що було зроблено правильний вибір алгоритму з цієї категорії.

Цей додаток є чудовою платформою для вдосконалення та перевірки різних еволюційних алгоритмів та порівняння їх продуктивності. Також це є показовим приклад того, що не тільки нейронні мережі можуть використовуватися у задачах, де потрібно навчитися виконувати якусь комплексну операцію з нуля.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Back T, Schwefel HP. Огляд еволюційного алгоритму оптимізації параметрів. Еволюційне обчислення. 1993 рік. – С. 1–23.
2. Cantú-Paz E. A summary of research on parallel genetic algorithms. Illinois Genetic Algorithm Lab., Univ. Urbana, IL: Illinois Genetic Algorithm Lab., Univ. Illinois Urbana-Champaign; 1995. p. 950076. Tech. Rep. – С. 22- 35.
3. Charalampakis, AE, and Tsiatas, GC (2019). Критична оцінка метаевристичних агоритмів. – С. 101 – 105.
4. Eiben, A., Hinterding, R., and Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. IEEE Trans. Evol. Comput. 3. – С. 120–130.
5. Islas R, Heine T, Ito K, Schleyer P, v. R., and Merino G. Boron rings enclosing planar hypercoordinate group 14 elements. Journal of the American Chemical Society. 2007. – С. 50–55.
6. Lienig J. A parallel genetic algorithm for performance-driven VLSI routing. IEEE Transactions on Evolutionary Computation. 1997. – С. 29–39.
7. Mallipeddi, R., Suganthan, P. N., Pan, Q. K., and Tasgetiren, M. F. (2011). Differential evolution algorithm with ensemble of parameters and mutation strategies. – С. 170–190.
8. Michael Affenzeller, Stephan Winkler, Stefan Wagner. Evolutionary System Identification: New Algorithmic Concepts and Applications // ResearchGate 2008. – С. 30–45.
9. Notredame C, Higgins DG. SAGA: Sequence alignment by genetic algorithm. Nucleic Acids Research. 1996. – С. 1515–1524.
10. Riccardo Poli, William B. Langdon, and Nicholas F. McPhee. A Field Guide to Genetic Programming // ResearchGate 2008. – С. 29-83.

11. S. P. Lim і H. Haron, «Ефективність різних методів, застосованих у генетичному алгоритмі для порівняльних функцій», ACIIDS 2013, LNAI 7802, частина I, 2013. – С. 202–212.
12. Shukla, R., Hazela, B., Shukla, S., Prakash, R., and Mishra, K. K. (2017). “Variant of differential evolution algorithm”. – С. 601 – 610.
13. Storn,R., and Price, K.(1995). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. J.Glob. –С. 22–60.
14. Storn, R., and Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. – С. 41–59.
15. Veldhuize DAV, Zydallis JB, Lamont GB. Considerations in engineering parallel multiobjective evolutionary algorithms. IEEE Transactions on Evolutionary Computation. 2003. – С. 144–173.
16. Zhao S, Jin R, Abroshan H, Zeng C, Zhang H, House SD. Gold nanoclusters promote electrocatalytic water oxidation at the nanocluster/cose2 interface. Journal of the American Chemical Society. 2017. – С. 27–37.
17. Бадер С.Д. Колоквіум: Можливості наномагнетизму. Огляди сучасної фізики. 2006 рік. – С. 30–35.
18. Брест Дж., Грейнер С., Бошкович Б., Мернік М. та Зумер В. (2006). Самоадаптація контрольних параметрів у диференціальній еволюції: порівняльне дослідження числових контрольних задач.
19. Гамперле, Р., Мллер, С.Д., і Кумуцакос, П. (2002). Дослідження параметрів для диференціальної еволюції . Технічний звіт WSEAS NNA-FSFS-EC 2002. – С. 35–45. Інтерлакен.
20. Голланд Дж. Х. Адаптація в природних і штучних системах. Ann Arbor, MI: Univ Michigan Press; 1975 рік. – С. 22–27.

21. Де Йонг К.А. Аналіз поведінки класу генетично адаптивної системи. Докторська дисертація. Мічиганський університет; 1975 рік. – С. 22–31.
22. Девіс Л.Д. Довідник з генетичних алгоритмів. Нью-Йорк: Van Nostrand Reinhold; 1991 рік. – С. 10–17.
23. Мезура-Монтес, Е., Вельскес-Рейес, Дж., і Коелло, САС (2006). «Порівняльне дослідження варіантів диференціальної еволюції для глобальної оптимізації», у GECCO '06: Матеріали 8-ї щорічної конференції з генетичних і еволюційних обчислень (Сіетл, Вашингтон). – С. 470–490.
24. Пелегріні М, Паррейра Р.Л.Т., Феррао ЛФА, Караморі Г.Ф., Ортолан А.О., Сільва Е.Х. Розкладання гідразину на малому платиновому кластері: роль проміжного продукту. 2016 рік. – С. 29–33.
25. Рехенберг І. Кібернетичний шлях вирішення експериментальної задачі. Royal Aircraft Establishment (UK): Міністерство авіації; 1965 рік.–С.50–62.
26. Т. Мантере, (2007). Мінімально-максимальний гібрид DE/GA для проблем із обмеженнями. [Електронний ресурс] . – Режим доступа: www.uva.fi/~timan/Min_max_GA_2007.pdf.
27. Фогель Д.Б. Еволюційні обчислення: до нової філософії машинного інтелекту. Піскатавей, Нью-Джерсі: IEEE Press; 1995 рік. – С. 47–60.
28. Фрідман Г.Дж. Цифрове моделювання еволюційного процесу. Загальний системний щорічник. 1959 рік. – С. 171–184.
29. Хао Дж. К., Купол Р. Новий популяційний метод для проблем задовільності. праць. 11-ї Європейської конференції зі штучного інтелекту. Нью-Йорк: Wiley; 1994. – С. 135–139.
30. Швефель Х.П. Evolutions Strategie and Numerische Optimierung. доктор філософії Дипломна робота. Берлін: Технічний університет; 1975 рік. – С. 44–51.

ДОДАТОК А

Функ.	Метод	Мін час	Макс час	Середній час
f1(x)	GA	0.0008	0.1978	0.0641
	DE	0.0040	0.3654	0.4951
	PSO	0.0099	0.0719	0.5206
f2(x)	GA	0.0169	3.6098	0.0770
	DE	0.5545	4.5842	0.7646
	PSO	0.8398	2.3545	0.7648
f3(x)	GA	0.0563	19.1326	0.0752
	DE	5.0154	12.3525	0.7385
	PSO	46.1047	116.1930	0.7605
f4(x)	GA	0.2972	1.6587	0.0662
	DE	0.0649	1.9861	0.5206
	PSO	0.0613	0.6920	0.5595
f5(x)	GA	0.0019	1.5947	0.0629
	DE	0.0032	0.7700	0.5124
	PSO	0.1685	1.1863	0.5301
f6(x)	GA	2414.3400	2420.5900	0.0837
	DE	91.0157	206.6490	0.9084
	PSO	361.0020	1081.0000	1.1379
f7(x)	GA	1.0000E-06	5.4100E-04	0.0660
	DE	8.9361E-11	1.9694E-06	0.4943
	PSO	1.6500E-16	2.4500E-14	0.5278

ДОДАТОК Б

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

МАГІСТЕРСЬКА РОБОТА

Кафедра інженерії програмного забезпечення

«МОДЕЛЮВАННЯ ПРОЦЕСУ АВТОМАТИЗОВАНОГО ПАРКУВАННЯ АВТОМОБІЛЯ НА ОСНОВІ ГЕНЕТИЧНОГО АЛГОРИТМУ»

Виконав: студент групи ПДМ-61 Даценко Михайло Андрійович

Керівник: Завідувач кафедри Інженерії програмного забезпечення, к.т.н., доцент, Негоденко Олена Василівна

Київ - 2022

МЕТА, ОБ'ЄКТ, ПРЕДМЕТ ДОСЛІДЖЕННЯ

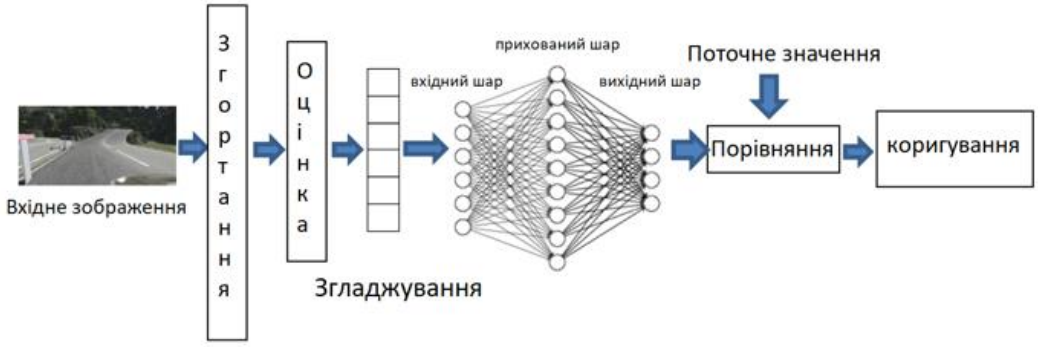
Мета роботи: підвищення точності паркування автомобіля за рахунок використання в атоматизованій системі генетичного алгоритму

Об'єкт дослідження: паркування автомобіля

Предмет дослідження: генерація геному автомобіля за допомогою генетичного алгоритму та сигмоїдної функції

АНАЛОГИ ГЕНЕТИЧНОГО АЛГОРИТМУ З ЇХ НЕДОЛІКАМИ

Згорткові нейронні мережі



- Труднощі з класифікацією зображень під кутом
- Не мають систем координат, які є основним компонентом людського зору
- Процес навчання займає багато часу

АНАЛОГИ ГЕНЕТИЧНОГО АЛГОРИТМУ З ЇХ НЕДОЛІКАМИ

Якісне навчання або Q-lerning

- Не ефективно навчатиметься в середовищі, яке швидко змінюється
- Результат може застрягти в локальному мінімумі, оскільки моделі обирають лише найкоротший шлях

Продуктивність не підтверджена при зміні середовища

ГЕНЕТИЧНИЙ АЛГОРИТМ

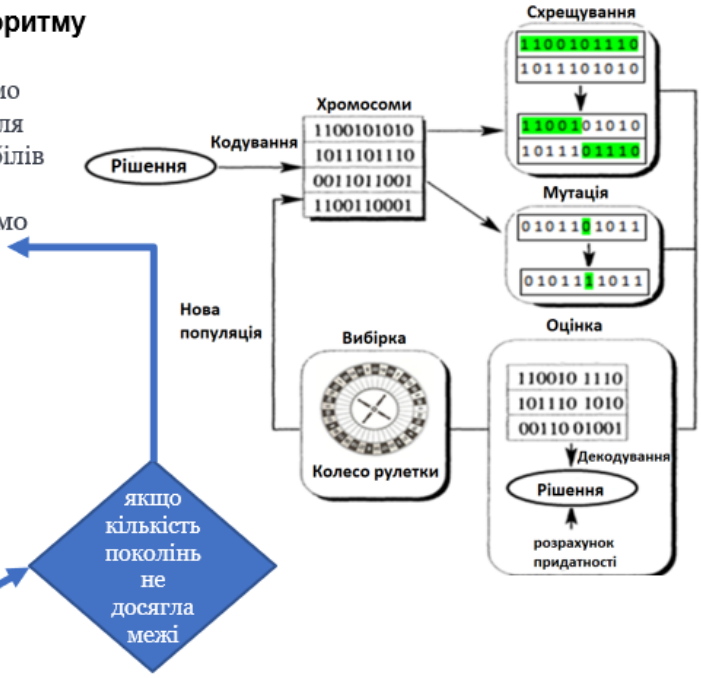
4 кроки роботи алгоритму

1. СТВОРЕННЯ - генеруємо набір випадкових геномів для першого покоління автомобілів

2. СЕЛЕКЦІЯ - вибираємо найбільш пристосовані автомобілі із поточного покоління

3. НОВЕ ПОКОЛІННЯ - створення більш розумного автомобіля наслідуючи найкраще від попередніх рішень

4.МУТАЦІЯ – змінюємо кілька випадкових бітів генному

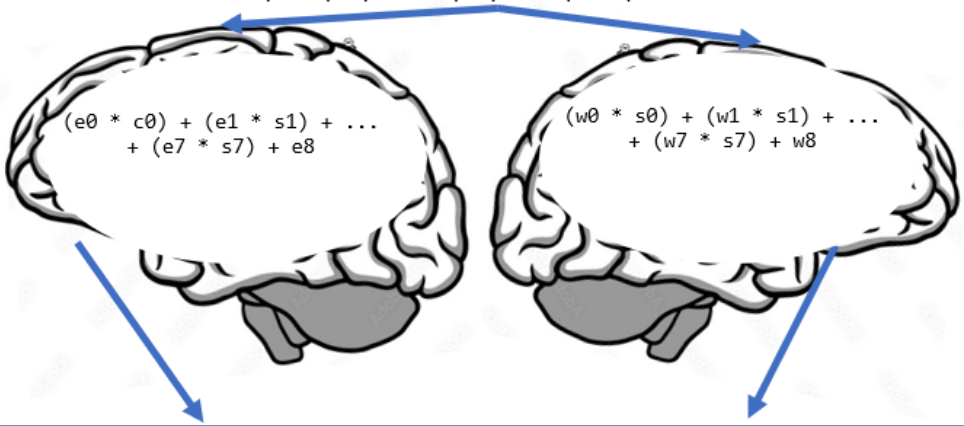


ГЕНЕТИЧНИЙ АЛГОРИТМ У ДЕТАЛЯХ: КРОК 1

Два лінійних многочлена з параметрами від сенсорів

Отримуємо значення від сенсорів кожну 1/10 сек.

0 | 0.7 | 5 | 0.003 | 0 | 2.76 | 0.2 | 3.245

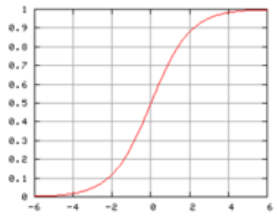


0.123	5321.22
Коефіцієнт для двигуна	Коефіцієнт для керма ↔

ГЕНЕТИЧНИЙ АЛГОРИТМ У ДЕТАЛЯХ: КРОК 2

Сігмоїд перетворює широкий діапазон чисел з плаваючою комою (вісь x) у числа з плаваючою комою з обмеженим діапазоном (0...1) (вісь y)

$$\sigma(x) = \frac{1}{1+e^{-x}}.$$



$$\sigma(x) = \frac{1}{1+e^{-x}}.$$

0.07
Коефіцієнт для двигуна

0.42
Коефіцієнт для керма

7

ГЕНЕТИЧНИЙ АЛГОРИТМ У ДЕТАЛЯХ: КРОК 3

Перетворення значень у діапазоні (0...1) на сигнали органів керування
(-1, 0, 1)

Якщо сигмоїда < 0.1 → -1

Якщо сигмоїда > 0.9 → 1

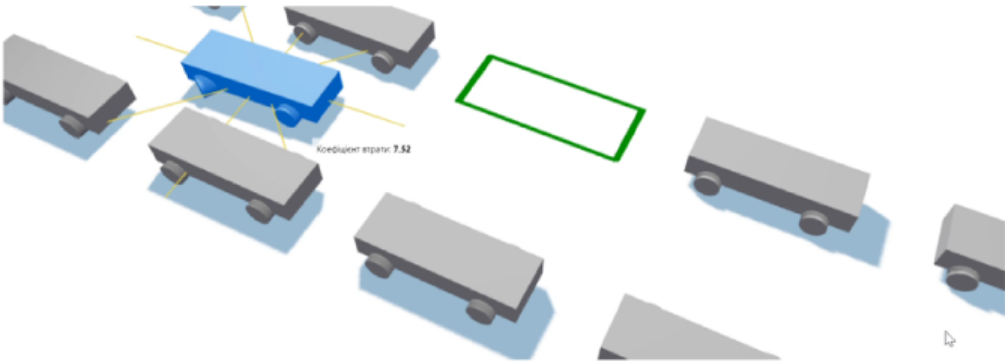
Якщо сигмоїда < 0.1 → 0

1
Коефіцієнт для двигуна
Їдь вперед

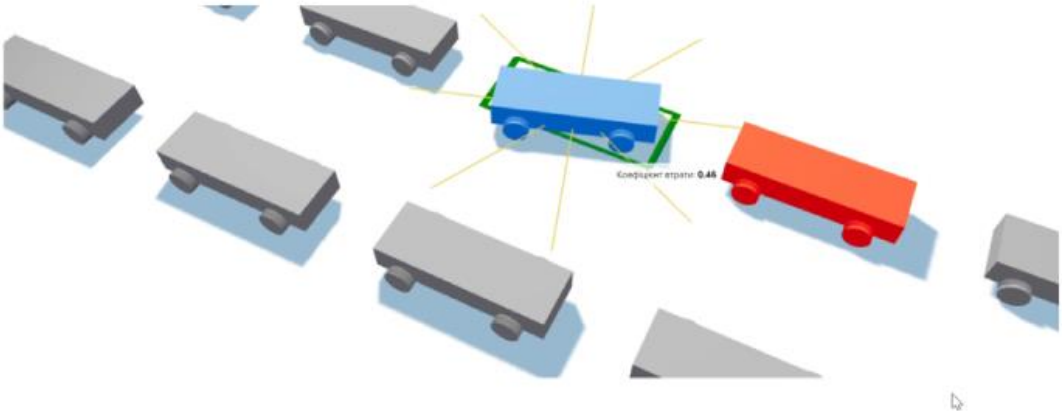
-1
Коефіцієнт для керма
Поверни вліво

8

ПРАКТИЧНИЙ РЕЗУЛЬТАТ: початок еволюції



ПРАКТИЧНИЙ РЕЗУЛЬТАТ: еволюція плинула довше доби



ПОРІВНЯННЯ ПРОДУКТИВНОСТІ ОБРОБКИ ДАННИХ

- DE - Диференціальна еволюція
- PSO - Оптимізація рою частинок

- GA- Генетичний алгоритм

Функ.	Метод	Мін час	Макс час	Середній час
$f_1(x)$	GA	0.0008	0.1978	0.0641
	DE	0.0040	0.3654	0.4951
	PSO	0.0099	0.0719	0.5206
$f_2(x)$	GA	0.0169	3.6098	0.0770
	DE	0.5545	4.5842	0.7646
	PSO	0.8398	2.3545	0.7648
$f_3(x)$	GA	0.0563	19.1326	0.0752
	DE	5.0154	12.3525	0.7385
	PSO	46.1047	116.1930	0.7605
$f_4(x)$	GA	0.2972	1.6587	0.0662
	DE	0.0649	1.9861	0.5206
	PSO	0.0613	0.6920	0.5595
$f_5(x)$	GA	0.0019	1.5947	0.0629
	DE	0.0032	0.7700	0.5124
	PSO	0.1685	1.1863	0.5301
$f_6(x)$	GA	2414.3400	2420.5900	0.0837
	DE	91.0157	206.6490	0.9084
	PSO	361.0020	1081.0000	1.1379
$f_7(x)$	GA	1.0000E-06	5.4100E-04	0.0660
	DE	8.9361E-11	1.9694E-06	0.4943
	PSO	1.6500E-16	2.4500E-14	0.5278

Генетичний алгоритм працює стабільно швидше

ВИСНОВКИ

1. Розроблено 3D модель паркування автомобіля на базі генетичного алгоритму
2. Доведено можливість використання лінійних рівнянь замість нейронних мереж у задачах оптимізації
3. Згенеровано оптимальний геном автомобіля, який паркується на місце з точністю до метра
4. Встановлено що генетичний алгоритм більш ефективний ніж диференціальна еволюція та оптимізація рою частинок

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези доповідей:

1. XV НАУКОВО-ТЕХНІЧНА КОНФЕРЕНЦІЯ «СУЧАСНІ ІНФОКОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ» – Київ: ДУТ, публікація запланована на кінець грудня
2. НАУКОВО-ПРАКТИЧНА КОНФЕРЕНЦІЯ «ПРОБЛЕМИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ» – Київ: ДУТ, збірник ще не опубліковано

ДЯКУЮ ЗА УВАГУ!