КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп’ютерних наук і кібернетики
Кафедра теоретичної кібернетики

Кваліфікаційна робота
на здобуття ступеня бакалавра
за спеціальністю 122 Комп’ютерні науки
на тему:

Використання генетичних алгоритмів в прикладних задачах

Виконав студент 4-го курсу
ЄРМОЛЕНКО Олександр ______________

Науковий керівник:
професор, доктор фіз.-мат. наук
ПАШКО Анатолій ______________

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент ______________

Роботу розглянуто й допущено до захисту на засіданні кафедри теоретичної кібернетики
«___» __________________ 202__ р.,
Протокол №_____
Завідувач кафедри
Юрій КРАК ______________

Київ – 2023

РЕФЕРАТ

Обсяг роботи 41 сторінка, 22 ілюстрації, 8 використаних джерел.

Об'єктом дослідження є процес оптимізації в транспортній логістиці, а точніше, використання генетичних алгоритмів для вирішення задач оптимізації в цій сфері.

Метою цієї дипломної роботи є дослідження можливості використання генетичних алгоритмів для оптимізації процесів в транспортній логістиці. Особливий акцент роботи буде зроблено на вирішенні задач планування маршрутів. Крім того, робота передбачає розробку програмного забезпечення, яке демонструє застосування генетичних алгоритмів до цих задач.

Інструменти розробки: середовище програмування PyCharm, мова програмування Python 3.9.4.

У цій роботі проведено теоретичне дослідження генетичних алгоритмів та їх потенційного застосування в транспортній логістиці. Зокрема, були досліджені задачі планування маршрутів, та як генетичні алгоритми можуть бути використані для їх оптимізації. Було розроблено програмне забезпечення, яке реалізує генетичний алгоритм для розв'язання цих задач. Крім того, було зроблено висновки про доцільність використання генетичних алгоритмів в контексті практичного застосування в транспортній логістиці.

СПИСОК УМОВНИХ СКОРОЧЕНЬ

ГА – генетичний алгоритми

ТЛ – транспортна логістика

Зміст

РЕФЕРАТ **2**

СПИСОК УМОВНИХ СКОРОЧЕНЬ **3**

ЗМІСТ **4**

ВСТУП **6**

РОЗДІЛ 1. ТЕОРЕТИЧНА ЧАСТИНА **9**

1. Розгляд основних понять 9
   - 1.1 Основи генетичних алгоритмів 9
   - 1.2 Прикладні задачі генетичних алгоритмів 10
   - 1.3 Генетичні алгоритми в логістиці 11
2. Огляд генетичних алгоритмів 13
   - 2.1 Історія генетичних алгоритмів 13
   - 2.2 Структура генетичних алгоритмів 13
     - 2.2.1 Популяція 15
     - 2.2.2 Фітнес-функція 16
     - 2.2.3 Представлення генотипу 16
     - 2.2.4 Селекція 17
   - 2.3 Види генетичних алгоритмів 18

РОЗДІЛ 2. ПРАКТИЧНА ЧАСТИНА **20**

1. Постановка задачі 20
   - 1.1 Огляд програмного забезпечення 20
   - 1.2 Вхідні дані 20
2. Реалізація 21
   - 2.1 Структура вхідних даних 21
   - 2.2 Алгоритм повного перебору 22
     - 2.2.1 Часова оцінка 25
   - 2.3 Генетичний алгоритм 27
     - 2.3.1 Опис моделей 27
     - 2.3.2 Функція фітнесу 29
     - 2.3.3 Еволюція 30
     - 2.3.4 Мутація 31
     - 2.3.5 Кросовер 32
     - 2.3.6 Селекція 33
       - 2.3.6.1 Рулетковий відбір 33

2.3.6.2 Турнірна селекція 34
2.3.6.3 Рангова селекція 34
2.3.6 Результати 35
2.3.7 Можливі покращення 38
ВИСНОВОК 40
ДЖЕРЕЛА ІНФОРМАЦІЇ 41

ВСТУП

Генетичні алгоритми — це потужний інструмент для розв'язання складних проблем оптимізації і пошуку, що використовують принципи еволюції і натурального відбору. Ці методи отримали широке застосування в різних областях, включаючи машинне навчання, обробку сигналів, комп'ютерний дизайн і багато іншого.

Еволюційні процеси, які підтримують життя через адаптацію і розвиток, є основною концепцією, на яку спираються генетичні алгоритми як методи оптимізації. Центральними елементами еволюції є принципи випадкової мутації та генетичної рекомбінації, що стають фундаментальними основами для роботи генетичних алгоритмів.

Процес створення та використання генетичних алгоритмів (ГА) зазвичай містить декілька ключових етапів:

1. **Ініціалізація**: На цьому етапі створюється початкова популяція можливих рішень (індивідів). Кожне рішення часто представляється у вигляді хромосоми — набору генів, які кодують деталі рішення.
2. **Оцінка**: Кожен індивід в популяції оцінюється за допомогою функції приспособлености, яка визначає, наскільки добре рішення вирішує поставлену задачу.
3. **Селекція**: На цьому етапі індивіди вибираються для участі у генетичних операціях на основі їх приспособлености. Часто використовуються тактики, які більш ймовірно вибирають індивідів з вищою приспособленістю.
4. **Кросовер (схрещування)**: Вибрані індивіди комбінуються для створення нового покоління індивідів. Цей процес може бути дуже різним і залежить від конкретного ГА, але зазвичай він включає обмін генами між двома батьківськими хромосомами.
5. **Мутація**: Після кросоверу нові хромосоми можуть піддаватися мутації, яка випадковим чином змінює деякі з їх генів. Це допомагає зберегти різноманітність в популяції та запобігає застрягенню в

локальних мінімумах.

6. **Ітерація**: Процес повторюється з новим поколінням індивідів, починаючи з оцінки.

Ці етапи продовжуються до тих пір, поки не буде досягнуто критерії зупинки, такі як вичерпання максимального числа поколінь або досягнення достатнього рівня пристосованості.

Однією з важливих областей застосування ГА є задачі оптимізації та планування.

Транспортна логістика є важливою складовою сучасного економічного простору, що включає планування, координацію та контроль переміщення товарів та послуг від початку до кінця. Вона включає різні аспекти, включаючи управління запасами, складування, транспортування, а також координацію між різними учасниками ланцюга поставок. Генетичні алгоритми (ГА) можуть виявитися корисними у цій сфері, особливо при вирішенні складних проблем оптимізації, таких як планування маршрутів доставки, оптимізація використання складського простору, розкладу, і так далі.

Ця дипломна робота фокусується на дослідженні та використанні генетичних алгоритмів в транспортній логістиці. Ми розглянемо теоретичні аспекти ГА, їхні параметри та механізми, а також розробимо та впровадимо ГА для розв'язання конкретних задач в ТЛ.

Як результат цієї дипломної роботи, ми сподіваємося розробити ефективний генетичний алгоритм, який може бути використаний для оптимізації різних аспектів ТЛ. Мета цієї роботи - не тільки розробити практичне рішення для проблем логістики, але і продемонструвати, як генетичні алгоритми можуть бути ефективно використані в реальних прикладних ситуаціях.

Транспортна логістика, безумовно, може бути представлена у вигляді послідовності етапів. Деякі з них можуть включати:

1. **Планування**: Цей етап включає в себе визначення стратегії логістики, вибір найефективніших маршрутів доставки, розробку графіків і термінів, та інше. За допомогою ГА можна оптимізувати цей процес,

знаходячи найкращі маршрути та графіки.

2. **Складування**: Задачі, пов'язані із складуванням, можуть включати оптимальне розміщення товарів на складі, управління запасами, оптимізацію розміщення для швидкого доступу до популярних товарів, та інше. ГА можуть допомогти в розв'язанні цих задач, знаходячи оптимальні стратегії розміщення та управління запасами.
3. **Транспортування**: Тут ГА можуть бути використані для розробки оптимальних маршрутів доставки, що мінімізують витрати на паливо, час доставки, або інші критерії. Вони також можуть допомогти в плануванні розвантаження та завантаження транспортних засобів.
4. **Моніторинг і контроль**: На цьому етапі важливо відслідковувати виконання плану, виявляти та реагувати на відхилення. ГА можуть допомогти в розробці систем, які динамічно адаптуються до змін умов.
5. **Аналіз та вдосконалення**: Після виконання логістичних операцій необхідно проаналізувати результати, виявити можливості для вдосконалення та застосувати ці вдосконалення для покращення майбутніх операцій. ГА можуть допомогти в цьому процесі, використовуючи історичні дані для покращення стратегій планування та виконання.

У своїй дипломній роботі я також хочу розглянути, як ГА можуть бути застосовані на кожному з цих етапів для покращення загальної ефективності та продуктивності транспортної логістики.

РОЗДІЛ 1. ТЕОРЕТИЧНА ЧАСТИНА

1. Розгляд основних понять

1.1 Основи генетичних алгоритмів

Генетичні алгоритми (ГА) є евристичною технологією оптимізації, що виникла в середині 20-го століття, яка набула популярності завдяки своїй здатності розв'язувати складні оптимізаційні задачі. Вони засновані на принципах дарвінівської еволюції і біологічного вибору, включаючи селекцію, кросовер (схрещування) і мутацію. Суть генетичних алгоритмів полягає в використанні еволюційних процесів для пошуку оптимальних або наближено оптимальних рішень у великому просторі пошуку. Кожне рішення в генетичному алгоритмі представлене як хромосома, яка складається з набору генів, а наявність різних генів визначає властивості рішення.

Оптимізація є важливою галуззю математики, яка займається визначенням найкращого (оптимального) рішення з усіх можливих варіантів. "Найкраще" тут може мати різне значення в залежності від контексту: це може бути мінімум (наприклад, витрати, час) або максимум (наприклад, прибуток, ефективність).

Оптимізація потрібна для вирішення реальних проблем в широкому діапазоні областей, включаючи інженерію, науку, бізнес, фінанси та логістику. Оптимізація дозволяє нам використовувати наші ресурси найбільш ефективним чином та досягти наших цілей з мінімальними витратами.

Евристична оптимізація - це підгалузь оптимізації, яка використовує евристики для знаходження прийнятних або наближених рішень для складних оптимізаційних проблем, де точні або аналітичні методи недостатньо ефективні або неможливі. Евристики - це прості, інтуїтивні правила або стратегії, які не гарантують оптимального рішення, але надають швидке та просте виконання. Вони є особливо корисними в великих та складних проблемах оптимізації, де прямі методи можуть бути надто часозатратними або навіть неможливими для виконання.

ГА є прикладом методу евристичної оптимізації. Вони використовують біологічно натхненні стратегії, такі як мутація, схрещування та вибір, щоб шукати оптимальні або наближено оптимальні рішення для складних проблем.

1.2 Прикладні задачі генетичних алгоритмів

ГА використовуються в широкому спектрі дисциплін та в різноманітних прикладних задачах. Вони особливо ефективні в задачах, де простір пошуку великий або де простір пошуку має складну структуру, яку важко аналізувати аналітично. Наприклад, вони можуть бути використані для вирішення проблем оптимізації, таких як планування виробництва, планування розкладу, управління інвентарем, і розв'язування проблем транспортної логістики. Вони також можуть бути використані для розв'язання проблем оптимізації в науці та інженерії, включаючи дизайн машин, автоматизацію проектування, моделювання і прогнозування.

Генетичні алгоритми (ГА) широко використовуються в різних областях застосування. Ось декілька прикладів:

1. **Транспортна логістика**: Як уже було згадано, ГА можуть використовуватися для вирішення задач маршрутизації, таких як задача комівояжера або задача маршрутизації вантажівок. Наприклад, компанії, які займаються доставкою, можуть використовувати ГА для оптимізації маршрутів доставки з метою мінімізації витрат на паливо або часу доставки.
2. **Біоінформатика**: ГА можуть використовуватися для вирішення різних задач в біоінформатиці, включаючи прогнозування структури білка, вирішення задачі вирізання ДНК та інших.
3. **Машинне навчання та штучний інтелект**: ГА використовуються для вибору оптимального набору параметрів моделей машинного навчання, для автоматичного генерування коду, в задачах підбору оптимальних архітектур нейронних мереж та інших задачах оптимізації.

4. **Контроль та оптимізація промислових процесів**: ГА можуть використовуватися для оптимізації промислових процесів, таких як планування виробництва, управління запасами або оптимізація процесів енерговикористання.
5. **Дизайн і інженерія**: ГА використовуються для автоматичного проектування, наприклад, для оптимізації форми та структури об'єктів з метою максимізації їхньої ефективності.

1.3 Генетичні алгоритми в логістиці

Транспортна логістика (ТЛ) - це важлива область, де генетичні алгоритми можуть знайти широке застосування. Логістика включає рух товарів від місця виробництва до місця споживання, а це включає такі процеси, як транспортування, складування, управління запасами, планування маршрутів і т.д. Застосування генетичних алгоритмів в логістиці може включати оптимізацію маршрутів для зменшення вартості перевезення, планування розкладу для максимізації ефективності складу, або вирішення складних задач, таких як проблема комівояжера.

Приклади ГА в логістиці

1. **Задача комівояжера (TSP)**: це класична задача оптимізації, в якій необхідно знайти найкоротший можливий маршрут, який відвідує кожне місто (або місце) один раз і повертається до вихідного міста. ГА можуть бути використані для ефективного пошуку розв'язків для TSP.
2. **Задача маршрутизації вантажівок (VRP)**: в цій задачі необхідно планувати маршрути для кількох вантажівок, які доставляють товари від центру розподілу до різних місць. ГА також можуть бути використані для ефективного розв'язання VRP.
3. **Задача маршрутизації зі збором та доставкою (Pickup and Delivery Problem, PDP)**: Це є розширенням Задачі маршрутизації вантажівок, де крім місць доставки, є також місця збору вантажів. ГА можуть бути

використані для розв'язання PDP, мінімізуючи загальний час роботи вантажівок та/або загальну відстань, яку вони проїжджають.

4. **Задача маршрутизації з часовими обмеженнями (Vehicle Routing Problem with Time Windows, VRPTW)**: В цій задачі вантажівки повинні відвідувати різні місця в межах заданих часових вікон. ГА можуть допомогти вирішити VRPTW, оптимізуючи маршрути вантажівок з урахуванням часових обмежень.
5. **Задача планування маршрутів для мультимодальних перевезень**: Це складніший варіант задачі маршрутизації, де товари можуть бути перевезені різними видами транспорту (наприклад, вантажівками, потягами, кораблями, літаками). ГА можуть бути використані для оптимізації маршрутів в мультимодальних логістичних мережах, мінімізуючи загальні витрати або час перевезення.

Багато великих логістичних компаній, таких як DHL, FedEx і UPS, використовують алгоритми, базовані на ГА, для планування та оптимізації маршрутів доставки.

Переваги використання ГА в логістиці:

1. **Здатність до глобальної оптимізації**: ГА добре працюють для вирішення складних задач оптимізації, де є багато локальних мінімумів, але необхідно знайти глобальний мінімум.
2. **Робота з багатооб'єктними задачами**: ГА можуть бути адаптовані для роботи з багатооб'єктними задачами, де необхідно оптимізувати кілька цілей одночасно.
3. **Паралельні обчислення**: ГА можуть ефективно використовувати паралельні обчислення, оскільки багато роботи може бути розподілено між різними обчислювальними одиницями.

Недоліки використання ГА в логістиці

1. **Час обчислень**: Залежно від розміру задачі, ГА можуть бути обчислювально важкими і потребувати значних ресурсів обчислень.
2. **Не гарантія знайти абсолютно оптимальний розв'язок**: ГА є методами приблизної оптимізації, тому вони можуть не завжди знайти абсолютно оптимальний розв'язок.
3. **Потреба в налаштуванні**: ГА мають ряд параметрів, таких як розмір популяції, ймовірність мутації, які потребують налаштування для кожної конкретної задачі.

2. Огляд генетичних алгоритмів

2.1 Історія генетичних алгоритмів

Генетичні алгоритми були розроблені в 1960-х роках в рамках більш широкого дослідження в області штучного життя. Ці алгоритми були створені в рамках пошуку нових підходів до розв'язання складних оптимізаційних проблем. Вони були натхненні процесами природного відбору, в яких найкращі особини виживають та передають свої гени наступним поколінням.

Саме цей механізм був запозичений в генетичних алгоритмах, де потенційні розв'язки задачі представлені в формі хромосом або структур, які можна мутувати та перехрещувати для створення нових розв'язків. З часом найкращі розв'язки "виживають" і передають свої "гени" наступним поколінням розв'язків.

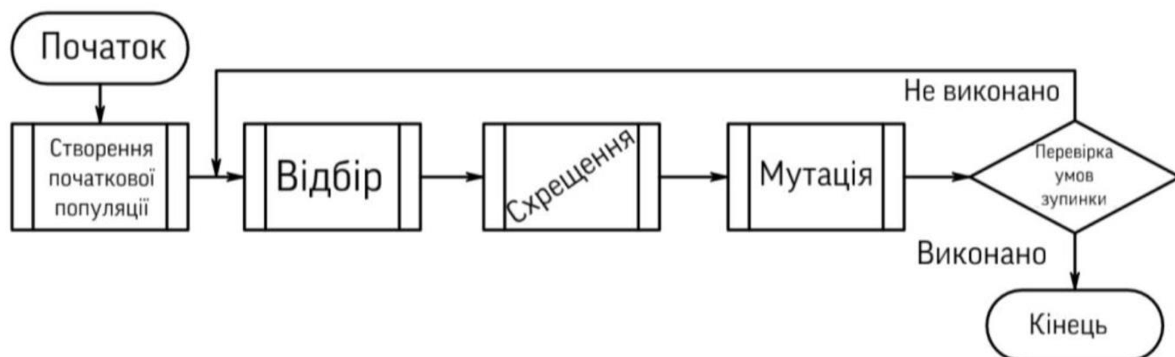
2.2 Структура генетичних алгоритмів

В основі генетичних алгоритмів лежить концепція еволюції в природі, де найкраще пристосовані індивіди виживають і передають свої гени наступним поколінням. Таким чином, генетичні алгоритми використовують стохастичний пошук для знаходження оптимальних або близьких до оптимальних розв'язків в складних просторах пошуку.

Основні елементи генетичного алгоритму включають популяцію потенційних розв'язків, функцію пристосування для оцінки якості кожного розв'язку, а також оператори мутації та перехрещування для модифікації розв'язків.

Процес роботи генетичного алгоритму включає наступні кроки:

1. **Ініціалізація**: Початкова популяція розв'язків (індивідів) генерується випадковим чином. Кожний розв'язок представляється хромосомою, яка є строкою бінарних, цілих або реальних чисел.
2. **Оцінка**: Кожному розв'язку призначається величина пристосованості, яка вимірює якість розв'язку. Це відбувається шляхом застосування функції пристосованості до кожного розв'язку. Функція пристосованості відображає особливості конкретної задачі оптимізації.
3. **Селекція**: Індивіди вибираються для участі в операторах генетичного перехрещування та мутації на основі їх пристосованості. Частіше вибираються більш пристосовані індивіди.
4. **Перехрещування та мутація**: Перехрещування (або рекомбінація) комбінує гени двох батьків, щоб створити нових потомків. Мутація випадково змінює гени в хромосомі.
5. **Заміна**: Нова популяція, що складається з потомків, замінює поточну популяцію.



Головними компонентами генетичного алгоритму є функція пристосованості, оператори селекції, перехрещування та мутації, а також

стратегія заміни. Усі ці компоненти можна налаштувати для покращення ефективності генетичного алгоритму в конкретній задачі оптимізації.

Слід зазначити, що генетичні алгоритми є глобальними методами пошуку, що означає, що вони здатні знаходити оптимальні розв'язки в просторах пошуку з багатьма локальними оптимумами, де детерміновані методи пошуку часто провалилися. Однак вони також вимагають значних обчислювальних ресурсів, особливо для великих задач.

2.2.1 Популяція

Популяція в контексті генетичних алгоритмів відноситься до групи потенційних рішень для даної оптимізаційної проблеми. Ці рішення, також відомі як "особини" або "хромосоми", можуть бути представлені в різних форматах, таких як бінарні рядки, дійсні числа, чи інші структури даних, в залежності від вимог задачі. Кожна особина представляє потенційне рішення проблеми і має власну "фітнес-функцію", яка оцінює її доброту чи придатність.

Популяція має важливе значення в генетичних алгоритмах. З одного боку, вона дає можливість для більш широкого пошуку простору рішень, оскільки кожна особина може експлорувати різні точки в просторі рішень. З іншого боку, еволюційні операції, такі як перехрещення і мутація, виконуються в межах популяції, сприяючи різноманітності і відбору найкращих рішень.

Реалізація популяції залежить від природи проблеми і способу кодування рішень. Деякі алгоритми можуть використовувати статичну популяцію, де розмір популяції залишається сталим протягом всього процесу еволюції, тоді як інші можуть використовувати динамічну популяцію, де розмір популяції може змінюватися в часі.

Моделі населення в генетичних алгоритмах також можуть варіюватися. У найпростішій моделі, відомій як панміксія, всі особини мають однакову можливість бути обрані для відбору і кросоверу. У більш складних моделях, таких як острівні моделі або моделі сусідства, особини поділяються на декілька підпопуляцій, і відбір і кросовер обмежуються в межах цих підпопуляцій.

2.2.2 Фітнес-функція

Фітнес-функція в генетичних алгоритмах відіграє вирішальну роль, оскільки вона визначає, наскільки "добре" потенційне рішення задовольняє критерії проблеми. Вона є мірою "придатності" кожної особини в популяції, враховуючи її якість чи ефективність в контексті вирішуваної проблеми.

Фітнес-функція потрібна для проведення відбору в генетичних алгоритмах. Вона є основою для визначення, які особини будуть вибрані для створення наступного покоління через генетичні операції, такі як перехрещення та мутації. Вища вартість фітнес-функції зазвичай відповідає кращому рішенню проблеми, а тому такі особини мають більшу ймовірність бути обрані.

Методи реалізації фітнес-функції варіюються в залежності від характеру проблеми. Фітнес-функція повинна бути добре визначеною та спроможною чітко оцінити якість рішення. Наприклад, для проблеми маршрутизації, фітнес-функція може вимірювати загальну відстань маршруту чи час, необхідний для завершення маршруту.

Додатково, важливо відзначити, що вибір фітнес-функції може значно вплинути на ефективність генетичного алгоритму. Неадекватна чи слабо визначена фітнес-функція може призвести до невдалої конвергенції або надмірної адаптації.

2.2.3 Представлення генотипу

Генотип в генетичних алгоритмах - це внутрішнє представлення особини, що використовується в процесі еволюції. Він представляє собою стрічку даних (нерідко називану "хромосомою"), яка кодує інформацію про "траїт" особини. Перетворення генотипу в "фенотип" - зовнішнє представлення особини, що використовується для оцінки фітнес-функції, - є критичною складовою генетичних алгоритмів.

Представлення генотипу визначає структуру пошукового простору і може суттєво вплинути на ефективність генетичного алгоритму. Є кілька звичайних методів представлення генотипу:

1. Двійкове представлення - це найпростіше і найбільш традиційне представлення. Кожен ген в хромосомі представлений двійковим числом (0 або 1). Двійкові хромосоми прості в реалізації та інтерпретації, але можуть бути недостатніми для деяких складних проблем.
2. Реальне представлення - це більш гнучкий метод, де гени представлені реальними числами. Це корисно для проблем оптимізації в реальному числовому просторі, але вимагає більш складних операцій перехрещення та мутації.
3. Цілочисельне представлення - схоже на реальне представлення, але замість реальних чисел використовуються цілі числа. Це може бути корисним для проблем, де рішення мають бути цілими числами.
4. Представлення перестановок - особливо корисне для задач оптимізації, де порядок елементів має значення, наприклад, в задачі комівояжера. Генотип в цьому випадку представляє собою перестановку елементів.

Вибір конкретного представлення генотипу залежить від специфіки задачі. Найкраще вибирати таке представлення, яке найбільше відповідає природі проблеми і сприяє збіжності генетичного алгоритму.

2.2.4 Селекція

Селекція батьків - це важливий етап в процесі генетичного алгоритму. Вона забезпечує вибір особин з популяції для участі в операторах перехрещення та мутації з метою створення нового покоління. Процес селекції відіграє важливу роль в генетичному алгоритмі, оскільки він визначає, яким чином будуть здійснюватися перехрещення та мутації.

Є декілька різних методів селекції, кожен з яких має свої переваги та недоліки:

1. **Селекція рулетки**: цей метод селекції базується на принципі "виживання найсильніших". Кожна особина отримує слот на "рулетці" пропорційно до свого фітнес-значення, і потім випадково обираються

батьки для наступного покоління. Головна перевага цього методу полягає в тому, що він підтримує різноманітність популяції, але в той же час, він може бути підданий "проблемі прематурної конвергенції", коли декілька надзвичайно високих особин домінують над популяцією.

2. **Турнірна селекція**: цей метод включає вибір випадкового підмножини з популяції і вибір найкращого елемента з цієї підмножини. Цей процес повторюється для кожного батька, який потрібно вибрати. Турнірна селекція має просту реалізацію, здатну підтримувати різноманітність популяції та уникнути проблеми прематурної конвергенції.
3. **Селекція рангу**: в цьому методі особини ранжуються за їхніми фітнес-значеннями, і вибір здійснюється на основі цього рангу, а не безпосередньо фітнес-значення. Цей метод зменшує тиск селекції на особини з високим фітнесом, що допомагає уникнути проблеми прематурної конвергенції. Вибір методу селекції залежить від специфіки задачі, структури популяції та потреби в різноманітності популяції.

2.3 Види генетичних алгоритмів

Існує багато варіантів генетичних алгоритмів, які відрізняються за типом представлення розв'язків, операторами мутації та перехрещування, стратегією селекції та механізмом заміни.

Ось найпопулярніші ГА:

1. **Стандартний генетичний алгоритм (SGA):** Це основна форма ГА, що включає операції селекції, схрещування (кросоверу) та мутації. SGA може бути добрим вибором для початку, оскільки він простий для розуміння та імплементації.
2. **Мультиоб'єктний генетичний алгоритм (MOGA)**: Якщо задача вимагає оптимізації декількох критеріїв одночасно (наприклад,

мінімізації витрат та максимізації ефективності), то MOGA може бути відповідним вибором.

3. **Генетичний алгоритм з паралельними популяціями**: Цей варіант ГА використовує декілька популяцій, які працюють паралельно та обмінюються індивідами з часом. Це може бути корисно для покращення швидкості пошуку та збереження різноманітності популяції.

РОЗДІЛ 2. ПРАКТИЧНА ЧАСТИНА

1 Постановка задачі

Розглядаємо задачу оптимізації маршруту доставки товарів за допомогою генетичних алгоритмів. Задача включає дві основні складові: доставка товарів до певних точок призначення та максимізація використання вмістимості транспорту. Вони будуть відрізнятись за порядком значень, які присвоюються різним факторам, таким як вмістимість автомобіля, кількість товару для доставки та відстань між точками призначення.

Також буде враховуватись різна кількість величин, що будуть розглядатись, таких як кількість водіїв, кількість точок призначення та обсяг товару для перевезення. Від цих параметрів залежатиме швидкість та точність роботи розробленого алгоритму.

Поставлена задача буде розв'язуватися за допомогою різних алгоритмів, дозволяючи оцінити різницю між їх швидкістю та точністю в залежності від вхідних даних. Це дозволить вибрати найбільш ефективний алгоритм для вирішення конкретної задачі в контексті транспортної логістики.

1.1 Огляд програмного забезпечення

Для розробки та тестування швидкодії алгоритмів використовується ноутбук Xiaomi Mi Notebook Pro 15.6 з процесором Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 3.40 GHz та оперативною пам’яттю 8 GB. Алгоритми розроблені у середовищі програмування PyCharm 2022.1, мова розробки – Python 3.9. Використовуються такі стандартні бібліотеки Python, як random, copy, itertools, numpy, math.

1.2 Вхідні дані

Для нашого прикладу, розглянемо місто Дніпро, де ми будемо виконувати доставку до таких точок, як: ж/м Покровський, Діївка, Перемога, Тополь, Дафі,

Лівобережний.

Кожна з цих точок має різний обсяг товарів для доставки. Водночас маємо наявний парк вантажних автомобілів з різними водіями та вмістимостями.

На основі цього набору даних будемо створювати хромосоми, що відображають маршрут водія, і будемо використовувати генетичний алгоритм для пошуку найбільш ефективних маршрутів для виконання цієї задачі.

Ціль - максимізувати ефективність використання автомобілів і мінімізувати час доставки.

2 Реалізація

2.1 Структура вхідних даних

Вхідні дані будуть представлені у вигляді двох основних структур: графу точок доставки та списку автомобілів.

Граф точок доставки можна представити у вигляді матриці відстаней. Наприклад:

```
# Матриця відстаней між точками (у кілометрах)
distance_matrix = [
    [0, 7, 5, 9, 2, 3],
    [7, 0, 2, 3, 8, 10],
    [5, 2, 0, 1, 6, 4],
    [9, 3, 1, 0, 7, 3],
    [2, 8, 6, 7, 0, 5],
    [3, 10, 4, 3, 5, 0]
]
# Точки: Ж/м Покровський, Діївка, Перемога, Тополь, Дафі, Лівобережний
```

Список автомобілів з параметрами може бути представлений у вигляді масиву об'єктів:

```
vehicles = [
    {"model": "Renaul Kangoo 2007", "average_speed": 60, "capacity": 800},
    {"model": "Ford Transit 2015", "average_speed": 80, "capacity": 1200},
    {"model": "Renault Kangoo 2021", "average_speed": 70, "capacity": 800},
    {"model": "Renault Megan 2010", "average_speed": 90, "capacity": 600},
    {"model": "Volkswagen Transporter 2020", "average_speed": 110, "capacity": 1500}
]
```

Для кожної точки вкажемо товари, які потрібно доставити:

```
delivery_points = {
    "Ж/м Покровський": {"goods": 500},
    "Діївка": {"goods": 700},
    "Перемога": {"goods": 800},
    "Тополь": {"goods": 600},
    "Дафі": {"goods": 400},
    "Лівобережний": {"goods": 1200}
}
```

Ці дані будуть служити вхідними для нашого генетичного алгоритму для розв'язання задачі доставки.

2.2 Алгоритм повного перебору

Алгоритм повного перебору є одним із простих і базових методів пошуку рішень. Цей алгоритм використовується, коли потрібно перебрати всі можливі комбінації заданих елементів. Перевагою цього алгоритму є те, що він забезпечує 100% гарантію знаходження найкращого рішення, оскільки перебирається абсолютно всі можливі варіанти. Проте, основним недоліком є велика часова складність, особливо при великій кількості елементів, що пов'язано з тим, що кількість перебраних комбінацій зростає факторіально.

Для таких вихідних даних алгоритм повного перебору можна розбити на чотири основні кроки:

1. Генерація усіх можливих комбінацій маршрутів: Враховуючи, що маршрут визначається набором точок, до яких потрібно відвідати, ми

генеруємо всі можливі перестановки цих точок. Це означає, що для точок A, B і C ми маємо такі можливі маршрути: {A, B, C}, {A-B, C}, {A-C, B}, {B-C, A}, {A-B-C}, {A-C-B}, {B-A-C}, {B-C-A}, {C-A-B}, {C-B-A}. Щоб згенерувати усі можливі комбінації маршрутів з урахуванням порядку точок у маршруті, можна використати комбінацію itertools.combinations та itertools.permutations в Python.

2. Генерація усіх можливих пар водій-маршрут: Далі ми генеруємо всі можливі пари водій-маршрут. Тобто для кожного маршруту, який ми згенерували на попередньому кроці, ми створюємо пару з кожним водієм.

```
def generate_all_routes(points):
    all_routes = []
    for r in range(1, len(points) + 1):
        combinations = list(itertools.combinations(points, r))
        for combination in combinations:
            permutations = list(itertools.permutations(combination, r))
            for permutation in permutations:
                all_routes.append(list(permutation))
    return all_routes


def calculate_travel_time(route, distance_matrix, points, vehicle):
    total_distance = 0
    for i in range(len(route) - 1):
        start = points.index(route[i])
        end = points.index(route[i + 1])
        print("For vehicle: " + str(vehicle['model']) + " distance: " + str(distance_matrix[start][end]))
        total_distance += distance_matrix[start][end]
    return total_distance / vehicle['average_speed']
```

3. Розрахунок максимального часу доставки для кожного набору: Для кожного набору водій-маршрут ми розраховуємо час, який потрібен для доставки. Це робиться шляхом додавання відстаней між точками на маршруті та ділення загальної відстані на середню швидкість водія. Важливо відзначити, що ми розглядаємо максимальний час доставки, оскільки ми хочемо знайти найшвидший можливий час доставки для всіх маршрутів.
4. Вибір набору маршрутів з найменшим максимальним часом доставки: Нарешті, з усіх можливих наборів водій-маршрут ми обираємо той, який має найменший максимальний час доставки.

```python
# Основний алгоритм
def full_search_algorithm(vehicles, delivery_points, distance_matrix):
    points = list(delivery_points.keys())
    all_possible_routes = generate_all_routes(points)

    minimum_time = float('inf')
    optimal_schedule = {}

    for routes in itertools.combinations(all_possible_routes, len(vehicles)):
        # check if each point is included in the routes
        flattened_routes = [point for route in routes for point in route]
        if sorted(flattened_routes) != sorted(points):
            continue

        # check if any vehicle is overloaded
        overloaded = False
        for vehicle, route in zip(vehicles, routes):
            total_goods = sum([delivery_points[point]['goods'] for point in route])
            if total_goods > vehicle['capacity']:
                overloaded = True
                break
        if overloaded:
            continue

        # calculate the maximum time taken by any vehicle
        times = [calculate_travel_time(route, distance_matrix, points, vehicle)
                 for vehicle, route in zip(vehicles, routes)]
        max_time = max(times)

        # if this schedule is better than the current best, update the optimal schedule
        if max_time < minimum_time:
            minimum_time = max_time
            optimal_schedule = dict(zip([v['model'] for v in vehicles], routes))

    return optimal_schedule, minimum_time
```

Важливо відзначити, що цей алгоритм повного перебору має велику обчислювальну складність, особливо якщо кількість точок доставки або водіїв велика. Більш того, цей алгоритм не враховує обмеження на максимальну вмістимість автомобілів, що може вплинути на можливість виконання деяких маршрутів.

Алгоритм повного перебору, який ми тут використовуємо, має досить велику обчислювальну складність. Його складність складається з кількох частин.

1. Спочатку ми генеруємо всі можливі маршрути для доставки. Ця частина алгоритму має факторіальну складність, тому що ми генеруємо всі можливі перестановки для кожної комбінації точок доставки. Таким чином, складність цього етапу можна оцінити як O(n!), де n - кількість точок доставки.
2. Далі ми генеруємо всі можливі способи призначити маршрути автомобілям. Ця частина також має факторіальну складність, тому що ми генеруємо всі можливі перестановки для списку маршрутів. Отже, складність цього етапу теж можна оцінити як O(m!), де m - кількість маршрутів.
3. Нарешті, для кожної можливої комбінації ми розраховуємо максимальний час доставки, що вимагає O(n) часу.

Таким чином, загальна складність алгоритму досить велика і залежить від кількості точок доставки та кількості автомобілів. Вона може бути оцінена як O(n! * m! * n), що є досить великою для навіть відносно невеликої кількості точок доставки і автомобілів.

Це означає, що цей алгоритм не підходить для великих проблем, але для невеликих проблем він може знайти оптимальне рішення.

2.2.1 Часова оцінка

Для набору з 4х автомобілів та 4х точок доставки маємо такий результат:

```
For vehicle: Renaul Kangoo 2007 distance: 100
For vehicle: Ford Transit 2015 distance: 50
For vehicle: Renault Kangoo 2021 distance: 80
For vehicle: Renault Megan 2010 distance: 70
Кількість ітерацій: 635376
Оптимальний графік:
Renaul Kangoo 2007: ['Ж/м Покровський']
Ford Transit 2015: ['Діївка']
Renault Kangoo 2021: ['Перемога']
Renault Megan 2010: ['Тополь']
Мінімальний час: 16.666666666666668
--- Час виконання: 1.2143373489379883 секунд ---
```

Для набору з 4х автомобілів та 5 точок доставки маємо такий результат:

```
For vehicle: Renault Megan 2010 distance: 100
Кількість ітерацій: 456326325
Оптимальний графік:
Renaul Kangoo 2007: ['Ж/м Покровський']
Ford Transit 2015: ['Діївка']
Renault Kangoo 2021: ['Дафі']
Renault Megan 2010: ['Перемога', 'Тополь']
Мінімальний час: 1.6666666666666667
--- Час виконання: 1253.8111321926117 секунд ---
```

Для набору з 3х автомобілів та 5 точок:

```
For vehicle: Renault Kangoo 2021 distance: 80
For vehicle: Renault Kangoo 2021 distance: 50
For vehicle: Renaul Kangoo 2007 distance: 90
For vehicle: Ford Transit 2015 distance: 70
For vehicle: Ford Transit 2015 distance: 100
For vehicle: Renault Kangoo 2021 distance: 50
For vehicle: Renault Kangoo 2021 distance: 80
For vehicle: Renaul Kangoo 2007 distance: 90
For vehicle: Ford Transit 2015 distance: 70
For vehicle: Ford Transit 2015 distance: 100
For vehicle: Renault Kangoo 2021 distance: 80
For vehicle: Renault Kangoo 2021 distance: 50
Кількість ітерацій: 5668650
Оптимальний графік:
Renaul Kangoo 2007: ['Перемога']
Ford Transit 2015: ['Ж/м Покровський', 'Тополь']
Renault Kangoo 2021: ['Діївка', 'Дафі']
Мінімальний час: 2.125
--- Час виконання: 22.558861255645752 секунд ---
```

Для набору з 3х автомобілів та 6 точок доставки маємо такий результат:

```
Кількість ітерацій: 1245342820
Оптимальний графік:
Renaul Kangoo 2007: ['Ж/м Покровський', 'Лівобережний']
Ford Transit 2015: ['Діївка', 'Дафі']
Renault Kangoo 2021: ['Перемога', 'Тополь']
Мінімальний час: 2.142857142857143
--- Час виконання: 3377.6878395080566 секунд ---
```

2.3 Генетичний алгоритм

2.3.1 Опис моделей

Модель хромосоми з водіями та точками доставки може бути представлена у вигляді набору об'єктів, кожен з яких представляє собою водія та його маршрут доставки. Кожен об'єкт містить наступну інформацію:

1. Водій: Ідентифікатор водія, може бути унікальним числовим або текстовим значенням. Також можуть бути вказані додаткові параметри водія, такі як середня швидкість руху транспортного засобу, вмістимість автомобіля та інші характеристики.
2. Маршрут доставки: Це список точок доставки, які водій повинен відвідати у певному порядку. Кожна точка представлена унікальним ідентифікатором або назвою. Для кожної точки можуть бути вказані додаткові параметри, такі як вага товарів, час доставки або відстань від попередньої точки.

Модель хромосоми може бути представлена у вигляді масиву об'єктів, де кожен об'єкт представляє окремого водія та його маршрут доставки. Порядок об'єктів в масиві відповідає порядку, в якому водії будуть виконувати свої маршрути.

Обираючи модель хромосоми, необхідно враховувати особливості самої задачі і забезпечувати зручність та ефективність подальшої обробки та маніпулювання хромосомами. Модель хромосоми з водіями та точками доставки має свої переваги:

1. Зручне представлення даних: Вона дозволяє явно вказати кожного водія та його маршрут доставки. Це допомагає легко розрізняти між водіями та точками, а також знаходити необхідну інформацію про кожного водія та його доставки.
2. Гнучкість: Модель дозволяє працювати з різними кількостями водіїв та точок доставки, а також змінювати маршрути та склад водіїв. Вона може адаптуватися до змін у вихідних даних без необхідності внесення великої кількості змін в структуру даних.

3. Легкість маніпуляцій: Завдяки моделі хромосоми з об'єктами водіїв та точок доставки, ми можемо легко здійснювати операції додавання, видалення та зміни точок доставки для кожного водія. Це дозволяє легко змінювати розклади та склад водіїв для отримання оптимального розв'язку.
4. Чіткість та зрозумілість: Модель хромосоми з водіями та точками доставки є інтуїтивно зрозумілою та легко читабельною. Вона дозволяє чітко виразити структуру та зв'язки між водіями та точками доставки, що спрощує розуміння інших осіб та спільну роботу з кодом.

Ці переваги роблять модель хромосоми з водіями та точками доставки зручною та ефективною для розв'язання задачі організації маршрутів доставки. Однак, вибір моделі хромосоми може залежати від конкретних вимог та обмежень задачі, тому важливо зробити аналіз та вибір моделі, яка найкраще відповідає поставленим завданням.

```
class Chromosome:
    def __init__(self, drivers):
        self.drivers = drivers

    def __repr__(self):
        return repr(self.drivers)


class Driver:
    def __init__(self, name, vehicle, delivery_points):
        self.name = name
        self.vehicle = vehicle
        self.delivery_points = delivery_points

    def __repr__(self):
        return f"{self.name} ({self.vehicle}): {self.delivery_points}"
```

2.3.2 Функція фітнесу

Для моделі хромосоми з водіями та точками доставки можна реалізувати функцію фітнесу, яка враховуватиме кількість водіїв, загальний час доставки та використання вмістимості автомобілів. Основною метою функції фітнесу є оцінка якості розв'язку, тобто визначення того, наскільки добре дана хромосома задовольняє вимогам задачі.

Для досягнення найшвидшої доставки ви можете використати функцію фітнесу, яка буде максимізувати швидкість доставки або мінімізувати час доставки. Оскільки менший час доставки є бажаним результатом, то варто використати функцію фітнесу, яка буде мінімізувати час доставки.

Нижче наведено приклад коду для функції фітнесу, яка мінімізує час доставки:

```
def calculate_fitness_score(chromosome):
    total_time = 0
    for route in chromosome.routes:
        total_time += calculate_delivery_time(route, chromosome.driver.average_speed)
    return total_time


def calculate_delivery_time(route, average_speed):
    # Розрахунок часу доставки для маршруту
    total_distance = 0
    for i in range(len(route) - 1):
        current_point = delivery_point_index(route[i])
        next_point = delivery_point_index(route[i + 1])
        total_distance += distance_matrix[current_point][next_point]
    # Повернення часу доставки, враховуючи швидкість водія та відстань
    return total_distance / average_speed
```

Функція calculate_fitness_score обчислює фітнес-оцінку для заданої хромосоми. У цьому випадку, фітнес-оцінка представляє собою загальний час доставки для всіх маршрутів, що містяться в хромосомі.

Функція calculate_delivery_time використовується для обчислення часу доставки для окремого маршруту. Вона приймає маршрут та середню швидкість водія як вхідні параметри. Спочатку вона обчислює загальну відстань для маршруту, проходячи по всіх точках маршруту і додаючи відстані між ними з

матриці відстаней. Потім час доставки обчислюється як відношення загальної відстані до середньої швидкості водія.

Функція calculate_fitness_score проходить по всіх маршрутах в хромосомі та обчислює загальний час доставки для всіх маршрутів, викликаючи calculate_delivery_time для кожного маршруту з відповідною середньою швидкістю водія. Загальний час доставки стає значенням фітнес-оцінки і повертається як результат функції.

Така функція фітнесу дозволяє оцінити якість розкладу та маршруту доставки на основі загального часу доставки. Мінімізація функції фітнесу сприяє знаходженню оптимальних розкладів та маршрутів, які забезпечують мінімальний час доставки.

2.3.3 Еволюція

Функція evolve(population, mutation_rate) виконує еволюційний процес, використовуючи оператори селекції, кросовера і мутації. У цій функції нова популяція створюється на основі поточної популяції.

```
def evolve(population, mutation_rate):
    # Застосування операторів селекції, кросовера та мутації
    new_population = []
    while len(new_population) < len(population):
        parent1 = roulette_wheel_selection(population)
        parent2 = roulette_wheel_selection(population)
        child1, child2 = crossover(parent1, parent2)
        child1 = mutate(child1, mutation_rate)
        child2 = mutate(child2, mutation_rate)
        new_population.append(child1)
        new_population.append(child2)
    return new_population
```

Процес складається з наступних етапів:

1. Вибір батьків за допомогою оператора селекції. Один з можливих варіантів селекції - рулетковий відбір, де кращі особини мають більшу ймовірність бути вибраними.

2. Застосування кросовера до вибраних батьків для створення нових потомків.
3. Застосування мутації до потомків з заданою ймовірністю.
4. Додавання новостворених потомків до нової популяції.
5. Повторення цих кроків, поки нова популяція не досягне бажаного розміру.

2.3.4 Мутація

Ця функція виконує оператор мутації для внесення випадкових змін у маршрути хромосоми. Мутація дозволяє вносити різноманітність у популяцію і допомагає уникнути застрягання в локальних оптимумах.

```
def mutate(chromosome, mutation_rate):
    # Оператор мутації для внесення випадкових змін у маршрути
    mutated_routes = []
    for route in chromosome.routes:
        if random.random() < mutation_rate:
            random.shuffle(route)
        mutated_routes.append(route)
    mutated_chromosome = Chromosome(chromosome.driver, mutated_routes)
    return mutated_chromosome
```

Опис алгоритму виконання функції:

1. Створення порожнього списку mutated_routes, в якому будуть зберігатись мутовані маршрути. Для кожного маршруту в хромосомі:
   1.1. Перевірка, чи виконується умова мутації за допомогою генератора випадкових чисел random.random() < mutation_rate. Якщо умова виконується, то мутація застосовується до цього маршруту.
   1.2. Мутація полягає в перемішуванні точок маршруту за допомогою функції random.shuffle(route). Це дозволяє змінювати послідовність доставки товарів.
   1.3. Додавання мутованого маршруту до списку mutated_routes.
2. Створення нової хромосоми mutated_chromosome з мутованими маршрутами і тим самим водієм, який був у вихідній хромосомі.

3. Повернення мутованої хромосоми mutated_chromosome.

Ця функція дозволяє внести різноманітність у маршрути хромосоми, що допомагає знаходити оптимальні рішення в еволюційному процесі.

2.3.5 Кросовер

Ця функція виконує оператор кросовера для створення двох потомків на основі двох батьківських хромосом.

```
def crossover(parent1, parent2):
    # Оператор кросовера для створення двох потомків
    crossover_point = random.randint(1, len(parent1.routes) - 1)
    child1_routes = parent1.routes[:crossover_point] + parent2.routes[crossover_point:]
    child2_routes = parent2.routes[:crossover_point] + parent1.routes[crossover_point:]
    child1 = Chromosome(parent1.driver, child1_routes)
    child2 = Chromosome(parent2.driver, child2_routes)
    return child1, child2
```

Кросовер є ключовим етапом в еволюційному процесі, де генетичний матеріал обмінюється між батьківськими хромосомами для створення нових комбінацій.

Опис алгоритму виконання функції:

1. Вибір випадкової точки перетину crossover_point, яка визначає місце розриву між маршрутами батьківських хромосом. Ця точка вибирається випадковим чином в межах від 1 до len(parent1.routes) - 1. Такий діапазон виключає початкову та кінцеву точку маршруту для забезпечення правильності кросовера.
2. Створення двох маршрутів потомків child1_routes і child2_routes шляхом комбінування маршрутів батьків. Для child1_routes береться частина маршруту з першого батька до точки перетину, а потім додається частина маршруту з другого батька починаючи з точки перетину. Аналогічно для child2_routes - частина маршруту з другого батька до точки перетину, а потім частина маршруту з першого батька починаючи з точки перетину.

3. Створення двох потомків child1 і child2 на основі маршрутів child1_routes і child2_routes, використовуючи водіїв батьківських хромосом.
4. Повернення двох потомків child1 і child2.

Ця функція допомагає створити нові комбінації маршрутів на основі батьківських хромосом, що сприяє рознообразність і пошук оптимальних рішень у еволюційному процесі.

2.3.6 Селекція

2.3.6.1 Рулетковий відбір

Ця функція виконує відбір батьків за допомогою рулеткового відбору, що базується на ймовірності обрання хромосоми пропорційно її фітнес-оцінці.

```
def roulette_wheel_selection(population):
    # Відбір батьків за допомогою рулеткового відбору
    total_fitness = sum(calculate_fitness_score(chromosome) for chromosome in population)
    random_fitness = random.uniform(0, total_fitness)
    cumulative_fitness = 0
    for chromosome in population:
        cumulative_fitness += calculate_fitness_score(chromosome)
        if cumulative_fitness >= random_fitness:
            return chromosome
```

Опис алгоритму виконання функції:

1. Обчислення загальної фітнес-оцінки для всіх хромосом у популяції population шляхом сумування фітнес-оцінок для кожної хромосоми. Це дає загальну міру пристосованості популяції до вирішення задачі.
2. Генерація випадкового числа random_fitness в діапазоні від 0 до загальної фітнес-оцінки total_fitness. Це число використовується для випадкового вибору хромосоми.
3. Обчислення накопиченої фітнес-оцінки cumulative_fitness, яка починається зі значення 0.
4. Проходження через кожну хромосому в популяції population і додавання її фітнес-оцінки до cumulative_fitness. Порівняння cumulative_fitness з random_fitness для визначення обраної хромосоми.

5. Якщо cumulative_fitness стає більшим або рівним random_fitness, повернення обраної хромосоми. Це означає, що хромосома вибрана з ймовірністю, пропорційною її фітнес-оцінці.

Ця функція допомагає вибрати батьків для формування нових поколінь у процесі еволюції. Вибір здійснюється таким чином, що хромосоми з більшою фітнес-оцінкою мають вищі шанси бути обраними, проте всі хромосоми мають шанс бути вибраними залежно від їхньої пристосованості.

2.3.6.2 Турнірна селекція

Один із варіантів іншого методу селекції може бути метод турнірної селекції. Він використовується для вибору батьків шляхом проведення турніру між випадково вибраними особинами з популяції. Особина з кращою пристосованістю перемагає в турнірі і вибирається як батько або мати.

Опишемо цей метод:

```
def tournament_selection(population, tournament_size):
    participants = random.sample(population, tournament_size)
    best_participant = max(participants, key=lambda x: calculate_fitness_score(x))
    return best_participant
```

У даній реалізації методу tournament_selection спочатку вибирається випадкова підмножина tournament_size особин з популяції. Потім серед цих учасників проводиться турнір, де перемагає особина з найкращою пристосованістю. В результаті повертається особина з кращою пристосованістю.

2.3.6.3 Рангова селекція

Метод рангової селекції використовується для вибору батьків на основі їх рангу в популяції. Чим вищий ранг у особини, тим більша ймовірність вона буде вибрана.

У даній реалізації методу rank_selection спочатку популяція сортується за пристосованістю (фітнесом), потім для кожної особини вираховується ранг, а потім обчислюється ймовірність вибору особини на основі її рангу і ваги рангу rank_weights. На основі цих ймовірностей використовується функція random.choices для вибору двох батьків з популяції.

Опишемо цей метод:

```
def rank_selection(population, rank_weights):
    sorted_population = sorted(population, key=lambda x: calculate_fitness_score(x))
    ranks = range(1, len(population) + 1)
    probabilities = [rank_weights[rank - 1] for rank in ranks]
    selected_chromosomes = random.choices(sorted_population, probabilities, k=2)
    return selected_chromosomes
```

2.3.6 Результати

Опис алгоритму по пунктах:

1. Ініціалізація: Початкові дані задаються, включаючи список водіїв, точки доставки та параметри алгоритму, такі як розмір популяції, швидкість мутації та кількість поколінь.
2. Створення початкової популяції: За допомогою функції generate_initial_population створюється початкова популяція хромосом, де кожна хромосома представляє собою комбінацію водіїв та маршрутів.
3. Оцінка фітнесу: Для кожної хромосоми обчислюється значення функції фітнесу за допомогою функції calculate_fitness_score, яка враховує час доставки та інші критерії ефективності.
4. Вибір найкращих особин: За допомогою функції вибору родичів (наприклад, турнірний відбір) відбираються найкращі хромосоми для наступного покоління.
5. Створення нового покоління: За допомогою операторів розмноження (наприклад, одноточковий або двоточковий кросовер) і мутації створюються нащадки для наступного покоління.
6. Оцінка фітнесу нового покоління: Знову обчислюється функція фітнесу для новостворених хромосом.
7. Вибір найкращих рішень: За допомогою функції вибору найкращих рішень (наприклад, елітний відбір) обираються найкращі хромосоми, які перейдуть до наступного покоління.

8. Перевірка критерію зупинки: Перевіряється, чи досягнутий критерій зупинки (наприклад, максимальна кількість поколінь або збільшення фітнесу).
9. Повторення кроків 4-8: Якщо критерій зупинки не досягнутий, повторюються кроки 4-8 для створення нового покоління.
10. Вибір найкращого рішення: По завершенні алгоритму вибирається найкраща хромосома, яка представляє оптимальний розклад та маршрут доставки.

Цей алгоритм використовує генетичні оператори, такі як розмноження та мутація, для поступового покращення розв'язку протягом кількох поколінь. Під час виконання алгоритму шукається найкраще поєднання водіїв та маршрутів доставки, що призводить до знаходження оптимального розкладу та маршруту доставки.

```
# Налаштування параметрів алгоритму
population_size = 100
mutation_rate = 0.1
num_generations = 100

# Знаходження оптимального розв'язку
optimal_solution = find_optimal_solution(drivers, delivery_points, population_size, mutation_rate, num_generations)

# Виведення результатів
print("Optimal Solution:")
print(optimal_solution)
print("Delivery Time:", calculate_fitness_score(optimal_solution))

print("--- Час виконання: %s секунд ---" % (time.time() - start_time))
```

Для набору з 5ти водіїв та 6ти точок маємо такий результат. Треба звернути увагу, що кількість генерацій 100. Таким чином ми маємо дуже швидку, проте не дуже стабільну програму. Також наведені приклади з кількістю генерацій 1000 та 10000.

```
Population size:  100
Mutation rate:  0.1
Number of generations:  100
Optimal Solution:
Renault Kangoo 2021 (70): {'Ж/м Покровський': {'g
Delivery Time: 9.72857142857143
--- Час виконання: 0.884150505065918 секунд ---
```

1000 генерацій:

```
Population size:  100
Mutation rate:  0.1
Number of generations:  1000
Optimal Solution:
Renault Megan 2010 (90): {'Ж/м Покровський': {'goods
Delivery Time: 7.433333333333336
--- Час виконання: 6.910162448883057 секунд ---
```

10000 генерацій:

```
Population size:  100
Mutation rate:  0.1
Number of generations:  10000
Optimal Solution:
Volkswagen Transporter 2020 (70): {'Ж/м Покровський':
Delivery Time: 9.52857142857143
--- Час виконання: 78.01845145225525 секунд ---
```

При виконанні цього алгоритму багато раз, я помітив що результат нестабільний при кількості генерацій 100, 1000 та 10000.

Нестабільні результати можуть бути спричинені декількома факторами:

1. Випадковість: Генетичний алгоритм використовує випадковість при виборі батьків, кросовері та мутації. Це означає, що кінцевий результат може залежати від випадкових факторів. Велика випадковість може призводити до різних результатів при кожному запуску алгоритму.
2. Недостатня кількість генерацій: При недостатній кількості генерацій алгоритм може не мати достатньо часу для знаходження оптимального рішення. Генетичний алгоритм вимагає багатократних ітерацій та еволюційних кроків для поступового поліпшення рішення. Зазвичай, для складних задач рекомендується використовувати більшу кількість генерацій.
3. Недостатня розмірність популяції: Мала кількість хромосом у популяції може призвести до недостатньої різноманітності та збіднення простору

рішень. Це може обмежити можливість алгоритму знайти оптимальне рішення.

4. Неправильні параметри алгоритму: Неправильно вибрані параметри алгоритму, такі як розмір популяції, ймовірності кросовера та мутації, можуть призвести до нестабільних результатів. Важливо провести належний аналіз та налаштування параметрів алгоритму для конкретної задачі.

Рекомендується провести подальше дослідження та експерименти з параметрами алгоритму, такими як кількість генерацій, розмір популяції та інші параметри, для досягнення стабільних та задовільних результатів. Також варто звернути увагу на вибір адекватної функції фітнесу та правильне налаштування інших елементів алгоритму.

2.3.7 Можливі покращення

Існує кілька способів покращити ефективність генетичного алгоритму (ГА) для задачі комівояжера. Ось декілька можливих покращень:

1. Розширення пошуку: Використання механізмів розширеного пошуку, таких як локальний пошук, ітеративне поглиблення, випадкове перебудовування, може допомогти знаходити кращі розв'язки. Ці методи можуть бути застосовані після отримання результатів від ГА для поліпшення знайденого маршруту.
2. Тюнінг параметрів: Варто налаштувати параметри ГА, такі як розмір популяції, ймовірність кросовера та мутації, для досягнення кращих результатів. Експериментуйте з різними значеннями цих параметрів та спостерігайте їх вплив на результати.
3. Ініціалізація популяції: Оптимальний вибір початкової популяції може покращити швидкість збіжності алгоритму. Розгляньте використання евристик для створення початкових розв'язків, таких як метод найближчого сусіда або метод випадкового пошуку, щоб отримати більш різноманітну та пристосовану популяцію.

4. Механізм елітизму: Збереження кращих розв'язків з попередньої популяції (найкращих хромосом) може допомогти уникнути втрати цінної інформації та сприяти швидшій збіжності алгоритму. Резервування кількох найкращих хромосом у нову популяцію без змін може забезпечити стабільний прогрес.
5. Паралелізація: Використання паралельних обчислень може прискорити обробку поколінь ГА. Розгляньте можливість розпаралелювання обчислень для ефективнішого використання обчислювальних ресурсів.
6. Методи вибору: Спробуйте інші методи вибору батьків, такі як турнірний відбір або ранговий відбір, для збільшення різноманітності і покращення експлорації простору розв'язків.
7. Локальний пошук: Після отримання кінцевої популяції можна застосувати локальний пошук, щоб покращити знайдені маршрути. Локальні методи, такі як 2-Opt, 3-Opt або Local Search with Perturbations, можуть допомогти відшліфувати розв'язки та знизити загальну вартість маршруту.

Загалом, комбінація цих підходів може покращити ефективність ГА для задачі комівояжера та допомогти знайти кращі розв'язки. Рекомендується експериментувати з різними варіантами та параметрами для знаходження оптимального рішення в конкретному випадку.

ВИСНОВОК

Генетичні алгоритми працюють з задачами великої розмірності та забезпечують велику кількість розв‘язань, але неможливо довести їх оптимальність, а якість розв‘язку та швидкість роботи сильно залежать від обраних початкових значень

У даній роботі було проведено дослідження щодо пошуку оптимальних розв’язків для транспортної задачі. Були розглянуті історичні аспекти формулювання проблеми транспортування та побудови найдешевшого маршруту.

Теоретично були описані та розглянуті точні (алгоритм повного перебору) та наближені (генетичні алгоритми) способи знаходження оптимального маршруту за певним критерієм. Для експериментального дослідження були реалізовані алгоритми, такі як повний перебір, генетичний алгоритм з використанням мови програмування Python.

Отримані результати дозволяють зробити наступні висновки:

- Для невеликої кількості точок маршруту ($n < 5$) алгоритм повного перебору є ефективним, оскільки він забезпечує точний результат та працює достатньо швидко.
- Алгоритми з використанням генерації випадкових послідовностей та порівнянням поточного згенерованого маршруту з попереднім мінімальним є простішими у реалізації, але можуть працювати повільніше та давати гірші результати при сильних обмеженнях на час обчислення.

Порівняння знайдених маршрутів для кожного методу дозволяє наочно переконатись, що знайдені шляхи є оптимальними згідно з встановленими критеріями. Робота підтверджує важливість використання ефективних алгоритмів для розв'язання транспортної задачі та надає висновки щодо найкращих підходів для досягнення оптимальних результатів.

ДЖЕРЕЛА ІНФОРМАЦІЇ

1. Goldschmidt O., Laugier A., Olinick E.V. SONET/SDH ring assignment with capacity constraints // Discrete Applied Mathematics, 2003. Vol. 129. P. 99–128.
2. Ахо, А. Структуры данных и алгоритмы / А. Ахо, Дж. Хопкрофт, Д. Ульман. – М.: Издательский дом «Вильямс», 2001. – 384 с.
3. Гэри, М. Вычислительные машины и труднорешаемые задачи / М. Гэри, Д. Джонсон. – М.: Мир, 1982. – 419 с.
4. A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems, ст. 497-520.
5. The traveling salesman problem and its variations / G. Gutin, A. Punnen, (eds.) // Combinatorial optimization. – Nowell: Kluwer, 2002.
6. Bianchi L., Gambardella L.M., Dorigo M. An ant colony optimization approach to the probabilistic traveling salesman problem // Proceedings of PPSN-VII, Sev- enth international conference on parallel problem solving from nature. – Berlin.: Springer, 2002.
7. S. Lin & B. W. Kernighan, “An Effective Heuristic Algorithm for the Traveling-Salesman Problem”, Oper. Res. 21, 498-516 (1973).
8. Riadi, I.C.J & Nefti-Meziani, 2011, Cognitive Ant Colony Optimization: A New Framework In Swarm Intelligence, Proceeding of the 2nd Computing, Science and Engineering Postgraduate Research Doctoral School Conference, 2011