ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка
до магістерської роботи
на ступінь вищої освіти магістр
на тему: «**ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ СИСТЕМИ ПЛАНУВАННЯ ТА ОПТИМІЗАЦІЇ ТРАНСПОРТНИХ ПЕРЕВЕЗЕНЬ НА ОСНОВІ ГЕНЕТИЧНОГО МЕТОДУ**»

Виконав: студент 6 курсу, групи ПДМ-61
спеціальності:
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

Бондаренко П.В.
(прізвище та ініціали)

Керівник Негоденко О.В.
(прізвище та ініціали)

Рецензент_____
(прізвище та ініціали)

Нормоконтроль_____
(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти - «Магістр»
Спеціальність – 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення

Негоденко О.В.

“____” _____2022 року

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Бондаренку Павлу Вячеславовичу
(прізвище, ім’я, по батькові)

1. Тема роботи: «ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ СИСТЕМИ ПЛАНУВАННЯ ТА ОПТИМІЗАЦІЇ ТРАНСПОРТНИХ ПЕРЕВЕЗЕНЬ НА ОСНОВІ ГЕНЕТИЧНОГО МЕТОДУ»
Керівник роботи Негоденко О.В., завідувач кафедри ІПЗ, к.т.н., доц. кафедри ІПЗ
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від “11” жовтня 2021 року № 170.

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи:

3.1 Вимоги до кваліфікаційної роботи магістра з актуальних завдань спеціальності;_____

3.2 Нормативні матеріали (стандарти, Гости);_____

3.3 Технічні вимоги;_____

3.4 Науково-технічна література з питань, пов’язаних з темою роботи._____

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):

4.1 Провести порівняльний аналіз ефективності існуючих методів вирішення технічного завдання;

4.2 Побудувати генетичні оператори, що відображають специфіку розв’язуваної задачі за великої кількості клієнтів (понад 50);

4.3 Дослідити методи розпаралелювання основних операторів генетичного алгоритму на основі TBB та CUDA, оцінюваних на різних наборах даних

відповідно до варіацій операторів GA, включаючи довжину хромосом, кількість поколінь та розмір популяції;

4.4 Порівняти продуктивність платформ TBB і CUDA у вирішенні проблеми ТЗ з паралельним ГА та визначити загальну ефективність використання генетичного алгоритму для вирішення ТЗ.

4.5 Висновки.

5. Перелік графічного матеріалу.

5.1 Титульний слайд.

5.2 Мета дослідження – підвищення ефективності системи планування та оптимізації транспортних перевезень на основі генетичного методу.

5.3 Об'єкт дослідження – планування та оптимізація транспортних перевезень.

5.4 Предмет дослідження – алгоритми планування та методи оптимізації транспортних перевезень.

5.5 Модель класичної маршрутизації транспортних засобів.

5.6 Математична постановка транспортної задачі з обмеженням за часом.

5.7 Структура генетичного алгоритму.

5.8 Сутність програмної платформи CUDA.

5.9 Сутність бібліотеки паралельного програмування TBB.

5.10 Паралельні підходи для запуску генетичного алгоритму.

5.11 Автомобільні хмарні обчислення для використання запропонованого способу.

5.12 Вплив чисельності популяції на прискорення паралельних методів.

5.13 Час реалізації генетичного алгоритму.

5.14 Приклад реалізації програми еталонної моделі ГА.

5.15 Висновки

6. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи до	Примітка
1	Вибір теми магістерської роботи	29.10-07.11	
2	Підбір науково-технічної літератури	08.11-10.11	
3	Складання плану роботи	11.11-18.11	
4	Проведення порівняльного аналізу ефективності існуючих методів	19.11-28.11	
5	Побудова генетичних операторів, що відображають специфіку розв'язуваної задачі	29.11-04.12	
6	Дослідження методів розпаралелювання основних операторів генетичного алгоритму	05.12-8.12	
7	Порівняння продуктивності платформ	8.12-12.12	
8	Вступ, висновки, реферат	12.12-19.12	
9	Розробка обов'язкових матеріалів	21.12	
10	Попередній захист роботи	24.12	
11	Пред'явлення роботи в деканат	29.10-07.11	

Студент _____ Бондаренко П.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Негоденко О.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина магістерської роботи 121 с., 29 рис., 5 табл., 77 літературних джерел.

Об'єкт дослідження – планування та оптимізація транспортних перевезень.

Предмет дослідження – алгоритми планування та методи оптимізації транспортних перевезень.

Мета дослідження - підвищення ефективності системи планування та оптимізації транспортних перевезень на основі генетичного методу.

Метод дослідження – методи теорії ймовірності, математичної статистики та математичне імітаційне моделювання.

Під час дослідження був здійснений аналіз постановки транспортної задачі в різних модифікаціях. Реалізований огляд класичних та метаевристичних алгоритмів розв'язання задач маршрутизації транспорту. Серед цих алгоритмів виділена група еволюційних алгоритмів як найбільш ефективних, де найбільш практичним розроблюваним є генетичний алгоритм. Далі детально розглянуті всі властивості генетичного алгоритму та уточнення до методу розпаралелювання генетичного алгоритму транспортної задачі. Обговорюваний метод може значно прискорити вирішення еквівалентної транспортної задачі для багатьох складних проблем маршрутизації транспортних засобів (VRP) у хмарній реалізації інтелектуальних транспортних систем.

Галузь використання – сучасні транспортні системи України.

ТРАНСПОРТНА ЗАДАЧА, ЗАВДАННЯ МАРШРУТИЗАЦІЇ ТРАНСПОРТУ, ГЕНЕТИЧНИЙ АЛГОРИТМ, ВИХІДНА ПОПУЛЯЦІЯ, ФІТНЕС-ФУНКЦІЯ, ОПЕРАТОР КРОСИНГОВЕРУ, ХМАРНА РЕАЛІАЗЦІЯ ІНТЕЛЕКТУАЛЬНИХ ТРАНСПОРТНИХ СИСТЕМ, РОЗПАРАЛЕЛЮВАННЯ АЛГОРИТМІВ.

ЗМІСТ

ВСТУП

Актуальність. Завдання маршрутизації є одним з найбільш важливих і одночасно складних типів завдань комбінаторної оптимізації в галузі транспортної логістики. Піонерами в даному напрямку вважаються Г. Данциг і Дж. Рамсер [1], які запропонували в 1959 році математичну постановку і алгоритмічний підхід до вирішення практичного завдання поставки бензину від кінцевої станції магістрального трубопроводу до великої кількості обслуговуючих терміналів (завдання диспетчеризації вантажівок). Через кілька років Кларк і Райт [2] вперше включили більше одного ТЗ в постановку задачі, а також запропонували більш ефективну евристику на основі жодного алгоритму. З тих пір було запропоновано велику кількість моделей і алгоритмів, присвячених пошуку точного і наближеного рішення багатьох варіантів даного завдання, надалі об'єднаних в загальну групу завдань маршрутизації транспорту (Vehicle Routing Problem, VRP).

Підвищений інтерес до VRP викликаний одночасно її практичною значущістю і значною складністю. Завдання найбільш часто зустрічається серед компаній, що виконують розвезення товару з деякого пункту виробництва або складу до пунктів споживання або точок роздрібної торгівлі, але існують і інші області застосування, найбільш типові серед них: сервісне обслуговування, кур'єри і поштові служби, збір відходів, очищення вулиць, приватний перевезення, маршрутизація громадського транспорту, доставка швидкого харчування.

Будь-яке завдання класу VRP є узагальненням однієї з найважливіших в області комбінаторної оптимізації завдання комівояжера (Traveling Salesman Problem, TSP), в якій зазвичай потрібно знайти для комівояжера найвигідніший шлях, що проходить через зазначені міста по одному разу з наступним поверненням в початковий місто [3]. Принципова відмінність будь VRP від TSP полягає в тому, що рішення може включати одночасно декількох замкнутих маршрутів. Обидва завдання є NP-важкими, тобто не існує методів знаходження їх точних рішень і перевірки оптимальності наближених рішень за поліноміальний час [4]. VRP так

само, як і TSP, може бути описана в термінах теорії графів, при цьому об'єктами комбінаторної оптимізації стають графові структури.

Точні методи вирішення транспортних задач (ТЗ), дозволяють знайти рішення тільки для задач з малою кількістю клієнтів (тобто до 50-ти клієнтів). Для розв'язання задач великої розмірності точні методи є неефективними в зв'язку з їх великими часовими витратами.

Одним з актуальних варіантів є оптимально паралелізований генетичний алгоритм (ГА) вирішення транспортної задачі у VRP. Наближені рішення знайшли широке застосування в наукових та технічних задачах. Генетичний алгоритм - це клас еволюційних алгоритмів, який використовується для пошуку наближених рішень пошукових та оптимізаційних задач.

Вузькими обчислювальними місцями ГА є функції пристосованості, мутації, кросовер і вибір. Поширеним рішенням цієї проблеми є перенесення обчислень цих функцій до паралельні машини [4]. Тому, що стосується широкого використання ГА у різноманітних проблемах, таких як оптимізація, обробка зображень, навчання штучних нейронних мереж та системи на основі правил, багато дослідників намагаються досліджувати методи розпаралелювання для ГА на звичайних та графічних процесорах.

Вищесказане обумовлює актуальність і необхідність проведення досліджень в цьому напрямку.

Метою дослідження є підвищення ефективності різних параметрів рішень транспортної задачі (ТЗ) на основі генетичного алгоритму (ГА) на продуктивність паралельних ядер як в багатоядерних центральних процесорів (CPU), так і в багатоядерних графічних процесорів (GPU) та уточнення методу розпаралелювання основних операторів генетичного алгоритму. Досліджуваний паралелізм заснований на структурі багатоядерних центральних процесорів (CPU) і багатоядерних графічних процесорів (GPU) і намагається порівняти потужність двох процесорів при розпаралелюванні генетичних алгоритмів.

Об’єкт дослідження – планування та оптимізація транспортних перевезень.

Предмет дослідження – алгоритми планування та методи оптимізації транспортних перевезень.

Мета дослідження - підвищення ефективності системи планування та оптимізації транспортних перевезень на основі генетичного методу.

Метод дослідження – методи теорії ймовірності, математичної статистики та математичне імітаційне моделювання.

Для досягнення поставленої мети потрібно вирішити такі завдання:

1. Провести порівняльний аналіз ефективності існуючих методів вирішення технічного завдання.

2. Побудувати генетичні оператори, що відображають специфіку розв’язуваної задачі за великої кількості клієнтів (понад 50).

3. Дослідити методи розпаралелювання основних операторів генетичного алгоритму на основі TBB та CUDA, оцінюваних на різних наборах даних відповідно до варіацій операторів GA, включаючи довжину хромосом, кількість поколінь та розмір популяції.

4. Порівняти продуктивність платформ TBB і CUDA у вирішенні проблеми ТЗ з паралельним ГА та визначити загальну ефективність використання генетичного алгоритму для вирішення ТЗ.

1 СТАН ПРОБЛЕМИ ТА АНАЛІЗ МЕТОДІВ РІШЕННЯ ТРАНСПОРТНИХ ЗАДАЧ

1.1 Класифікація транспортних задач

1.1.1 Основні характеристики завдань оптимізації маршрутів транспорту

Завдання визначення раціональних маршрутів при організації транспортних перевезень, поряд із завданнями про завантаження транспорту і розміщення транспортних агентів, є одними з найбільш важливих в транспортній логістиці. В цілому, завдання оптимізації маршрутів транспорту можуть бути розділені на основі їх параметрів, обмежень і умов.

Для системного уявлення широкої множини завдань оптимізації маршрутів транспорту була взята за основу ієрархічна класифікація їх характеристик, запропонована [5], і доповнена рядом інших базових ознак, що зустрічаються на практиці і часто згадуваних у вітчизняній і зарубіжній літературі за останній час, серед яких: багатокритеріальність [6, 7], випадкові параметри [8, 9], обмеження по відстані [10], обмеження по пунктам виробництва і споживання [11], умова збалансованості [12], періодичність планування [13], крос-докінг [14], топологічні особливості графа [15], вимога вивезення вантажу з пунктів споживання [16]. Важливість факторів, відображених даними характеристиками, показана у відповідних джерелах. В результаті можна виділені наступні 23 основні характеристики завдань оптимізації маршрутів транспорту:

1) Кількість пунктів виробництва:

а) пункт виробництва один (характерно для ситуації, коли є склад з якого здійснюються поставки різним споживачам або їх обслуговування);

б) пунктів виробництва кілька (ситуація найчастіше виникає, якщо вантажі перевозяться від декількох виробників до споживачів.

2) Кількість пунктів споживання:

а) пункт споживання один (зокрема, це так, якщо продукція з різних місць повинна доставлятися на склад або в пункт сертифікації);

б) пунктів споживання кілька [5].

3) Кількість критеріїв:

а) однокритеріальних (при оцінці оптимальності маршрутів враховується тільки один критерій - або відстань, або час, або інша характеристика);

б) багатокритеріальні (використовується кілька, як правило, конфліктуючих критеріїв).

4) Параметри шляхів:

а) дорога характеризується одним параметром;

б) дорога характеризується кількома параметрами (відстань, пропускна здатність, габарити, мито чи інша фіксована плата, обмеження по швидкості і ін.):

- все що враховуються характеристики адитивно залежать від шляху;

- серед характеристик є неадитивні (наприклад, пропускна здатність).

5) Кількість вантажу:

а) вантаж відсутній (в задачі не мається на увазі переміщення будь-яких матеріальних предметів, речовин тощо., наприклад, в разі виконання сервісних робіт);

б) кількість вантажу характеризується дійсним числом (безперервна завдання):

- одним;

- декількома;

в) кількість вантажу характеризується цілим числом (дискретна завдання):

- одним;

- декількома.

6) Види вантажу:

а) однотипний вантаж;

б) багатономенклатурними вантаж:

- всі вантажі дозволені;

- є вантажі, заборонені до перевезення тих чи інших ТЗ;

- для кожного ТЗ є обмеження по різним видам вантажів, що перевозяться;

- існують заборони на спільні перевезення вантажів деяких видів одним ТЗ.

7) Кількість транспортних підприємств [5]:

а) одне (парк транспорту зосереджений в одному місці);

б) кілька (парк транспорту територіально розподілений);

в) збігаються з пунктами виробництва.

8) Кількість ТЗ:

а) єдине ТЗ;

б) фіксована кількість доступних ТЗ;

в) необмежену кількість доступних ТС.

9) Однорідність парку ТЗ:

а) всі ТЗ однакові;

б) різні типи ТС мають відмінні характеристики:

- відрізняються по одній характеристиці (наприклад, місткістю);

- відрізняються за багатьма характеристик (вантажопідйомність, вартість використання, швидкість та ін.).

10) Обмеження на місткість ТЗ:

а) без обмежень (кількістю вантажу можна знехтувати);

б) передбачені обмеження тільки по одному параметру;

в) передбачені обмеження за кількома параметрами.

11) Вимоги до перевезення:

а) при перевезенні вантажу з кожного пункту виробництва в кожен пункт споживання може використовуватися один ТЗ незалежно від вантажу, що перевозиться;

б) при перевезенні вантажу з кожного пункту виробництва один ТЗ може розвозити вантажі в різні пункти споживання;

в) в кожен пункт споживання вантажі з різних пунктів виробництва доставляє один ТЗ;

г) ТЗ можуть збирати вантаж в різних пунктах виробництва і перевозити в різні пункти споживання.

12) Додаткові умови:

а) кожен вид вантажу повинен бути доставлений з деякого пункту виробництва в певний пункт споживання (зокрема, під час перевезення пасажирів);

б) не допускається порожній пробіг (крім початкового і кінцевого відрізків маршруту);

в) кожне ТЗ може зробити тільки одну поїздку;

г) деякі пункти споживання є також пунктами виробництва:

- вивезення з пунктів споживання здійснюється паралельно доставці;

- вивезення з пунктів споживання здійснюється після доставки

13) Обмеження за часом [11]:

а) максимальна тривалість маршрутів перевезень обмежена;

б) час безперервного перебування ТЗ у шляху обмежена;

в) в ремінні вікна:

- доставка в пункти споживання здійснюється в зазначений інтервал часу;

- вивезення з пунктів виробництва здійснюється в зазначений інтервал часу;

г) крос-докінг.

14) Обмеження по відстані:

а) загальна довжина всіх маршрутів обмежена;

б) максимальне пройдене ТС відстань обмежена.

15) Обмеження по пунктам виробництва і споживання:

а) деякі ТЗ не можуть відвідувати певні пункти;

б) пункти розподілені між ТЗ.

16) Обмеження збалансування:

а) збалансування тривалості маршрутів;

б) збалансування довжин маршрутів;

в) збалансування кількості обслуговуваних пунктів;

г) збалансування кількості доставляється вантажу.

17) Послідовність перевезень:

а) заданий відповідність пунктів виробництва до пунктів споживання;

б) поділ фаз доставки і вивезення;

в) обмеження на послідовність завантаження-вивантаження вантажу - стековий принцип (LIFO);

г) задані інші вимоги до послідовності перевезень.

18) Періодичність:

а) планування маршрутів для одноразового виконання;

б) планування здійснюється на тривалий період;

в) планування регулярних маршрутів.

19) Фактори, що визначають вартість перевезення:

а) залежність вартості перевезення тільки від самого шляху;

б) залежність вартості перевезення від шляху і завантаження ТС:

- пропорційність залежність;

- складна функціональна залежність.

20) Умова повернення [14]:

а) ТЗ повинні повертатися на вихідну базу;

б) ТЗ повинні завершувати маршрут на який-небудь базі;

в) ТЗ можуть завершувати маршрут в будь-якому місці.

21) Топологія:

а) симетрична задача (характеристики шляху не залежать від вибору напрямку):

- метрична (відстань описується евклідовій метрикою);

- Чи не метрична (відстань не описується евклідовій метрикою);

б) асиметрична завдання (характеристики шляху залежать від вибору напрямку);

в) складна топологія.

22) стохастичну:

а) детерміновані дані;

б) випадкові дані (компоненти завдання мають випадкове поводження):

- випадкові параметри доріг;

- випадкові запити;

- випадкові умови (обмеження);

- випадкові характеристики ТЗ.

23) Визначеність:

а) статична задача, тобто замовлення в процесі доставки не змінюються;

б) динамічна задача, тобто замовлення можуть додаватися або скасовуватися.

Класифікація пов'язана, головним чином, з автомобільним транспортом, але основна маса представлених характеристик справедлива також для інших видів. Наведене сімейство властивостей дозволяє сформувати широку множину завдань транспортної логістики, в тому числі задач маршрутизації.

1.1.2 Опис класичної задачі маршрутизації транспорту

Класична VRP (Vehicle Routing Problem, переклад з англійської – маршрутизація транспортних засобів) - задача комбінаторної оптимізації, в якій для парку однотипних транспортних засобів (ТЗ) потрібно визначити оптимальний набір замкнутих маршрутів від єдиного депо до множини віддалених клієнтів. На практиці критерій оптимальності може виражатися будь-якими витратами на об'їзд клієнтів, але частіше за все відповідає довжині маршрутів. На рисунку 1.1 показаний типовий приклад побудови маршрутів для парку з трьох ТС при мінімізації сумарної довжини маршрутів. У центрі малюнка розташоване депо, в якому спочатку знаходиться парк з декількох однотипних ТЗ, а на деякій відстані від нього розташовані клієнти, яких потрібно відвідати. Відстані між усіма пунктами вважається відомим. Оптимальне рішення являє собою набір найкоротших маршрутів для ТЗ через всіх клієнтів з поверненням в депо.

Класична VRP може бути представлена на графі G = (V, E) з множиною вершин V = {0, n} і множиною ребер E. Вершини i = 1, n відповідають клієнтам, тоді як вершина 0 відповідає депо. Також задається матриця n x n невід'ємних вартостей cij, пов'язаних з кожним ребром (i, j) e E і визначають витрати на пересування між вершинами i та j.

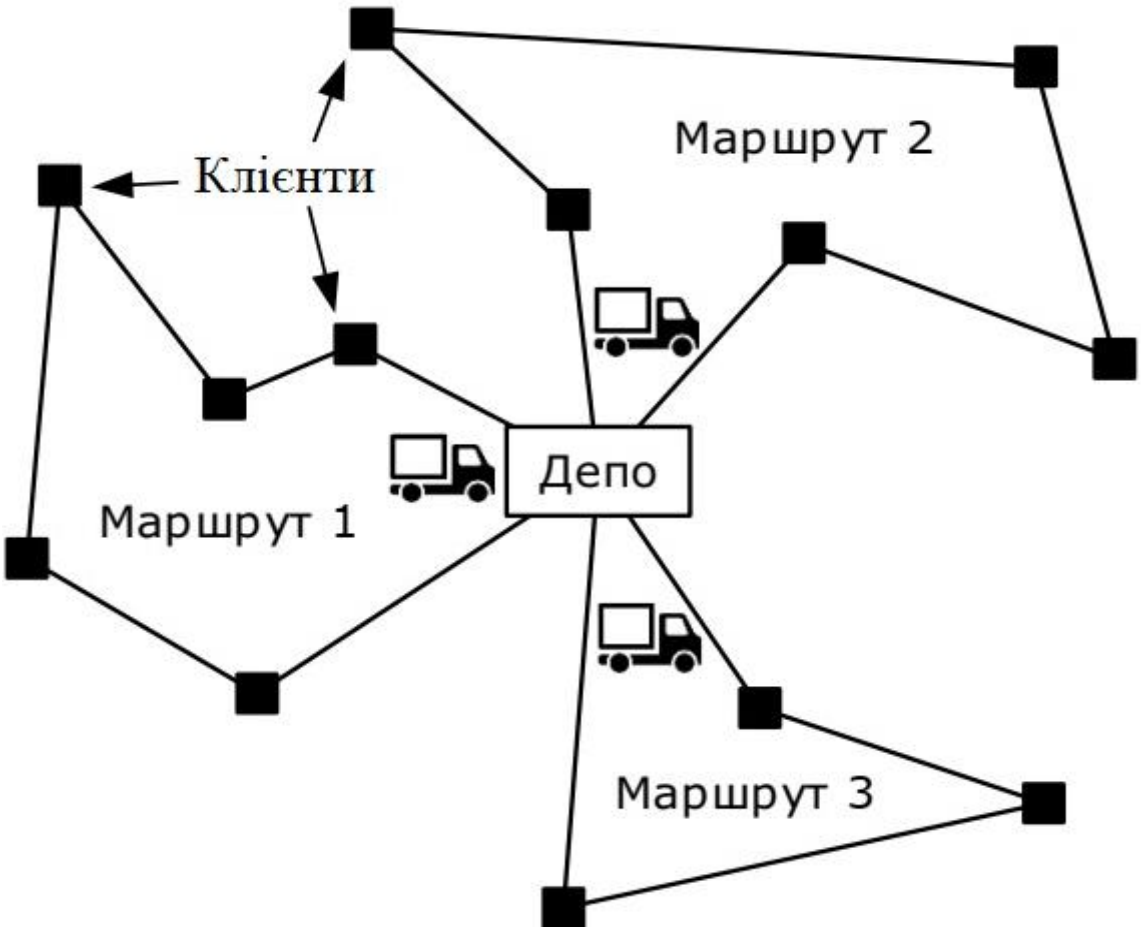


Рис.1.1. - Приклад рішення класичної VRP

Тоді в термінах теорії графів рішення класичної VRP зводиться до побудови декількох гамільтонових циклів мінімальної довжини на підграфі графу G з однією спільною вершиною-депо [18]. При відсутності обмеження вантажопідйомності і довільній кількості ТЗ k завдання зводиться до побудови k циклів із загальною вершиною 0, які в сукупності містять всі вершини графа і мінімізують суму вартостей об'їзду клієнтів, тобто отримуємо множинну завдання комівояжера (TSP з k ізольованими контурам з загальною точкою 0). Таке завдання можливо перетворити до звичайної TSP додаванням k - 1 додаткових копій вершини 0 і суміжних з нею ребер.

В силу обмеженості класична постановка слабо співвідноситься з практикою. Сьогодні існує велика кількість різновидів VRP, велика частина яких є комбінаціями декількох основних розширень класичного варіанту. Всі вони

відрізняються, головним чином, різними реальними обмеженнями, що накладаються на одержуване рішення [19].

1.1.3 Огляд і аналіз узагальнень та розширень завдання маршрутизації транспорту

Asymmetric VRP (AVRP) - асиметрична завдання маршрутизації транспорту [14]. У загальному випадку, асиметрична завдання відрізняється від симетричною тим, що вона моделюється орієнтованим графом [5]. Відповідно, матриця вартостей є асиметричною. У більшості наукових робіт, присвячених дослідженню VRP, розглядається симетричний варіант завдання, що передбачає симетричність матриці вартостей [20]. Таке припущення часто не співвідноситься з реальними умовами і веде до розриву між теорією і практикою: найкоротший шлях між двома точками дорожньої мережі, як правило, відрізняється в залежності від напрямку. Найбільш істотні такі відмінності при малих масштабах, наприклад, в межах одного міста. З іншого боку, так само істотно збільшується простір пошуку, відповідно, ускладнюється знаходження оптимального рішення. Тому, в залежності від конкретної прикладної задачі, вартості (ваги дуг) для протилежних напрямків можуть бути зведені до деякого усередненого значення.

Capacitated VRP (CVRP) - задача маршрутизації транспорту з обмеженням по вантажопідйомності [21]. Схожа на класичну з тим лише обмеженням, що обсяг вантажів на кожному маршруті не повинен перевищувати задану величину Q, однакову для всіх ТЗ. Фіксований парк ТС однакової місткості із загального депо повинен з мінімальними витратами задовольнити попит на товар кожного клієнта, і при цьому не перевищити власну вантажопідйомність. Попит і вантажопідйомність задаються цілочисельними або речовими значеннями. Цей найбільш вивчений варіант завдання транспортної маршрутизації був вперше описаний в роботі Данцига і Рамсер [1], а в 1979 році була дана формалізація у вигляді задачі лінійного програмування [22].

Distance-Constrained VRP (DCVRP) - задача маршрутизації транспорту з обмеженням по відстані [23]. Довжина маршруту не може перевищувати задане

значення. Такий варіант постановки зручно використовувати, коли ТС може заправлятися тільки в депо і необхідно врахувати обмежений обсяг паливного бака. Обмеження по відстані зазвичай розглядається спільно з обмеженням вантажопідйомності [24].

VRP with Time Windows (VRPTW) - задача маршрутизації транспорту з тимчасовими вікнами [25]. Використовується при обмеженому часовому діапазоні прийому або вивезення продукції. Для виконання запиту кожного / -го клієнта існує відомий проміжок часу, визначений як інтервал $[a_i, b_i]$. У разі прибуття раніше нижньої межі інтервалу, враховується час очікування її настання [26]. Прибуття пізніше верхньої межі інтервалу неприпустимо [27], однак в деяких випадках обслуговування клієнта в певному заздалегідь тимчасовому вікні не є критично важливою умовою, але його порушення додає деякий штрафне значення до цільової функції -VRP with Soft Time Windows (VRPSTW). Також може бути врахований сервісний час, необхідний для обслуговування клієнта. Крім усього іншого, рішення VRPTW дозволяє підібрати час виїзду автотранспорту з депо і тим самим уникнути непотрібних простоїв в точках доставки. Постановка завдання є більш складною, але в ряді випадків більш повно описують реальний процес, тому що в багатьох практичних завданнях доставки товарів час прибуття до клієнта і час обслуговування клієнта грають істотну роль .

Multi-Depot VRP (MDVRP) - задача маршрутизації транспорту з кількома депо [28]. У класичній моделі VRP передбачається наявність єдиного депо, в якому повинні починатися і закінчуватися маршрути всіх ТЗ. І хоча такий варіант завдання привернув найбільшу увагу, він не підходить для ряду випадків [29], таких як розподіл продукції декількома постачальниками загальній групі споживачів. MDVRP є узагальненням класичної VRP, в якому існує більше одного депо, з яких здійснюється обслуговування клієнтів, при цьому кожне ТЗ починає і закінчує маршрут в з власним депо. Рішення вважається допустимим, якщо виконуються стандартні умови VRP, проте часто використовуються додаткові обмеження (вантажопідйомності, відстані, часу і т.п.).

Очевидно, що такий варіант завдання є більш складним, оскільки виникає необхідність розподілити споживачів за різними депо. Для цього зазвичай особі, що приймає рішення, доводиться визначати, які клієнти ставляться до кожного депо, або використовуються двофазні алгоритми, в яких спочатку здійснюється розбивкою графа на підграфи, а потім по-окремо будуються маршрути для всіх депо. В ідеальному випадку більш ефективно здійснювати обидва кроку одночасно, проте при вирішенні задач великих розмірностей це робити важко [30].

У розширенні завдання з проміжними депо MDVRP with Inter-Depot Routes (MDVRPI) може здійснюватися дозавантаження ТЗ в будь-якому депо по шляху проходження [31]. В системі розподілу ці депо є складами, а в системі збору - пунктами розвантаження.

Periodic VRP (PVRP) - періодична задача маршрутизації транспорту. На відміну від класичної VRP в задачах з періодичної маршрутизацією використовується розширений період планування до декількох днів. Для різних клієнтів потрібно різне число відвідувань в зазначений період, причому дні обслуговування заздалегідь не визначені, але заданий список можливих дат відвідування для кожного клієнта. Таким чином, завдання маршрутизації вирішується для кожного дня планування. У ряді випадків ця особливість має важливе значення, зокрема, при вирішенні проблеми збору відходів.

Існує також розширення завдання з вибірковим обслуговуванням - PVRP with Service Choice (PVRP-SC), де частота відвідування клієнтів встановлюється по ходу рішення задачі [32]. Це дозволяє отримати більш оптимальні маршрути і дає переваги при обслуговуванні клієнтів.

VRP with Pickup and Delivery (VRPPD) - задача маршрутизації транспорту з вивезенням і доставкою [33]. Узагальнення завдання з обмеженням вантажопідйомності, в якому клієнти можуть як отримувати, так і відправляти товари. При цьому зазвичай мається на увазі, що товари не перевозяться від одного клієнта до іншого, а спочатку відправляються з депо, або в кінцевому рахунку надходять в депо.

VRPPD підрозділяється залежно від порядку доставки та вивозу, які можуть здійснюватися послідовно, змішано або одночасно:

• Delivery-First, Pickup-Second VRP: усі товари повинні бути доставлені клієнтам-споживачам, перш ніж відбудеться будь-вивезення від клієнтів-постачальників (рисунок 1.2.а). Таким чином, рішення задачі поділяється на дві фази. З практичної точки зору це вимога пояснюється тим фактом, що все навантаження зазвичай здійснюються ззаду транспорту і перестановка вантажів є неприйнятною або витратної за часом. У той же час зростає ризик перевищення вантажопідйомності ТЗ, незалежно від співвідношення рівнів попиту на вивезення та доставку. Інше відоме назва - VRP with Backhauls (VRPB) - задача маршрутизації транспорту з зворотним транзитом (з поверненням товарів) [34];

• Mixed Pickup and Delivery VRP (VRPMPD): вивезення та доставка можуть проводитися в будь-якій послідовності за маршрутом ТЗ (рисунок 1.2.б), проте клієнти також як і в попередньому випадку розділені на постачальників і споживачів. Такий варіант дозволяє отримати більш оптимальні маршрути, однак ускладнює вантаження і вивантаження товарів. Іноді також використовується позначення Mixed VRP with Backhauls (VRPMB) - змішана задача маршрутизації транспорту з зворотним транзитом [36];

• VRP with Simultaneous Pick-up and Delivery (VRPSPD): одні й ті ж клієнти одночасно виступають в якості споживачів і постачальників (рисунок 1.2.в), повертаючи певний товар в депо [36]. На практиці це потрібно, наприклад, при доставці в продуктові магазини, коли багаторазові піддони або контейнери використовуються для транспортування товарів. Обидва останніх варіанти можуть бути описані за допомогою загальної моделі: VRPMPD як VRPSPD, в якій попит на доставку або вивезення дорівнює нулю, а VRPMPD як VRPSPD, якщо кожного клієнта розбити на отримувати й надсилати товар (рисунок 1.2.г).

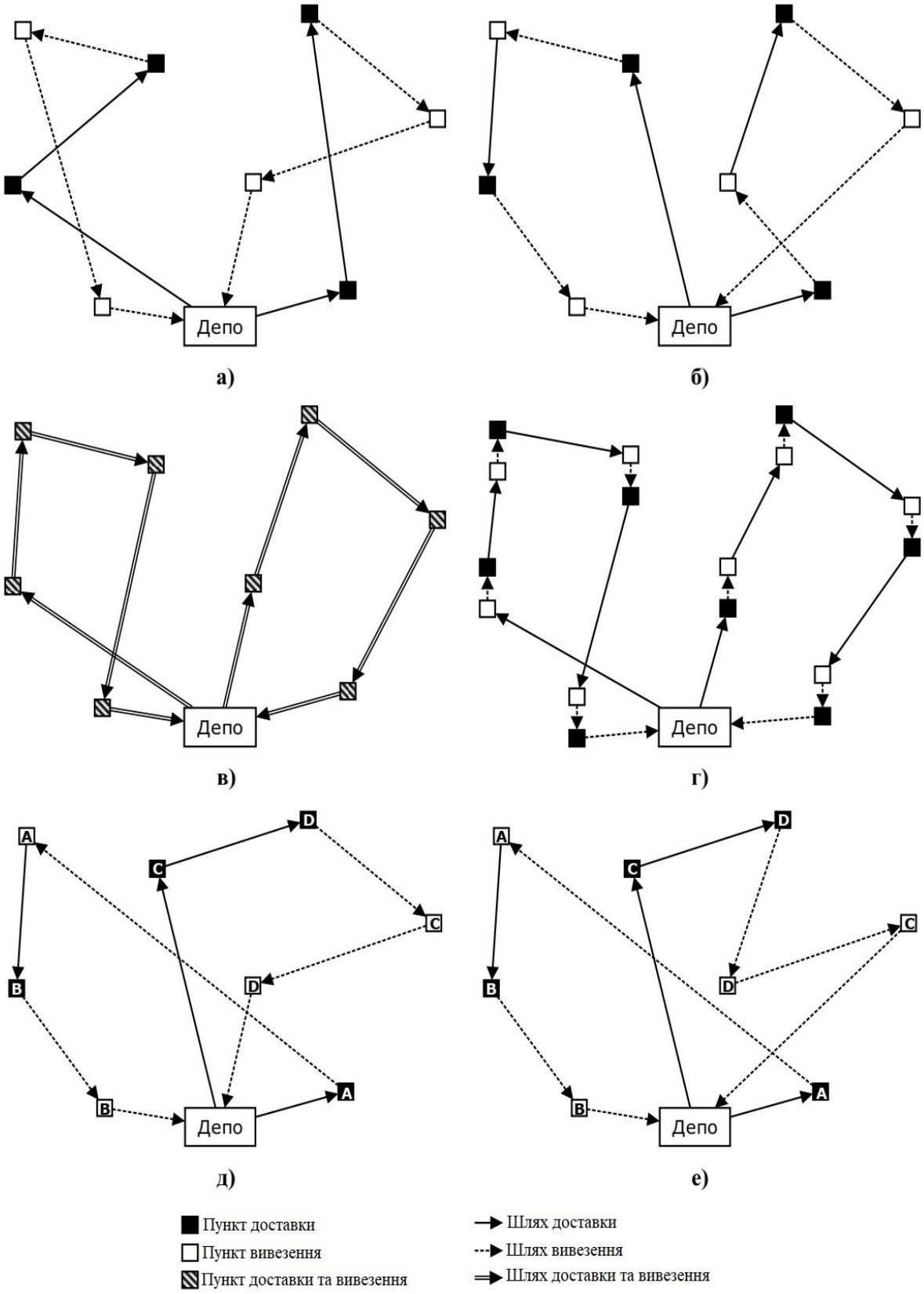


Рис. 1.2. - Рішення задач маршрутизації транспорту з вивезенням і доставкою: а - VRPB; б - VRPMB; в - VRPSPD; г - VRPSPD з розбиттям; д - DARP; е - VRP LIFO [36].

У розглянутих вище варіантах VRPPD передбачалося, що запити на вивезення та доставку відносяться як один-до-багатьох-до-одного (one-to-many-to-one problem), коли товари транспортуються від депо до клієнтів і в зворотному напрямку. Однак деякі автори також відносять до VRPPD різновиди зі зв'язками багато-до-багатьох (many-to-many problem), в яких будь-який клієнт може відправляти і отримувати будь-який товар, і зі зв'язками один-до-одного (one-to-one problem) в яких встановлюється попарне відповідність між пунктами навантаження і пунктами доставки у вигляді запитів [14]. У першому випадку товар вивозиться від одного з багатьох постачальників і доставлятися одному з багатьох споживачів (рисунок 2.б за умови вивезення та доставки товару від одних клієнтів до інших). У другому випадку кожен запит визначається пунктом навантаження, відповідним йому пунктом доставки і заданою кількістю вантажу, яке потрібно перемістити між зазначеними пунктами (рисунок 1.2.д). Добре відомими прикладами завдань з парними зв'язками є Pickup and Delivery Problem (PDP), що застосовується під час перевезення товарів [57], і Dial-A-Ride Problem (DARP) [36], що застосовується під час перевезення людей, що виражається в різних додаткових обмеженнях і цілях.

Загальна задача вивезення та доставки General Pick-up and Delivery Problem (GPDP) дозволяє визначити багато завдань з вивезенням і доставкою як окремі випадки її формулювання [37].

Варто окремо відзначити варіант VRP зі стековим принципом «останнім прийшов - першим вийшов» - VRP with LIFO [38]. Аналогічна VRPPD за винятком додаткового обмеження: розвантаження ТЗ може здійснюватися тільки в зворотному порядку завантаження, тобто першим завжди вивантажується останній завантажений товар (рисунок 1.2). Зазвичай використовується при мультіноменклатурном вантажі для скорочення часу завантаження і розвантаження ТЗ, тому що відпадає необхідність переставляти товар. Однак в літературі добре вивчений тільки випадок з одним ТЗ - TSP with Pickup and Delivery and LIFO (TSPPDL) [39] - завдання комівояжера з вивезенням і доставками за принципом LIFO.

Split Delivery VRP (SDVRP) - задача маршрутизації транспорту з роздільним доставкою [40]. У задачі знімається загальний для всіх VRP заборона на багаторазове відвідування клієнта. Тобто один і той же клієнт може бути обслужений безліч разів декількома ТЗ, якщо це дозволяє зменшити загальні витрати. Така постановка особливо виправдана, якщо запити клієнтів перевищують вантажопідйомність ТС [41]. Експериментально показано, що в залежності від характеристик завдання може бути отриманий різний виграш в довжині маршрутів при використанні роздільної доставки [42]. При розгляді завдання простим підходом є розбиття кожного клієнта на безліч близько розташованих з меншими запитами, тим самим завдання зводиться до звичайної VRP. Однак, як правило, для завдання маршрутизації з різними видами транспорту отримати оптимальне рішення складніше, ніж для класичної VRP.

Stochastic VRP (SVRP) - задача маршрутизації транспорту з випадковими даними [43]. Один або кілька компонентів завдання можуть мати випадкове поводження:

• VRP with Stochastic Demands (VRPSD): попит кожного клієнта пов'язаний із заданим імовірнісним розподілом, замість конкретного значення, а дійсне значення визначається тільки по прибуттю ТЗ;

• VRP with Stochastic Clients (VRPSC): число клієнтів точно не відомо, кожен клієнт існує з певною ймовірністю;

• VRP with Stochastic Travel Time (VRPST): часи поїздок (відстані) між пунктами детермінований;

• VRP with Stochastic Service Time (VRPSST): час обслуговування кожного клієнта не детерміновано.

SVRP відрізняється від класичної VRP в ряді аспектів. Методологія вирішення є більш складною і поєднує в собі особливості стохастичного і цілочисельного програмування. Більш того, часто завдання даного типу виявляються нерозв'язними [44]. Лише в рідкісних випадках вдається отримати оптимальне значення, тому розробка і застосування хороших евристик складні. Рішення SVRP зазвичай відбувається в два підходи. Перший етап дає рішення без

урахування випадкових змінних. На другому етапі, коли стають відомими випадкові значення, відбувається корекція раніше отриманого рішення.

Fuzzy VRP (FVRP) - нечітка задача маршрутизації транспорту [45]. На практиці буває важко отримати точні значення запитів, часу шляху, кількості та місця розташування клієнтів, меж тимчасових вікон і інших величин, якщо вони підкоряються імовірнісним законам. У деяких нових системах також складно описати параметри завдання як випадкові величини через недостатність даних для аналізу розподілу. Використання методів теорії нечітких множин дозволяє успішно моделювати завдання, що містять елементи невизначеності і суб'єктивності. Основними різновидами FVRP є:

• VRP with Fuzzy Demands (VRPFD): нечіткий попит клієнтів на товар;

• VRP with Fuzzy Travel Time (VRPFT): нечіткі часи поїздок (відстані) між пунктами;

• VRP with Fuzzy Service Time (VRPSST): нечітке час обслуговування кожного клієнта;

• VRP with Fuzzy Time Windows (VRPFTW): нечіткі межі часових вікон, в межах яких можуть обслуговуватися клієнти.

Dynamic VRP (DVRP) - динамічна задача маршрутизації транспорту [46]. Передбачається, що можуть відбуватися деякі зміни параметрів завдання в процесі її рішення. До таких зазвичай ставляться поломки ТЗ, дорожні затори, нові замовлення, раптові виклики і т.д. Найчастіше розглядається випадок, коли нові клієнти можуть з'являтися протягом дня, тобто після того, як ТЗ покине депо. Таким чином, у міру обслуговування клієнтів умови задачі ме ються і необхідно динамічно перераховувати маршрути з урахуванням нової інформації, що надходить. Як правило, заздалегідь відома статистика появи нових замовлень, на основі якої створюються імітаційні моделі для розробки і випробування алгоритмів.

Open VRP (OVRP) - відкрита задача маршрутизації транспорту [47]. Незамкнений варіант завдання, в якому не потрібно повертатися в депо в кінці маршруту. Таким чином, віддаленість останнього відвіданого клієнта від депо не

впливає на загальну вартість рішення. Найчастіше OVRP використовується в тому випадку, коли підприємство не володіє парком ТЗ, а укладає контракт із зовнішніми кур'єрами. Відповідно, наймані ТС не повинні повертатися в розподільний центр підприємства (депо) і можуть закінчити маршрут в будь-якому місці. Рішення повинно забезпечити мінімальний набір ТЗ, які повинні бути найняті для обслуговування всіх клієнтів з мінімальними витратами на дорогу. Крім того, при наявності власного парку ТЗ і значних коливаннях споживчого попиту з плином часу, рішення дозволяє визначити підходяще поєднання власних і найнятих ТС.

Варто також відзначити, що зняття типової для всіх VRP вимоги повернення до депо не робить задачу більш простою, OVRP також залишається NP-важкою [48]. У той же час є принципова відмінність: в OVRP будуються Гамільтона шляху, що починаються в депо, а в звичайній VRP - Гамільтона цикли. У зв'язку з цим алгоритми вирішення замкнутих завдань часто виявляються неефективними при вирішенні незамкнутих. Проте незамкнений варіант завдання зводиться до замкнутого шляхом заміни ваг дуг, що входять в депо, на число 0. Оптимальний замкнутий маршрут в такому графі відповідає оптимальному незамкнутому маршруту в вихідному графі.

Heterogeneous VRP (HVRP) - задача маршрутизації транспорту з різнорідним парком [49]. Узагальнення класичної VRP, в якому клієнти обслуговуються декількома типами ТЗ з відмінними характеристиками, такими як вантажопідйомність, швидкість, вартість використання і т.п. Як правило, парк ТЗ в моделях VRP є однорідним, що рідко відповідає реальній практиці. Таке припущення дозволяє спростити пошук рішення, але в логістичних операціях часто потрібно брати до уваги характеристики кожного конкретного ТЗ.

У літературі вивчені три версії завдання з різнорідним транспортом:

• Fleet Size and Mix VRP (FSMVRP) [50], Vehicle Fleet Mix (VFM) [43] або Fleet Size and Composition VRP (FSCVRP) [44]: змішаний парк ТС різних типів, що відрізняються вантажопідйомністю та вартість, причому кількість доступних ТС кожного типу не обмежена;

• Heterogeneous Fleet VRP (HFVRP) [35] або Mix Fleet VRP (MFVRP) [76]: аналогічна попередній, але, крім іншого, для кожного типу вводиться різна вартість пересування (за одиницю відстані / часу), тобто використовується безліч матриць вартостей для всіх типів ТС;

• Heterogeneous Fixed Fleet VRP (HFFVRP) [42]: узагальнення попередніх версій з обмеженням кількості доступних ТС кожного типу (фіксований парк).

VRP with Satellite Facilities (VRPSF) - задача маршрутизації транспорту з супутніми складами [41]. Класична задача VRP передбачає, що кожен маршрут починається і закінчується в депо. Однією з причин повернення в депо є обмежена вантажопідйомність - коли машина розвозить усі товари, вона повинна повернутися в депо за новою порцією товарів. Однак, в деяких випадках вигідніше провести дозагрузку на маршруті в додаткових проміжних пунктах поповнення (супутніх складах), без повернення в депо [45]. Даний варіант застосовується в ситуаціях, коли центральний постачальник повинен забезпечувати товаром велике число споживачів на регулярній основі. VRPSF є складовою частиною завдання розподілу товару - Inventory Routing Problem (IRP) [81].

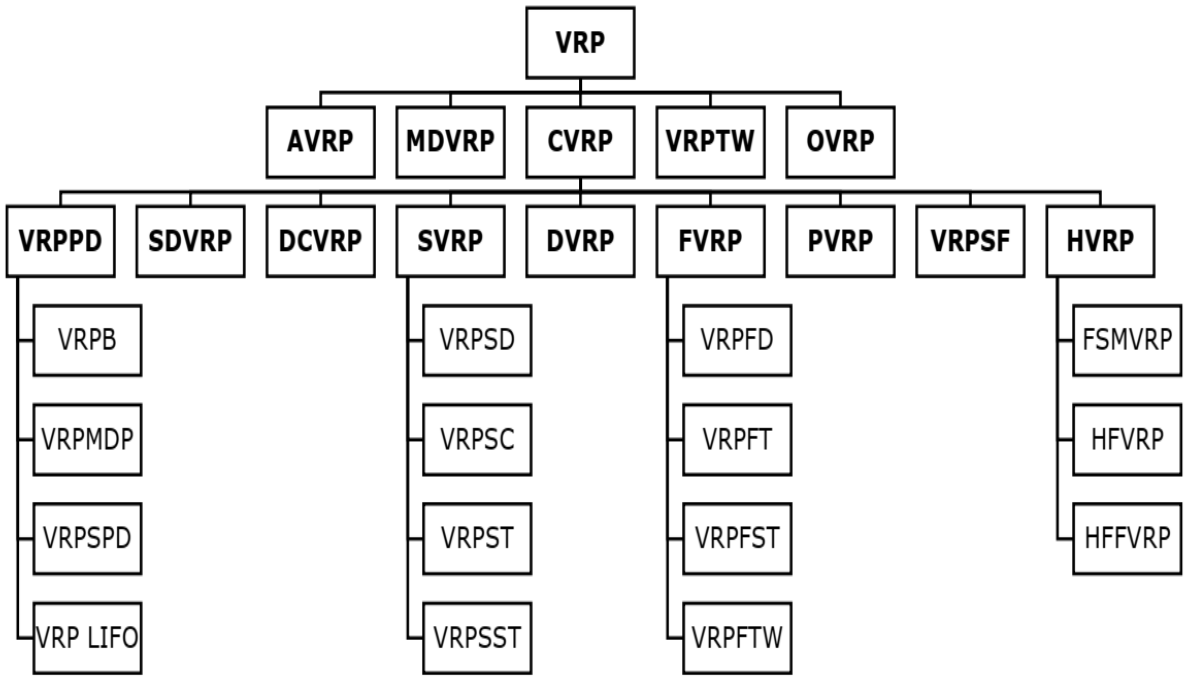


Рис.1.3. - Ієрархічна схема узагальнень і розширень VRP

Розглянуті різновиди VRP можна представити у вигляді ієрархічної схеми (рисунок 1.3). З неї видно, що в основі більшості різновидів VRP мається на увазі базове обмеження вантажопідйомності (CVRP), яке в явному вигляді при описі може не вказуватися. Очевидно, за рахунок сполучень розглянутих вище різновидів VRP можна отримати безліч варіантів її постановок.

На рисунку 1.4 в якості прикладу показано зв'язок декількох відомих похідних варіантів VRP, узагальнюючих чотирьох базових розширення: відкриту (OVRP); з безліччю депо (MDVRP); з обмеженням вантажопідйомності (CVRP); з тимчасовими вікнами (VRPTW).

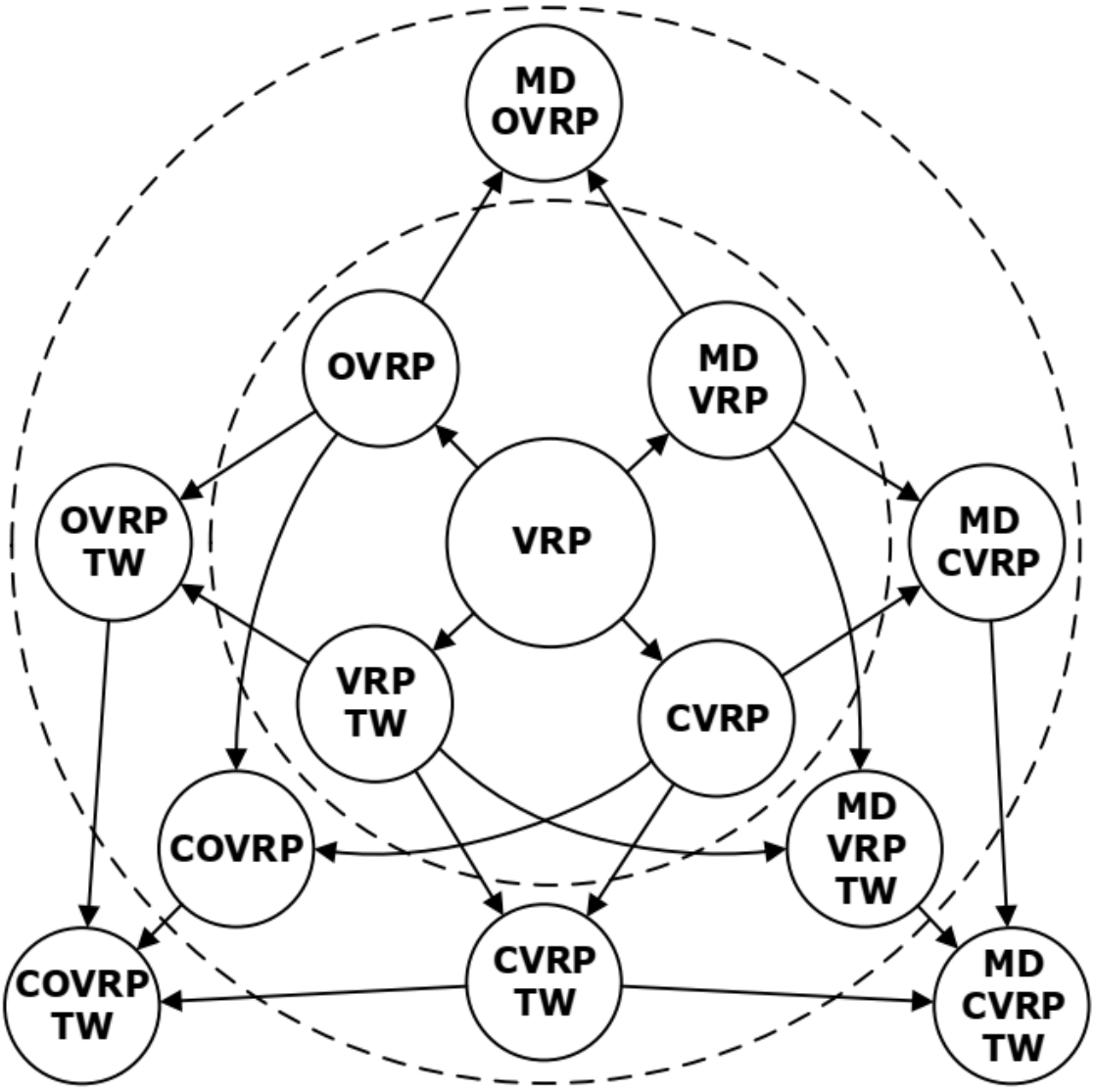


Рис.1.4. - Схема зв’язку узагальнюючих різновидів VRP

У літературі їх позначення прийнято формувати за принципом додавання до абревіатурі букв, що характеризують складові розширення. Відповідно до загальноприйнятої системі назву завдання і її скорочення можуть виходити досить довгими, наприклад, Multi-Depot Capacitated Vehicle Routing Problem with Time Window (MDC VRPTW) - задача маршрутизації транспорту з безліччю депо, обмеженням по вантажопідйомності і тимчасовими вікнами.

У різних роботах розглядаються конкретні комбінації умов і обмежень для визначення нових областей застосування VRP. В даний час такі комплексні варіанти (англ. «Rich VRP») вважаються найцікавішими для дослідження, оскільки дозволяють наблизитися до реальних умов [46].

В таблиці 1.1 наведені різновиди завдання, які можуть розглядатися як узагальнення класичного варіанту VRP, із зазначенням достатніх умов відомості до окремого випадку відповідно ієрархічної схемою (рисунок 3). Існують відомі моделі розглянутих різновидів VRP із загальним підходом до формулювання, наприклад, як задач цілочисельного програмування [33].

Для вирішення всіх найбільш відомих узагальнень і розширень VRP існує величезна кількість методів, кожен з яких має свої переваги і недоліки - в цьому ключі кожен варіант завдання заслуговує окремого огляду.

У джерелах, зазначених для кожного варіанта завдання, різними авторами пропонуються приклади розв'язання і практичного застосування, тому в наступних підрозділах розглядаються тільки основні підходи до вирішення завдань VRP.

В результаті аналізу можна зробити висновок, що найбільш поширені узагальнення і розширення VRP можуть бути об'єднані в рамках узагальненої постановки, що дозволило б розробляти рішення на основі єдиної моделі.

ТЗ з обмеженням за часом описується наступним чином. Є деяка кількість автотранспорту, один склад (депо) і деяку кількість клієнтів. Для кожного транспортного засобу потрібно скласти маршрут, протягом якого транспортний засіб відвідує ряд клієнтів (наприклад, з метою доставки будь-якого вантажу).

Таблиця 1.1 - Зв'язок узагальнень VRP з класичним варіантом

Задача	Особливість	Умова взведення до класичної VRP
AVRP	Асиметрична матриця вартостей	Ваги дуг для взаємно протилежних напрямків рівні
CVRP	Обмеження вантажопідйомності	Вантажопідйомність кожного ТЗ більше або дорівнює сумарному попиту всіх клієнтів
DCVRP	Обмеження відстані	Допустима довжина маршруту більше або дорівнює максимальній довжині Гамільтона циклу через всі вершини графа
VRPTW	Часові вікна	Нижнє значення часового вікна менше або дорівнює часу шляху від вершини-депо до вершини-клієнта, верхнє - найбільш пізнього з усіх можливих
MDVRP	Множина депо	Ваги ребер, що з'єднують вершини-депо з вершинами клієнтами, приймаються однаковими для всіх депо, тобто розташування всіх депо збігається
PVRP	Розширений період планування	Можливі відвідування всіх клієнтів обмежені одним єдиним днем
VRPPD	Крім доставки потрібно вивезення	Величина запитів на вивезення для всіх вершин-клієнтів дорівнює нулю
SVRP	Випадкові параметри	Щільність ймовірності випадкового параметра більше нуля тільки для певного вузького діапазону
DVRP	Динамічні параметри	Час до зміни параметрів одно або перевищує початковий час виконання завдання
HVRP	Різні ТС	Параметри транспорту і ваги ребер графа для всіх видів транспорту рівні

На маршрут кожного транспортного засобу накладається ряд обмежень. Кожен маршрут повинен починатися і закінчуватися в депо. Загальна кількість товарів, необхідних для доставки клієнтам на даному маршруті даного транспортного засобу, не повинно перевищувати його вантажопідйомність. Кожен клієнт обслуговується лише одним транспортним засобом і лише одного разу, тобто не допускається відвідування одного клієнта двома і більше транспортними засобами. Кожен клієнт повинен бути обслужений в певний проміжок часу, цей проміжок визначається двома значеннями; перше значення визначає час прибуття транспортного засобу до клієнта, друге - час відправлення. Для даної задачі сформульовано такі цілі (цільові функції): первинна мета - мінімізувати загальну кількість транспортних засобів, необхідних для обслуговування всіх клієнтів; вторинні цілі - мінімізувати загальний час обслуговування всіх клієнтів і загальна відстань, пройдену всіма транспортними засобами [2].

Наведена постановка описує статичну версію ТЗ з обмеженням за часом, тобто передбачається, що всі параметри ТЗ відомі заздалегідь до початку виконання завдання. Однак на практиці часто умову задачі змінюється в процесі її рішення і (або) під час виконання отриманого в результаті рішення - транспортного плану. Такі ТЗ відносяться до класу динамічних транспортних завдань (Dynamic Vehicle Routing Problem, DVRP). В наступній частині буде дана постановка динамічної ТЗ (ДТЗ); також буде введена міра оцінки динамічної складової ТЗ.

1.2 Динамічна транспортна задача. Проблема оцінки динамічної складової транспортної задачі

На відміну від статичної ТЗ з обмеженням за часом, параметри якої відомі до початку її рішення і не змінюються ні під час її вирішення, ні під час виконання, в динамічної ТЗ з обмеженням за часом умови задачі можуть змінюватися під час двох перерахованих вище етапів. Виникає новий клас задач - динамічні транспортні завдання з обмеженням за часом (DVRPTW, Dynamic Vehicle Routing Problems with Time Windows) [9-10]. Далі розглянемо основні параметри, зміна яких перетворює

статичну ТЗ з обмеженням за часом в динамічну. Згідно [11] в літературі виділяють два головних динамічних елемента ТЗ: динамічні запити клієнтів і динамічний час двтісенія від одного клієнта до іншого. Динамічні запити клієнтів - це нові запити, які виникають під час виконання завдання і які повинні бути виконані. Динамічне час руху - це час руху від одного клієнта до іншого, яке може змінюватися в межах будь-якого інтервалу, або зміняться заздалегідь (до початку виконання завдання) невідомим чином. На зміну часу руху між двома клієнтами можуть впливати різні причини: завантаженість дорожніх трас, випадкові аварії, погода, місце розташування і конструкція доріг і т.д. Дані причини в різній мірі впливають на можливість передбачення часу руху. Як правило, розрізняють три типи динамічного часу руху:

• Час руху можна передбачити заздалегідь (long-term forecasts). Час руху свідомо залежить від часу доби або від дня тижня (time-depend) [12]. Хоча даний час руху і змінюється, ці зміни відомі заздалегідь і можуть бути описані до початку виконання завдання і, по суті, є статичним [18].

• Час руху може бути передбачене па короткий проміжок часу (short-term forecasts), коли є більш повна інформація про поточну ситуацію на дорозі [9]. Вважається що таку інформацію, наприклад, про час руху автомобіля з пункту / в з пунктом можна отримати тільки після прибуття автомобіля в пункт и [19].

• Час руху не може бути передбачене і розглядається стохастическим. Причиною зміни даного часу можуть стати непередбачені аварії і поломки в дорозі, а також інші випадковості на дорогах, цей час є істинно динамічним компонентом часу пересування [20].

Динамічні елементи ТЗ, що виділяються в математичної моделі динамічної транспортної задачі, є її важливою і невід'ємною частиною. При описі математичної моделі задачі визначається, який вид динамізму розглядається в даній задачі. Як правило, окремий алгоритм вирішення ДТЗ з обмеженням за часом приймає до розгляду один або кілька динамічних елементів, описаних вище [9]. У більшості реальних ТЗ предметом зміни є запити клієнтів. Це означає, що не всі запити клієнтів відомі заздалегідь (до початку вирішення ТЗ або до початку

виконання отриманого в результаті рішення, транспортного плану), тобто вважається, що інформація про кількість запитів змінюється і оперативно доводиться до центру диспетчеризації - місця, де вирішується поточна ТЗ. Дані запити (якщо вони не виходять за рамки тимчасових обмежень) повинні бути задоволені в рамках вирішення поточної ТЗ.

Для того, щоб оцінити ефективність того чи іншого алгоритму для вирішення ДТЗ необхідно формалізувати і виміряти динамічну складову в задачах, на яких оцінюється продуктивність тестованого алгоритму. Для оцінки динамічної складової вводять міру - рівень (ступінь) динамічної складової завдання (degree of dynamism) [24]. Далі наведемо оцінки ступеня динамічної складової для ТЗ різного виду згідно [24]. У даній оцінці-враховується тільки динамічна зміна кількості запитів від клієнтів, формула (1.1).

$$DD = \frac{R_d}{R_t} \tag{1.1}$$

де DD - ступінь динамізму ТЗ без обмежень;

R_d - кількість динамічних запитів (запити, що надійшли під час рішення задачі);

R_t - загальна кількість запитів (кількість запитів відомих до початку рішення задачі (статичні запити) плюс динамічні запити - R_d.

Дана оцінка (1.1) ступеня динамічної складової ТЗ не враховує час появи динамічного запиту по відношенню до інтервалу транспортування ТЗ (planning horizon) - періоду часу, за який транспортні засоби повинні обслужити всіх клієнтів і повернутися в депо. Тобто запити, що з'явилися на початку даного інтервалу і в його кінці, вважаються рівнозначними за складністю їх обробки, але це не так, тому що час реакції на запит, отриманий пізніше менше, ніж на запит, отриманий раніше, і це ускладнює процес пошуку рішення. Для обліку часу отримання динамічного запиту застосовують наступний спосіб оцінки ступеню динамічної складової ТЗ [24]:

$$EDD = \frac{\sum_{i=1}^{R_d} \frac{t_i}{T}}{R_t} \tag{1.2}$$

де EDD - ступінь динамізму ТЗ без обмежень з урахуванням часу надходження динамічного запиту;

[0; T] - період часу, за який транспортні засоби повинні обслужити всіх клієнтів і повернутися в депо, всі динамічні запити виникають тільки в даний проміжок часу;

t_i- момент часу отримання / -го динамічного запиту, 0 <t_i <T.

З урахуванням тимчасових обмежень, що накладаються на період часу обслуговування запитів клієнтів, формула для оцінки ступеня динамічної складової ТЗ з обмеженням за часом має вигляд [24]:

$$EDD_{tw} = \frac{1}{R_t}\sum_{i=1}^{R_t}\frac{T-(l_i-t_i)}{T} \quad (1.3)$$

де EDD_{tw} - ступінь динамізму ТЗ з обмеженням за часом;

l_i - верхня межа тимчасового вікна, протягом якого повинен бути обслужений відповідний і-й клієнт.

Дана оцінка EDD буде використовуватися в подальшому для оцінки ефективності алгоритму для вирішення ДТЗ з обмеженням за часом. Оскільки ДТЗ є серією статичних задач (де чергове завдання виникає при зміні параметрів (появі нових запитів) вихідної задачі, то для її вирішення використовуються методи вирішення статичних задач, адаптованих під динамічну задачу [28]. Тому далі буде дано історичний огляд і аналіз ефективності методів вирішення ТЗ з обмеженням за часом.

1.3 Аналіз методів рішення транспортних задач з обмеженням за часом

1.3.1 Огляд класичних алгоритмів розв'язання задач маршрутизації транспорту

VRP є завданням комбінаторної оптимізації, в якій число допустимих маршрутів експоненціально зростає при збільшенні числа клієнтів, і належить до класу складності NP. Повний перебір можливих її рішень хоча і дозволяє знайти оптимум, але вимагає колосальної часу обчислень навіть при відносно невеликій

розмірності задачі (14 і більше). Пропонувалися такі точні методи вирішення VRP, як, наприклад, метод гілок і меж [51], метод гілок з відсіканням. Вони є розвитком методу повного перебору, на відміну від останнього - з відсівом підмножин допустимих рішень, свідомо не містять оптимальних рішень. Однак час їх обчислень все одно зростає занадто швидко, а в гіршому випадку - як і при повному переборі.

Більш розумним представляється пошук в напрямку наближених алгоритмів. Відомо досить багато шляхів вирішення VRP, велика частина яких - евристичні методи. Відомі підходи зазвичай орієнтуються на загальне формулювання VRP, в якій передбачається симетрична або несиметрична матриця відстаней, що не заданий жорстко кількість транспортних засобів, і відстежується тільки обмеження по їх вантажопідйомності або максимальної довжини маршруту.

Класичні алгоритми можна розбити на три групи:

1. Конструктивні алгоритми. Виконують поступову побудову рішення, відстежуючи зростання його вартості, але не мають фази подальшого поліпшення:

• алгоритм Кларка-Райта (заощаджень) [45],

• метод, заснований на збіги [52].

2. Двофазні (кластерні) алгоритми. Завдання розбивається на дві операції - угруповання вершин для кожного майбутнього маршруту (кластеризація) і рішення TSP для кожної отриманої групи:

• алгоритм Фішера і Джайкумара [53],

• алгоритм замітання (Sweep Algorithm) [54],

• алгоритм пелюсток (Petal Algorithm) [55],

• алгоритм Османа [56].

3. Покращуючі алгоритми: спочатку ведеться пошук деякого рішення, а потім робляться спроби обміну вершин (ребер) всередині кожного маршруту або між маршрутами:

• евристики поліпшення багатьох маршрутів [57].

У двофазних алгоритмах можлива наявність зворотного зв'язку між етапами рішення. Окремо можна згадати зворотний двофазний підхід, коли спочатку

здійснюється рішення TSP, а потім поділ одного маршруту на кілька. При цьому TSP вирішується для всіх вершин вихідної множини.

Пошук рішень VRP почався в 60-і роки XX століття. Евристичні методи, які в наші дні називають класичними, розроблені переважно між 1960 і 1990 роком. Протягом останніх двадцяти років велика кількість успішних і надзвичайно ефективних евристик були запропоновані для вирішення VRP. Нові підходи дають все більш точні результати. Але при цьому більшість з них не в змозі забезпечити хороший компроміс між якістю і продуктивністю. Одночасне вдосконалення алгоритмів як з точки зору ефективності, так і з точки зору продуктивності є відкритою завданням.

В даний час зусилля спрямовані переважно на новий напрямок в теорії і практиці штучного інтелекту - так звані метаевристикі.

1.3.2 Аналіз методів метаеврістичної оптимізації

Термін «метаевристика», що походить від грецького прийменника «meta» (в значенні «вищий рівень») і «heuristic» (від грец. SvpiaKsiv, «шукати»), був вперше введений Фредом Гловером в 1986 р для позначення алгоритмічних схем вищого рівня, спрямованих на ефективне вивчення простору пошуку складних оптимізаційних задач [57]. Перші метаевристичні методи стали з'являтися ще до введення даного терміну, і для їх позначення використовувалися більш широкі терміни «оптимізація чорного ящика» (англ. Black-Box Optimization) або «слабка оптимізація» (англ. Weak Methods). Насправді метаевристикі описують великий (а також основний) підрозділ в стохастичній оптимізації - великому класі алгоритмів і методів, які в тій чи іншій мірі використовується випадковість для пошуку оптимального (близького до оптимального або субоптимального) рішення складних завдань.

Метаевристикі є сучасним потужним і надзвичайно популярним класом оптимізаційних методів, що дозволяють знаходити рішення для широкого кола завдань з різних додатків. Сила метаевристик полягає в їх здатності вирішувати складні завдання без знання простору пошуку, саме тому ці методи дають можливість вирішувати складні для задачі оптимізації. У метаевристичних методах упор робиться на ретельному вивченні найбільш перспективних частин простору рішень. Якість одержуваних рішень виходить вище, ніж у отриманих класичними евристиками. Особливість метаевристичних алгоритмів в тому, що вони не дають точного опису порядку дій для вирішення завдання, і кожен з них повинен бути додатково конкретизовано шляхом підбору значень керуючих параметрів.

Метаевристика - це такий метод вирішення обчислювальних задач шляхом комбінування існуючих процедур з відкритим інтерфейсом і закритою реалізацією, яке призводить до максимально ефективному вирішенню. Метаевристичний підхід зазвичай застосовується для вирішення завдань, що не мають задовільного специфічного для завдання алгоритму, або в тому випадку, коли немає практичної необхідності реалізовувати такий алгоритм. Найбільш часто метаевристикі використовуються в рішенні задач комбінаторної оптимізації, але також вони можуть застосовуватися до будь-яким іншим, які можна звести до вирішення логічних рівнянь. На відміну від традиційних алгоритмів оптимізації та ітераційних методів, метаевристикі не гарантують, що в глобальному масштабі оптимальне рішення може бути знайдено для деякого класу задач в силу стохастичності. Однак при пошуку по великим набором допустимих рішень метаевристикі часто дозволяють знайти гарне рішення з меншими обчислювальними і тимчасовими витратами.

Виділяють наступні властивості метаевристик [57]:

- метаевристика є стратегією, яка управляє процесом пошуку;

- метою метаевристикі є ефективне дослідження простору пошуку для знаходження (суб) оптимального рішення;

- методи, які реалізуються метаевристичним алгоритмом, варіюються від простого локального пошуку до складного процесу навчання;

- метаевристичний алгоритм є наближеним і зазвичай недетермінованим;

- в метаевристикі закладається механізм, що запобігає застрявання в локальному оптимумі;

- основні положення метаевристик допускають абстрактне опис;

- метаевристика може використовувати проблемно-орієнтоване знання в формі евристик, керованих високорівневою стратегією;

- передові метаевристикі використовують досвід, накопичений в процесі пошуку і представлений у вигляді пам'яті, для управління пошуком.

Основна частина літератури по темі метаевристик має експериментальний характер, описуючи емпіричні результати, засновані на імітаційному моделюванні і програмному експерименті з алгоритмами. Однак існують також деякі формальні теоретичні результати, що обгрунтовують можливість знаходження глобального оптимуму.

Вивчення процесів при відпалі металів, спостереження за розмноженням особин в деякій популяції і організація мурашиних колоній послужили основою цілої серії ідей таких метаевристик, як алгоритм імітації відпалу, генетичний алгоритм і мурашиний алгоритм, що застосовуються в тому числі для вирішення VRP. Наведемо приклади метаевристичних алгоритмів, які вже успішно застосовувалися для вирішення VRP:

• пошук з винятками [58],

• модельований відпал [59],

• генетичний алгоритм [60],

• мурашиний алгоритм [61],

• нейронні мережі [62].

Сильною стороною розглянутих алгоритмів є швидкість роботи навіть при великій розмірності задачі і можливість подолання локального мінімуму в процесі пошуку. Відповідно до думки дослідників такі алгоритми, як генетичні і мурашині, зараз ще відносно слабко вивчені, хоча містять великий простір для подальшого пошуку поліпшень, потенційно здатних привести до помітного зростання якості.

Перешкодою до практичної реалізації більшості метаевристик є або їх абстрактний опис (просте перерахування найменування етапів алгоритму), або опис, орієнтований на вирішення тільки однієї певної проблеми (наприклад, оптимізації або числових функцій, або комбінаторної оптимізації), тому для вирішення різних завдань VRP потрібна модифікація існуючих метаевристик або розробки принципово нових.

Метод пошуку з заборонами. Привабливість цього методу в тому що, хоча в його основі і лежить метод локального пошуку, метод заборон дозволяє продовжувати пошук після знаходження локального оптимуму, тим самим розширюючи простір пошуку в надії знайти рішення, близьке до оптимального. Для цієї мети метод заборон використовує список заборон (Tabu list); саме цей основний елемент методу заборон використовується для запобігання зациклення (повернення до попередніх рішень) і для розширення простору пошуку [38]. Також основними поняттями в методі пошуку із заборонами, запозиченими з методів локального пошуку, є простір пошуку (search space) і механізм побудови околиці рішення (neighborhood structure) [38]. Розглянемо ці поняття більш детально. Простір пошуку - це всі можливі рішення, які можуть бути отримані в результаті виконання завдання, в даному випадку - маршрути транспортних засобів. Простір пошуку тісно пов'язане з механізмом побудови околиці даного рішення. Околицею поточного рішення називають всі рішення, які можуть бути отримані з поточного шляхом застосування до поточного рішенням певних операторів; процес застосування цих операторів і називається побудовою околиці поточного рішення [52-53]. Список заборон - це ключове поняття, що відрізняє метод заборон від методів локального пошуку. У списку заборон містяться переміщення (moves), які застосовувалися на попередніх етапах рішення. Список заборонених переміщень може бути як фіксованою, так і змінної довжини. Час, протягом якого дане переміщення міститься в списку заборон, називається тривалістю заборони (tabu tenure). Після закінчення цього часу переміщення видаляється зі списку заборон. Однак список заборон може надати занадто жорстке обмеження на процес пошуку, він може забороняти переміщення, які не призведуть до зациклення алгоритму, а

навпаки, є кращими. Для запобігання цьому використовується спосіб, який називається аспіраційний критерій (aspiration criteria). Суть методу полягає в тому, що переміщення дозволяється, навіть якщо воно знаходиться в списку заборон, якщо рішення, отримане при застосуванні даного переміщення, краще в сенсі ЦФ поточного оптимального рішення; відповідно отримане рішення стає поточним оптимальним рішенням. описані вище елементи є основними в методі пошуку з заборонами. Через велику кількість алгоритмів на основі методу пошуку з заборонами наведемо їх короткий опис (метод, який використовували при побудові початкового рішення, метод побудови околиці рішення) і додаткові особливості (якщо вони є) тільки найбільш ефективних алгоритмів згідно з результатами вирішення тестових завдань, опублікованих в літературі.

У 1995 Рочат (Rochat) і Тейлард (Taillard) запропонували алгоритм на основі методу пошуку заборонами. Основною особливістю даного алгоритму було використання адаптивної пам'яті (adaptive memory) [54]. Адаптивна пам'ять - сукупність (набір) кращих маршрутів, отриманих в результаті роботи алгоритму, пам'ять призначена для формування нового стартового рішення для даного алгоритму; шляхом комбінування маршрутів, обраних за певним критерієм з адаптивною пам'яті. До обраним маршрутами застосовувалася процедура методу пошуку з заборонами, і отримане рішення містилося назад в адаптивну пам'ять. Для побудови початкових рішень використовувався метод конструювання Соломона. Для отримання нового рішення з поточного (процедура отримання околиці рішення) використовувався метод 2-opt перестановки [38].

У 1997 році був запропонований алгоритм Чанга (Chiang) і Рассел (Russell) [55]. В даному алгоритмі динамічно змінювалася довжина списку заборон залежно від стадії виконання алгоритму для того, щоб запобігти зациклення пошуку і надмірно не обмежувати його. Для локального пошуку використовувалася процедура λ-перестановок, описана вище [38].

У 2001 році був запропонований алгоритм Кордео (Cordeau) [56]. В даному алгоритмі допускалося, що в процесі пошуку рішення може порушувати частина обмежень (тимчасові обмеження, обмеження на вантажопідйомність машини);

порушення обмежень збільшувало ЦФ на деяке значення в залежності від типу обмеження. Для локального пошуку використовувалася процедура простий вставки клієнта, описана вище [38].

Генетичні і еволюційні алгоритми. ГА клас метаевристичних алгоритмів, які застосовуються для вирішення широкого кола комбінаторних задач [57], в тому числі і для вирішення ТЗ. Дані алгоритми є випадково-спрямованими пошуковими алгоритмами і засновані на ітеративній адаптації популяції (індивідів, хромосом, рішень) протягом певного проміжку часу, заданого у вигляді кількості ітерацій роботи алгоритму. Це означає, що спочатку створюється популяція рішень. Потім на кожній ітерації алгоритму до вихідної популяції застосовуються оператори мутації, кроссинговер а й селекції, в результаті чого популяція рішень піддається зміні (еволюції); краще рішення в кінцевій отриманої популяції є остаточним рішенням завдання. Процес створення і зміни популяції на кожній ітерації алгоритму складається з наступних кроків. Створення популяції - кодування рішення. Зміна популяції - застосування операторів селекції, кросинговеру (рекомбінація) і мутації. Застосування оператора кросинговеру до рішень, обраним з поточної популяції (процедура вибору рішень називається селекції і здійснюється на основі оцінки ЦФ рішення), призводить до того, що в результаті отримуємо рішення-нащадки, з яких формується нова популяція; потім до рішень нової популяції застосовується оператор мутації, дана нова популяція на наступній ітерації буде вихідної для отримання наступної популяції і т.д. Застосування оператора кросинговеру має бути побудовано таким чином, щоб закріпити в нащадках кращі властивості рішень-батьків (у разі ТЗ рішенням-нащадкам повинні передаватися «кращі» маршрути рішень-батьків), застосування оператора мутації має розширити простір пошуку за рахунок часткового довільного зміни рішень.

Процес формування нової популяції намагаються будувати таким чином, щоб ЦФ рішень нової популяції (популяції, отриманої з вихідної шляхом застосування до неї операторів селекції, кросинговеру і мутації) були краще (менше або більше) ЦФ вихідної популяції. Оцінку рішень в популяції (яке рішення краще,

гірше) виробляють на основі ЦФ завдання часто в літературі цю функцію називають «функція придатності рішення» (fitness function).

Танго (Thangiah) в 1994 році запропонував кілька метаевристичних алгоритмів [36]. В основі цих алгоритмів лежить поділ клієнтів на сектори з використанням ГА. Після поділу клієнтів в межах одного сектора формується маршрут, який містить клієнтів тільки з даного сектора. До отриманого рішенням застосовується процедура локального пошуку - λ- перестановок [57].

Джанг (Jung) і Мун (Moon) в 2002 році запропонували ГА для вирішення ТЗ з обмеженням за часом [62]. Особливістю даного алгоритму було те, що рішення уявлялося у вигляді двовимірного відображення маршрутів (дуг) на площині. Оператор рекомбінації для поділу рішення на дві частини використовував різні криві на двовимірної площині, на якій знаходилися клієнти і маршрути [63].

Хомбергер (Homberger) і Герінг (Gehring) в 1999 році запропонували два еволюційних алгоритму для вирішення ТЗ з обмеженням за часом, результати кращого алгоритму наведені в таблиці 3 [63-64]. Оператор мутації в даному алгоритмі заснований на Or-opt, 2-opt і λ-перестановках (Л = 1). В результаті застосування оператора схрещування з декількох батьків виходить тільки одне рішення-нащадок. Відмінність між цими двома алгоритму від іншого полягає у відсутності (присутності) оператора схрещування.

Местер (Mester) в 2005 [65] розробив еволюційний алгоритм подібний алгоритму Хомбергера і Герінга. Оператор мутації в даному алгоритмі реалізований наступним чином: з рішення, до якого застосовується даний оператор, видаляються клієнти по одному з кожного маршруту (або кілька клієнтів по заздалегідь заданому критерію) і заново вставляються з використанням іншого евристичного алгоритму вставки з малими тимчасовими витратами. Для розв'язання задач великої розмірності використовується стратегія поділу задачі на підзадачі меншої розмірності.

2 ГЕНЕТИЧНИЙ АЛГОРИТМ РІШЕННЯ ТРАНСПОРТНОЇ ЗАДАЧІ З ОБМЕЖЕНОЇ ЗА ЧАСОМ

2.1 Математична постановка транспортної задачі з обмеженням за часом

Спочатку дамо математичну постановку розв'язуваної задачі - ТЗ з обмеженням за часом (VRPTW, Vehicle Routing Problem with Time Window).

ТЗ з обмеженням за часом належить до класу задач маршрутизації автотранспорту (VRP - Vehicle Routing Problem). Завдання даного типу можна описати таким чином. Є деяка кількість автотранспорту, один склад (депо) і деяка кількість клієнтів. Для кожного транспортного засобу потрібно скласти маршрут, протягом якого транспортний засіб відвідує ряд клієнтів (наприклад, з метою доставки будь-якого вантажу). На маршрут кожного транспортного засобу накладається ряд обмежень. Кожен маршрут повинен починатися і закінчуватися в депо. Загальна кількість товарів, необхідних для доставки клієнтам на даному маршруті даного транспортного засобу, не повинно перевищувати його вантажопідйомність. Кожен клієнт обслуговується лише одним транспортним засобом і лише одного разу, тобто не допускається відвідування одного клієнта двома і більше транспортними засобами. Кожен клієнт повинен бути обслужений в певний проміжок часу, цей проміжок визначається двома значеннями, перше значення визначає час прибуття транспортного засобу до клієнта, друге - час відправлення. Для даної задачі сформульовано такі цілі (цільові функції): первинна мета - мінімізувати загальну кількість транспортних засобів, необхідних для обслуговування всіх клієнтів; вторинна - мінімізувати загальну відстань, пройдену всіма транспортними засобами [2].

Згідно [2, 3] математично ТЗ з обмеженням за часом можна представити у вигляді графа:

$$G = (N, A)$$

G = (N, A), де: N - множина вершин, відповідних набору клієнтів (customers) (вершини 1,2, ..., n) і вихідного депо (depot), в якому починають і закінчують свій маршрут всі автомобілі (вершини 0 і n + 1);

A - набір дуг, що з'єднують вершини графа.

Введемо позначення: C - множина клієнтів, $|C| = n$;

i, j - *i*-й і *j*-й клієнти, $i \in C, j \in C$;

(i, j) є *A*-дуга, що з'єднує *i*-у і *j*-у вершини графа;

d_i - попит *i*-го клієнта;

$t_{i,j}$ - час переміщення по дузі *(i, j)*, що складається з часу обслуговування клієнта *i* та часу переміщення автомобіля від клієнта *i* до кліенту *j*;

c_{ij} - вартість переміщення автомобіля від клієнта / до кліентуу;

V - кількість ідентичних автомобілів вантажопідйомністю q;

k - *к*-й автомобіль, $к \in V$;

$[a_i, b_j]$ - «тимчасове вікно» (time window) - проміжок часу, протягом якого повинен бути обслужений *i*-й клієнт;

S_i^k - час прибуття *k*-го автомобіля к *i*-му клієнту; час відправлення з депо для всіх автомобілів дорівнює 0 (тобто $S_0^k = 0 \forall \in V$);

X_{ij}^k - змінна, що приймає значення {0,1} і характеризує напрямок руху автомобіля: X_{ij}^k = 1 - від клієнта *i* до клієнту j, X_{ij}^k =0 - в зворотному напрямку.

З урахуванням цих позначень математична формулювання ТЗ з обмеженням за часом таке: необхідно мінімізувати цільову функцію (1) при обмеженнях (2.1) - (2.9) [2, 3]:

$$\sum_{k \in V} \sum_{(i.j) \in A} c_{ij} X_{ij}^k \quad (2.1)$$

$$\sum_{k \in V} \sum_{\mathrm{j} \in N} X_{ij}^k = 1, \forall i \in C \quad (2.2)$$

$$\sum_{i \in C} d_i \sum_{\mathrm{j} \in N} X_{ij}^k \leq q, \forall k \in V \quad (2.3)$$

$$\sum_{\mathrm{j} \in N} X_{0j}^k = 1, \forall k \in V \quad (2.4)$$

$$\sum_{\mathrm{j} \in N} X_{ih}^k - \sum_{\mathrm{j} \in N} X_{ih}^k = 0, \forall h \in C, \forall k \in V \quad (2.5)$$

$$\sum_{\mathrm{j} \in N} X_{i,h+1}^k = 1, \forall k \in V \quad (2.6)$$

$$\sum_{j\in N} X_{ij}^{k}\left(S_i^k + t_{ji} - S_j^k\right) \le 0, \forall (i,j) \in A, \forall k \in V \quad (2.7)$$

$$a_i \le S_j^k \le b_j, \forall i \in N, \forall k \in V \quad (2.8)$$

$$X_{ij}^{k} \in \{0,1\}, \forall (i,j) \in A, \forall k \in V \quad (2.9)$$

Цільова функція (2.1) визначає ціну всіх маршрутів всіх транспортних засобів (загальна ціна транспортного плану). Обмеження (2.2) вважає, що кожен клієнт обслуговується тільки одним транспортним засобом і тільки один раз. Обмеження (2.3) визначає, що транспортний засіб не може обслужити більше клієнтів, ніж дозволяє його вантажопідйомність. Обмеження (2.4) означає, що кожен автомобіль залишає депо один раз. Обмеження (2.5) показує, що автомобіль може покинути вершину h, тільки якщо він прибув в цю вершину. Аналогічно обмеження (2.4), обмеження (2.6) означає, що всі транспортні засоби повертаються до депо, причому один раз. Це обмеження випливає з обмежень (2.4) і (2.5). Обмеження (2.7) означає що, якщо автомобіль рухається з вершини *i* до *j*, то час прибуття автомобіля до *j* не може бути менше суми часу прибуття автомобіля в пункт *i* (S_i^k) і часу руху автомобіля з пункту *i* до пункту *j* (t_{jj}). Обмеження (2.8) - це обмеження за часом, прибуття автомобіля до клієнта повинно бути в межах тимчасового вікна

2.2 Визначення та основні властивості генетичних алгоритмів

Генетичні алгоритми були засновані в Мічиганському університеті американським дослідником Холландом і спочатку розроблені для задач оптимізації в якості досить ефективного механізму комбінаторного перебору варіантів рішення. На відміну від багатьох інших робіт, метою Холланда було не тільки рішення конкретних завдань, але дослідження явища адаптації в біологічних системах і застосування його в обчислювальних системах. При цьому потенційне рішення - особина представляється хромосомою - двійковим кодом. Популяція містить множини особин. В процесі еволюції використовуються три основних генетичних оператора: репродукція, кросинговер і мутація. Голдберг (учень

Холланда) успішно розвинув генетичний алгоритм і розширив області його застосування.

Основи теорії генетичних алгоритмів (genetic algorithm) сформульовані Холландом (Holland) в основній роботі [1] і в подальшому були розвинені рядом дослідників. Найбільш відомою і цитованої в даний час є монографія Голдберга (Goldberg) [2], де систематично викладені основні результати та області практичного застосування генетичних алгоритмів.

Також в 60-х роках в Німеччині Рехенберг заклав основи "еволюційних стратегій" при вирішенні задачі оптимізації речових параметрів у розрахунку ліній електропередачі. Це напрямок розвивався довгі роки незалежно і тут були отримані важливі фундаментальні результати. У еволюційних стратегіях потенційне рішення (особина) є вектором дійсних чисел, популяція складається з двох особин і основним генетичним оператором є мутація.

Фогель незалежно від інших дослідників заснував еволюційне програмування, де потенційне рішення представляється кінцевим автоматом. Основним генетичним оператором тут також є мутація, яка випадковим чином змінює таблицю переходів-виходів автомата.

Трохи пізніше Коза в Массачусетському технологічному інституті США заклав основи генетичного програмування. Тут в якості особи виступала програма на LISP, яка представлялася деревовидною структурою. На цих структурах були розроблені генетичні оператори кросинговеру і мутації.

На даний час вказані напрями об'єднані до «еволюційних обчислень», які успішно застосовуються при вирішенні багатьох проблем. При цьому окремі напрямки взаємодіють за рахунок запозичення кращих рис один у одного.

Генетичні алгоритми використовують принципи і термінологію, запозичені у біологічної науки - генетики. У генетичному алгоритмі кожна особина представляє потенційне рішення деякої проблеми. У класичному генетичному алгоритмі особина кодується рядком двійкових символів - хромосомою, кожен біт якої називається геном. Безліч особин - потенційних рішень становить популяцію. Пошук (суб) оптимального вирішення проблеми виконується в процесі еволюції

популяції послідовний перетворення одного кінцевого безлічі рішень в інше за допомогою генетичних операторів репродукції, кросинговера і мутації.

Еволюційні обчислення використовують такі механізми природної еволюції:

1. Перший принцип заснований на концепції виживання найсильніших і природного відбору за Дарвіном, який був сформульований ним в 1859 році в книзі «Походження видів шляхом природного відбору». Згідно Дарвіну особини, які краще здатні вирішувати завдання в своєму середовищі, виживають і більше розмножуються (репродукують). У генетичних алгоритмах кожна особина є рішення деяких й проблеми. За аналогією з цим принципом особини з кращими значеннями цільової (фітнес) функції мають великі шанси вижити і репродукувати. Формалізація цього принципу, як ми побачимо далі, дає оператор репродукції.

2. Другий принцип обумовлений тим фактом, що хромосома нащадка складається з частин отриманих з хромосом батьків. Цей принцип був відкритий в 1865 році Менделем. Його формалізація дає основу для оператора схрещування (кросинговеру).

3. Третій принцип заснований на концепції мутації, відкритої в 1900 році де Вре. Спочатку цей термін використовувався для опису істотних (різких) змін властивостей нащадків і придбання ними властивостей, відсутніх у батьків. За аналогією з цим принципом генетичні алгоритми використовують подібний механізм для різкої зміни властивостей нащадків і тим самим, підвищують різноманітність (мінливість) особин в популяції (безлічі рішень).

Ці три принципи складають ядро еволюційних обчислень. Використовуючи їх, популяція (безліч варіантів розв'язання проблеми) еволюціонує від покоління до покоління.

Еволюцію штучної популяції - пошуку безлічі рішень деякої проблеми формально можна описати алгоритмом, який представлений на рисунку 2.1.

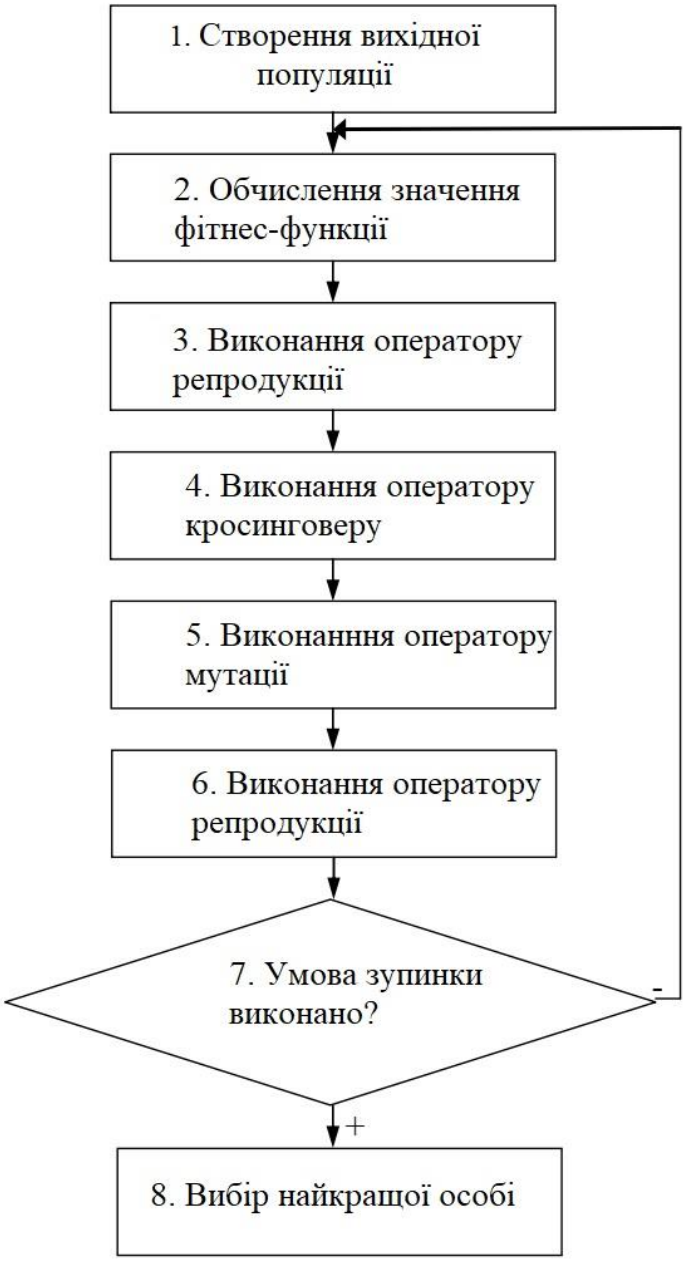


Рис. 2.1. - Структура генетичного алгоритму

Генетичний алгоритм бере безліч параметрів оптимізаційної проблеми і кодує їх послідовностями кінцевої довжини в деякому кінцевому алфавіті (в найпростішому випадку двійковий алфавіт «0» і «1»).

Попередньо простий генетичний алгоритм випадковим чином генерує початкову популяцію стрінгів (хромосом). Потім алгоритм генерує наступне покоління (популяцію), за допомогою трьох основних генетичних операторів:

- оператор репродукції;

- оператор схрещування (кросинговеру);

- оператор мутації.

Генетичний алгоритм працює до тих пір, поки не буде виконано задану кількість поколінь (ітерацій) процесу еволюції або на деякій генерації буде отримано задану якість або внаслідок передчасної збіжності при попаданні в деякий локальний оптимум. У кожному поколінні безліч штучних особин створюється з використанням старих і додаванням нових з хорошими властивостями.

Генетичні оператори є математичної формалізацією наведених вище трьох основоположних принципів Дарвіна, Менделя і де Вре природної еволюції.

Створення вихідної популяції. Популяція - це множина особин (рішень), які представляються хромосомами. Хромосома - ця впорядкована структура генів. Для завдання оптимізації числової функції як генів хромосоми виступають аргументи цієї функції. Для вирішення завдання пошуку оптимального маршруту в якості генів хромосоми виступають міста (вершини графа).

Існують три основні способи створення популяції:

- стратегія «ковдри» (формування повної популяції). Практично не реалізуємо, внаслідок великої обчислювальної складності;

- стратегія «дробовика» (формування досить великого підмножини повної популяції). Використовується найчастіше;

- стратегія «фокусування» (формування популяції з різновидів одного рішення). Використовується, якщо є припущення щодо рішення. У цьому випадку алгоритм почне роботу в околиці оптимуму.

Потужність популяції K є найважливішим параметром генетичного алгоритму. Чим більше K, тим більше різноманітність потенційних рішень (при хорошій схемою ініціалізації, що забезпечує однорідний розподіл часток). Велике число особин дозволяє покрити більшу частину простору пошуку за ітерацію. З іншого боку велика кількість особин підвищує обчислювальну складність ітерації і при цьому генетичний алгоритм може виродитися в випадковий паралельний пошук. Якщо K мало, то генетичний алгоритм працює швидко, але при цьому

збільшується небезпека передчасної збіжності до локального екстремуму. Велика потужність популяції збільшує генофонд, але процес пошуку сповільнюється. Зазвичай вважають K е [30; 200]. На різних етапах роботи генетичного алгоритму оптимальне значення K може бути різним. На початковому етапі K має бути великим, а на заключному K можна зменшити.

Фітнес-функція визначає пристосованість даної особини в популяції. На кожній ітерації генетичного алгоритму пристосованість кожної особини популяції оцінюється за допомогою фітнес-функції.

У разі пошуку мінімуму функції $f(x), x \in [x^{min}, x^{max}]$, фітнес-функція може бути представлена у вигляді $F(x) = f(x) \to min$

У разі пошуку оптимального маршруту фітнес-функція представлена у вигляді:

$$F(x) = d_{x_M,x_1} + \sum_{i=1}^{M-1} d_{x_i,x_{i+1}} \to min$$

де d_{x_M,x_1} - вага ребра;

V = {1, ..., M} - множина вершин;

x - вектор вершин;

M - довжина вектору вершин.

Слід зазначити, що в загальному випадку цільова функція і фітнес-функція можуть відрізнятися. Цільова функція призначена для оцінки характеристик особи щодо кінцевої мети (наприклад, екстремумів). Фітнес-функція призначена, перш за все, для відбору особин для подальшої репродукції і тут важливі характеристики якості однієї особини щодо інших особин. Після декодування хромосоми, де виконується перетворення генотип →фенотип (наприклад, двійковий код перетвориться в дійсне число), отримане значення далі використовується як аргумент для фітнес-функції. Далі для кожної особини популяції обчислюються значення фітнес-функції, які ранжують ці особі відносно один одного в сенсі перспективності отримання з них хорошого рішення.

Визначення відповідної фітнес-функції є вирішальним для коректної роботи генетичного алгоритму. Зокрема, вид фітнес-функції може залежати від накладених обмежень при вирішенні оптимізаційних задач. Відзначимо, що оператори кросинговеру і мутації не враховують, чи потрапляють новозбудовані особини - нащадки в область допустимих рішень, яка обумовлена накладаються обмеженнями.

У генетичному алгоритмі використовуються чотири основні методи для обліку накладених обмежень при вирішенні оптимізаційних задач. Ймовірно, найпростішим способом є метод відхилення (відкидання), де неприпустимі хромосоми (що не задовольняють обмеженням) виключаються з подальшої еволюції. Другий метод заснований на використанні процедури відновлення, яка перетворює отримане неприпустиме рішення в допустимий. Іншою альтернативою є застосування проблемно-орієнтованих генетичних операторів, які породжують тільки допустимі рішення.

Розглянуті методи не будують неприпустимих рішень. Але це не завжди дає хороші результати. Наприклад, в тому випадку, коли оптимальні рішення лежать на межі допустимої області, зазначені методи можуть давати неоптимальні рішення. Одним з можливих варіантів подолання цієї проблеми є виконання процедури відновлення тільки для деякого підмножини рішень (наприклад, 10% особин).

Для вирішення оптимізаційних завдань зі складними обмеженнями іноді дозволяють вести пошук рішення і в неприпустимих областях. Реалізується це підхід часто за допомогою методу штрафних функцій, що дозволяє розширити простір пошуку рішень. Слід зазначити, що часто неприпустима точка, близька до оптимального рішення, містить більше корисної інформації, ніж допустима точка, далека від оптимуму. З іншого боку, побудова штрафних функцій є досить складною проблемою, яка сильно залежить від розв'язуваної задачі. Зазвичай немає апріорної інформації про відстані до оптимальних точок, є тільки відстань до кордону області допустимих рішень. Тому, як правило, штрафні функції

використовують відстань до кордонів допустимої області. Штрафи, засновані на порушенні окремих обмежень, працюють зазвичай не дуже добре.

Розроблені два основних способи побудови штрафних функцій зі штрафним термом: адитивна і мультиплікативна форма. У першій формі функція представляється у вигляді $g(x) = f(x) + p(x)$, де при максимізації для допустимих точок $p(x) = 0$ і в іншому випадку $p(x) < 0$. Максимум значення $p(x)$ по абсолютній величині не може бути більше, ніж мінімальне значення $f(x)$ по абсолютній величині для будь-якої генерації, щоб уникнути негативних значень фітнес-функції. Мультиплікативна форма являє функцію у вигляді $g(x) = f(x) \cdot p(x)$, де при максимізації $p(x) = 1$ для допустимих точок і $0 < p(x) < 1$ в іншому випадку.

При цьому штрафний терм повинен змінюватися не тільки в залежності від ступеня порушення обмеження, але і від номера покоління генетичного алгоритму. Поряд з порушенням обмеження, штрафний терм зазвичай містить штрафні коефіцієнти (по одному для кожного обмеження). На практиці велику роль відіграють значення цих коефіцієнтів. Маленькі значення можуть привести до неприпустимих значень, в той час як великі значення повністю відкидають неприпустимі підпростори. В середньому абсолютні значення цільової і штрафний функції повинні бути порівнянні. При такому підході параметри штрафний функції можна включити в параметри генетичного алгоритму, що дозволяє розробити адаптивний метод, де значення коефіцієнтів регулюються в процесі пошуку рішення.

В цілому, на вибір (побудова) фітнес-функції впливають такі чинники:

1. Тип завдання - максимізація або мінімізація.

2. Зміст шумів навколишнього середовища в фітнес-функції.

3. Можливість динамічного зміни фітнес-функції в процесі виконання завдання.

4. Обсяг допустимих обчислювальних ресурсів - чи допускається використовувати більш точні методи і значні ресурси або можливі тільки наближені апроксимації, які не потребують великих ресурсів.

5. Наскільки різні значення для особин повинна давати фітнес-функція для полегшення відбору батьківських особин.

6. Чи повинна фітнес-функція містити обмеження розв'язуваної задачі.

7. Чи може фітнес-функція поєднувати різні підцілі (наприклад, для багатокритеріальних задач).

У генетичному алгоритмі фітнес-функція використовується у вигляді чорного ящика: для даної хромосоми вона обчислює значення, що визначає якість даної особини. Усередині вона може бути реалізована по-різному: у вигляді математичної функції, програми моделювання (в тому числі імітаційного), нейронної мережі, або навіть експертної оцінки.

2.2.1 Оператори репродукції (селекції) та кросинговеру (кросовера, рекомбінації)

Оператор репродукції створює проміжну популяцію шляхом відбору особин з поточної популяції з наступним їх копіюванням. При відборі використовується фітнес-функція, відповідно до значень якої особини попередньо упорядковано (упорядковуються).

В якості оператора репродукції можуть використовуватися:

1. Пропорційний відбір (рулетка). Для цього виду відбору в ряді випадків доводиться виконувати масштабування фітнес-функції:

а) зрушення (у разі пошуку мінімуму функції і оптимального маршруту)

$$F1(x_1) = maxF(x_1) - F(x_1)$$

б) лінійне (для виділення кращих особин)

$$a = \frac{\frac{1}{K}\sum_{i=1}^{K} F(x_i)}{\frac{1}{K}\sum_{i=1}^{K} F(x_i) - \min\limits_i F(x_i)}, b = -\frac{\left(\frac{1}{K}\sum_{i=1}^{K} F(x_i)\right)\min\limits_i F(x_i)}{\frac{1}{K}\sum_{i=1}^{K} F(x_i) - \min\limits_i F(x_i)}$$

причому a, b задовольняють трьом умовам

$$\frac{1}{K}\sum_{i=1}^{K} F1(x_i) = \frac{1}{K}\sum_{i=1}^{K} F(x_i)$$

де K - потужність популяції;

в) сигма-відсікання (для виділення кращих особин)

$$F1(x_i) = max(0, F(x_1) + (m - c\sigma)),$$

де m, σ - математичне сподівання і середньоквадратичне відхилення по популяції, c = 1÷5 - натуральна константа;

г) статечне (для виділення кращих особин)

$$F1(x_i) = F^k(x_i),$$

де k> 1 - константа (зазвичай k = 2÷3)

д) Больцмана (імітація відпалу) (для виділення кращих особин)

$$F1(x_i) = exp\left(\frac{F(x_i)}{T(n)}\right)$$

де $T(n)$ - температура на ітерації n.

Потім особі упорядковуються по фітнес-функції і визначається ймовірність вибору кожної i-ої хромосоми у вигляді:

$$P(x_i) = \frac{F1(x_i)}{\sum_{s=1}^{K} F1(x_s)}$$

Використовується на кінцевих стадіях роботи генетичного алгоритму, оскільки забезпечує спрямованість пошуку (поточні кращі хромосоми зберігаються), але не досліджує весь простір пошуку (вибираються тільки поточні кращі хромосоми і глобально найкраща хромосома може бути не знайдена). Цей відбір вимагає масштабування і не може використовуватися при мінімізації фітнес-функції.

2. Лінійно упорядкований відбір (лінійне ранжування). Перевагою цього методу є те, що він може використовуватися для пошуку мінімуму.

Ймовірність вибору кожної і -й хромосоми визначається у вигляді:

$$p(x_i) = \frac{1}{K}\left(a - (2a - 2)\frac{i - 1}{K - 1}\right), i \in \overline{1, K}$$

де a - випадкове число, $a \in [1,2]$.

Використовується на кінцевих стадіях роботи генетичного алгоритму, оскільки забезпечує спрямованість пошуку (поточні кращі хромосоми

зберігаються), але не досліджує весь простір пошуку (вибираються тільки поточні кращі хромосоми і глобально найкраща хромосома може бути не знайдена). Цей відбір не вимагає масштабування і може використовуватися при мінімізації фітнес-функції.

3. Нелінійно упорядкований відбір (нелінійне ранжування). Спочатку особини упорядковуються по фітнес-функції. Потім визначається ймовірність вибору кожної і -й хромосоми у вигляді

$$p(x_i) = \frac{1}{cK}\left(a - (a-b)\left(\frac{i-1}{K-1}\right)^2\right), c = a - (a-b)\frac{2K-1}{6K-6}, i \in \overline{1,K}$$

де a, b - випадкове число, $0 < a < b$

або

$$p(x_i) = a(1-a)^{K-i}, i \in \overline{1,K}$$

де a - випадкове число, $0 < a < 1$.

Використовується на кінцевих стадіях роботи генетичного алгоритму, оскільки забезпечує спрямованість пошуку (поточні кращі хромосоми зберігаються), але не досліджує весь простір пошуку (вибираються тільки поточні кращі хромосоми і глобально найкраща хромосома може бути не знайдена). Цей відбір не вимагає масштабування і може використовуватися при мінімізації фітнес-функції. Використовується рідше лінійного відбору.

4. Випадковий (однорідний, рівноймовірності, рівномірний) відбір. Є окремим випадком ранжирування при $a = 1$. Можливість вибору кожної i-ої хромосоми визначається у вигляді

$$p(x_i) = \frac{1}{K}, i \in \overline{1,K}$$

Використовується на початкових стадіях роботи генетичного алгоритму, оскільки забезпечує дослідження всього простору пошуку (випадковий вибір хромосом), але відсутня спрямованість пошуку (поточні кращі хромосоми можуть бути замінені гіршими і найкраща хромосома може загубитися). Цей відбір не вимагає масштабування і може використовуватися при мінімізації фітнес-функції.

5. Турнірний відбір. Перевагою цього методу є те, що він може використовуватися для пошуку мінімуму, і не вимагає масштабування. Популяція випадковим чином розбивається на групи розміру m, зазвичай m = 2÷3. Потім в кожній групі вибирається одна особина. Якщо вибір детермінований, то з групи завжди вибирається тільки особина з кращим значенням фітнес-функції. Якщо вибір випадковий, то для групи розміром $m = 2$ ймовірність вибору обох її особин i та j визначається у вигляді:

$$p(x_i) = \frac{1}{1 + exp\left(\frac{f(x_j) - f(x_i)}{T(n)}\right)} \text{ та } p(x_j) = \frac{1}{1 + exp\left(\frac{f(x_i) - f(x_j)}{T(n)}\right)}$$

де $T(n)$ - температура на ітерації n.

Обрані особини копіюються (число копій m) і поміщаються до проміжної популяції. Цей метод використовується як на початкових, так і на кінцевих стадіях роботи генетичного алгоритму.

6. Комбінація випадкового і пропорційного відбору з імітацією відпалу. Особи упорядковуються по фітнес-функції і ймовірність вибору кожної i-ої хромосоми визначається у вигляді:

$$p(x_i) = \frac{1}{K} exp\left(-\frac{1}{T(n)}\right) + \frac{F1(x_i)}{\sum_{s=1}^{K} F1(x_s)}\left(1 - exp\left(-\frac{1}{T(n)}\right)\right), i \in \overline{1, K}$$

$T(n) = \beta T(n-1)$, $0 < \beta < 1$, $T(0) = T_0$, $T_0 > 0$, n - номер ітерації.

Використовується як на початкових, так і на кінцевих стадіях роботи генетичного алгоритму, оскільки на ранніх стадіях роботи генетичного алгоритму використовується рівноймовірний відбір, що забезпечує дослідження всього простору пошуку (випадковий вибір хромосом), а на заключних стадіях використовується пропорційний відбір, який робить пошук спрямованим (поточні кращі хромосоми зберігаються). Ця комбінація вимагає масштабування і не може використовуватися при мінімізації фітнес-функції.

7. Комбінація випадкового і лінійно впорядкованого відбору з імітацією відпалу. Особи упорядковуються по фітнес-функції і ймовірність вибору кожної і-ї хромосоми визначається у вигляді

$$p(x_i) = \frac{1}{K} exp\left(-\frac{1}{T(n)}\right) + \frac{1}{K}\left(a - (2a-2)\frac{i-1}{K-1}\right)\left(1 - exp\left(-\frac{1}{T(n)}\right)\right)$$

T(n) = βT(n -1), 0 <β<1, T(0) = T_0, T_0> 0, n - номер ітерації.

Використовується як на початкових, так і на кінцевих стадіях роботи генетичного алгоритму, оскільки на ранніх стадіях роботи генетичного алгоритму використовується рівноймовірний відбір, що забезпечує дослідження всього простору пошуку (випадковий вибір хромосом), а на заключних стадіях використовується лінійно упорядкований відбір, який робить пошук спрямованим (поточні кращі хромосоми зберігаються). Ця комбінація не вимагає масштабування і може використовуватися при мінімізації фітнес-функції.

Для операторів 1-4 після визначення ймовірності вибору кожної і -й хромосоми на відрізку [0,1] для кожної особини будуються відрізки, довжини яких відповідають можливостям вибору особин. Приклад для пропорційного відбору наведено на рисунку 2.2 для лінійно впорядкованого відбору різниця між сусідніми довжинами буде однакова, а для нелінійно впорядкованого відбору буде неоднакова і залежить від номера і. Для випадкового відбору довжини будуть однаковими. Далі до-раз випадково генеруються числа з [0,1] і в проміжну популяцію вибираються ті особини, в чиї відрізки потрапляють ці випадкові числа.

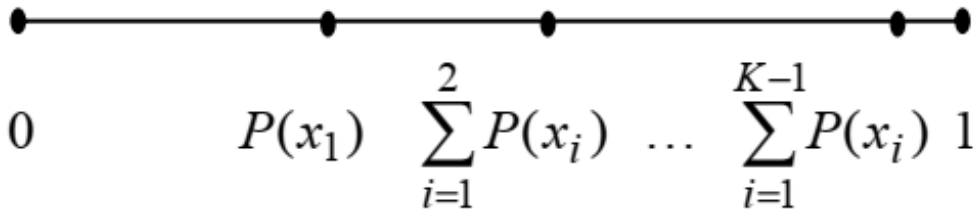


Рис. 2.2. - Ймовірності вибору особин для пропорційного відбору

Особи, отримані шляхом репродукції, необхідно схрестити між собою. Перед схрещуванням необхідно попередньо вибрати батьків, які повинні бути різні.

Розглянемо методи вибору пар для схрещування.

1. Випадковий (однорідний, рівноймовірності, рівномірний) відбір Вибір обох батьків відбувається з ймовірністю

$$p(x_i) = \frac{1}{K}$$

Якщо не використовується імітація відпалу, то цей кросинговер над вибраними батьками відбувається зазвичай з ймовірністю 0.5 Для імітації відпалу кросинговер відбувається з імовірністю $p_0 = p_0 exp\ (-1/T\ (n)),\ T(n) = \beta T\ (n-1),\ 0 < \beta < 1,\ T(0)=T_0,\ T_0 > 0$, де T_0 - початкова ймовірність кросинговеру, n - номер ітерації.

Використовується на початкових стадіях роботи генетичного алгоритму, оскільки забезпечує дослідження всього простору пошуку (випадковий вибір хромосом), але відсутня спрямованість пошуку (поточні кращі хромосоми можуть бути замінені гіршими і найкраща хромосома може загубитися). Цей відбір не вимагає масштабування і може використовуватися при мінімізації фітнес-функції.

2. Селективний відбір. Є окремим випадком схрещування кращих хромосом з кращими хромосомами (пропорційного відбору).

Вибір обох батьків відбувається, якщо значення їх фітнес-функції не менше середнього значення по проміжній популяції.

Якщо не використовується імітація відпалу, то цей кросинговер над вибраними батьками відбувається зазвичай з ймовірністю 0.5. Для імітації відпалу кросинговер відбувається з імовірністю $p_0 = p_0 exp\ (-1/T\ (n)),\ T(n) = \beta T\ (n-1),\ 0 < \beta < 1,\ T(0)=T_0,\ T_0 > 0$, де T_0 - початкова ймовірність кросинговеру, n - номер ітерації.

Використовується на кінцевих стадіях роботи генетичного алгоритму, оскільки забезпечує спрямованість пошуку (поточні кращі хромосоми зберігаються), але не досліджує весь простір пошуку (вибираються тільки поточні кращі хромосоми і глобально найкраща хромосома може бути не знайдена). Цей відбір вимагає масштабування і не може використовуватися при мінімізації фітнес-функції.

3. Схрещування кращих хромосом з гіршими хромосомами Вибір першого (кращого) батька відбувається з ймовірністю:

$$p(x_i) = \frac{F(x_i)}{\sum_{s=1}^{K} F(x_s)}$$

Вибір другого (гіршого) батька відбувається з ймовірністю

$$p(x_r) = \frac{F(x_r)}{\sum_{s=1}^{K} F(x_s)}$$

Якщо не використовується імітація відпалу, то цей кросинговер над вибраними батьками відбувається зазвичай з ймовірністю 0.5 Для імітації відпалу кросинговер відбувається з імовірністю $p_0 = p_0 exp\ (-1/T\ (n)),\ T(n) = \beta T\ (n-1),\ 0 < \beta < 1,\ T(0)=T_0,\ T_0 > 0$, де $T_0 > 0$. Використовується на початкових стадіях роботи генетичного алгоритму, оскільки забезпечує дослідження всього простору пошуку (схрещування кращих хромосом з гіршими), але відсутня спрямованість пошуку (відсутня схрещування кращих хромосом з кращими). Цей відбір вимагає масштабування і не може використовуватися при мінімізації фітнес-функції.

4. Схрещування кращих хромосом з кращими хромосомами (пропорційний відбір)

Вибір обох батьків відбувається з ймовірністю

$$p(x_i) = \frac{F(x_i)}{\sum_{s=1}^{K} F(x_s)}$$

Якщо не використовується імітація відпалу, то цей кросинговер над вибраними батьками відбувається зазвичай з ймовірністю 0.5

Для імітації відпалу кросинговер відбувається з імовірністю $p_0 = p_0 exp\ (-1/T\ (n)),\ T(n) = \beta T\ (n-1),\ 0 < \beta < 1,\ T(0)=T_0,\ T_0 > 0$, де $T_0 > 0$.

Використовується на кінцевих стадіях роботи генетичного алгоритму, оскільки забезпечує спрямованість пошуку (схрещування кращих хромосом з кращими), але не досліджує весь простір пошуку (відсутня схрещування кращих хромосом з гіршими). Цей відбір вимагає масштабування і не може використовуватися при мінімізації фітнес-функції.

5. Аутбридинг («далеке спорідненість»). Перший батько вибирається випадково, а другий - як максимально далекий від першого або як що знаходиться від першого на відстані більшій, ніж задане s. Як відстань між батьками може використовуватися відстань Манхеттена.

Якщо не використовується імітація відпалу, то цей кросинговер над вибраними батьками відбувається зазвичай з ймовірністю 0.5

Для імітації відпалу кросинговер над вибраними батьками відбувається з ймовірністю $p_0 = p_0 exp\ (-1/T\ (n)),\ T(n) = \beta T\ (n-1),\ 0 < \beta < 1,\ T(0)=T_0,\ T_0 > 0$, де $T_0 > 0$.

Використовується на початкових стадіях роботи генетичного алгоритму, оскільки забезпечує дослідження всього простору пошуку (схрещування далеких батьків), але відсутня спрямованість пошуку (відсутня схрещування найближчих батьків). Цей відбір не вимагає масштабування і може використовуватися при мінімізації фітнес-функції.

6. Інбридинг («близьку спорідненість»). Перший батько вибирається випадково, а другий - як максимально близький до першого або як що знаходиться від першого на відстані меншій, ніж заданий s.

Якщо не використовується імітація відпалу, то цей кросинговер над вибраними батьками відбувається зазвичай з ймовірністю 0.5.

Для імітації відпалу кросинговер над вибраними батьками відбувається з ймовірністю $p_0 = p_0 exp\ (-1/T\ (n)),\ T(n) = \beta T\ (n-1),\ 0 < \beta < 1,\ T(0)=T_0,\ T_0 > 0$, де $T_0 > 0$.

Використовується на кінцевих стадіях роботи генетичного алгоритму, оскільки забезпечує спрямованість пошуку (схрещування найближчих батьків), але не досліджує весь простір пошуку (відсутня схрещування далеких батьків). Цей відбір не вимагає масштабування і може використовуватися при мінімізації фітнес-функції.

Таким чином, можливі два ефективні комбінації:

1. На ранніх стадіях роботи генетичного алгоритму використовується аутбридинг, що забезпечує дослідження всього простору пошуку, а на заключних стадіях використовується інбридинг, що робить пошук спрямованим. Ця комбінація краще, оскільки не вимагає масштабування і може використовуватися при мінімізації фітнес-функції.

2. На ранніх стадіях роботи генетичного алгоритму використовується схрещування кращих хромосом з гіршими, забезпечує дослідження всього простору пошуку, а на заключних стадіях використовуються схрещування кращих хромосом з кращими, що робить пошук спрямованим. Ця комбінація вимагає масштабування і не може використовуватися при мінімізації фітнес-функції.

Розглянемо далі детально кросинговер над цілими хромосомами для завдання пошуку оптимального маршруту

В якості оператора кросинговеру можуть використовуватися: 1. Частково відповідний кросинговер Нехай *c1*, *c2* - обрані випадковим чином точки схрещування (дві вершини), причому *c1*<*c2*, і хромосоми батьків представлені як:

$$x_1 = x_{11} \dots x_{1,c1} x_{1,c1+1} \dots x_{1,c2} x_{1,c2+1} \dots x_{1M}$$

$$x_2 = x_{21} \dots x_{2,c1} x_{2,c1+1} \dots x_{2,c2} x_{2,c2+1} \dots x_{2M}$$

Тоді нащадок x_3 створюється у вигляді:

1. $x_{3j} = x_{1j}, j \in \overline{c1+1, c2}$

2. $j = c2 + 1$

3. Якщо, $\{x_{1,c1+1}, \dots, x_{1,c2}\} \cap \{x_{2j}\} = \emptyset$, то $x_{3j} = x_{2j}$

4. Якщо, $\{x_{1,c1+1}, \dots, x_{1,c2}\} \cap \{x_{2j}\} = \{x_{1m}\}$, то $x_{3j} = x_{2m}$

5. $j = j + 1$

6. Якщо $j > M$, то $j = 1$

7. Якщо $j = c1 + 1$, то зупинка, інакше перехід до кроку 3.

Нащадок x_4 створюється у вигляді:

1. $x_{4j} = x_{2j}, j \in \overline{c1 + c2}$

2. $j = c2 + 1$

3. Якщо, $\{x_{2,c1+1}, \dots, x_{2,c2}\} \cap \{x_{1j}\} = \emptyset$, то $x_{4j} = x_{1j}$

4. Якщо, $\{x_{2,c1+1}, \dots, x_{2,c2}\} \cap \{x_{1j}\} = \{x_{2m}\}$, то $x_{4j} = x_{1m}$

5. $j = j + 1$

6. Якщо $j > M$, то $j = 1$

7. Якщо $j = c1 + 1$, то зупинка, інакше перехід до кроку 3

2. Упорядкований кросинговер. Нехай *c1*, *c2* - обрані випадковим чином точки схрещування (дві вершини), причому *c1*<*c2*, і хромосоми батьків представлені як:

$$x_1 = x_{11} \dots x_{1,c1} x_{1,c1+1} \dots x_{1,c2} x_{1,c2+1} \dots x_{1M}$$

$$x_2 = x_{21} \dots x_{2,c1} x_{2,c1+1} \dots x_{2,c2} x_{2,c2+1} \dots x_{2M}$$

Тоді нащадок x_3 створюється у вигляді:

1. $x_{3j} = x_{1j}, j \in \overline{c1 + c2}$

2. $j = c2 + 1, r = c2 + 1$

3. Якщо $x_{2r} \in \{x_{1,c1+1} \dots x_{1,c2}\}$, то перехід до кроку 7

4. $x_{3j} = x_{2r}, j = j + 1$

5. Якщо $j > M$, то $j = 1$

6. Якщо $j = c1 + 1$, то зупинка

7. $r = r + 1$

8. Якщо $r > M$, то $r = 1$, перехід до кроку 3 Нащадок x_4 створюється у вигляді:

1. $x_{4j} = x_{2j}, j \in \overline{c1 + c2}$

2. $j = c2 + 1, r = c2 + 1$

3. Якщо $x_{1r} \in \{x_{2,c1+1} \dots x_{2,c2}\}$, то перехід до кроку 7

4. $x_{4j} = x_1 r, j = j + 1$

5. Якщо $j > M$, то $j = 1$

6. Якщо $j = c1 + 1$, то зупинка

7. $r = r + 1$

8. Якщо $r > M$, то $r = 1$, перехід до кроку 3

3. Циклічний кросинговер

Нехай хромосоми батьків представлені як:

$x_1 = x_{11} \dots x_{1M}$

$x_2 = x_{21} \dots x_{2M}$

Тоді нащадок x_3 створюється у вигляді:

1. $x_{31} = (0, \dots, 0)$

2. $x_{31} = x_{11}, s = 1, V^{tabu} = \{x_{11}\}$

3. Якщо, $\{x_{11}, \dots, x_{1M}\} \cap \{x_{2s}\} = \{x_{1r}\}$ та $x_{1r} \notin V^{tabu}$, то $x_{3r} = x_{1r}, s = r, V^{tabu} = V^{tabu} \cup \{x_{1r}\}$, перехід на крок 3

4. $s = 1, r = 1$

5. Якщо $x_{2r} \in V^{tabu}$, то $r = r + 1$, перехід на крок 5

6. Якщо $x_{3s} > 0$, то $s = s + 1$, перехід на крок 6

7. $x_{3s} = x_{2r}, V^{tabu} = V^{tabu} U \{x_{2r}\}$

8. Якщо $|V^{tabu}| < M$, то $r = r + 1$, $s = s + 1$, перехід на крок 5

Нащадок x_4 створюється у вигляді:

1. $x_{41} = (0, ..., 0)$

2. $x_{41} = x_{21}$, $s = 1$, $V = \{x_{21}\}$

3. Якщо $\{x_{21}, \dots, x_{2M}\} \cap \{x_{1s}\} = \{x_{2r}\}$ та $x_{2r} \notin V^{tabu}$, то $x_{4r} = x_{2r}, s = r, V^{tabu} = V^{tabu} \cup \{x_{2r}\}$, перехід на крок 3

4. $s = 1$, $r = 1$

5. Якщо $x_{1r} \in V^{tabu}$, то $r = r + 1$, перехід на крок 5

6. Якщо $x_{4s} > 0$, то $s = s + 1$, перехід на крок 6

7. $x_{4s} = x_{1r}$, $V^{tabu} = V^{tabu} \cup \{x_{1r}\}$

8. Якщо $|V^{tabu}| < M$, то $r = r + 1$, $s = s + 1$, перехід на крок 5 Найбільш ефективним вважається частково відповідний кросинговер.

2.2.2 Оператор мутації

Оператор мутації дозволяє отримати з хромосом, отриманих на етапі кросинговеру, нові хромосоми з різко відрізняються властивостями.

Розглянемо детально далі мутацію над цілими хромосомами для завдання пошуку оптимального маршруту

В якості оператора мутації можуть використовуватися:

1. Мутація на основі обміну. Випадково вибирається хромосома. Випадковим чином вибираються з набору генів цієї хромосоми два гена *c1* і *c2*, причому вибір цих генів триває до тих пір, поки не буде виконана умова:

$$1 < c2 - c1 < M - 1.$$

Ці гени обмінюються місцями

2. Мутація на основі вставки. Випадково вибирається хромосома. Випадковим чином вибираються з набору генів цієї хромосоми два гена *c1* і *c2*, причому вибір цих генів триває до тих пір, поки не буде виконана умова

$$1 < c2 - c1 < M - 1$$

Ген с2 ставиться перед геном (або після гена) с1

3. Мутація на основі переміщення. Випадково вибирається хромосома. Випадковим чином вибираються з набору генів цієї хромосоми гена попарно різні гени *c1*, *c2* і *c3*, причому вибір цих генів триває до тих пір, поки не буде виконана умова *c1 <c2 <c3 або c3 <c1 <c2*. Гени *c1, ..., c2* ставляться перед геном (або після гена) *c1*

4. Мутація на основі перестановки 2-opt. Випадково вибирається хромосома. Випадковим чином вибираються з набору генів цієї хромосоми два гена *c1* і *c2*, причому вибір цих генів триває до тих пір, поки не буде виконана умова *1 <c2 c1 <M - 1*.

На основі хромосоми

$$x_1 = x_{11} \dots x_{1,C1-1} x_{1,C1} \dots x_{1,C2} x_{1,C2+1} \dots x_{1M}$$

створюється хромосома

$$x_2 = x_{11} \dots x_{1,C1-1} x_{1,C2} \dots x_{1,C1} x_{1,C2+1} \dots x_{1M}$$

тобто гени $x_{1,C1} \dots x_{1,C2}$ переставляються в зворотному порядку.

5. Мутація на основі перестановки 3-opt. Випадково вибирається хромосома. Випадковим чином вибираються з набору генів цієї хромосоми два гена *c1* і *c2*, причому вибір цих генів триває до тих пір, поки не буде виконана умова *1<c2 c1 <M-1*.

На основі хромосоми

$$x_1 = x_{11} \dots x_{1,C1-1} x_{1,C1} \dots x_{1,C2} x_{1,C2+1} \dots x_{1M}$$

створюється хромосома

$$x_2 = x_{11} \dots x_{1,C1-1} x_{1,C2} \dots x_{1,M} x_{1,C1} \dots x_{1,C2-1}$$

тобто гени $x_{1,C1} \dots x_{1,C2}$ і $x_{1,C2} \dots x_{1,M}$ міняються місцями.

Слід зазначити, що перестановки більше 3-opt не суттєво підвищують точність пошуку. Найчастіше в метаеврістіках як мутації використовується перестановка 2-opt.

Імовірність мутації може бути:

1. Постійної. Зазвичай $p_m = 0.05 \div 0.1$ або $p_m = 1 / K$.

2. Перемінної, на основі імітації відпалу

$p_m = p_0 exp\ (-1 / T\ (n)),\ T\ (n) = pT\ (n-1),\ 0 < \beta < 1,\ T\ (0) = T_0,\ T_0 > 0$

Перевага змінної ймовірності мутації - на ранніх стадіях роботи генетичного алгоритму з високою ймовірністю відбувається мутація, що забезпечує дослідження всього простору пошуку, а на заключних стадіях ймовірність мутації прагнуть до нуля, що робить пошук спрямованим.

2.2.3 Оператор редукції

Оператор редукції дозволяє сформувати нову популяцію на основі попередньої популяції і нащадків (особин, отриманих шляхом кросинговеру і мутації), причому потужність нової популяції повинна збігатися з потужністю попередньої популяції, тобто повинна дорівнювати *K*. У наведених нижче схемах операторів редукції символ «μ» позначає потужність попередньої популяції, тобто $\mu = K$, символ «λ» позначає кількість нащадків, символ «+» позначає об'єднання, символ «,» позначає заміну.

В якості оператора редукції можуть використовуватися:

1. Схема ($\mu + \lambda$) (селективна схема). Особи попередньої популяції і нащадки об'єднуються і упорядковуються за значенням фітнес-функції. У нову популяцію відбирається μ перших кращих особин.

Використовується на кінцевих стадіях роботи генетичного алгоритму, оскільки забезпечує спрямованість пошуку (поточні кращі хромосоми зберігаються), але не досліджує весь простір пошуку (вибираються тільки поточні кращі хромосоми і глобально найкраща хромосома може бути не знайдена). Ця схема не вимагає масштабування і може використовуватися при мінімізації фітнес-функції.

2. Схема (μ, λ). Всі особини попередньої популяції замінюються кращими (за значенням фітнес-функції) нащадками. Кількість нащадків λ має бути більше потужності популяції μ.

У порівнянні зі схемою ($\mu + \lambda$) ця схема досліджує простір пошуку більш широко (більшу різноманітність популяції за рахунок її постійної зміни), але спрямованість пошуку гірше (хоча відбираються найкращі нащадки, але не

зберігаються кращі хромосоми попередньої популяції, частина з яких може перевершувати кращих нащадків). У порівнянні з випадковим відбором у цієї схеми спрямованість пошуку краще (відбираються найкращі нащадки), але досліджує простір пошуку більш вузько (меншу різноманітність популяції за рахунок відбору кращих нащадків). Ця схема не вимагає масштабування і може використовуватися при мінімізації фітнес-функції.

3. Схема (μ, μ). Є окремим випадком схеми (μ, λ). Всі нащадки заміняють всі особини попередньої популяції. Кількість нащадків λ має збігатися з потужністю популяції μ.

У порівнянні зі схемами ($\mu + \lambda$) і (μ, λ) ця схема досліджує весь простір пошуку (випадкова заміна хромосом), але відсутня спрямованість пошуку (поточні кращі хромосоми можуть бути замінені гіршими, і найкраща хромосома може загубитися). Ця схема не вимагає масштабування і може використовуватися при мінімізації фітнес-функції.

4. Випадкова (однорідна, рівноймовірності, рівномірна) схема Особи попередньої популяції і нащадки об'єднуються. Випадковим чином (з однаковою ймовірністю) в нову популяцію відбирається K особин, причому кожна особина може бути обрана з об'єднання тільки один раз.

Використовується на початкових стадіях роботи генетичного алгоритму, оскільки забезпечує дослідження всього простору пошуку (випадковий вибір хромосом), але відсутня спрямованість пошуку (поточні кращі хромосоми можуть бути замінені гіршими і найкраща хромосома може загубитися). Ця схема не вимагає масштабування і може використовуватися при мінімізації фітнес-функції.

5. Турнірна схема. Особи попередньої популяції і нащадки об'єднуються. Отримане об'єднання випадковим чином розбивається на групи розміру m, зазвичай m= 2÷3. Потім в кожній групі вибирається особина з кращим значенням фітнес-функції. Обрані особини утворюють нову популяцію. Кількість нащадків X збігається з потужністю популяції ц або кратно їй, при цьому ($\lambda + \mu$) / $\mu = m$. Ця схема не вимагає масштабування і може використовуватися при мінімізації фітнес-

функції. Використовується як на початкових, так і на кінцевих стадіях роботи генетичного алгоритму.

6. Комбінація селективної схеми і випадкової схеми з імітацією відпалу. На ранніх стадіях роботи генетичного алгоритму використовується випадкова схема (випадковий вибір хромосом), що забезпечує дослідження всього простору пошуку, а на заключних стадіях використовується селективна схема, що робить пошук спрямованим (поточні кращі хромосоми зберігаються). Ця комбінація не вимагає масштабування і може використовуватися при мінімізації фітнес-функції.

Можливість вибору випадкової схеми визначена за допомогою імітації відпалу у вигляді

$p_r = p_0 exp\ (-1 / T\ (n)),\ T\ (n) = \beta T\ (n-1),\ 0 < \beta < 1,\ T(0) = T_0,\ T_0 > 0,$

де p_0 - початкова ймовірність редукції.

Як умова зупинки використовуються:

- перевищення максимальної кількості ітерацій;

- перевищення кількості поколінь, протягом яких не покращується результат.

2.2.4 Використання генетичних алгоритмів для умовної оптимізації

Нехай функція мети і обмеження (лінійні і нелінійні) представлені у вигляді:

$$f(x) \to min$$

У найпростішому випадку особини, що не задовольняють обмеженням, просто видаляються з популяції. Такий підхід зазвичай використовують, якщо область допустимих рішень є опуклою. В іншому випадку це може привести до тривалого пошуку.

Найчастіше для вирішення завдань умовної оптимізації використовується штрафна функція F_w. Виділяють наступні групи штрафних функцій:

- статичні (не залежить від номера ітерації);

- динамічні (залежать від номера ітерації);

- адаптивні (параметри штрафний функції модифікуються з урахуванням популяції);

- напівадаптивні (штрафна функція розділена на дві частини - зважена сума значень обмежують функцій і зважена кількість порушених обмежень);

- на основі імітації відпалу.

Найбільш поширені такі штрафні функції:

1. Статична штрафна функція, запропонована Хомейфа (Homaifar), Леєм (Lai) і Ki (Qi):

$$F_w(x) = \sum_{z=1}^{Z_1} w_z |h_z(x)| + \sum_{z=1}^{Z_2} w_{z+Z_1} max\{0, g_z(x)\}$$

де w_z - штрафні коефіцієнти (ваги).

В цьому випадку фітнес-функція $F(x) = f(x) + F_w(x)$

2. Динамічна штрафна функція, запропонована Джойнесом (Joines) і Хуком (Houck)

$$F_w(x) = (\gamma \cdot n)\alpha \left(\sum_{z=1}^{Z_1} |h_z(x)|^\beta + \sum_{z=1}^{Z_2} |max\{0, g_z(x)\}|^\beta \right)$$

де α, β, γ - параметри (зазвичай $\alpha, \beta \in \{1,2\}$, $\gamma = 0.5$), n - номер ітерації.

В цьому випадку фітнес-функція $F(x) = f(x) + F_w(x)$

3. Динамічна штрафна функція, запропонована Казарлісом (Kazarlis) і Петрідісом (Petridis):

$$F_w(x) = \left(\frac{n}{N}\right)^2 \left(\alpha \left(\sum_{z=1}^{Z_1} w_z |h_z(x)| + \sum_{z=1}^{Z_2} w_z max\{0, g_z(x)\} \right) + \beta \right) \delta(x)$$

$$\delta(x) = \begin{cases} 1, \sum_{z=1}^{Z_1} |h_z(x)| + \sum_{z=1}^{Z_2} max\{0, g_z(x)\} \neq 0 \\ 0, \sum_{z=1}^{Z_1} |h_z(x)| + \sum_{z=1}^{Z_2} max\{0, g_z(x)\} = 0 \end{cases}$$

де α, β - параметри (зазвичай a = 1000, p = 0), n - номер ітерації, N - максимальне число ітерацій, w_z - штрафні коефіцієнти (ваги). У цьому випадку фітнес-функція $F(x) = f(x) + F_w(x)$

4. Адаптивна штрафна функція, запропонована Геном (Gen) і Ченгом (Cheng)

Нехай $w1_z^{max} = \max_{x \in P} |h_z(x)|, w2_z^{max} = \max_{x \in P} \{0, g_z(x)\}$

де P - поточна популяція.

Якщо $w1_z^{max} > 0$, $w2_z^{max} > 0$, то

$$F_w(x) = 1 - \frac{1}{Z_1 + Z_2}\left(\sum_{z=1}^{Z_1}\left|\frac{h_z(x)}{w1_z^{max}}\right|^{\alpha} + \sum_{z=1}^{Z_2}\left(\frac{max\{0, g_z(x)\}}{w2_z^{max}}\right)^{\alpha}\right)$$

де α - параметр, $\alpha > 1$.

Якщо $w1_z^{max} = 0$, $w2_z^{max} > 0$, то

$$F_w(x) = 1 - \frac{1}{Z_1 + Z_2}\left(\sum_{z=1}^{Z_2}\left(\frac{max\{0, g_z(x)\}}{w2_z^{max}}\right)^{\alpha}\right)$$

Якщо $w1_z^{max} > 0$, $w2_z^{max} = 0$, то

$$F_w(x) = 1 - \frac{1}{Z_1 + Z_2}\left(\sum_{z=1}^{Z_1}\left|\frac{h_z(x)}{w1_z^{max}}\right|^{\alpha}\right)$$

Якщо $w1_z^{max} > 0$, $w2_z^{max} = 0$, то $F_w = 1$.

В цьому випадку фітнес-функція $F(x) = f(x)F_w(x)$

5. Адаптивна штрафна функція, запропонована Хадж-Алоуаном (Hadj-Alouane) і Біном (Bean)

$$F_w(x) = \psi(n)\left(\sum_{z=1}^{Z_1}|h_z(x)| + \sum_{z=1}^{Z_2} max\{0, g_z(x)\}\right)$$

$$\psi(n) = \begin{cases} \alpha^{-1}\psi(n-1), x^* \in P, n - \tau + 1 \le n^* \le n \\ \beta\psi(n-1), x^* \notin P, n - \tau + 1 \le n^* \le n \\ \psi(n-1), \text{інакше} \end{cases}$$

де α, β, τ - параметри, $\alpha > \beta > 1$, τ - натуральне число,

n - номер ітерації з кращим рішенням x^*;

P - поточна популяція.

В цьому випадку фітнес-функція $F(x) = f(x) + F_w(x)$

6. Адаптивна штрафна функція, запропонована Хінтедіном (Hinterding)

$$F_w(x) = \psi(n)\left(\sum_{z=1}^{Z_1}|h_z(x)| + \sum_{z=1}^{Z_2} max\{0, g_z(x)\}\right)$$

Якщо $n \bmod \tau \neq 0$, то $\psi(n) = \psi(n-1)$

Якщо $n \bmod \tau = 0$, то $\psi(n) = \begin{cases} \alpha^{-1}\psi(n-1), p > 0{,}25 \\ \alpha\psi(n-1), p < 0{,}15 \\ \psi(n-1), 0{,}15 < p < 0{,}25 \end{cases}$

$$p = \frac{K2}{K}$$

де *K* - потужність популяції,

K2 - кількість особин поточної популяції, які відповідають всім умовам,

α, τ - параметри (зазвичай a = 1.3), τ - натуральне число.

В цьому випадку фітнес-функція $F(x) = f(x) + F_w(x)$

7. Адаптивна штрафна функція, запропонована Смітом (Smith) і Тейтем (Tate)

$$F_w(x) = \left(f(x1^{opt}) - f(x2^{opt})\right)\frac{\left(\sum_{z=1}^{Z_1}|h_z(x)| + \sum_{z=1}^{Z_2} max\{0, g_z(x)\}\right)^{\alpha}}{\psi^{\alpha}(n)}$$

$$\psi(n) = \frac{\gamma}{1 + n\beta}$$

де α, β, γ - параметри, $\alpha > 1$, $\gamma > 0$, $\beta > 0$,

$x1^{opt}$ - найкраще рішення без врахування обмежень,

$x2^{opt}$ - краще рішення з урахуванням обмежень, n - номер ітерації

В цьому випадку фітнес-функція $F(x) = f(x) + F_w(x)$

2.3 Порівняння продуктивності для різних операторів кросовера на різному розмірі популяції

Далі порівнюється ефективність операторів схрещування з базовими операторами (на різних розмірах популяції, кількості оцінок придатності тощо) і детально розбираємо, як інші параметри, такі як швидкість мутації, впливають на ефективність найкращих операторів, знайдених для транспортної задачі.

У цій групі експериментів вивчається, як діють різні оператори кросовера на прикладах задач у різному розмірі популяції (з урахуванням фіксованої кількості оцінок придатності). Операторами кросоверу, які розглядаються в цьому наборі експериментів, є одноточковий кросовер (ONE_POINT), двоточковий кросовер (TWO_POINT), триточковий кросовер (THREE_POINT), кросовер на основі позиції (ATC) та їх відповідні версії, позначені з додаванням _SHUFFLE. Мутація застосовується шляхом застосування однієї операції заміни, де випадковим чином вибираються два гени та обмінюються їх значеннями. Імовірність мутації встановлена на рівні 20%.

Таблиця 2.1 - Уточнення задач на завдання, використаних для порівняння [13].

Номер задачі	Загальна кількість агентів	Необхідна кількість агентів	Число завдань
1	10	10	4
2	20	20	4
3	20	20	8
4	30	30	8
5	30	30	12
6	60	60	12
7	80	80	20
8	96	96	30
9	400	400	100
10	800	800	100
11	1600	1600	100
12	1600	1600	100
13	400	400	200

Результати цих експериментів показані на рисунках 2.3 – 2.8. Перші три фігури зображують експерименти над задачею №9, де числа оцінок придатності встановлені на 10000, 40000 і 80000 відповідно. На рисунках 2.3 – 2.8 проілюстровані результати експериментів на прикладах задач № 10-12, з кількістю оцінки фітнесу встановленоъ на 10000. На рисунках 2.3 – 2.8 початковий розмір популяції, зображений на горизонтальній осі, змінюється від 2 до 1024. Вертикальна вісь відображає якість розчинів, яка визначається як середнє відсоткове відхилення від найкращого значення рішення, отриманого після 320000 оцінок фітнесу.

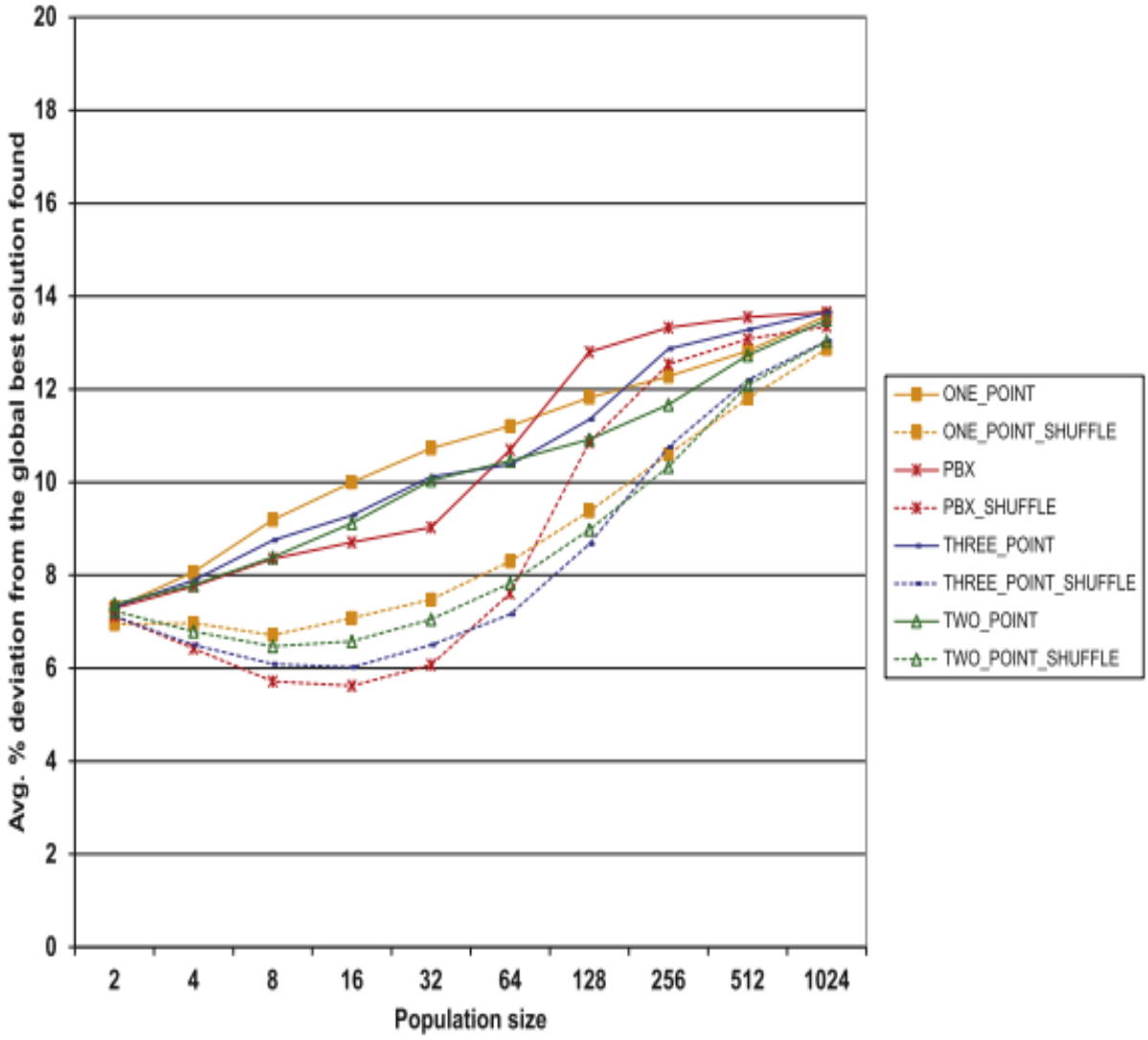


Рис. 2.3. - Вплив початкового розміру сукупності на якість розв’язків (результати задачі №9 з кількістю оцінок придатності = 10000)

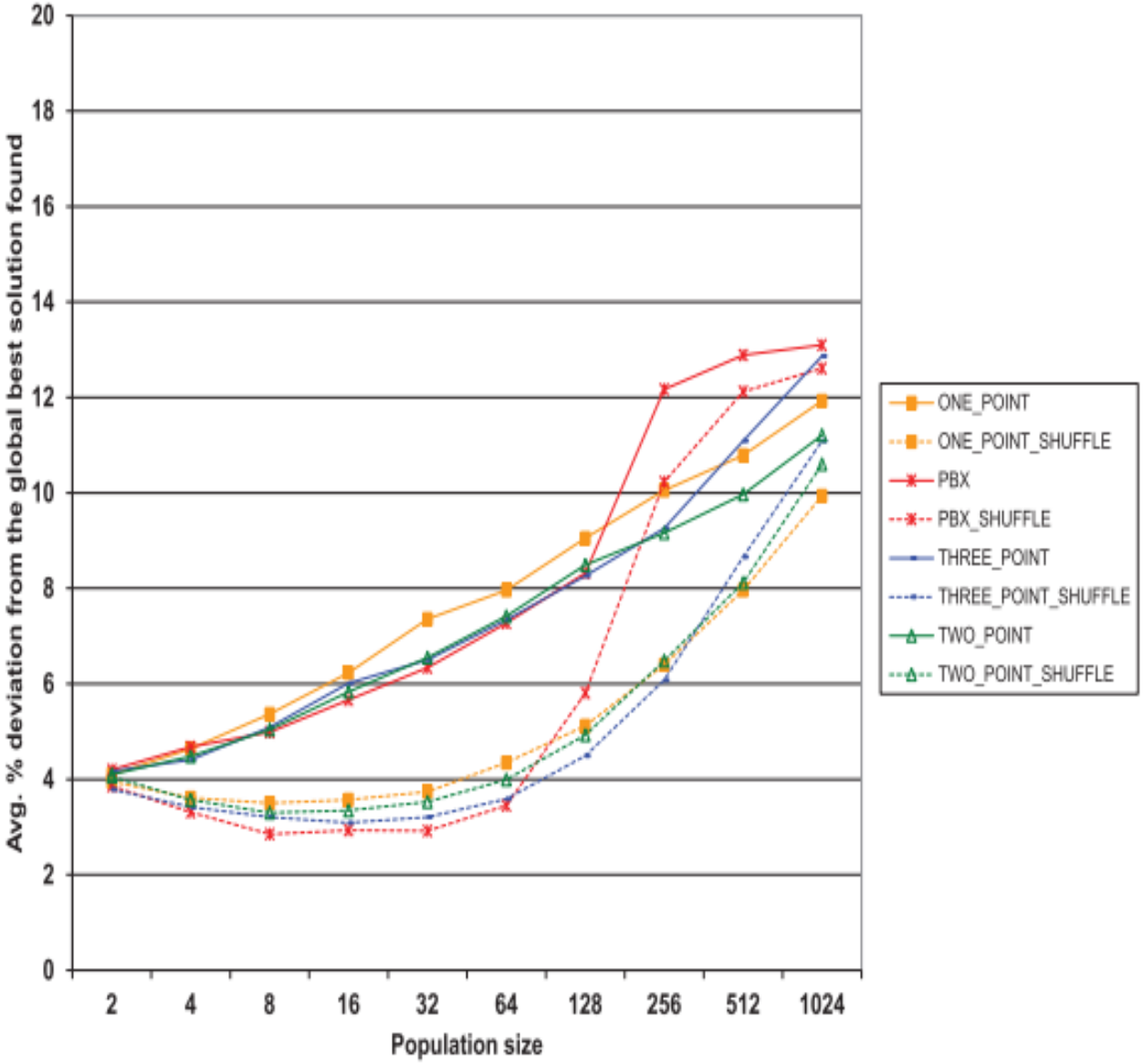


Рис. 2.4. - Вплив початкового розміру сукупності на якість розв'язків (результати задачі №9 з кількістю оцінок придатності = 40000)

Результати розрахунку GA з оператором перехресного переміщення одноточкового списку для задачі №10 і різною кількістю нащадків. Агенти працюють в командах без взаємодії, а оптимальне рішення виходить за допомогою угорського методу. avgDev та a – середнє значення та стандартне відхилення (для 10 повторень) dev. (відносне відхилення від оптимального рішення) , відповідно.

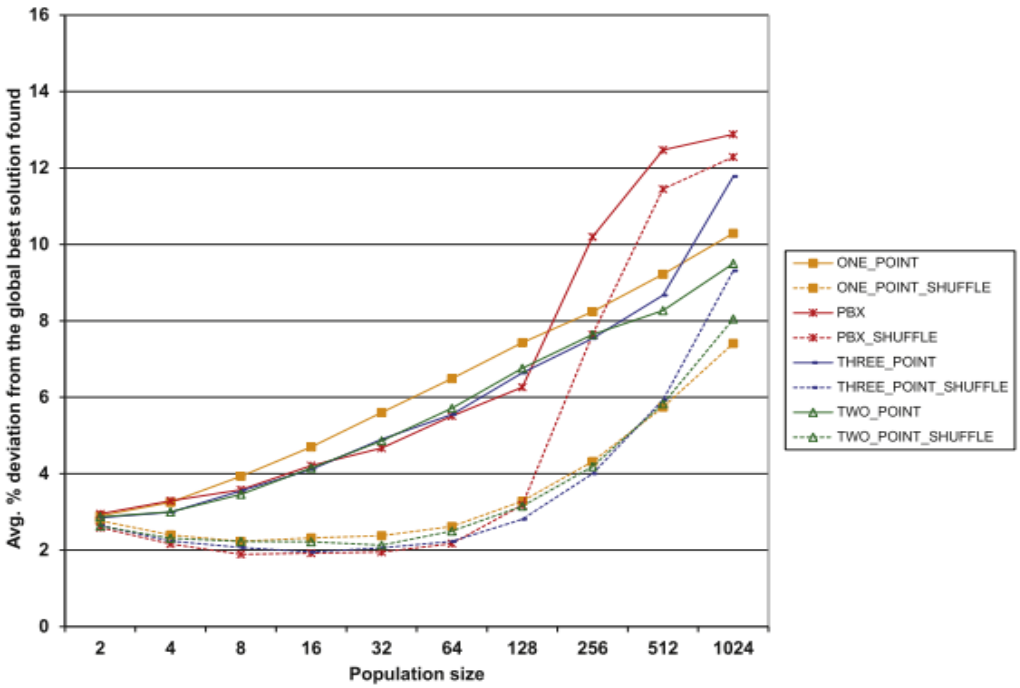


Рис. 2.5. - Вплив початкового розміру популяції на якість рішень (результати для задачі з кількістю оцінок фітнесу = 80000)

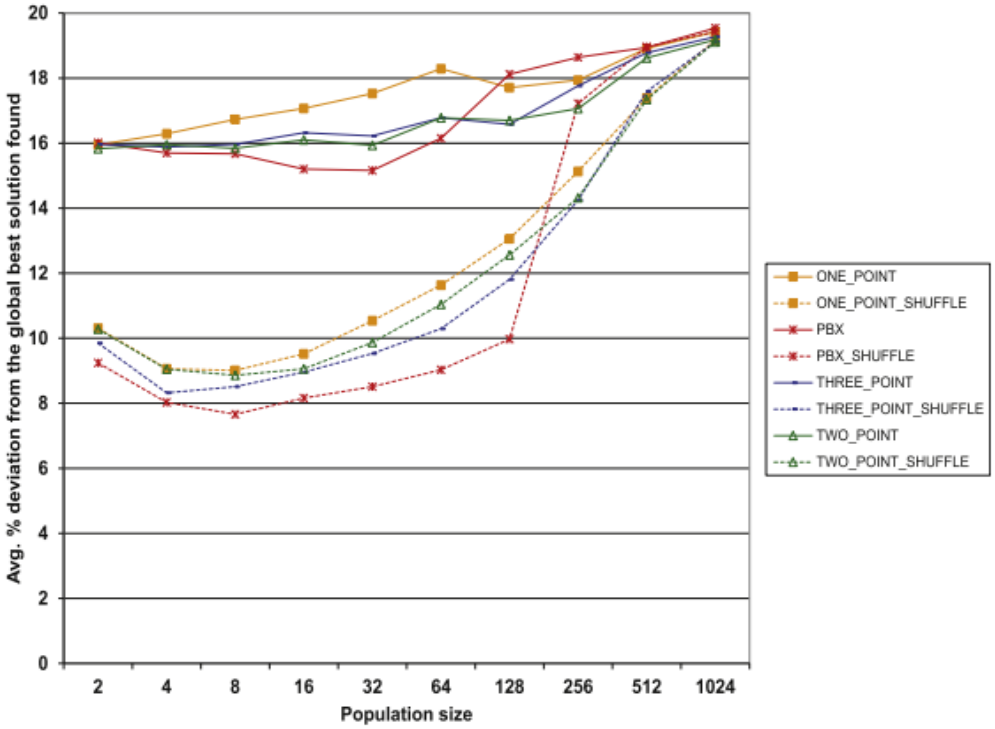


Рис. 2.6. - Вплив початкового розміру сукупності на якість розв’язків (результати задачі №10 з кількістю оцінок придатності = 10000).

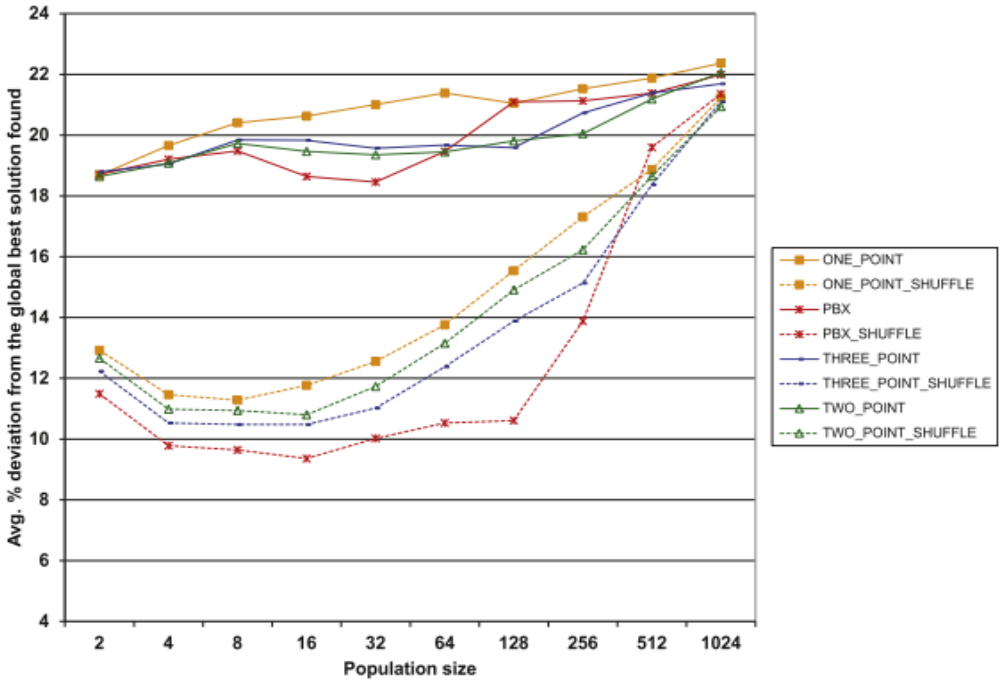


Рис. 2.7. - Вплив початкового розміру сукупності на якість розв'язків (результати задачі №11 з кількістю оцінок придатності = 10000)

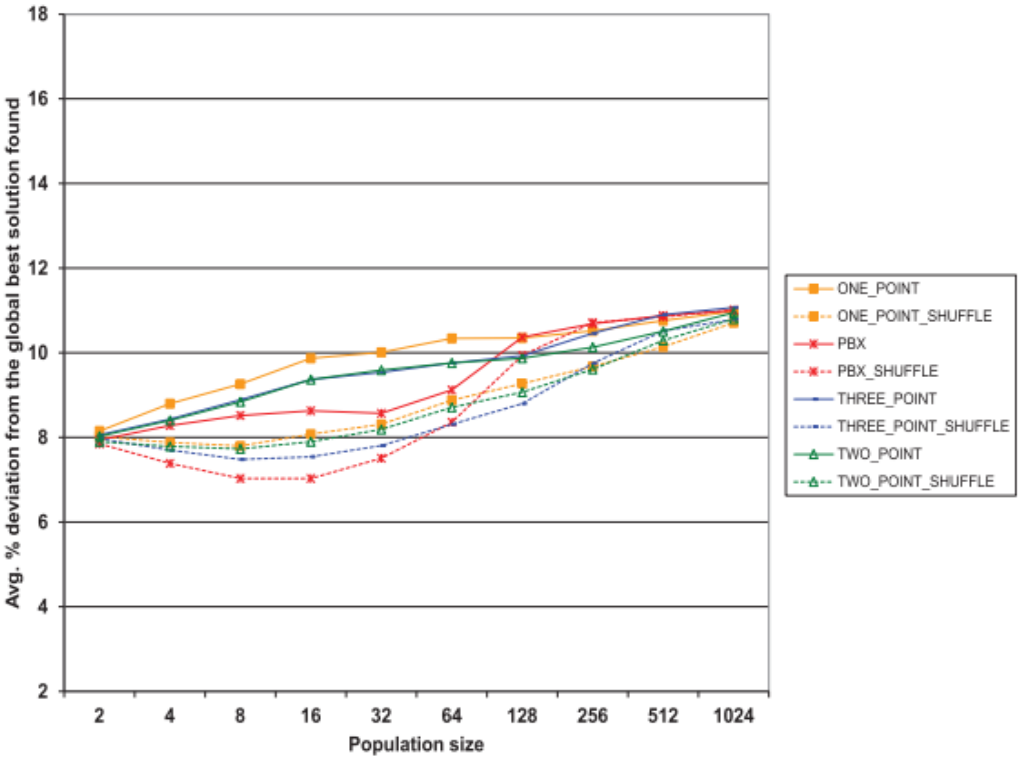


Рис. 2.8. - Вплив початкового розміру сукупності на якість рішень (результати задачі №12 з кількістю оцінок придатності = 10000)

Рисунки 2.3 – 2.8 показують, що прості оператори кросовера (без будь-якого перемішаного списку) продовжують деградувати, коли ми збільшуємо початковий розмір сукупності для заданої кількості оцінок придатності. Наприклад, на рисунку 2.3 середні відсоткові відхилення для оператора перехресного переходу на основі позиції (УАТС) становлять 7,29, 8,35 і 10,7, коли початкові розміри популяції становлять 2, 16 і 64 відповідно. У випадку операторів із перемішаним списком якість рішень покращується, оскільки ми збільшуємо початковий розмір сукупності до певної межі, а потім вона продовжує погіршуватися. Наприклад, у випадку оператора кросовера перемішування списку на основі позиції (PBX_SHUFFLE) якість рішень покращується, коли початковий розмір популяції змінюється від 2 до 16, а потім погіршується. Середні відсоткові відхилення становлять 7,13, 5,62 і 7,61, коли початкові розміри популяції становлять 2, 16 і 64 відповідно (рисунок 2.3).

Таку поведінку можна пояснити тим, що для фіксованої кількості оцінок придатності більший розмір популяції вносить у метод пошуку сильнішу властивість дослідження. Оператори без перемішаного списку вже страждають від відсутності достатньої експлуатації, і це посилюється зі збільшенням чисельності популяції. Більшість операторів із перемішаним списком забезпечують найкращі результати, коли початковий розмір популяції становить від 8 до 16. Їхня продуктивність починає знижуватися, коли початковий розмір популяції збільшується до 32 і вище.

Далі проаналізуємо результати, зображені на рисунках 2.3 – 2.8, щоб порівняти вплив перемішаного списку на різних операторів, особливо в цікавому діапазоні чисельності населення (тобто від 8 до 64).

Наприклад, на рисунку 2.3, коли початковий розмір сукупності дорівнює 8, середнє відсоткове відхилення для оператора простого одноточкового перехресування становить 9,19, тоді як для одноточкового кросовера зі змішаним списком воно зменшується до 6,71 (майже 27% покращення якості розчину). Аналогічно, результати для двоточкового кросовера без та з перемішаним списком становлять 8,39 та 6,15 відповідно. Інші оператори також демонструють подібну

поведінку, тобто версії з перемішаним списком забезпечують кращі результати порівняно з версії без перемішаного списку. Це також можна спостерігати на рисунках 2.4 і 2.5, де загальна кількість оцінок придатності встановлено на 40000 і 80000 відповідно.

На рисунках 2.6 і 2.7, де загальна кількість агентів більша за необхідну кількість агентів (задача №10 і задача №11). Наприклад, у задачі №10, коли початковий розмір популяції дорівнює 8, середнє відсоткове відхилення для простого одноточкового оператора кросовера становить 16,73, тоді як воно зменшується до 9,01 для одноточкового кросовера з перетасованим списком (покращення якості рішення майже на 46%).

Підсумовуючи рисунки 2.3 – 2.8, ми можемо зробити висновок, що перехресні оператори зі змішаними списками досягають кращої продуктивності, ніж відповідні оператори без перемішаних списків для всіх початкових розмірів популяції, особливо коли розмір популяції знаходиться в діапазоні від 8 до 64. Більше того, всі оператори зі змішаними списками працюють краще, ніж усі без перемішаних списків (за винятком кросовера на основі позиції зі змішаним списком, який погіршується для початкових розмірів популяції більше 128).

Для фіксованого розміру сукупності (до певної межі) оператори в кожній групі перехресних операторів (з та без перемішаних списків) можна впорядкувати за їх продуктивністю. Рисунки 2.3 – 2.8 підтверджують, що оператори з більшою кількістю точок перетину, як правило, демонструють кращу продуктивність. Порядок такий: одноточковий кросовер, двоточковий кросовер, трьохточковий кросовер, кросовер на основі позиції. Той самий порядок зберігається у відповідних операторах з перемішаними списки. Для оператора кросинговеру на основі позиції кількість точок кросинговеру становить приблизно n/2, де n — загальна кількість генів. Це число значно більше, ніж кількість точок перетину для інших операторів.

Щоб пояснити низьку продуктивність операторів з меншою кількістю точок перетину, перехресний перехід на основі позиції порівнюється з оператором одноточкового кросоверу. Якщо в першому випадку всі гени мають однакову ймовірність бути обраним, то в другому початкова послідовність генів від першого

батька з дуже високою ймовірністю передається майбутньому нащадку. Тобто успадкування генів є упередженим і залежить від їх положення в хромосомі. В результаті деякі гени можуть бути передані майбутнім поколінням, хоча це небажано. Це зміщення зменшується, коли кількість точок перетину збільшується. Позиційний кросовер повністю вільний від цього упередження.

Однак, показують рисунки 2.3 – 2.8, продуктивність операторів з більшою кількістю точок перехрестя погіршується при більш високих темпах, коли кількість населення збільшується, а кількість оцінок придатності залишається фіксованою. Наприклад, на рисунку 2.3, для початкового розміру популяції більше 64 оператор перемішаного списку на основі позиції дає погані результати порівняно з іншими операторами перемішаного списку. Одночасне збільшення чисельності популяції та кількості точок перетину призводить до більшого дослідження простору пошуку, ніж потрібно, що призводить до пізньої конвергенції.

3 ЕФЕКТИВНЕ РІШЕННЯ ПАРАЛЕЛЬНОГО ГЕНЕТИЧНОГО АЛГОРИТМУ ДЛЯ ПРОБЛЕМИ МАРШРУТИЗАЦІЇ ТРАНСПОРТНИХ ЗАСОБІВ У ХМАРНІЙ РЕАЛІЗАЦІЇ ІНТЕЛЕКТУАЛЬНИХ ТРАНСПОРТНИХ СИСТЕМ

3.1 Основні відомості про CUDA та TBB

Розглянемо ефективність розпаралелювання генетичного алгоритму (GA) задачі комівояжера (TSP). Обговорюваний метод може значно прискорити вирішення еквівалентного TSP багатьох складних проблем маршрутизації транспортних засобів (VRP) у хмарній реалізації інтелектуальних транспортних систем.

Далі наводиться короткий огляд пов’язаних концепцій, включаючи архітектуру GPU на платформі CUDA та основні інгредієнти архітектури TBB.

CUDA. Графічний процесор – це інструмент, призначений для відображення графічних зображень на робочих станціях, ігрових консолях або персональних комп’ютерах. Завдяки високій обробній потужності в неграфічних додатках була розроблена нова галузь інформатики під назвою GPGPU (General-Purpose Computing on Graphics Processing Unit). Відеокарти Nvidia включають одну зі структур Femi, Tesla і Kepler. Кожна структура має специфічні характеристики, такі як максимальна кількість потоків у блоках, розмір спільної пам'яті тощо. Однак, щоб полегшити програмування, був розроблений інтерфейс програмування. Програмування GPU, звичайно, є складним завданням. Тому компанія Nvidia у 2006 році представила програмну платформу під назвою CUDA для реалізації неграфічних обчислень на графічному процесорі [75].

CUDA надає розробникам можливості використання апаратних можливостей відеокарт Nvidia в неграфічних програмах і підвищення швидкості реалізації складних алгоритмів за допомогою можливостей GPU. CUDA підтримує основні фактори, які беруть участь у обчисленні з двох різних точок зору: хост і пристрій. Хост запускає основну програму, а пристрій допомагає в обробці. Типовий

сценарій полягає в тому, що центральний процесор розглядається як хост, а графічний процесор розглядається як допоміжний для процесора.

Будь-яка програма, написана на CUDA, може складатися з кількох ядер. Кожне ядро виконується сіткою, яка складається з кількох блоків. Кожен блок складається з кількох потоків. Ці потоки відповідають за реалізацію програми. На рисунку 3.1 зображено поняття нитки, блоку та сітки [76].

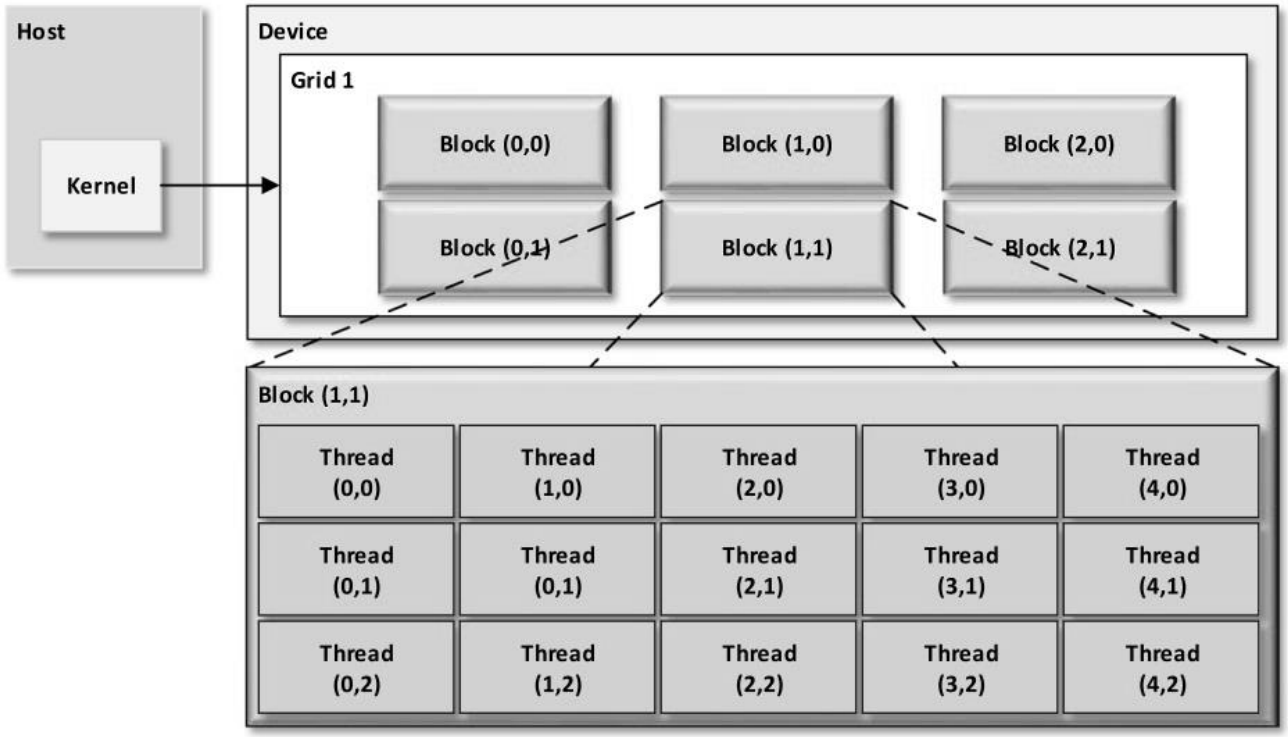


Рис. 3.1. - Сітка блоків ниток

TBB вперше був представлений Intel у 2006 році як бібліотека паралельного програмування. Рисунок 3.2 ілюструє загальну структуру TBB і як вона працює для створення потоків і балансування їх робочого навантаження. Ця бібліотека дозволяє інтерпретувати розпаралелювання як явно, так і неявно. У явному режимі спаунінгу надає програмісту повний контроль над виконанням кожного завдання. Неявного стану можна досягти за допомогою таких шаблонів, як parallel_for або parallel_reduce, які прискорюють написання коду. Завдання, створені явно або неявно, додаються до черги завдань потоку в абстрактному просторі, який називається Арена ниток (Threads Arena). Ці завдання виконуються головним

потоком або іншими працівниками за допомогою механізму, який називається крадіжкою. Далі ми розглянемо це поняття [12].

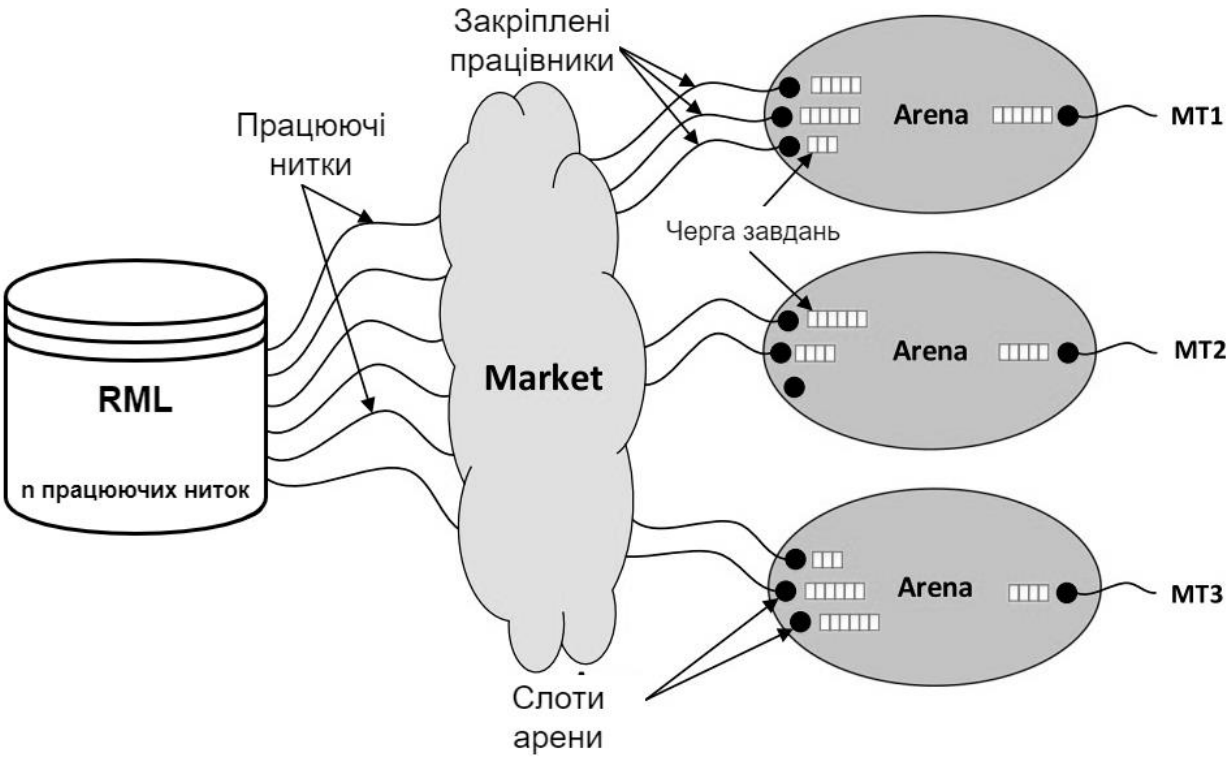


Рис. 3.2. - Компоненти планування завдань TBB

Головний потік TBB, представлений MT на цьому рисунку, є програмним потоком, який створює екземпляр об'єкта TBB::task_scheduler_init. Усі потоки, створені MT і використовуються для виконання завдання MT, називаються робочими потоками. Рівень керування ресурсами (RML) є хостом пулу робочих потоків. Роль Market полягає в розподілі робочих навантажень провідних потоків, а також у призначенні працівників на арени провідних потоків для виконання розподілених робочих навантажень. Кількість доступних робочих потоків завжди на один менше, ніж максимальний аргумент tbb::task_scheduler_ init і загальна кількість логічних ядер у системі

Наступна структура - це арена кожного Ниткового Магазину (MT), яка інкапсулює всі доступні завдання та ресурси (робочі потоки) для виконання головного потоку. Кожній арені призначається кількість слотів, які представляють

кількість робочих потоків, необхідних для виконання паралельних завдань MT. Якщо загальна кількість слотів, необхідних для всіх головних потоків, перевищує кількість працівників у пулі RML, розподіл слотів на арені MT буде адаптовано до потреб. Коли завдання головного потоку закінчується, потоки, створені під час створення кожної арени, або знищуються, або призначаються RML активним аренам.

Кожен робочий потік, коли виконується на арені, виконує процедуру планування під назвою wait_for_all(), яка складається з трьох вкладених циклів. Внутрішній цикл виконує поточне завдання, викликаючи метод execute(). Після завершення цього завдання, якщо подальше завдання не викликається, програма виходить з внутрішнього циклу. У середньому циклі метод get_task() намагається вивести локальні завдання з черги в порядку «останнього прийшов першим вийшов» (LIFO). У разі успіху внутрішній цикл буде викликано знову. В іншому випадку потік виходить із середнього циклу, а зовнішній цикл активує механізм крадіжки, викликаючи метод receive_or_steal_task(). Цей метод шукає всі завдання на цьому рівні. Пошук включає в себе надсилання завдань через механізм залежності завдання від потоку, перезавантаження завдань без пріоритету завантаження або перезавантаження завдань, залишених іншими працівниками. Якщо пошук не повертає завдання для виконання, цей метод викрадає потік жертви, який випадково вибирається в поточному місці. Якщо помилки крадіжки робочого потоку перевищують певний поріг (значення за замовчуванням 100), а арена MT порожня, невдаха робочий звільняється і повертається до пулу RML.

3.2 Пропонований підхід у вигляді паралельного GA для TSP

Загальний метод вирішення TSP за допомогою генетичного алгоритму представлений у блок-схемі на рисунку 3.3. Він має багато застосувань у різних областях інженерії, таких як хронометраж, маршрутизація, тощо [13].

Популяція: кожна хромосома складається з фіксованої кількості генів. У цьому випадку кожен ген є містом і кожну перестановку міст можна розглядати як хромосому.

Функція придатності початкової популяції: для кожної хромосоми функція дає ціле невід'ємне число, яке вказує на компетентність та індивідуальну здатність кожної хромосоми. Для обчислення пристосованості кожної хромосоми в TSP розглянемо матрицю координат міст. Відносно матриці координат відстань між містами хромосоми отримуємо з наступного рівняння [34]:

$$f(x) = \left(\sum_{i=2}^{n} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}\right) + \sqrt{(x_1 - x_n)^2 + (y_1 - y_n)^2} \qquad (3.1)$$

Кросовер: цей оператор відповідає за процес спарювання (обмін інформацією між парними хромосомами), а також швидкість зближення генетичного алгоритму. Зазвичай він діє з високою ймовірністю, тобто від 0,6 до 0,9. Це значення називається швидкістю кросовера і позначається Pc. У цьому випадку враховуються батьківський і випадкове положення між генами батьків. Потім всі гени, які були праворуч від позиції батька хромосома буде переміщена, щоб отримати нову хромосому.

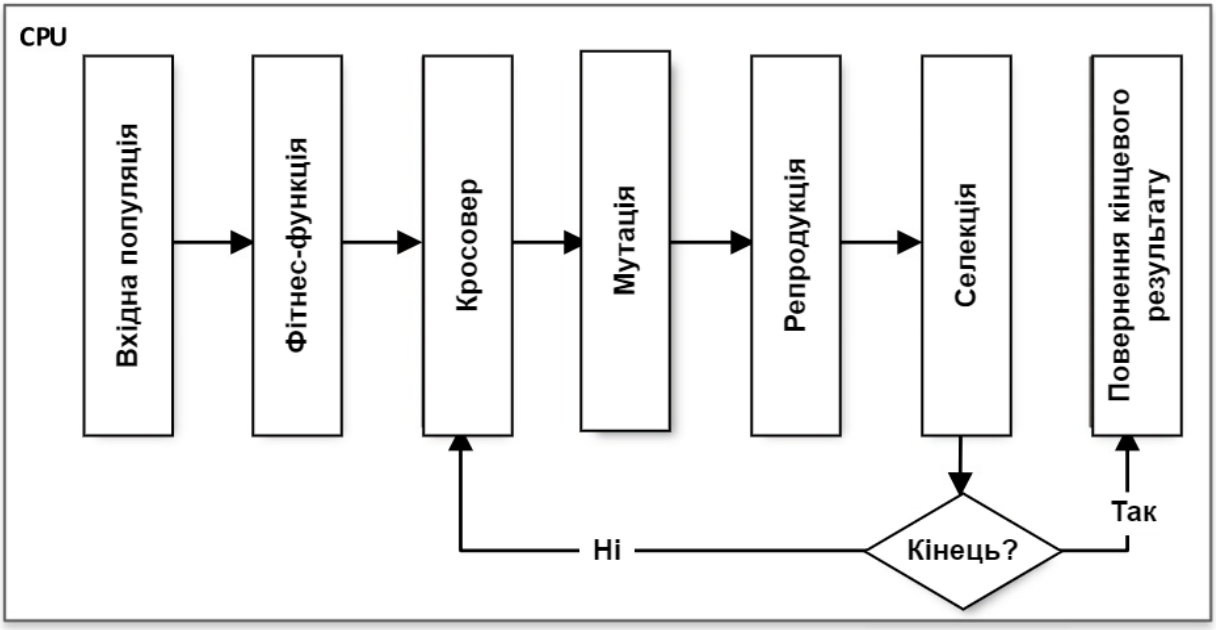


Рис. 3.3. - Послідовні підходи до запуску генетичного алгоритму

Мутація: це інший оператор, який відповідає за нову інформацію. Цей оператор з низькою ймовірністю 0,01 випадково змінює один із отриманих генів. Загальна ймовірність мутації в хромосомі називається швидкістю мутації, яка позначається Pm I. У цій статті два гени хромосоми випадково зміщені.

Розрахунок пристосованості нового покоління: функція пристосованості популяції розраховується за допомогою операторів схрещування та мутації так само, як і початкової сукупності.

Відбір: існують різні методи відбору найкращої хромосоми та її передачі наступному поколінню. При паралельному виконанні краще використовувати турнірний відбір. У цьому методі з популяції випадковим чином вибираються дві хромосоми. Тоді як r вибирається випадкове число від нуля до одиниці. якщо $r < k$ (k є параметром для випадку 0,8), то в якості батьків вибирається пристосована людина або менш пристосована людина. Ці двоє потім повертаються до початкової популяції і знову беруть участь у процесі відбору. Нарешті, вибрані хромосоми розпізнаються як наступне покоління і відправляються на наступний раунд реалізації алгоритму [10]. Статистичний аналіз методів відбору в генетичних алгоритмах, як і інших операторів, вже вивчається в багатьох дослідженнях. Ця проблема має значний вплив на розробку ефективних GA [13].

Загальна структура послідовного та паралельного генетичного алгоритму проілюстрована на рисунку 3.4. У всіх запропонованих паралельних ядрах початкова популяція однакова. Метою запропонованих ядер для паралельного GA є порівняння потужності розпаралелювання CUDA та TBB. У послідовному методі всі операції генетичного алгоритму виконуються послідовно. Однак у паралельному методі важливі операції GA реалізуються паралельно, що скорочує час виконання генетичного алгоритму. Такими операторами є пристосованість, кросовер, мутація та відбір. На рисунку 4 показана блок-схема пропонованого способу розпаралелювання GA.

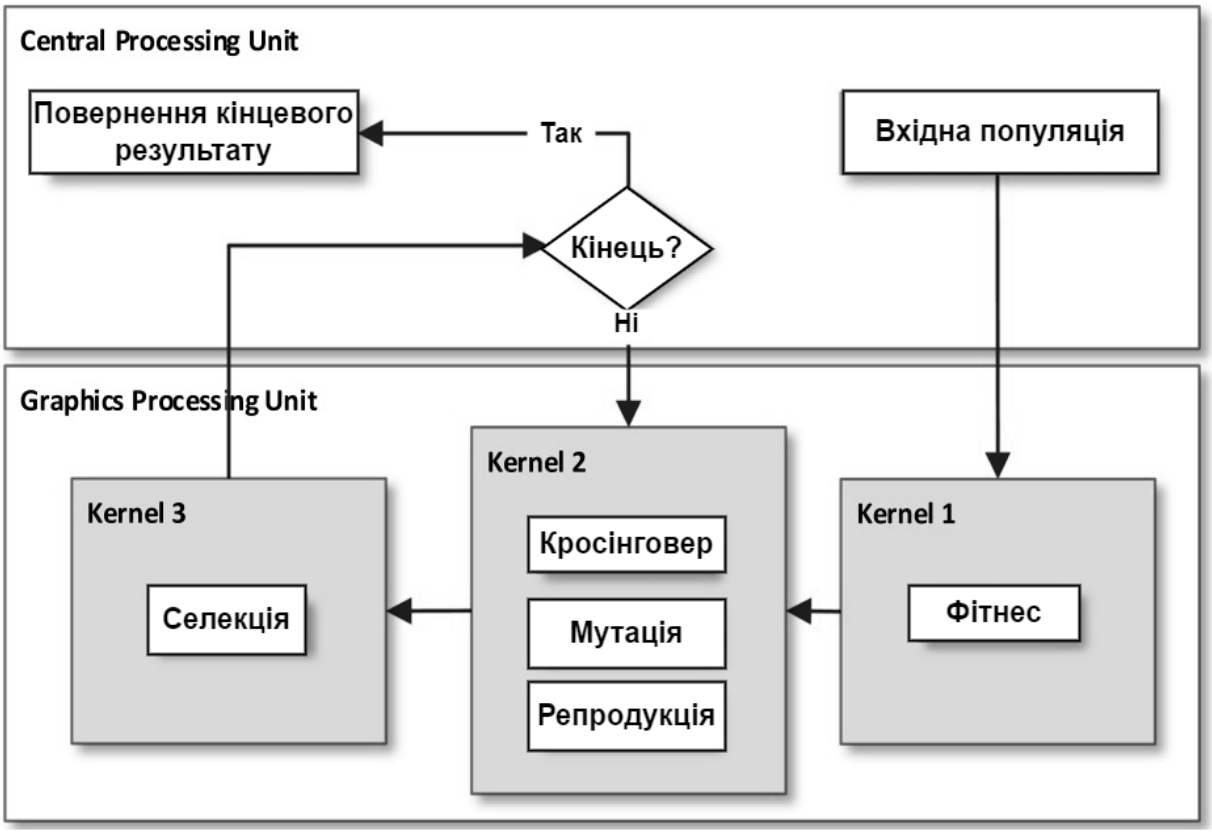


Рис. 3.4. - Паралельні підходи для запуску генетичного алгоритму.

Відповідно до структури графічних процесорів, можна синхронізувати лише потоки одного блоку. Синхронізація дуже важлива в генетичному алгоритмі, де деякі оператори повинні виконуватися послідовно. Найважливішою перевагою нашого методу перед останніми методами, такими як [18], є пропозиція трюку для вирішення проблеми синхронізації потоків і використання потоків більш ніж одного блоку. Для цього ми організуємо три різних ядра, кожне з яких відповідає окремій функції GA. Ці ядра будуть репліковані на різних блоках CUDA одночасно. При перемиканні між ядрами всі потоки збігаються. Основна проблема перемикання між ядрами - мінімізувати пов'язані з цим витрати. Враховуючи це, результати обчислень кожного ядра зберігаються в глобальній пам’яті графічного процесора, і на кожному кроці перемикання між хостом і пристроєм не відбуватиметься обмін даними. Тому час, необхідний для перемикання між ядрами, дуже малий. Нижче описано роботу кожного ядра.

У запропонованому паралелізмі спочатку паралельно розраховується відповідність первинної популяції першим ядром. У цьому ядрі кожен потік відповідає за обчислення пристосованість хромосоми. Потім, щоб створити нове покоління, оператори перехресних, мутаційних і відповідних функцій реалізуються паралельно другим ядром. У цьому ядрі кожен потік відповідає за створення нового дочірнього за допомогою різних операторів GA. Наступний оператор, який є виділенням, також виконується паралельно третім ядром і вибирає найкращі хромосоми поточного покоління для передачі наступному поколінню. У цьому ядрі кожен потік відповідає за вибір хромосоми. Наприкінці перевіряється стан кінця генерації. Якщо умова завершення генерації встановлена, ця операція припиняється, і в результаті популяцій повертається найкраща відповідь. В іншому випадку друге та третє ядра, які створюють нове покоління, а також виділення будуть виконуватися до кінця попередньо визначеної кількості поколінь. У наступному розділі ми розглянемо продуктивність паралельних методів на TSP.

3. 3 Інтеграція до транспортної хмари

Транспортні засоби та датчики в локальній місцевості виробляють вміст транспортного засобу. Цей вміст обробляється та використовується сусідніми транспортними засобами. Транспортні засоби можуть отримати доступ до цих ресурсів за допомогою транспортних хмарних обчислень (VCC) [6]. VCC дозволяє транспортним засобам обчислювати, обробляти, зберігати та спілкуватися один з одним. Відповідні ресурси забезпечують доцільність управління рухом та безпекою доріг. Дійсно, VCC відкриває нові можливості для управління рухом за допомогою ефективної маршрутизації транспортних засобів [8].

Нашу запропоновану систему маршрутизації можна використовувати на VCC. У цьому випадку системи можуть запропонувати новий напрямок розвитку систем управління рухом. Зауважте, що інтеграція VCC з іншими комерційними та придорожними хмарами може розширити можливості VCC. Різниця орендних типів хмар, які генеруються транспортними засобами, і відмінності між ними

залежать від базової мережевої інфраструктури та характеру злиття між хмарами. Тут ми зосереджуємось лише на загальній структурі платформи VCC, яка найкраще підходить для нашого методу, і пропускаємо проблеми та проблеми, які вона приносить.

На рисунку 3.5 показана запропонована структура VCC для реалізації запропонованого методу маршрутизації в мережах інтелектуальних транспортних засобів.

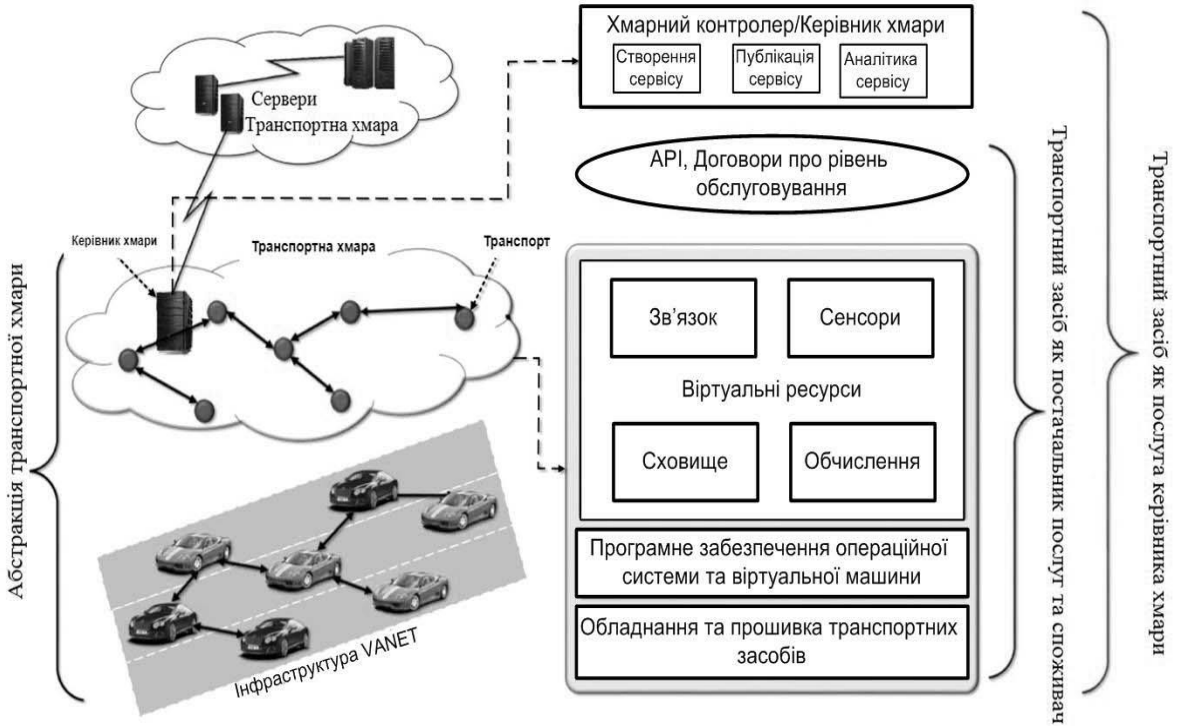


Рис. 3.5. - Автомобільні хмарні обчислення для використання запропонованого методу

У нижній частині діаграми представлена інфраструктура автономної мережі (VANET). Хмара показує з'єднання транспортних засобів-учасників. Один транспортний засіб виступає як лідер хмари або контролер хмари. Хмарний контролер також підключається до Інтернету для отримання додаткових послуг. Через Інтернет керівник спілкується з посадовими особами управління дорожнього руху, які запускають запропонований алгоритм на основі отриманої інформації про місцезнаходження транспортного засобу, надаючи необхідну інформацію про

маршрут (інформацію або інструкції, пов’язані з подією) про оптимальні маршрути для всієї дорожньої мережі транспортних засобів над містом.

Кожний транспортний засіб має набір ресурсів і знає основного постачальника послуг, який може надати інформацію про маршрути. Абстракція транспортного засобу, як показано на рисунку 3.5, наведена в правій частині діаграми.

Кожний транспортний засіб має операційну систему та апаратне забезпечення на первинному рівні, яким керує програмне забезпечення, що можна запускати на віртуальних машинах. Ця система збирає дані про місцезнаходження транспортного засобу та здійснює зв'язок. Ресурси автомобіля фактично стають доступними для хмари згідно з угодами про рівень обслуговування з лідером хмари. Керівник хмари ініціює, публікує та аналізує послуги для транспортних засобів, постійно оцінюючи та відстежуючи віртуальні ресурси транспортних засобів, що додаються.

3.4 Впровадження та оцінка ефективності

У цьому розділі спочатку описані характеристики використовуваних ЦП і графічних процесорів, а також значення різних параметрів GA. Далі на основі різних показників досліджується ефективність запропонованого методу розпаралелювання.

Експерименти проводилися на персональному комп’ютері Intel (R) Core i5-7600 3,50 ГГц з 8 ГБ оперативної пам’яті, оснащеному відеокартою NVIDIA GeForce GTX 1060. Цей графічний процесор має 1280 ядер, а його базова частота і частота прискорення становлять 1506 МГц і 1708 МГц відповідно. Фреймворки, що використовуються в цій реалізації, засновані на C++ CUDA 8.0 (V8.0.61) для багатоядерних GPU і TBB версії 2018 для багатоядерних процесорів. Кількість потоків на платформі CUDA було встановлено рівним числу хромосом, тоді як кількість потоків на багатоядерних ЦП було встановлено рівним кількості ядер. Тому в паралельному коді на основі TBB кількість ядер вражає. Щоб дослідити цей

ефект, запускається паралельний код на основі TBB на двохядерних і чотирьохядерних процесорах.

Імовірність оператора кросовера дорівнює 0,8, а ймовірність мутації – 0,02. Імовірність вибору оператора за умови, що оператор може вибрати вибірку з меншою придатністю, дорівнює 0,8. Оператор схрещування є однобатьковим і одноточковим, а оператор мутації — типу переміщення.

Отримані стандартні дані BCL380, RBU737, PBD984 для реалізації TSP були використані дані VLSI. Набори даних складалися з 380, 737 та 984 географічних регіонів міста [38]. TSP було вирішено з 1024 до 16 384 популяціями шляхом застосування 100, 200 до 3000 поколінь для кожної. Кількість нащадків, отриманих у кросовері, становила від 10% до 50% від розміру початкової популяції. Далі розглядаються результати, отримані в результаті експериментів, які повторювалися 10 разів.

Щоб оцінити запропоновані методи та порівняти їх ефективність, розглянуто вплив таких параметрів, як розмір популяції, кількість поколінь, кількість кросинговер-мутацій та розмір хромосом на ефективність кожного методу.

Критеріями оцінки є час впровадження ГА до запропонованих методів та прискорення паралельних версій послідовного методу. Спочатку досліджується вартість перемикання між запропонованими ядрами через платформу CUDA в різних сценаріях. У таблиці 3.1 наведено час, необхідний для вирішення ТСП з 380 містами з використанням запропонованого набору третинних ядер з різною чисельністю населення та різною кількістю поколінь. Для кожного випадку повідомляється час, необхідний для перемикання між ядрами, а також час їх виконання. Таблиця 3.2 показує час перемикання та виконання ядер за 100 поколінь з різною кількістю міст і різним чисельністю населення. У таблицях 3.1 і 3.2 кількість нащадків становить 40% і 30% популяції відповідно. Як пояснюється далі, оскільки немає передачі даних між ядрами і генераціями (з GPU на CPU і навпаки), збільшення кількості поколінь не впливає на час перемикання. Але зі збільшенням кількості міст і чисельності населення час перемикання збільшується за рахунок

перенесення початкового населення з CPU на GPU. Однак час перемикання є незначним у порівнянні з часом роботи ядра. Загальний час обчислень для будь-якого ядра CUDA насправді є сумою часу перемикання та часу обчислення ядра.

Таблиця 3.1 - Час перемикання та обчислення ядра (мс) у містах CUDA-380 (BCL380)

Населення		Генерація							
		100	200	400	800	1000	2000	2500	3000
1024	Switch	0.73822	0.75217	0.76018	0.75258	0.74694	0.75838	0.75547	0.75114
	Kernel	2183.76	4352.68	8687.09	17,355.7	21,697.6	43,378.4	54,213.4	65,068.2
2048	Switch	1.28195	1.47194	1.35182	1.26267	1.2215	1.37934	1.28483	1.28211
	Kernel	2268.34	4520.98	9024.91	18,036.1	22,541.3	45,064	56,323.4	67,587.5
3072	Switch	1.80883	1.80882	1.94954	1.81701	1.83845	2.31515	1.8183	1.80926
	Kernel	2359.75	4700.82	9390.72	18,762.1	23,439.8	46,859.3	58,596.2	70,285.9
4096	Switch	2.33739	2.33973	2.33702	2.52005	2.35171	2.3325	2.3419	2.28941
	Kernel	2362.82	4705.47	9399.08	18,779.1	23,461.8	46,914.9	58,661.3	70,357.7
5120	Switch	2.90762	2.92509	2.86219	2.91802	2.91189	2.97024	3.03762	2.88035
	Kernel	2384.06	4750.92	9484.7	18,944.5	23,687.8	47,371.8	59,201.4	71,052.9
6144	Switch	3.73773	3.4692	3.58517	3.48848	3.40894	3.45525	3.5112	3.60488
	Kernel	2399.68	4779.72	9544.65	19,071.8	23,829.4	47,643.4	59,560	71,438.4
8192	Switch	4.68266	4.48123	4.52781	4.87566	4.90762	4.45749	5.53869	4.46005
	Kernel	2443.86	4876.11	9721.46	19,423.6	24,309.7	48,604.7	60,656.3	72,888.9
10,240	Switch	5.55746	5.61781	5.79314	5.61149	5.62642	5.53069	5.70728	5.61958
	Kernel	2239.25	4434.78	8829.64	17,603.5	21,985.7	43,953.9	54,938.4	66,007.3
16,384	Switch	11.3822	9.32661	8.84053	8.85627	9.85898	9.05573	9.3819	8.99462
	Kernel	2434.82	4814.55	9593.75	19,134	23,902.9	48,031.8	60,033	72,071.8

Таблиця 3.2 - Час перемикання та обчислення ядра (мс) у поколіннях CUDA-100

Міста	Населення							
	Switch	Kernel	Switch	Kernel	Switch	Kernel	Switch	Kernel
	1024		**2048**		**4096**		**6144**	
380 (BCL380)	0.68951	2175.47	1.1975	2211.08	2.20462	2359.25	3.24037	2366.61
737 (RBU737)	1.15285	4060.45	2.1389	4114.11	3.83376	4367.86	5.60917	4377.8
984(PBD984)	1.57603	6200.72	3.11869	6301.57	6.60059	6714.43	8.52061	6727.66
	7168		**8192**		**9126**		**10,240**	
380 (BCL380)	3.67074	2138.17	4.96326	2400.57	4.65838	2179.79	5.00744	2198.41
737 (RBU737)	6.61571	4410.09	7.89123	4447.35	8.11848	4503.38	9.41981	4528.84
984(PBD984)	10.4104	6786.87	11.0019	6829.77	14.361	6917.51	13.9765	6953.94
	16,384		**20,480**		**32,768**		**65,536**	
380 (BCL380)	8.41643	2280.52	10.2355	2632.1	16.026	4905.64	32.1677	9212.24
737 (RBU737)	16.22	4694.13	194,048	4917.1	30.0229	9179.92	58.7788	17,204.9
984(PBD984)	24.4927	7204.22	28.8571	7572.09	44.0464	14,145.5	91.1372	26,544.6

На графіках на рисунку 3. 6 показано вплив збільшення розміру початкової сукупності та кількості поколінь на час виконання послідовного методу, паралельного методу на платформі CUDA та паралельних методів, утворених використанням TBB на двох /чотири ядра ЦП. У цьому експерименті розмір нової популяції, створеної в результаті операції кросовера, становить 40% від початкової сукупності. Збільшуючи кількість поколінь, час роботи TSP зріс у всіх методах.

Примітним моментом у цьому експерименті є вплив розміру популяції на час виконання паралельного коду на основі CUDA в порівнянні з паралельними кодами на основі TBB, що працюють на двоядерних і чотириядерних ЦП. Як показано на рисунку 3.6.(а, час виконання паралельного коду TSP на основі CUDA гірше, ніж паралельного коду на основі TBB на двоядерних і чотириядерних ЦП. У цьому експерименті були використані всі ресурси ЦП, тоді як у графічному процесорі було використано лише до 1024 потоків. Збільшення розміру популяцій збільшує рівень паралельності і, отже, покращується продуктивність коду GPU. Наприклад, якщо населення має 4096.

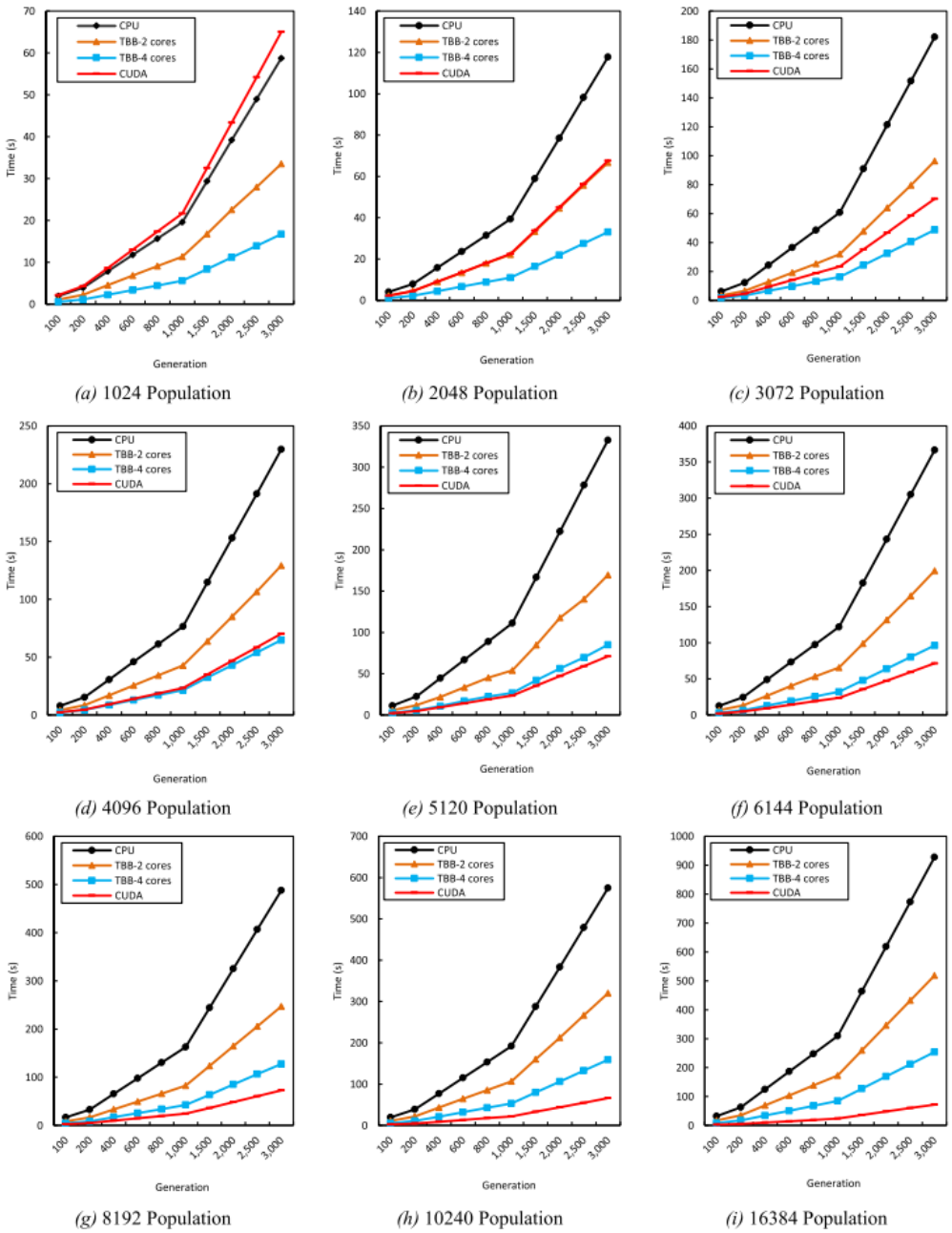


Рис.3.6. - Час роботи алгоритму для різної кількості поколінь і різної кількості населення.

Кожна підділянка позначена відповідним розміром популяції.

Паралельний TSP на основі CUDA був кращим, ніж TBB - на основі TSP. Ступінь цієї переваги досягає максимуму, коли чисельність населення досягає 16 384 особи. У цьому стані ресурси графічного процесора були добре використані.

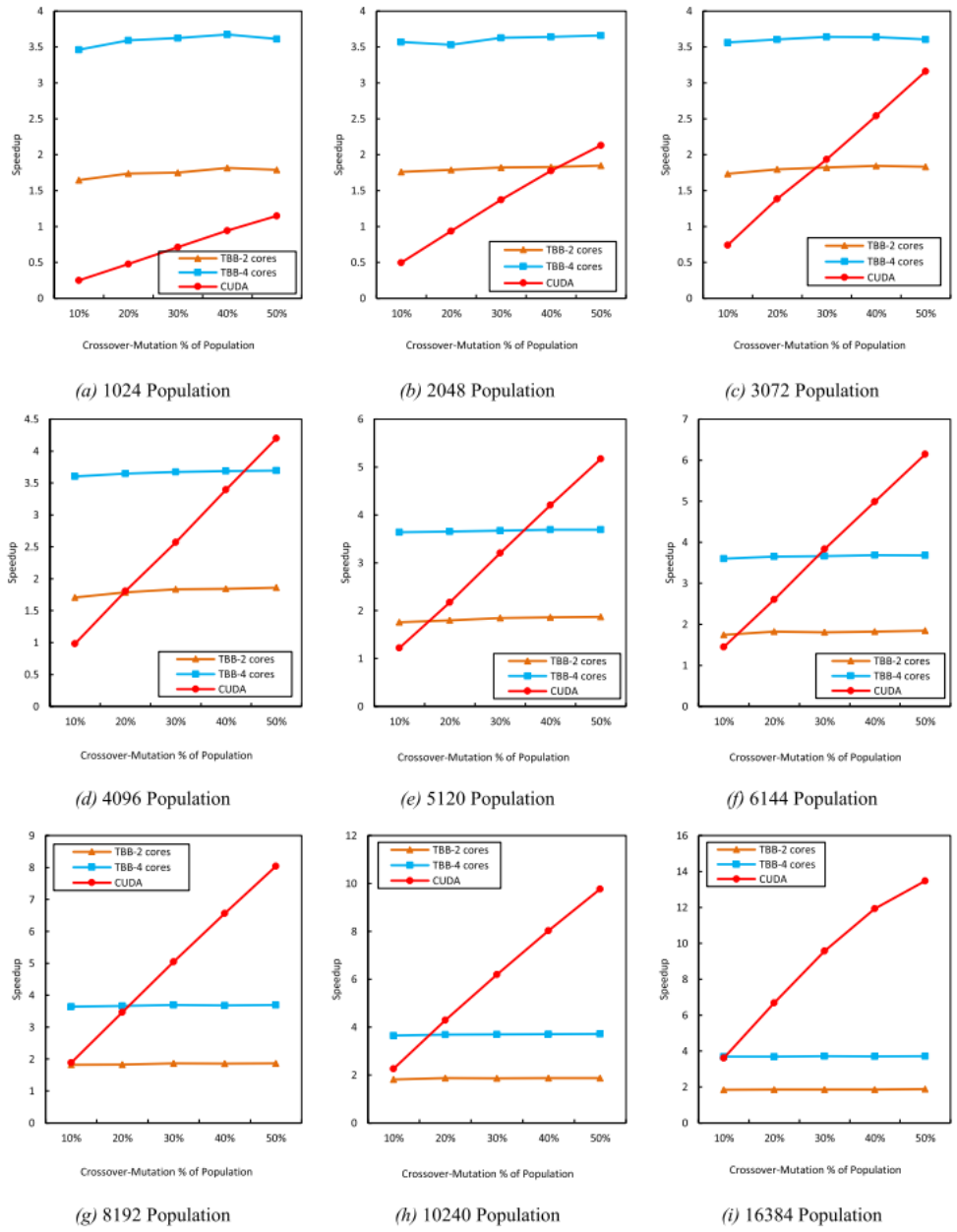


Рис. 3. 7. - Прискорення паралельних методів щодо різного співвідношення нащадків на різних популяціях у TSP з 380 містами.

Кожна підділянка позначена відповідним розміром популяції.

На рисунку 3.7 показано прискорення трьох методів паралелізації, а саме ядра TSP на основі TBB на двоядерних і чотириядерних ЦП та ядра GPU на основі CUDA щодо послідовного коду в різних розмірах популяції та нащадків, створених кросовером. Як звичайне налаштування в цьому експерименті, максимальна кількість поколінь GA встановлено на 100. У всіх випадках прискорення TBB на чотирьох ядрах більше, ніж прискорення TBB на двох ядрах. Крім того, загальний

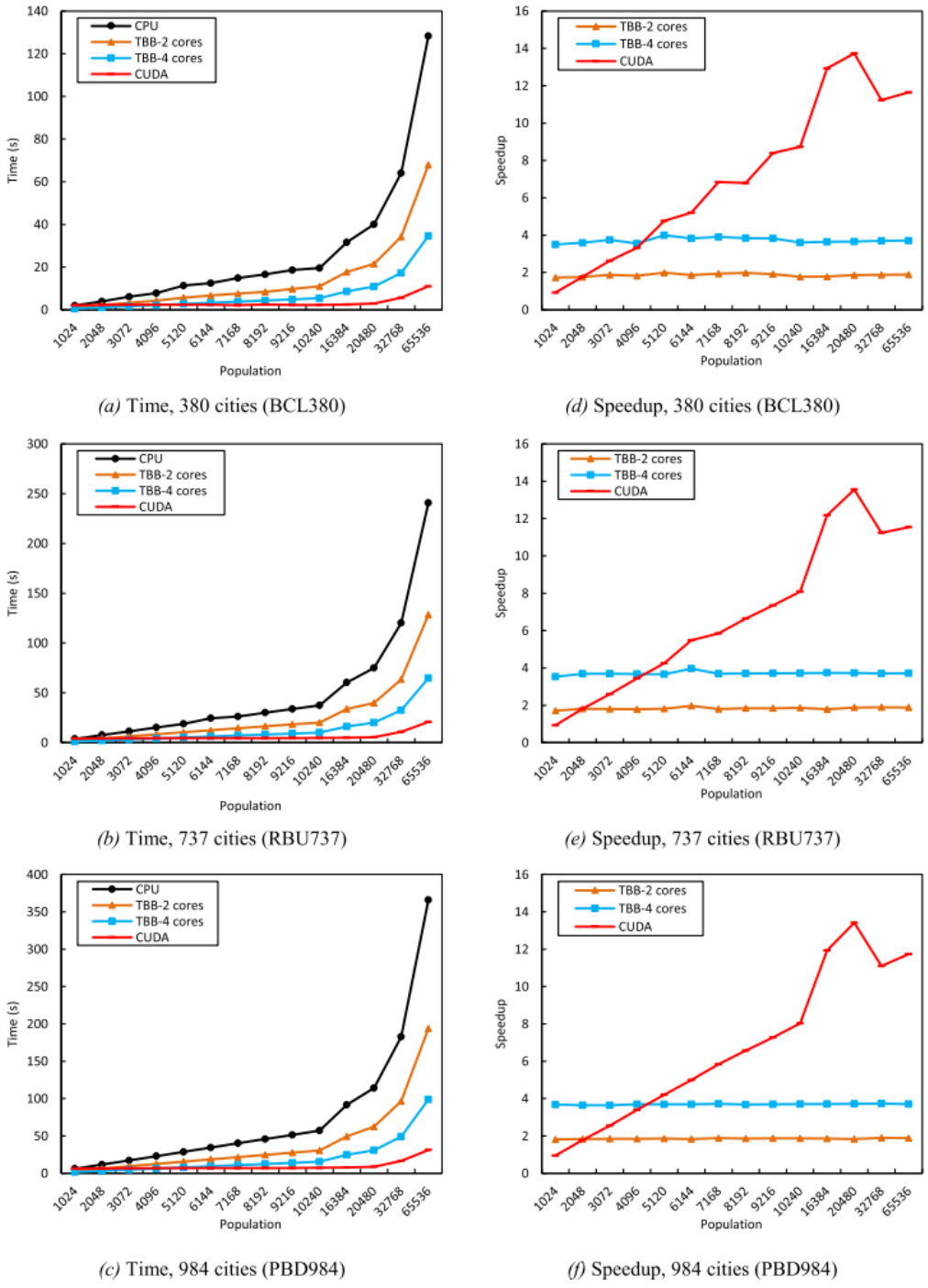
темп прискорення не змінився з урахуванням чисельності популяції та кількості потомства. Це в той час як у ядрі TSP на основі CUDA прискорення змінилося з обома цими параметрами. Причина в тому, що обидва параметри впливають на використання ресурсів графічний процесор.

Вплив кількості хромосом на час виконання паралельних кодів TSP проілюстровано на графіках на рисунку 3.8.

У відповідному експерименті кількість поколінь і кількість нащадків, отриманих в результаті кожного процесу схрещування, залишаються незмінними, в той час як розмір популяції дозволяється змінювати. Для цього експерименту було використано три різні набори даних із 380, 737 та 984 містами. Розмір потомства становить 40% від початкової популяції протягом 100 поколінь.

Як показано на рисунку 3.8, код на основі TBB на чотирьох ядрах у популяції менше або дорівнює 4096 має найменший час роботи та найвищу швидкість порівняно з іншими методами. Але коли населення перевищує 4096, CUDA має найкращу продуктивність порівняно з іншими методами. Цей стан досягається, коли CUDA обчислює рішення GA TSP з 20 480 населенням; в цьому випадку кожен потік обчислює хромосому. У міру зростання популяції кожен потік повинен досліджувати більше однієї хромосоми, що збільшує час роботи. Це показано на графіках прискорення на рисунку 3.8.

Відповідно до специфікацій графічного процесора, використаних у цьому дослідженні, максимальна кількість потоків для одночасного запуску становить 20 480 потоків. Таким чином, коли початкова сукупність перевищує 20 480, обчислювальне навантаження на потік зростає, що зменшує прискорення. Іншим примітним моментом у цьому експерименті є неефективність змін довжини хромосоми (кількості міст) у загальному часі роботи трьох паралельних методів, наданих GA. Максимальні значення прискорення, отримані в експерименті



(a) Time, 380 cities (BCL380)

(d) Speedup, 380 cities (BCL380)

(b) Time, 737 cities (RBU737)

(e) Speedup, 737 cities (RBU737)

(c) Time, 984 cities (PBD984)

(f) Speedup, 984 cities (PBD984)

Рис. 3.8. - Вплив чисельності популяції на прискорення паралельних методів.

Кожна підділянка позначена відповідним розміром популяції TSP на основі TBB для двоядерних і чотириядерних процесорів становить 1,99 с і 3,99 с відповідно, тоді як максимальне прискорення TSP на основі CUDA на 1280 ядрах становить 13,73 с.

Нарешті, згідно з експериментами, можна стверджувати, що коли генетичний алгоритм застосовується до невеликої популяції, метод на основі TBB має найкращу продуктивність. В іншому випадку, враховуючи здатність CUDA визначати максимальну необхідну кількість потоків для обчислень, використання CUDA є більш ефективним.

Для більш інтуїтивної ілюстрації ефективності запропонованого методу розпаралелювання обчислено та порівняно в таблиці 3 ефективність запропонованого розпаралелювання рішення GA TSP на платформі TBB з сучасними методами. Як показано в таблиці 3.3, ефективність запропонованого паралельного GA на чотириядерній системі з використанням платформи TBB є найвищою в порівнянні з іншими паралельними рішеннями GA TSP на багатоядерних системах. Крім того, ефективність багатоядерного розпаралелювання GA-рішення TSP за допомогою платформи TBB становить 0,9975, що значно вище, ніж у розпаралелювання на основі OpenMP, а також розпаралелювання на основі TBB, виконаного Чжу [16]. Цей результат показує, що запропонований метод розпаралелювання може більш оптимально використовувати паралельні ресурси багатоядерних ЦП і, отже, досягти більш високої ефективності.

Таблиця 3.3 - Порівняння ефективності сучасних паралелізацій GA рішень в TSP.

Рік	Метод	Посилання	Кількість ядер	Швидкість	Ефективність
2013	TBB	Джу [16]	4	2.55	0.6375
2019	OpenMP	Саксена [25]	4	2	0.5
2021	**TBB**	**Запропонований метод**	**4**	**3.99**	**0.9975**
2021	**TBB**	**Запропонований метод**	**2**	**1.99**	**0.995**

У запропонованому методі розпаралелювання паралельно реалізуються функції пристосованості, кросинговеру, мутації та відбору. Ключовим етапом дослідження стала організація синхронізованих ядер, які можуть використовувати

максимальні ресурси багатоядерних процесорів для прискорення обчислень. У зв’язку з цим три окремі ядра були розроблені для одночасного обчислення різних функцій генетичного алгоритму. Ці ядра можна реплікувати на різних блоках CUDA на GPU. Для синхронізації потоків цих блоків використовується механізм перемикання. Час, необхідний для перемикання між ядрами різних блоків, є незначним щодо часу роботи генетичного алгоритму. Таким чином, запропонований метод є високоефективним при синхронізації спільних потоків при обробці генетичних алгоритмів.

Запропоновані уточнення методу були протестовані для розпаралелювання транспортної задачі на основі генетичного алгоритму на платформах CUDA і TBB з однаковими налаштуваннями, включаючи ту саму кількість первинної популяції та поколінь, а також те саме відношення створених дітей. операторами кросинговеру та мутації на одному наборі даних.

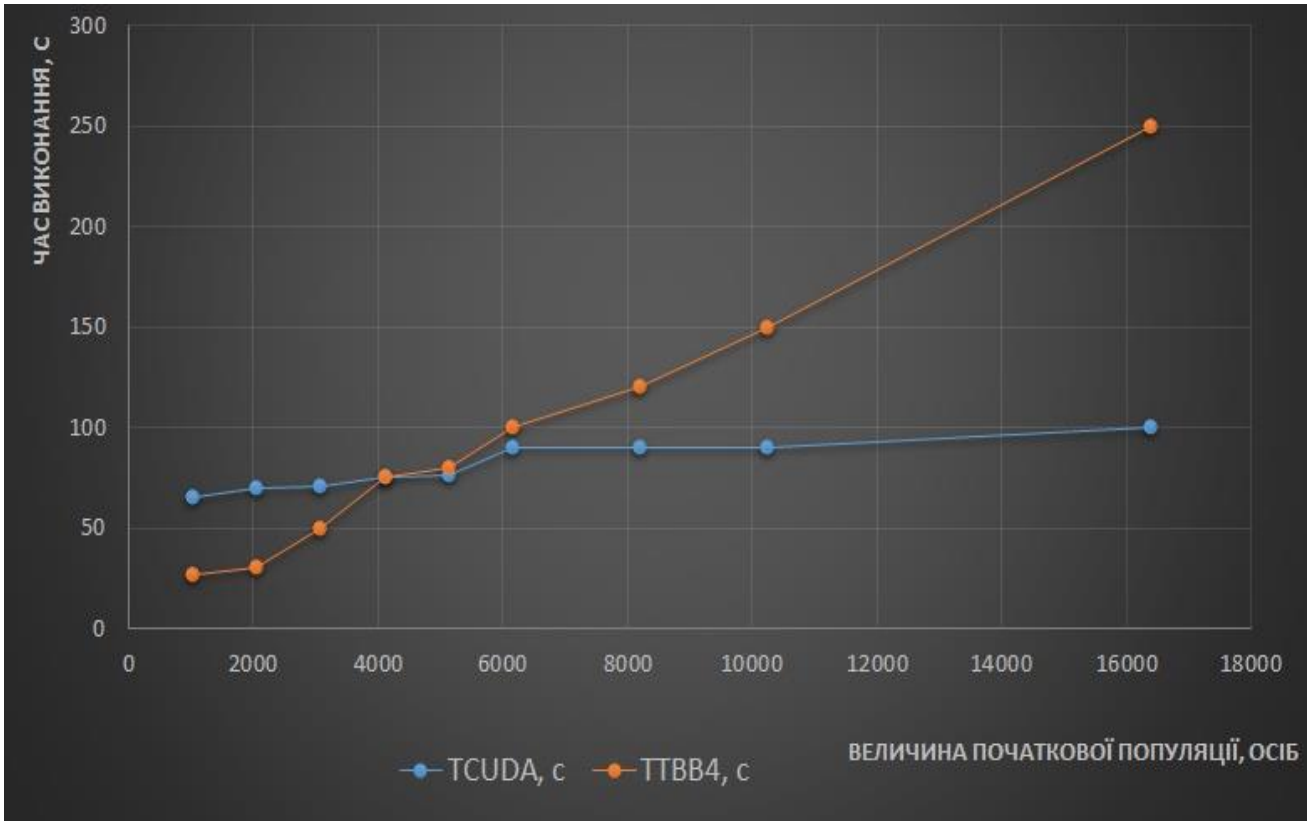


Рис. 3.9. - Порівняння часу реалізації генетичного алгоритму на платформах CUDA та TBB з 4 ядрами в залежності від початкової популяції.

Продуктивність цих двох платформ оцінювалася на основі такого важливого практичного критерію, як час роботи паралельної GA для кожної з них.

Було отримано підтвердження більш великого часу роботи CUDA для низької кількості популяцій у порівнянні з платформою TBB з 4 ядер.

Також було зроблено практичне уточнення щодо максимального часу реалізації для платформи CUDA за дуже великої кількості початкової популяції.

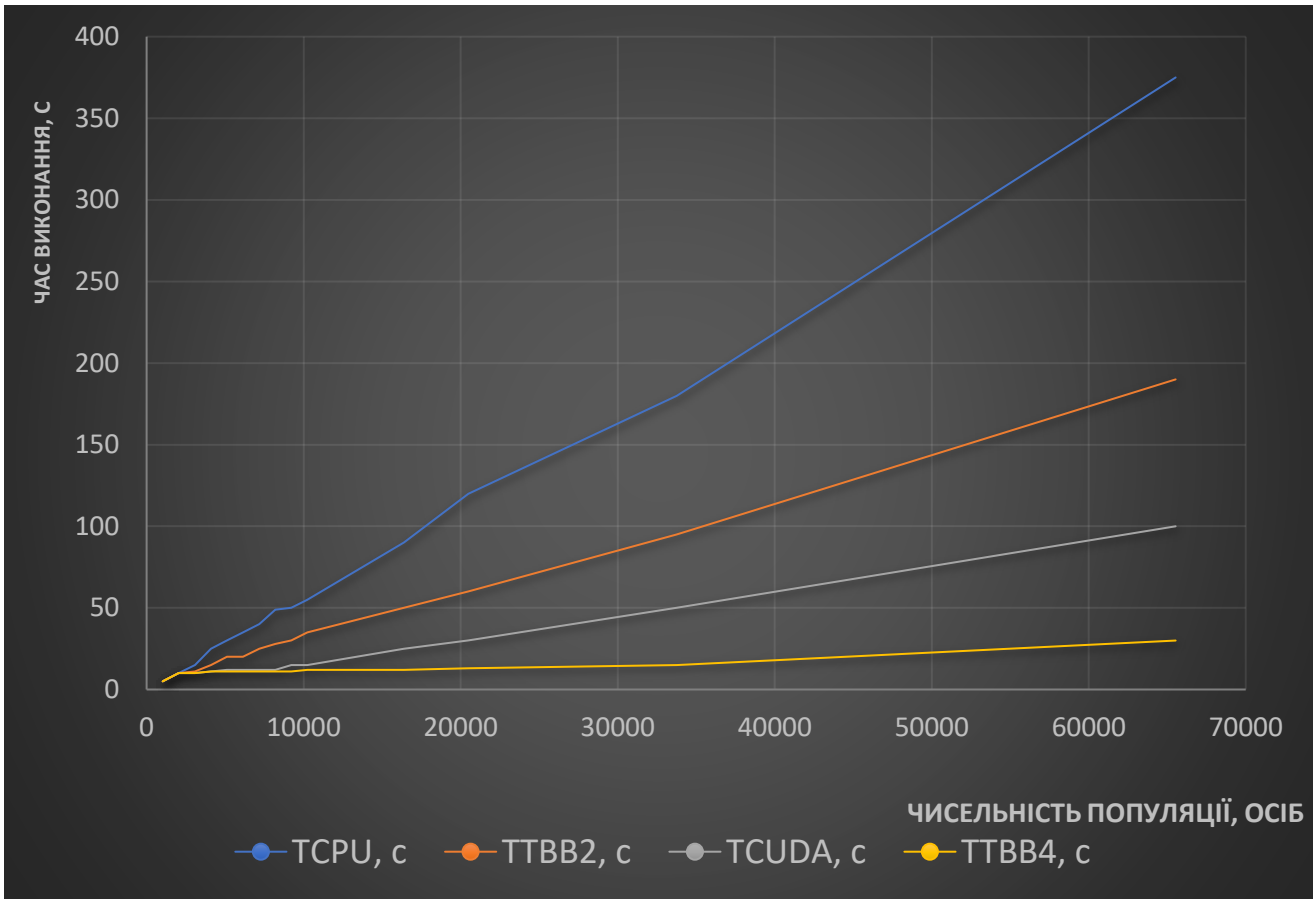


Рис. 3.10. Максимальний час реалізації генетичного алгоритму для системи з 918 міст у випадку CPU (375 с), TBB з 2 ядер (190 с), TBB з 4 ядер (100 с) та CUDA (30 с).

3.5 Опис базової програми виконання генетичного алгоритму

Розглянемо еталонну модель роботи генетичного алгоритму в програмі, зробленої на мові програмування Java та потім використаної для завантаження до спеціалізованої апаратної платформи, де стає практично можливим зробити запуск програми за розпаралелюваними алгоритмами окремо на графічному процесорі (GPU) та окремо на центральному процесорі (CPU).

Визначальним класом програми є клас World.java, де визначається графічні елементи програми.

World.java

```
public class World extends JPanel {

    private final TSPPopulation population;
    private final AtomicInteger generation;

    static final int WIDTH = 800;
    static final int HEIGHT = 600;

    private World() {
        setPreferredSize(new Dimension(WIDTH, HEIGHT));
        setBackground(Color.BLACK);
        this.population = new TSPPopulation(TSPUtils.CITIES, initialSize: 9000);
        this.generation = new AtomicInteger( initialValue: 0);
        final Timer timer = new Timer( delay: 5, (ActionEvent e) -> {
            this.population.update();
            repaint();
        });
        timer.start();
    }

    @Override
    public void paintComponent(final Graphics graphics) {
        super.paintComponent(graphics);
        final Graphics2D g = (Graphics2D) graphics;
        g.setColor(Color.CYAN);
        g.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
        g.drawString( str: "gen : " +this.generation.incrementAndGet(), x: 350, y: 15);
        g.drawString( str: "pop size : " +this.population.getPopulation().size(), x: 150, y: 15);
        g.drawString( str: "shortest path : "
                +String.format("%.2f", this.population.getAlpha().getDistance()), x: 500, y: 15);
        drawBestChromosome(g);
    }

    private void drawBestChromosome(final Graphics2D g) {
        final java.util.List<TSPGene> chromosome = this.population.getAlpha().getChromosome();
        g.setColor(Color.WHITE);
        for(int i = 0; i < chromosome.size() - 1; i++) {
            TSPGene gene = chromosome.get(i);
            TSPGene neighbor = chromosome.get(i + 1);
```

Рис. 3.9. - Клас World.java

В головному методі програми реалізується структура інтерфейсу програми за технологією swing:

```
public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
        final JFrame frame = new JFrame();
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        frame.setTitle("Genetic Algorithms");
        frame.setResizable(false);
        frame.add(new World(), BorderLayout.CENTER);
        frame.pack();
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    });
```

Структуру поточного гену визначається в класі TSPGene.java.

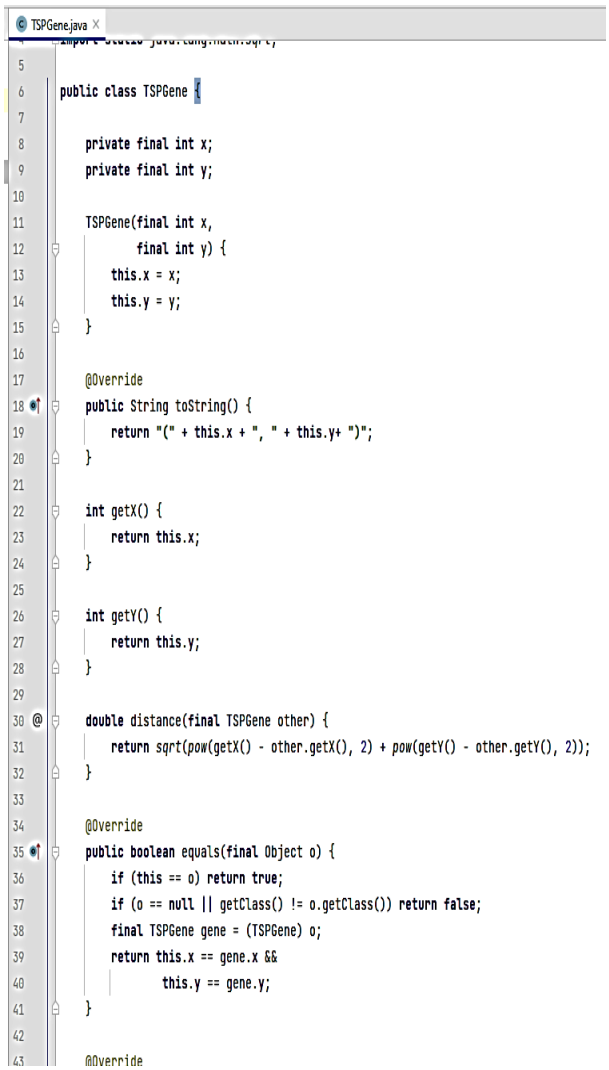


Рис. 3.10. - Клас TSPGene.java

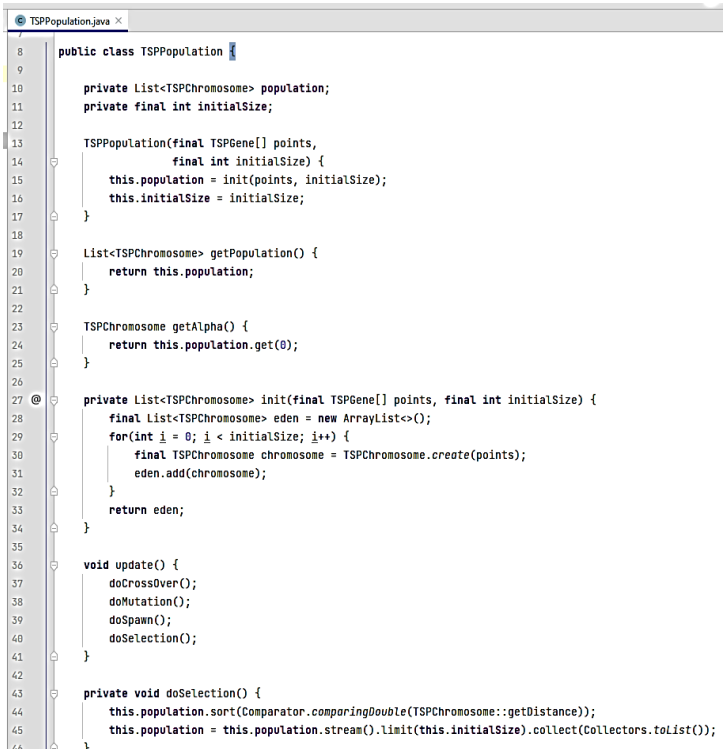
Клас TSPChromosome.java містить у собі перезавантажені методи виконання оператору кросоверу та виконання оператору мутації.

TSPChromosome.java

```
6   public class TSPChromosome {
7
8       private final List<TSPGene> chromosome;
9       private final double distance;
10
11      public double getDistance() {
12          return this.distance;
13      }
14
15      private TSPChromosome(final List<TSPGene> chromosome) {
16          this.chromosome = Collections.unmodifiableList(chromosome);
17          this.distance = calculateDistance();
18      }
19
20  @   static TSPChromosome create(final TSPGene[] points) {
21          final List<TSPGene> genes = Arrays.asList(Arrays.copyOf(points, points.length));
22          Collections.shuffle(genes);
23          return new TSPChromosome(genes);
24      }
25
26      @Override
27      public String toString() {
28          final StringBuilder builder = new StringBuilder();
29          for(final TSPGene gene : this.chromosome) {
30              builder.append(gene.toString()).append((" : "));
31          }
32          return builder.toString();
33      }
34
35      List<TSPGene> getChromosome() {
36          return this.chromosome;
37      }
38
39      double calculateDistance() {
40          double total = 0.0f;
41          for(int i = 0; i < this.chromosome.size() - 1; i++) {
42              total += this.chromosome.get(i).distance(this.chromosome.get(i+1));
43          }
44          return total;
```

Рис. 3.11. - Клас TSPChromosome.java

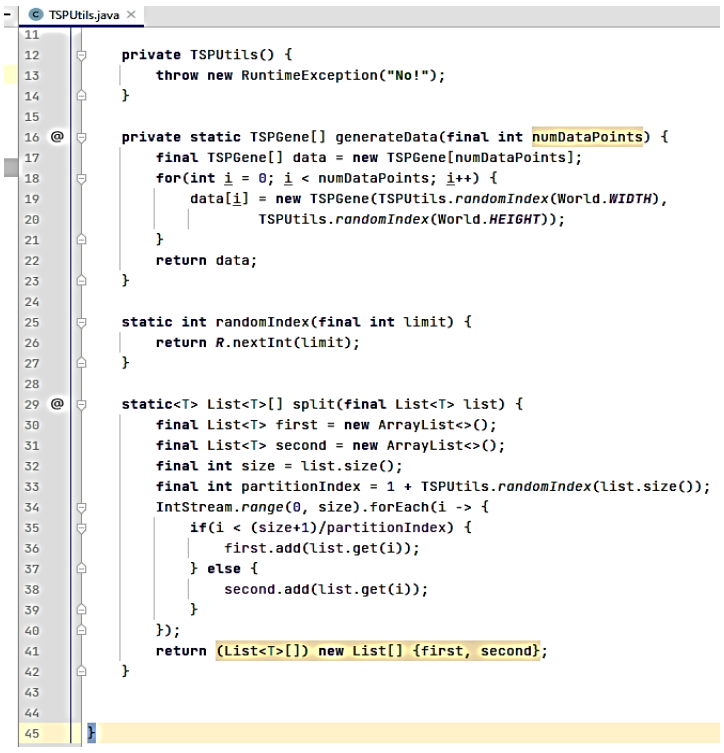
Клас TSPPopulation містить в собі методи реалізації оператору селекції, мутації та кросоверу.



```java
public class TSPPopulation {

    private List<TSPChromosome> population;
    private final int initialSize;

    TSPPopulation(final TSPGene[] points,
                  final int initialSize) {
        this.population = init(points, initialSize);
        this.initialSize = initialSize;
    }

    List<TSPChromosome> getPopulation() {
        return this.population;
    }

    TSPChromosome getAlpha() {
        return this.population.get(0);
    }

    private List<TSPChromosome> init(final TSPGene[] points, final int initialSize) {
        final List<TSPChromosome> eden = new ArrayList<>();
        for(int i = 0; i < initialSize; i++) {
            final TSPChromosome chromosome = TSPChromosome.create(points);
            eden.add(chromosome);
        }
        return eden;
    }

    void update() {
        doCrossOver();
        doMutation();
        doSpawn();
        doSelection();
    }

    private void doSelection() {
        this.population.sort(Comparator.comparingDouble(TSPChromosome::getDistance));
        this.population = this.population.stream().limit(this.initialSize).collect(Collectors.toList());
    }
```

Рис. 3.12. - Клас TSPPopulation.java

Клас Utils.java реалізує введення початкових параметрів у вигляді координат міст.



```java
    private TSPUtils() {
        throw new RuntimeException("No!");
    }

    private static TSPGene[] generateData(final int numDataPoints) {
        final TSPGene[] data = new TSPGene[numDataPoints];
        for(int i = 0; i < numDataPoints; i++) {
            data[i] = new TSPGene(TSPUtils.randomIndex(World.WIDTH),
                    TSPUtils.randomIndex(World.HEIGHT));
        }
        return data;
    }

    static int randomIndex(final int limit) {
        return R.nextInt(limit);
    }

    static<T> List<T>[] split(final List<T> list) {
        final List<T> first = new ArrayList<>();
        final List<T> second = new ArrayList<>();
        final int size = list.size();
        final int partitionIndex = 1 + TSPUtils.randomIndex(list.size());
        IntStream.range(0, size).forEach(i -> {
            if(i < (size+1)/partitionIndex) {
                first.add(list.get(i));
            } else {
                second.add(list.get(i));
            }
        });
        return (List<T>[]) new List[] {first, second};
    }


}
```

Рис. 3.13. - Клас TSPUtils.java

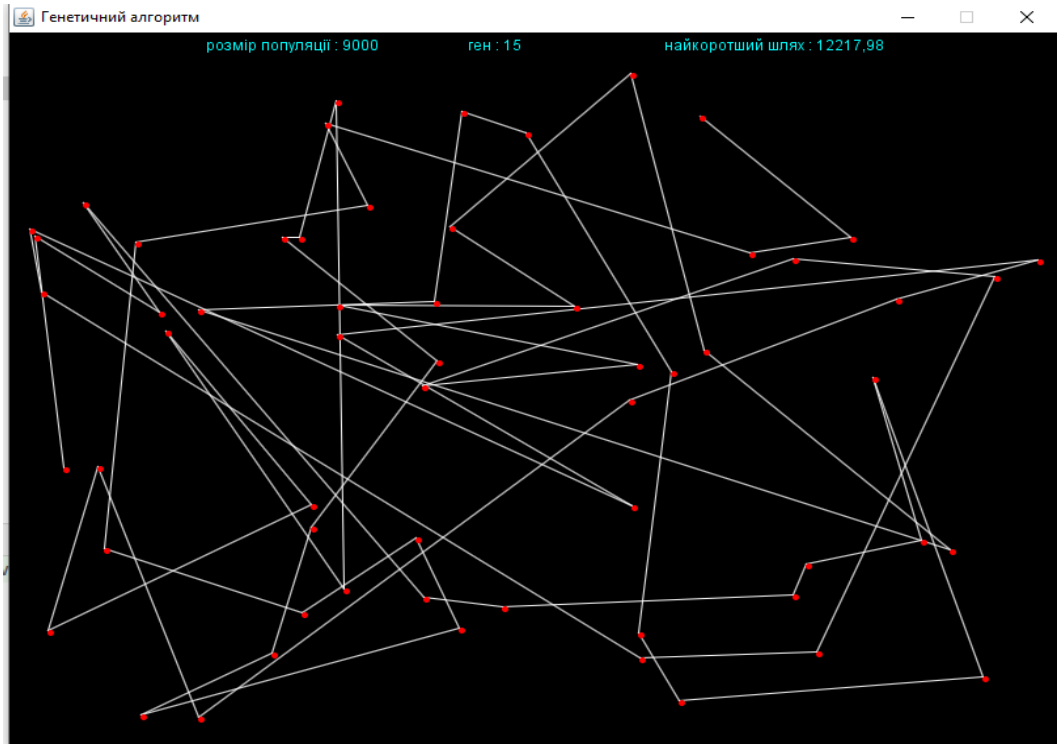


Рис. 3.14. - Скріншот реалізації програми TSP_GA. Початкова фаза генерації популяції шляхів між містами

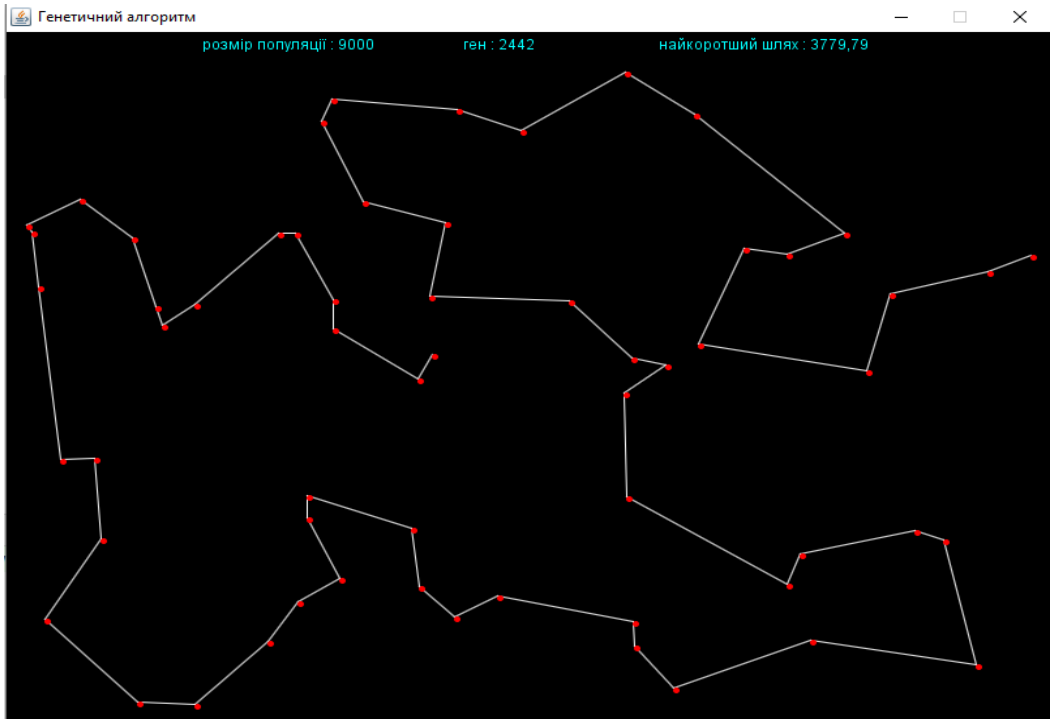


Рис. 3.15. - Скріншот реалізації програми TSP_GA. Кінцева фаза генерації популяції шляхів між містами з розрахунком найкоротшого шляху

ВИСНОВКИ

В результаті виконання магістерської роботи підвищено ефективність системи планування та оптимізації транспортних перевезень за допомогою генетичного методу. Використання ефективно розпараллелюваних оптимізаційних алгоритмів для вирішення проблем маршрутизації транспортних засобів є ключем до мінімізації витрат будь-якої інтелектуальної транспортної системи з обмеженою рентабельністю.

При виконанні роботи було виконано наступні задачі.

1. Проведено порівняльний аналіз методів реалізації ТЗ. Встановлено, що використання еволюційних алгоритмів, до яких відноситься й генетичний алгоритм, дозволяють знайти адекватні за часом виконання рішення для задач з великою кількістю клієнтів (понад 50-ти клієнтів).

2. Здійснено детальний аналіз складових генетичного алгоритму. Встановлено суттєвий вплив розміру початкової популяції на якість кінцевого рішення.

3. Досліджено метод розпаралелювання основних операторів генетичного алгоритму на основі TBB та CUDA, які оцінені на різних наборах даних відповідно до варіацій операторів ГА, включаючи довжину хромосом, кількість поколінь та розмір популяції.;

4. Згідно з експериментами, можна стверджувати, що коли генетичний алгоритм застосовується до невеликої популяції, метод на основі TBB має найкращу продуктивність. В іншому випадку, враховуючи здатність CUDA визначати максимальну необхідну кількість потоків для обчислень, використання CUDA є більш ефективним.

5. Максимальний час реалізації генетичного алгоритму за численності популяції у 65536 осіб при чисельності хромосом (міст) у 918 становив не більше 30 с для платформи CUDA, тоді як його аналоги мають максимальний час реалізації від 100 с до 375 с.

СПИСОК ЛІТЕРАТУРИ

1. Dantzig G.B., Ramser J.H. The Truck Dispatching Problem // Management science, Vol. 6, No. 1, 1959. pp. 80-91.

2. Clarke G., Wright J.W. Scheduling of vehicles from a central depot to a number of delivery points // Operations research, Vol. 12, No. 4, 1964. pp. 568-581.

3. Сергеев С.И., Сигал И.Х, Меламед И.И. Задача коммивояжера. Вопросы теории // Автоматика и телемеханика, № 9, 1989. С. 3-33.

4. Lenstra J.K., Kan A.H.G. Complexity of vehicle routing and scheduling problems // Networks, Vol. 11, No. 2, 1981. pp. 221-227.

5. Parragh S., Doerner K., Hartl R. A survey on pickup and delivery problems. Part I: Transportations between customers and depot // J. Betriebswirtschaft. 2008. V. 58. No 1. P. 21–51.

6. Jozefowiez N., Semet F., Talbi E.G. Multi-objective vehicle routing problems //European journal of operational research, Vol. 189, No. 2, 2008. pp. 293-309.

7. Никонов О.Я., Подоляка О.А., Подоляка А.Н., Скакалина Е.В. Математические методы решения многокритериальной задачи о назначениях // Вестник Харьковского национального автомобильно-дорожного университета, № 55, 2011. С. 103-111.

8. Shanmugam G., Ganesan P., Vanathi D.P.T. Meta heuristic algorithms for vehicle routing problem with stochastic demands // Journal of Computer Science, Vol. 7, No. 4, 2011. P. 533.

9. Lecluyse C., Van W.T., Peremans H. Vehicle routing with stochastic timedependent travel times // 4OR: A Quarterly Journal of Operations Research, Vol. 7, No. 4, 2009. pp. 363-377.

10. Kek A.G.H., Cheu R.L., Meng Q. Distance-constrained capacitated vehicle routing problems with flexible assignment of start and end depots // Mathematical and Computer Modelling, Vol. 47, No. 1, 2008. pp. 140-152.

11. Berbeglia G., Cordeau J.F., Gribkovskaia I., Laporte G. Static pickup and delivery problems: a classification scheme and survey // Top, Vol. 15, No. 1, 2007. pp. 1-31

12. Zhou W., Song T., He F., Liu X. Multiobjective vehicle routing problem with route balance based on genetic algorithm // Discrete Dynamics in Nature and Society, 2013.

13. Hemmelmayr V.C., Doerner K.F., Hartl R.F. A variable neighborhood search heuristic for periodic routing problems // European Journal of Operational Research, Vol. 195, No. 3, 2009. pp. 791-802.

14. Morais V.W.C., Mateus G.R., Noronha T.F. Iterated local search heuristics for the vehicle routing problem with cross-docking // Expert Systems with Applications, Vol. 41, No. 16, 2014. pp. 7495-7506.

15. Vigo D. A heuristic algorithm for the asymmetric capacitated vehicle routing problem // European Journal of Operational Research, Vol. 89, No. 1, 1996. pp. 108-126.

16. Parragh S.N., Doerner K.F., Hartl R.F. A survey on pickup and delivery problems // Journal für Betriebswirtschaft, Vol. 58, No. 1, 2008. pp. 21-51.

17. Toth P., Vigo D., eds. Vehicle routing: problems, methods, and applications. 2nd ed. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, 2014.

18. Toth P., Vigo D. The vehicle routing problem. Vol 9 of SIAM Monographs on Discrete Mathematics and Applications. Philadelphia. 2002.

19. Herrero R., Rodríguez A., Cáceres-Cruz J., Juan A.A. Solving vehicle routing problems with asymmetric costs and heterogeneous fleets // International Journal of Advanced Operations Management, Vol. 6, No. 1, 2014. pp. 58-80.

20. Toth P., Vigo D. Models, relaxations and exact approaches for the capacitated vehicle routing problem // Discrete Applied Mathematics, Vol. 123, No. 1, 2002. pp. 487-512.

21. Christofides N., Mingozzi A., Toth P. The vehicle routing problem // In: Combinatorial Optimization. Wiley, 1979. pp. 315–338.

22. Laporte G., Desrochers M., Nobert Y. Two exact algorithms for the distance-constrained vehicle routing problem // Networks, Vol. 14, No. 1, 1984. pp. 161-

23. Laporte G., Nobert Y., Desrochers M. Optimal routing under capacity and distance restrictions // Operations research, Vol. 33, No. 5, 1985. pp. 1050-1073.

24. Solomon M.M. Algorithms for the vehicle routing and scheduling problems with time window constraints // Operations research, Vol. 35, No. 2, 1987. pp. 254-265.

25. Cordeau J.F. The VRP with time windows // Montréal: Groupe d'études et de recherche en analyse des décisions, 2000.

26. Nagata Y., Bräysy O., Dullaert W. A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows // Computers & operations research, Vol. 37, No. 4, 2010. pp. 724-737.

27. Taillard É., Badeau P., Gendreau M., Guertin F., Potvin J.Y. A tabu search heuristic for the vehicle routing problem with soft time windows // Transportation science, Vol. 31, No. 2, 1997. pp. 170-186.

28. Jindal P. Towards the solution of variants of vehicle routing problem // Global Journal of Computer Science and Technology, Vol. 11, No. 15, 2011.

29. Chao I.M., Golden B.L., Wasil E. A new heuristic for the multi-depot vehicle routing problem that improves upon best-known solutions // American Journal of Mathematical and Management Sciences, Vol. 13, No. 3-4, 1993. pp. 371-406.

30. Ho W., Ho G.T., Ji P., Lau H.C. A hybrid genetic algorithm for the multi-depot vehicle routing problem // Engineering Applications of Artificial Intelligence, Vol. 21, No. 4, June 2008. pp. 548-557.

31. Daneshzand F. The vehicle-routing problem // Logistics Operations and Management, Vol. 8, 2011. pp. 127-153.

32. Crevier B., Cordeau J.F., Laporte G. The multi-depot vehicle routing problem with inter-depot routes // European Journal of Operational Research, Vol. 176, No. 2, 2007. pp. 756-773.

33. Christofides N., Beasley J.E. The period routing problem // Networks, Vol. 14, No. 2, 1984. pp. 237-256.

34. Francis P., Smilowitz K., Tzur M. The period vehicle routing problem with service choice // Transportation Science, Vol. 40, No. 4, 2006. pp. 439-454.

35. Francis P., Smilowitz K. Modeling techniques for periodic vehicle routing problems // Transportation Research Part B: Methodological, Vol. 40, No. 10, 2006. pp. 872-

36. Nagy G., Salhi S. Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries // European journal of operational research, Vol. 162, No. 1, 2005. pp. 126-141.

37. Savelsbergh M.W.P., Sol M. The general pickup and delivery problem //Transportation science, Vol. 29, No. 1, 1995. pp. 17-29.

38. Moura A., Oliveira J.F. An integrated approach to the vehicle routing and container loading problems // OR spectrum, Vol. 31, No. 4, 2009. pp. 775-800.

39. Cordeau, J.F., Iori M., Laporte G., Salazar González J.J. A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with LIFO loading // Networks, Vol. 55, No. 1, 2010. pp. 46-59.

40. Dror M., Laporte G., Trudeau P. Vehicle routing with split deliveries // Discrete Applied Mathematics, Vol. 50, No. 3, 1994. pp. 239-254.

41. Dror M., Trudeau P. Savings by split delivery routing // Transportation Science, Vol. 23, No. 2, 1989. pp. 141-145.

42. Archetti C., Savelsbergh M.W.P., Speranza M.G. To split or not to split: That is the question // Transportation Research Part E: Logistics and Transportation Review, Vol. 44, No. 1, 2008. pp. 114-123.

43. Gendreau M., Laporte G., Séguin R. Stochastic vehicle routing // European Journal of Operational Research, Vol. 88, No. 1, 1996. pp. 3-12.

44. Gendreau M., Laporte G., Séguin R. An exact algorithm for the vehicle routing problem with stochastic demands and customers // Transportation science, Vol. 29, No. 2, 1995. pp. 143-155.

45. Chen D., Chen D., Yang Y. Nondeterministic Vehicle Routing Problem: A Review // Advances in Information Sciences and Service Sciences, Vol. 5, No. 9, 2013. pp. 485-493.

46. Garrido P., Riff M.C. DVRP: a hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic // Journal of Heuristics, Vol. 16, No. 6, 2010. pp. 795-834.

47. Sariklis D., Powell S. A heuristic method for the open vehicle routing problem // Journal of the Operational Research Society, Vol. 51, No. 5, 2000. pp. 564-573.154

48. Brandão J. A tabu search algorithm for the open vehicle routing problem //European Journal of Operational Research, Vol. 157, No. 3, 2004. pp. 552-564.

49. Choi E., Tcha D.W. A column generation approach to the heterogeneous fleet vehicle routing problem // Computers & Operations Research, Vol. 34, No. 7, 2007. pp. 2080-2095.

50. Golden B., Assad A., Levy L., Gheysens F. The fleet size and mix vehicle routing problem // Computers & Operations Research, Vol. 11, No. 1, 1984. pp. 49-66.

51. Mitchell J.E. Branch-and-cut algorithms for combinatorial optimization problems //Handbook of applied optimization, 2002. pp. 65-77.

52. Shvaiko P., Euzenat J. A survey of schema-based matching approaches // Journal on data semantics IV. – Springer, Berlin, Heidelberg, 2005. pp. 146-171.

53. Fisher M.L., Jaikumar R. A generalized assignment heuristic for vehicle routing //Networks, Vol. 11, No. 2, 1981. pp. 109-124.

54. Gillett B.E., Miller L.R. A heuristic algorithm for the vehicle-dispatch problem //

Operations research, Vol. 22, No. 2, 1974. pp. 340-349.

55. Ryan D.M., Hjorring C., Glover F. Extensions of the petal method for vehicle routeing // Journal of the Operational Research Society, Vol. 44, No. 3, 1993. pp. 289-296.

56. Osman I.H. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem // Annals of operations research, Vol. 41, 1993. pp. 421-451.

57. Prosser P., Shaw P. Study of greedy search with multiple improvement heuristics for vehicle routing problems, University of Strathclyde, Department of Computer Science, Glasgow, Research Report 96/201 1996.

58. Amberg A., Domschke W., Voss S. Multiple center capacitated arc routing problems: A tabu search algorithm using capacitated trees // European Journal of Operational Research, Vol. 124, No. 2, 2000. pp. 360-376.

59. Arbelaitz O., Rodriguez C., Zamakola I. Low cost parallel solutions for the VRPTW optimization problem // Proceedings International Conference on Parallel Processing Workshops. – IEEE Computer Society. 2001. pp. 176-181.

60. Alba E., Dorronsoro B. Solving the vehicle routing problem by using cellular genetic algorithms // European Conference on Evolutionary Computation in Combinatorial Optimization. Berlin, Heidelberg. 2004. Vol. 3004. pp. 11-20.

61. Bullnheimer B., Hartl R.F., Strauss C. Applying the ant system to the vehicle routing problem // Meta-heuristics: Advances and trends in local search paradigms for optimization. Boston: Kluwer. 1999. pp. 285-296.

62. Ghaziri H.E.L. Solving routing problems by a self-organizing map. In In T. Kohonen, K. Makisara, O. Simula, J. Kangas, editors // Artificial neural network.– North-Holland, Amsterdam, 1991. pp. 829-834.

63. Скобцов Ю.А., Федоров Е.Е. Метаэвристики: монография. Донецк: Изд-во «Ноулидж» (Донецкое отделение), 2013. 426 с.

64. Wu J, Zhou L, Du Z, Lv Y (2019) Mixed steepest descent algorithm for the traveling salesman problem and application in air logistics. Transport Res Part E Logistic Transport Rev 126:87–102

65. Vidal T, Laporte G, Matl P (2019) A concise guide to existing and emerging vehicle routing problem variants. Eur J Oper Res

66. Whaiduzzaman M, Sookhak M, Gani A, Buyya R (2014) A survey on vehicular cloud computing. J Netw Comput Appl 40:325–344

67. Avraham E, Raviv T (2019) The data-driven time-dependent traveling salesperson problem

68. Giap CN, Ha DT Parallel genetic algorithm for minimum dominating set problem. In: Computing, Management and Telecommunications (ComManTel), 2014 International Conference on, 2014. IEEE, pp 165–169

69. Скобцов Ю.А., Федоров Е.Е. Метаэвристики: монография / Ю.А. Скобцов, Е.Е. Федоров. – Донецк: Изд-во «Ноулидж» (Донецкое отделение), 2013. – 426 с.

70. Zhu J, Li Q Application of Hybrid MPI+ TBB Parallel Programming Model for Traveling Salesman Problem. In: Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing, 2013. IEEE, pp 2164–2167

71. Sánchez LNG, Armenta JJT, Ramírez VHD (2015) Parallel genetic algorithms on a GPU to solve the travelling salesman problem. Difu100ci@ Revista en Ingeniería y Tecnología, UAZ 8 (2)

72. Kang S, Kim S-S, Won J, Kang Y-M (2016) GPU-based parallel genetic approach to large-scale travelling salesman problem. J Supercomput 72(11):4399–4414

73. Saxena R, Jain M, Sharma D, Jaidka S (2019) A review on VANET routing protocols and proposing a parallelized genetic algorithm based heuristic modification to mobicast routing for real time message passing. J Intell Fuzzy Systems 36(3):2387–2398

74. NVIDIA. NVIDIA CUDA (Compute Unified Device Architecture) Programming Guide, (accessed September 2019) http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf. Accessed 1 Sept 2019

75. Yip CM, Asaduzzaman A A promising CUDA-accelerated vehicular area network simulator using NS-3. In: Performance Computing and Communications Conference (IPCCC), 2014 IEEE International, 2014. IEEE, pp 1–2

76. Intel T (2018) Intel Threading Building Blocks. Available: http:// threadingbuildingblocks.org/. Accessed 16 Mar 2019

77. Hougardy S, Wilde M (2014) On the nearest neighbor rule for the metric traveling salesman problem. Discret Appl Math.

ДОДАТОК

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

МАГІСТЕРСЬКА РОБОТА
«Підвищення ефективності системи планування та оптимізації транспортних перевезень на основі генетичного методу»

Виконав: студент групи ПДМ – 61, Бондаренко Павєл Вячеславович

Керівник: , к.т.н., доцент Негоденко Олена Василівна

Київ - 2021

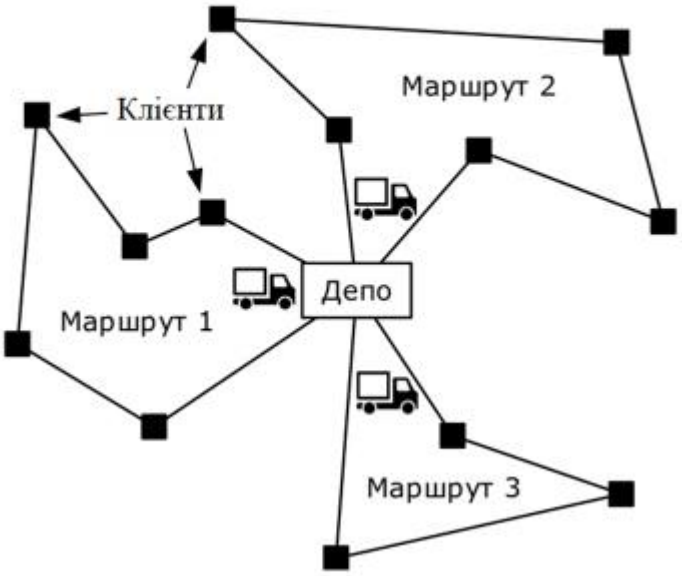
МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Об'єкт дослідження – планування та оптимізація транспортних перевезень.

Предмет дослідження – алгоритми планування та методи оптимізації транспортних перевезень.

Мета роботи - підвищення ефективності системи планування та оптимізації транспортних перевезень на основі генетичного методу.

МОДЕЛЬ КЛАСИЧНОЇ VRP
(Vehicle Routing Problem – Маршрутизація Транспортних Засобів)



МАТЕМАТИЧНА ПОСТАНОВКА ТРАНСПОРТНОЇ ЗАДАЧІ З ОБМЕЖЕННЯМ ЗА ЧАСОМ

Математично ТЗ з обмеженням за часом можна представити у вигляді графу:

$$G = (N, A)$$

G = (N, A), де: N - множина вершин, відповідних набору клієнтів (customers) (вершини 1,2, ..., n) і вихідного депо (depot), в якому починають і закінчують свій маршрут всі автомобілі (вершини 0 і n + 1); A - набір дуг, що з'єднують вершини графу.

C - множина клієнтів, $|C| = n$;

i, j - *i*-й і *j*-й клієнти, $i \in C, j \in C$;

(i, j) $\in A$-дуга, що з'єднує *i*-у і *j*-у вершини графа;

d_i - попит *i*-го клієнта;

t_{ij} - час переміщення по дузі *(i, j)*, що складається з часу обслуговування клієнта *i* та часу переміщення автомобіля від клієнта *i* до клієнту *j*;

c_{ij} - вартість переміщення автомобіля від клієнта *i* до клієнтуу;

V - кількість ідентичних автомобілів вантажопідйомністю q;

k - к-й автомобіль, $k \in V$;

$$\sum_{k \in V} \sum_{(i,j) \in A} c_{ij} X_{ij}^k \quad (2.1)$$

$$\sum_{k \in V} \sum_{j \in N} X_{ij}^k = 1, \forall i \in C \quad (2.2)$$

$$\sum_{i \in C} d_i \sum_{j \in N} X_{ij}^k \leq q, \forall k \in V \quad (2.3)$$

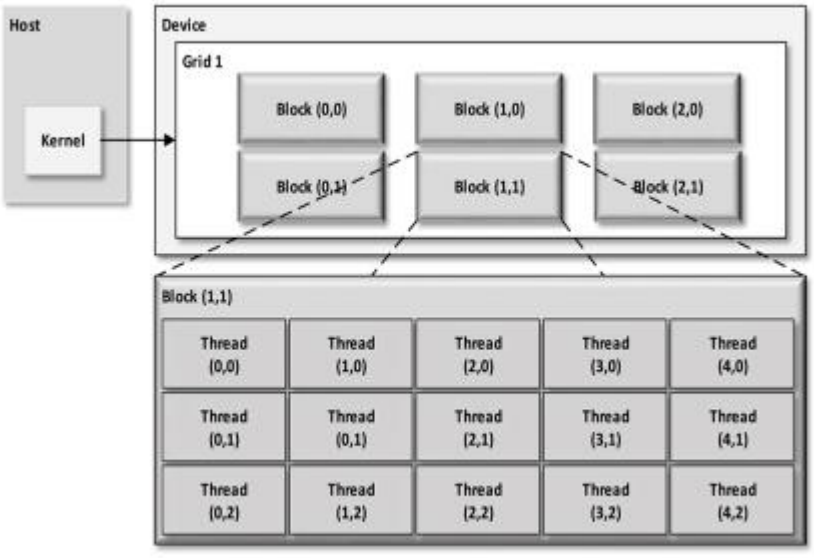
$$\sum_{j \in N} X_{0j}^k - 1, \forall k \in V \quad (2.4)$$

$$\sum_{j \in N} X_{ih}^k - \sum_{j \in N} X_{ih}^k = 0, \forall h \in C, \forall k \in V \quad (2.5)$$

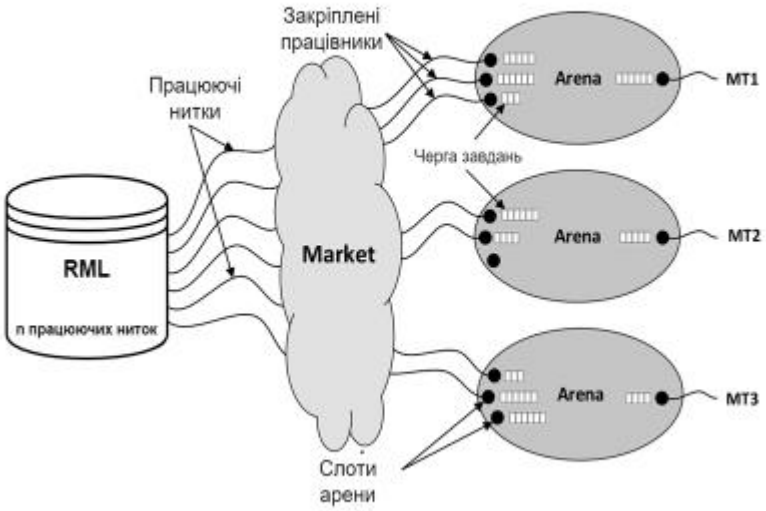
$$\sum_{j \in N} X_{i,n+1}^k = 1, \forall k \in V \quad (2.6)$$

СТРУКТУРА ГЕНЕТИЧНОГО АЛГОРИТМУ

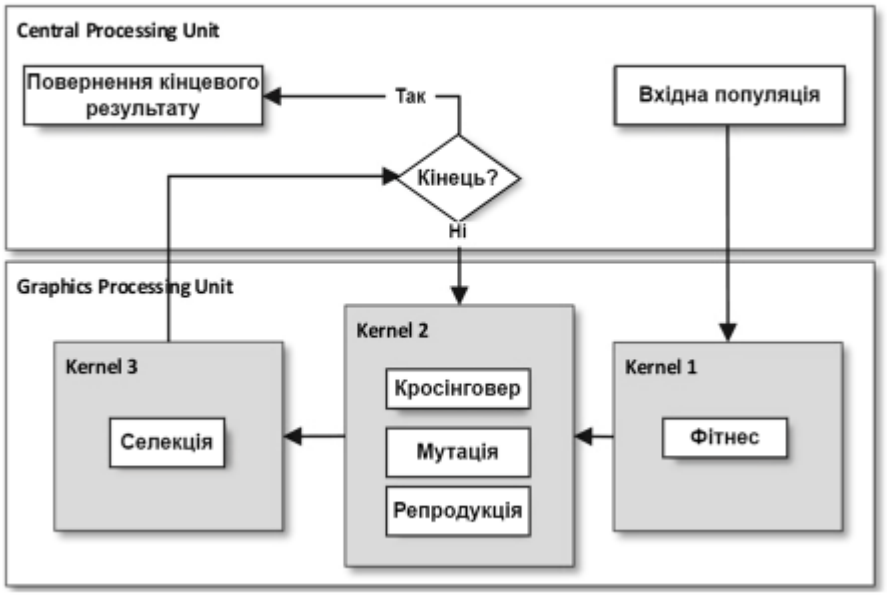
СУТНІСТЬ ПРОГРАМНОЇ ПЛАТФОРМИ (CUDA)



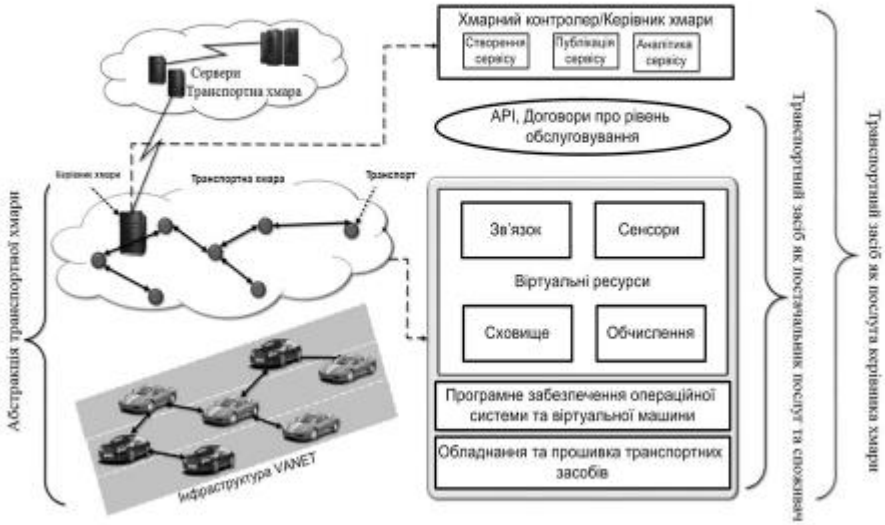
СУТНІСТЬ БІБЛІОТЕКИ ПАРАЛЕЛЬНОГО ПРОГРАМУВАННЯ (TBB)



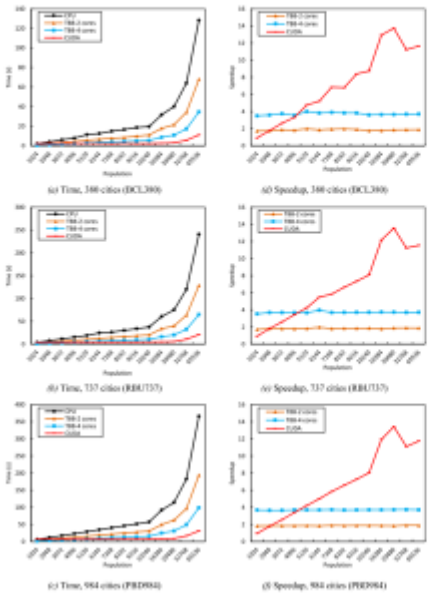
ПАРАЛЕЛЬНІ ПІДХОДИ ДЛЯ ЗАПУСКУ ГЕНЕТИЧНОГО АЛГОРИТМУ



АВТОМОБІЛЬНІ ХМАРИ ОБЧИСЛЕННЯ ДЛЯ ВИКОРИСТАННЯ ЗАПРОПОНОВАНОГО СПОСОБУ

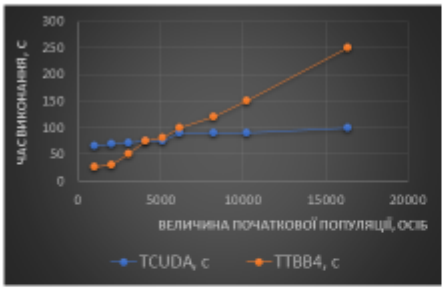


ВПЛИВ ЧИСЕЛЬНОСТІ ПОПУЛЯЦІЇ НА ПРИСКОРЕННЯ ПАРАЛЕЛЬНИХ МЕТОДІВ

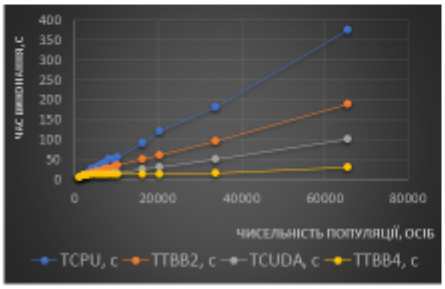


Кожна підділянка позначена відповідним розміром популяції TSP на основі TBB для двоядерних і чотириядерних процесорів становить 1,99 і 3,99 відповідно, тоді як максимальне прискорення TSP на основі CUDA на 1280 ядрах становить 13,73 с

ЧАС РЕАЛІЗАЦІЇ ГЕНЕТИЧНОГО АЛГОРИТМУ

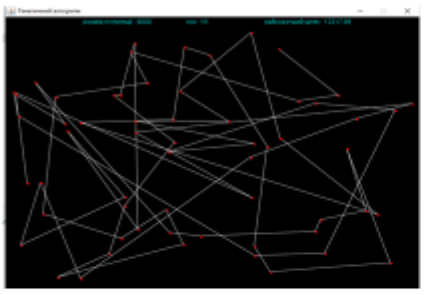


Порівняння часу реалізації генетичного алгоритму на платформах CUDA та TBB з 4 ядрами в залежності від початкової популяції

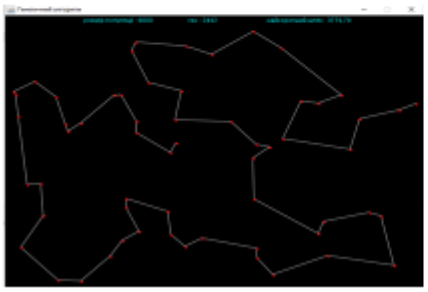


Максимальний час реалізації генетичного алгоритму системи з 918 міст у випадку CPU (375 с), TBB з 2 ядер (190 с), TBB з 4 ядер (100 с) та CUDA (30 с).

РЕАЛІЗАЦІЯ ПРОГРАМИ TSP_GA



Початкова фаза генерації популяції шляхів між містами



Кінцева фаза генерації популяції шляхів між містами з розрахунком найкоротшого шляху на прикладі реалізації програми еталонної моделі ГА

ВИСНОВКИ

При виконанні роботи було виконано наступні задачі.

1. Проведено порівняльний аналіз методів реалізації ТЗ. Встановлено, що використання еволюційних алгоритмів, до яких відноситься й генетичний алгоритм, дозволяють знайти адекватні за часом виконання рішення для задач з великою кількістю клієнтів (понад 50-ти клієнтів).

2. Здійснено детальний аналіз складових генетичного алгоритму. Встановлено суттєвий вплив розміру початкової популяції на якість кінцевого рішення.

3. Досліджено метод розпаралелювання основних операторів генетичного алгоритму на основі TBB та CUDA, які оцінені на різних наборах даних відповідно до варіацій операторів ГА, включаючи довжину хромосом, кількість поколінь та розмір популяції.;

4. Згідно з експериментами, можна стверджувати, що коли генетичний алгоритм застосовується до невеликої популяції, метод на основі TBB має найкращу продуктивність. В іншому випадку, враховуючи здатність CUDA визначати максимальну необхідну кількість потоків для обчислень, використання CUDA є більш ефективним.

5. Максимальний час реалізації генетичного алгоритму за численності популяції у 65536 осіб при чисельності хромосом (міст) у 918 становив не більше 30 с для платформи CUDA, тоді як його аналоги мають максимальний час реалізації від 100 с до 375 с.

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ

Статті:

1. Бондаренко П.В. Ефективне рішення пареллельного генетичного алгоритму для проблеми маршрутизації транспортних засобів у хмарній реалізації інтелектуальних транспортних систем // журнал «Телекомунікаційні та інформаційні технології», 2021.

Теза:

1. Бондаренко П.В. Проєктування мережі громадського транспорту з використанням генетичних алгоритмів // XIII НАУКОВО-ТЕХНІЧНА КОНФЕРЕНЦІЯ СТУДЕНТІВ ТА МОЛОДИХ ВЧЕНИХ НАВЧАЛЬНО-НАУКОВОГО ІНСТИТУТУ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЕРЖАВНОГО УНІВЕРСИТЕТУ ТЕЛЕКОМУНІКАЦІЙ, 10 грудня 2021 року, Державний університет телекомунікації, Київ, Україна.

ДЯКУЮ ЗА УВАГУ!