МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ

ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

другого (магістерського) рівня вищої освіти

на тему: **“Розробка та дослідження алгоритмів вирішення завдань комівояжера”**

Виконав: студент гр. Іт-61

Спеціальності 126 «Інформаційні системи та технології»
(шифр і назва)

Порвіш Святослав Любомирович
(Прізвище та ініціали)

Керівник: к.т.н., доц. Лиса О.В.
(Прізвище та ініціали)

Рецензенти: д.т.н., проф. Власовець В.М.
(Прізвище та ініціали)

ДУБЛЯНИ-2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ
ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Другий (магістерський) рівень вищої освіти
Спеціальність 126 «Інформаційні системи та технології»

"ЗАТВЕРДЖУЮ"
Завідувач кафедри_____
д.т.н., проф. А.М. Тригуба
"____"_____2023 р.

ЗАВДАННЯ

на кваліфікаційну роботу студенту

Порвіш Святослав Любомирович

1. Тема роботи: «Розробка та дослідження алгоритмів вирішення завдань комівояжера»

Керівник роботи Лиса Ольга Володимирівна, к.т.н., доцент.
Затверджені наказом по університету від 12.09 2024 року № 616 /к-с.

2. Строк подання студентом роботи 05.12.2024 р.

3. Вихідні дані до роботи: _ алгоритми вирішення задачі комівояжера, стандартна бібліотека C++, бібліотека функції рядкових перетворень Iconv, пакет для читання XML-файлів Xerces-C, бібліотека графічного виводу libgd2

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

Вступ.
1. Аналіз предметної області та завдання кваліфікаційної роботи.
2. Опис реалізації алгоритмів вирішення завдань комівояжера.
3. Результати розробки і дослідження алгоритмів вирішення завдань комівояжера.
4. Охорона праці та безпека у надзвичайних ситуаціях.
5. Визначення ефективності від використання алгоритмів вирішення завдань комівояжера.
Висновки та пропозиції.
Список використаної літератури

5. Перелік ілюстраційного матеріалу (з точним зазначенням обов'язкових слайдів): *формальна постановка задачі; представлення задачі k-комівояжерів у теорії графів; алгоритм розрізання "загального" маршруту; алгоритм розрізання маршруту для задачі багатьох комівояжерів; графічне зображення методу попереднього поділу міст на групи для кожного комівояжера; графічне зображення, яке демонструє обмінну оптимізацію поділу міст на дві групи; графічне зображення кутового сортування міст на площині; взаємозв'язки між розділами пакету TSP; тестування алгоритмів вирішення завдань комівояжера; ілюстрація результатів роботи алгоритму.*

6. Консультанти з розділів:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1, 2, 3, 5	*Лиса О.В., доцент кафедри інформаційних технологій*		
4	*Городецький І.М., доцент кафедри фізики, інженерної механіки та безпеки виробництва*		

7. Дата видачі завдання 30 червня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Приміт-ка
1	*Написання першого розділу*	30.06.24-04.07.24	
2	*Виконання другого розділу та аркушів ілюстраційного матеріалу до нього*	05.07.24-14.08.24	
3.	*Виконання третього розділу та аркушів ілюстраційного матеріалу до нього*	15.08.24-24.09.24	
4.	*Написання розділу «Охорона праці та безпека у надзвичайних ситуаціях»*	25.09.24-10.10.24	
5.	*Оцінення ефективності запропонованої системи*	11.10.24-31.10.24	
6.	*Завершення оформлення розрахунково-пояснювальної записки та презентації*	01.11.24-30.11.24	
7.	*Завершення роботи в цілому*	01.12.24-05.12.24	

Студент _____ Порвіш С.Л.
(підпис)

Керівник роботи _____ Лиса О.В.
(підпис)

УДК 621.311.1

Розробка та дослідження алгоритмів вирішення завдань комівояжера. Порвіш С.Л. Кафедра інформаційних технологій – Дубляни, Львівський НУП, 2024. Кваліфікаційна робота: 66 с. текст. част., 12 рис., 12 табл., 10 арк. ілюстраційного матеріалу, 30 джерел.

Задача k-комівояжерів є узагальненням класичної задачі комівояжера, коли необхідно визначити оптимальні маршрути для k комівояжерів, які починають і закінчують свої маршрути у визначених точках (або в одній загальній точці). Завдання полягає у мінімізації загальної відстані або часу, що витрачається всіма комівояжерами. Виконано формальну постановку задачі **k**-комівояжерів (З**k**К). Виконано аналіз алгоритмів вирішення задачі комівояжера. Охарактеризовано особливості наближених методів вирішення задачі комівояжера. Реалізовано програмний пакет вирішення задачі комівояжера, фундаментом для розробки пакету є: стандартна бібліотека C++, бібліотека функції рядкових перетворень Iconv, пакет для читання XML-файлів Xerces-C, бібліотека графічного виводу libgd2 та деякі функції операційної системи для роботи з файлами та вимірювання часу. Проведено обчислювальний експеримент для порівняння можливих методів вирішення ЗkК та виявлення тенденцій для продовження досліджень. Розроблено заходи стосовно охорони праці та безпеки у надзвичайних ситуаціях. Визначено економічну ефективність від розроблених програмних модулів.

Ключові слова: задача комівояжера, матриця відстаней, алгоритм розрізання "загального" маршруту, алгоритм поділу на групи, пакет TSP, обчислювальний експеримент.

ЗМІСТ

ВСТУП

В даний час у зв'язку з бурхливим розвитком логістичних мереж компаній, а також вимог сучасного темпу розвитку бізнесу та конкурентних відносин, важливо знаходити оптимальні рішення щодо зниження логістичних витрат, частиною яких є транспортні витрати.

Можливість їх зменшення завжди цікавить перевізників. Сформульовано кілька завдань, пов'язаних із цією проблемою. Серед них – завдання комівояжера (ЗК).

У цій роботі пропонується розглянути більш загальний варіант цієї задачі – завдання k комівояжерів (ЗkК). Її метою є мінімізація вартості об'їзду міст не одним, а k комівояжерами.

Насправді постановка ЗkК може мати безліч варіантів. Можливо, потрібно буде брати до уваги максимальне завантаження одного комівояжера, затримку при відвідуванні кожного міста, можливість існування кількох баз тощо. Тут ми не враховуватимемо такого роду деталі і спробуємо сформулювати спрощену постановку цього завдання. Це дозволить нам дослідити якість базових алгоритмів, які будуть основою практичної реалізації, що враховує всі тонкощі. Обмежимося випадком збалансованої постановки завдання, коли кількість міст для кожного комівояжера не повинна відрізнятися більш ніж на одиницю.

Відмінність збалансованої ЗЗkК від завдання комівояжера полягає в наступному:

1) має бути побудовано k (k > 1) замкнутих маршрутів, по одному для кожного комівояжера;

2) задається n + 1 місто, причому одне з них, зване базою, повинне входити у всі маршрути, а кожне з інших повинно входити рівно до одного з маршрутів;

3) кількість міст у маршрутах має відрізнятись один від одного не більше, ніж на одиницю;

4) сумарна вартість усіх маршрутів має бути мінімальною.

Як відомо, завдання комівояжера - одне з оптимізаційних NP-важких завдань, яке не вирішується за поліноміальний час. Вона може бути поставлена у комбінаторному формулюванні як завдання про знаходження послідовності з n міст із мінімальною вартістю циклічного об'їзду. Складність ЗК в тому, що спроба знайти деяке локальне рішення не дозволяє побудувати шлях для вирішення всього завдання.

В даний час методи пошуку точних та наближених рішень ЗК вивчалися багатьма дослідниками та запропоновано безліч алгоритмів, що дозволяють знайти такі рішення.

Ми будемо користуватися деякими з них при вирішенні ЗКК. Оскільки ці ці завдання мають багато спільного, то завдання k комівояжерів успадковує всі труднощі, пов'язані з пошуком рішень ЗК.

На підставі викладених фактів можна стверджувати, що тема кваліфікаційної роботи «Розробка та дослідження алгоритмів вирішення завдань комівояжера» є досить актуальною та своєчасною.

У дослідженнях, наведених у цій роботі, розглядатимуться лише наближені методи рішення ЗkК.

Для пошуку алгоритмів рішення ЗkК будемо проводити обчислювальний експеримент для порівняння можливих методів вирішення ЗkК та виявлення тенденцій для продовження досліджень.

Практична цінність полягає в тому, що запропоновані алгоритми дозволять знизити транспортні витрати, пов'язані з розвезенням продукції з точки зберігання до місць споживання.

Основними об'єктами дослідження під час вирішення поставленої завдання є самі алгоритми, з яких можна виконувати обробку даних. Технічні деталі реалізації та варіанти реальних даних не становлять особливої складності.

Поставлене завдання полягає у знаходженні якомога ефективніших алгоритмів рішення ЗkК. Головним критерієм оцінки є довжина обчислюваного шляху, але одержувані алгоритми повинні бути ефективними з точки зору часу роботи.

Предмет дослідження є алгоритми рішення завдання k комівояжерів.

Метою роботи є створення додатку, придатного для використання на підприємствах автотранспорту. Додаток має обробляти інформацію про міські магістралі з урахуванням їх завантаження та затримок на світлофорах, будувати граф шляхів руху, де кожне ребро має вагу, що вказує на вартість переїзду ним. На основі цього графа складається матриця вартості переїздів між усіма вершинами. Таку матрицю можна отримати відомими алгоритмами Дейкстри та Флойда і, використовуючи дані цієї матриці, додаток видаватиме послідовності точок для об'їзду кожному комівояжеру.

На цей час підготовлений набір утиліт, що дозволяє виконувати деякі операції над даними завдання k комівояжерів. До них відносяться утиліти пошуку рішення, утиліта порівняння якості та часу різних алгоритмів, утиліта візуального представлення об'їзду міст на площині та ін. Утиліти мають механізми збереження даних на диск у формат, що базується на XML. Самі алгоритми реалізовані у вигляді бібліотеки C++, що переноситься, яка супроводжується документацією з описом класів та їх методів.

РОЗДІЛ 1.
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ЗАВДАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1.1. Формальна постановка задачі

У загальному випадку, вартість переїзду з міста *i* в місто *j* задаватиметься матрицею, де потрібне значення зберігатиметься на перетині *i*-того рядка і *j*-того стовпця. Обмежимося варіантом, коли така матриця завжди симетрична. Формально розв'язуване завдання можна описати так:

1. На вхід алгоритму надходить матриця відстаней між *n* містами та множина значень відстаней від кожного міста до бази.

2. Необхідно знайти k замкнутих маршрутів, де k > 1, з наступними властивостями:

2.1. через кожне місто має проходити один і лише один маршрут, причому лише один раз;

2.2. всі маршрути мають проходити через базу;

2.3. для будь-якої пари отриманих маршрутів кількість міст у них не повинна відрізнятися більше ніж на одиницю.

3. Сумарна довжина отриманих маршрутів повинна бути якнайменшою.

Для створення матриці вартості переїздів (найкоротших шляхів між вершинами) алгоритм Дейкстри виконується кілька разів: кожна вершина графа по черзі розглядається як стартова вершина, результати записуються у відповідний рядок матриці вартості.

Етапи алгоритму для однієї стартової вершини:

1.Ініціалізація. Нехай n — кількість вершин графа. Початкова вершина позначається як S. Встановлюється масив відстаней D[i], де D[i]=∞ (нескінченність) для всіх вершин, окрім S, для якої D[S]=0.

Множина Q містить всі вершини графа.

2.Вибір вершини з мінімальною відстанню. З множини Q вибирається вершина u, для якої значення D[u] мінімальне.

3.Оновлення сусідніх вершин. Для кожної вершини v, яка є сусідньою для u, обчислюється:

$$D[v] = \min(D[v], D[u] + w(u, v)),$$

де w(u,v) — вага ребра між u та v.

4.Видалення вершини. Вершина u видаляється з множини Q.

5.Повторення. Кроки 2–4 повторюються, доки множина Q не стане порожньою.

6.Результат: Масив D[i] містить мінімальні відстані від вершини S до всіх інших вершин.

Матриця вартості. Алгоритм повторюється для кожної вершини як стартової, формуючи матрицю C, де C[i][j] — найкоротший шлях від вершини iii до вершини jjj.

Алгоритм Флойда-Воршелла (Floyd-Warshall Algorithm). Алгоритм Флойда-Воршелла знаходить найкоротші шляхи між усіма парами вершин у графі. На відміну від алгоритму Дейкстри, Флойд-Воршелл працює з графами, де можливі від'ємні ваги ребер, за умови, що немає циклів із від'ємною вагою.

Побудова матриці вартості переїздів.

Алгоритм Флойда-Воршелла обчислює матрицю найкоротших відстаней між усіма вершинами за один прохід. Це матриця D, де D[i][j] — мінімальна вартість шляху між вершинами i та j.

Етапи алгоритму:

1.Ініціалізація. Початкова матриця D визначається за матрицею суміжності графа:

- Якщо є ребро між i та j із вагою w, тоді D[i][j]=w.
- Якщо i=j, тоді D[i][j]=0 (відстань до себе).
- Якщо немає ребра між i та j, D[i][j]=∞.

2.Основна ітерація. Для кожної вершини k виконується оновлення матриці:

$$D[i][j] = \min(D[i][j], D[i][k] + D[k][j]),$$

де i та j — будь-які вершини графа. Ідея: шлях між i та j через k може бути коротшим, ніж прямий.

3.Повторення для всіх k від 1 до n, де n — кількість вершин.

4.Результат. Матриця D після завершення алгоритму містить найкоротші шляхи між усіма вершинами.

Таблиця 1.1

Порівняння алгоритмів.

Параметр	**Дейкстра**	**Флойд-Воршелл**
Тип графа	З невід'ємними вагами ребер	Може мати від'ємні ваги (без циклів)
Призначення	Від однієї вершини до всіх інших	Між усіма парами вершин
Часова складність	$O(V^2)$ або $O(V \log V + E)$ (з пріоритетною чергою)	$O(V^3)$
Простота реалізації	Простий для одного джерела	Простий для повної матриці
Особливості	Потрібно виконувати кілька разів для отримання матриці	Одноразове виконання для повної матриці

Дейкстра підходить для графів із невеликою кількістю ребер та необхідністю знайти найкоротші шляхи від однієї вершини.

Флойд-Воршелл використовується, якщо потрібно знайти всі найкоротші шляхи одразу або в графах із від'ємними вагами.

При постановці завдання ми визначили умову збалансованості результату. Якщо кількість міст ділиться на кількість комівояжерів без залишку, тоді всі комівояжери повинні проїхати однакову кількість міст. В іншому випадку у частини комівояжерів маршрути будуть довшими на одне місто.

Вхідні дані:

1. Граф G=(V,E)**:**

- V — множина вершин (|V|=n), що відповідають містам (точкам, які необхідно відвідати).
- E — множина ребер ((i,j)∈E), що відповідають маршрутам між вершинами.

2. Вартість маршруту **c(i,j)**:

- Функція вартості c:E→R_+, яка задає відстань, час або іншу метрику між вершинами i та j.

3. Кількість комівояжерів **k**:

- k — кількість доступних комівояжерів.

4. Вихідні умови:

Кожен комівояжер починає та завершує маршрут у фіксованій вершині v_0 (або різних для кожного комівояжера).

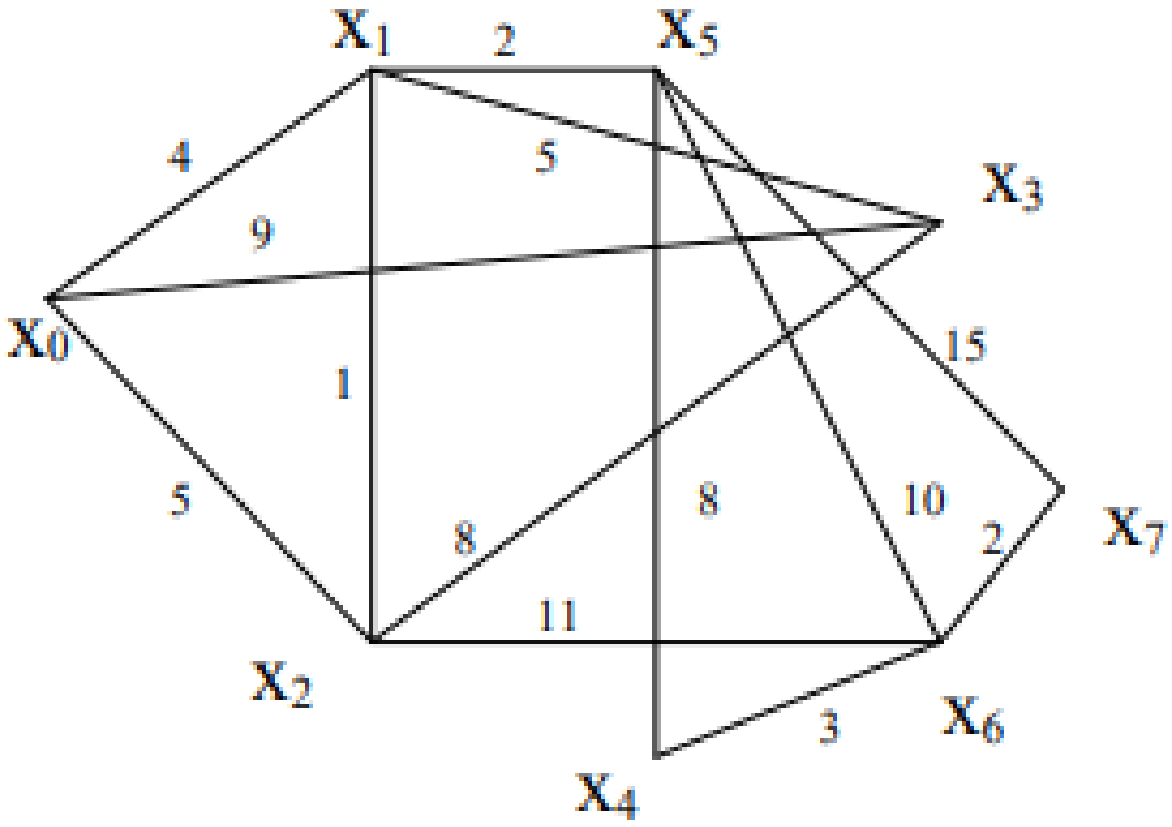

Рис.1.1. Представлення задачі комівояжера у теорії графів.

На рис.1.1 подано зображення, яке демонструє задачу комівояжера у теорії графів. На ньому показано повний зважений граф із вершинами, що представляють

міста, і ребрами з вагами, які позначають відстані або вартість. Найкоротший цикл виділено для кращої візуалізації.

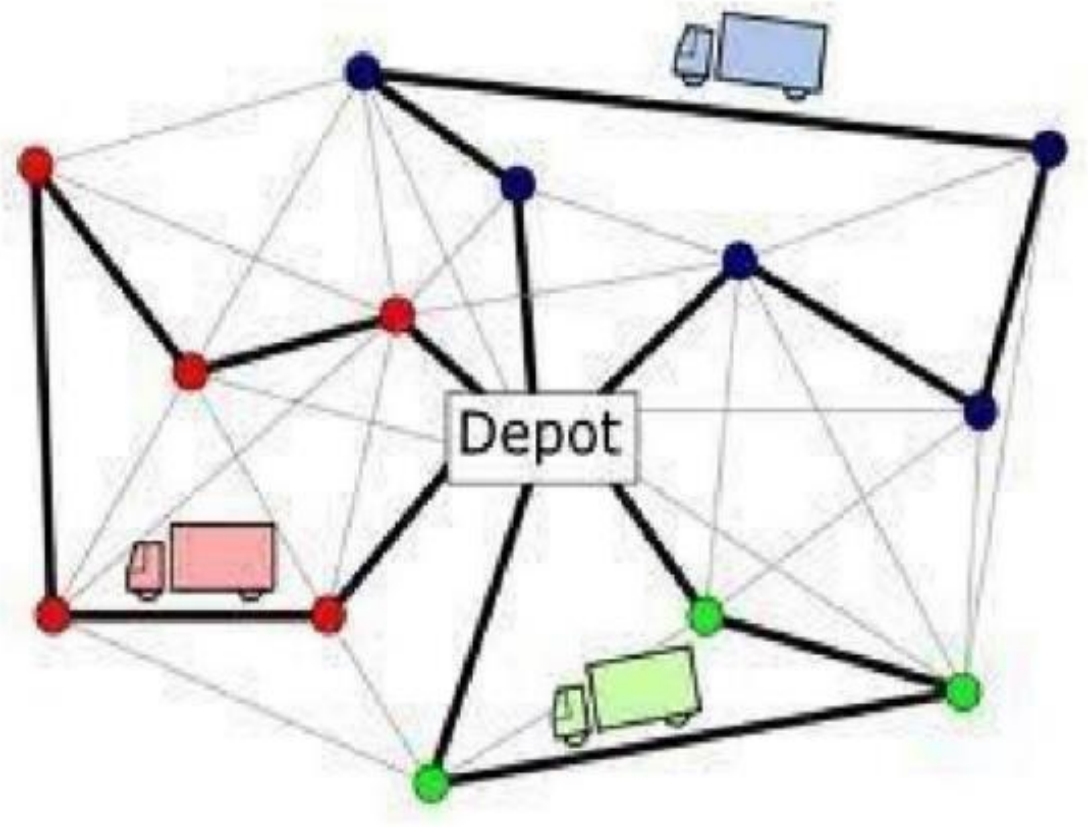

Рис.1.2. Представлення задачі k-комівояжерів у теорії графів.

На рис.1.2 подано, яке демонструє задачу багатьох комівояжерів (mTSP). Граф показує декілька стартових вузлів, що представляють різних комівояжерів. Їхні маршрути виділені різними кольорами, кожен з яких покриває свою частину міст, мінімізуючи загальну довжину шляху.

Обмеження:

1. Кожна вершина $i \in V \setminus \{v_0\}$ повинна бути відвідана рівно одним комівояжером.
2. Жодна вершина не може бути відвідана більше одного разу.

Кожен комівояжер може відвідати лише підмножину вершин V, що не перевищує його можливостей (за часом, ресурсами тощо, якщо такі обмеження задані).

Цільова функція:

Мінімізувати сумарну вартість (або іншу метрику), наприклад:

$$min \sum_{r=1}^{k} \sum_{(i.j)\in E} c(i.j) x_{i.j}^{T}$$

де: $x_{i.j}^{T} \in \{0,1\}$ бінарна змінна, яка дорівнює 1, якщо комівояжер r проходить ребро (i,j), і 0 в іншому випадку.

Математична модель:

1. Обмеження покриття вершин:

$$\sum_{r=1}^{k} \sum_{j\in V} x_{i.j}^{T} = 1, \qquad \forall i \in V \setminus \{v_0\}$$

Кожна вершина повинна бути відвідана рівно одним комівояжером.

2. Збереження маршруту: Для кожного комівояжера r:

$$\sum_{j\in V} x_{i.j}^{T} = \sum_{j\in V} x_{j.i}^{T}, \qquad \forall i \in V$$

Вхідні та вихідні маршрути для вершини і повинні збігатися.

3. Старт і фініш маршруту: Кожен комівояжер r починає та закінчує свій маршрут у початковій вершині v_0:

$$\sum_{j\in V} x_{v_0.j}^{r} = \sum_{j\in V} x_{j.v_0}^{r} = 1$$

4. Обмеження підциклів (для уникнення неповних маршрутів): Можуть бути застосовані умови використання допоміжних змінних u_i^r:

$$u_i^r - u_j^r + n \cdot x_{ij}^r \leq n-1, \quad \forall i,j \in V \setminus \{v_0\},\ r \in \{1,\ldots,k\}.$$

Вихідні дані:

1. Оптимальний розподіл маршрутів між k-комівояжерами.
2. Маршрути, що мінімізують цільову функцію.

Ця формалізація дозволяє використовувати методи оптимізації, такі як цілочислове програмування, для пошуку оптимального рішення.

1.2. Аналіз алгоритмів вирішення задачі комівояжера

Задача k-комівояжерів є узагальненням класичної задачі комівояжера (TSP, Traveling Salesman Problem), коли необхідно визначити оптимальні маршрути для k комівояжерів, які починають і закінчують свої маршрути у визначених точках (або в одній загальній точці). Завдання полягає у мінімізації загальної відстані або часу, що витрачається всіма комівояжерами. Для розв'язання цієї задачі існують різні підходи та алгоритми, які умовно можна розділити на точні методи, евристики та метаевристики.

Точні алгоритми

- Метод повного перебору (Brute Force) - перебираються всі можливі маршрути для k-комівояжерів, і вибирається оптимальний. Обчислювальна складність зростає експоненціально із зростанням кількості точок і комівояжерів. Застосовується лише для малих задач.
- Методи лінійного програмування використовуються для формулювання задачі як цілочислової задачі лінійного програмування (ILP). Найбільш поширені формулювання: модель Міллера-Така-Земліна (MTZ), модель Дантцига-Фулкерсона-Джонсона (DFJ). Навіть з сучасними оптимізаторами (як-от CPLEX, Gurobi) для великих задач обчислення є трудомістким.
- Динамічне програмування (алгоритм Беллмана-Хелда-Карпа) – це узагальнення класичного підходу до TSP на випадок k-комівояжерів, забезпечує точний результат, але має високу обчислювальну складність.

Евристичні алгоритми не гарантують оптимального результату, але забезпечують "достатньо хороші" рішення за прийнятний час.

- Жадібний алгоритм (Greedy Algorithm) полягає в тому, що кожен комівояжер вибирає найближчу невідвідану точку до поточної позиції. Переваги: швидкість виконання. Недоліки: висока ймовірність локально оптимальних, але глобально неоптимальних рішень.
- Алгоритм найближчого сусіда (Nearest Neighbor) - починаючи з випадкової точки, комівояжер послідовно відвідує найближчі точки, поки всі точки

не буде включено до маршруту. Придатний для швидкого отримання рішення, але має обмежену точність.

- Розподіл за кластерами (Clustering) - точки розбиваються на k груп (кластери) за допомогою алгоритмів кластеризації (наприклад, k-means). Для кожного кластера виконується окремий TSP. Недоліки: оптимальність рішення залежить від якості кластеризації.

Метаевристичні алгоритми імітують природні процеси та забезпечують високий рівень оптимальності навіть для великих задач.

- Генетичний алгоритм (GA) імітує процес природного відбору, генерує "популяцію" можливих рішень, виконує кросовери та мутації. Переваги: підходить для великих задач. Недоліки: потребує ретельного налаштування параметрів.
- Мурав'їний алгоритм (ACO, Ant Colony Optimization) імітує поведінку мурах, які залишають феромонний слід на оптимальних маршрутах. Переваги: добре працює з задачами розподілу. Недоліки: може застрягти в локальному мінімумі.
- Алгоритм імітації відпалу (Simulated Annealing) імітує процес охолодження металів. Рішення поступово покращується шляхом випадкових змін із певною ймовірністю прийняття погіршених рішень на початкових етапах. Переваги: простота реалізації. Недоліки: тривалий час пошуку для великих задач.
- Метод рою частинок (PSO, Particle Swarm Optimization) імітує колективну поведінку рою, кожна "частинка" шукає оптимум на основі свого досвіду та досвіду сусідів. Переваги: ефективність для неперервних задач. Недоліки: складність адаптації до дискретних задач.

Вибір підходу

1. Для малих задач ($n \leq 20$) доцільно використовувати точні алгоритми, як-от методи лінійного програмування.
2. Для середніх задач ($20 < n \leq 100$) ефективними є евристики або комбінація кластеризації з евристиками.
3. Для великих задач ($n > 100$) найкраще підходять метаевристичні алгоритми, такі як GA, ACO або PSO.

1.3. Аналіз наближених методів вирішення задачі комівояжера

Очевидно, що задача k комівояжерів тісно пов'язана із задачею комівояжера. У всіх методах на деякому етапі доведеться вирішувати задачу комівояжера, і у зв'язку з цим їх можна розділити на дві групи:

1. Спочатку вирішується задача комівояжера для всіх міст, а потім підбирається найкращий поділ результату для k комівояжерів.

2. Спочатку всі міста розбиваються на k груп, а потім задача комівояжера вирішується для кожної з них.

Як відомо, на практиці немає методу, який дозволяв би знаходити точне рішення задачі комівояжера за гарантований час для більш ніж 30-50 міст. Тому у разі, де потрібно буде вирішити задачу комівояжера, будемо використовувати наближені алгоритми.

Ітеративний алгоритм Ліна-Кернігана.

Суть алгоритму полягає у проведенні серії ітерацій [2]. Для початку можна взяти маршрут, побудований випадковим чином, чи отриманий в результаті роботи іншого алгоритму. Кожна ітерація намагається побудувати два списки ребер, один з яких міститиме елементи для видалення з вихідного маршруту, а інший – для додавання замість видалених. Зрозуміло, що сумарна довжина ребер для видалення повинна бути більшою за сумарну довжину ребер для додавання. Якщо в ході ітерації не вдалося знайти жодної пари таких ребер, робота алгоритму закінчується.

Розглянемо механізм побудови списків замін:

1. Нехай T – це маршрут, який складається із n міст. Множини X і Y - це ребра для видалення з маршруту та додавання до маршруту відповідно. Кожен елемент g[i] зберігає виграш, який можна отримати шляхом заміни ребра x[i] на y[i].

2. Нехай D* - це максимальний знайдений раніше виграш. Виберемо будь-яку вершину t[1] із T, і нехай x[1] – це будь-яке з двох ребер, що належать T, суміжних із t[1]. На початковому етапі і дорівнює 1.

3. Позначимо через t[2] вершину на протилежному кінці x[1], виберемо y[1] таким чином, щоб g[1]>0. Якщо такого y[1] немає, переходимо до пункту “6d”. (Це перше застосування обмеження обов'язкового отримання виграшу).

4. Нехай i=i+1. Виберемо x[i], яке зараз з'єднує t[2i-1] з t[2i] та y[i], як буде показано далі:

a) x[i] вибирається таким чином, що з'єднання t[2i] з t[1] ребром y[i] дозволяє закінчити побудову списків x та y.

Це застосування обмеження завершення побудови списків заміни. Воно гарантує, що ми можемо завжди завершити маршрут до циклу, якщо захочемо, простим з'єднанням t[2i] з t[1] для будь-якого i >=2. Вибір y[i-1] на кроці “4e” гарантуватиме, що існує такий x[i]. Іншими словами, при видаленні з вихідного маршруту T елементів множини X та заміна їх елементами з множини Y повинна давати замкнутий маршрут.

b) y[i] – це деяке ребро, зв’язане з вершиною t[2i]. Порядок його вибору буде розглянуто у пунктах "c", "d", "e". Якщо y[i] немає, то переходимо крок 5.

Вочевидь, що для отримання максимального виграшу на i-тому кроці, y[i] вибираємо максимально коротким.

c) При виборі y[i] необхідно враховувати, що елементи x[k1] і y[k2] для будь-яких k1і k2 при k1 не рівному k2 не повинні стикатися, і y[i] не може бути ребром, що належить множині X.

d) Позначимо через D[i] суму всіх елементів g від 1 до i. Ця величина необхідна для визначення критерію виграшу.

e) Вибір елемента y[i] має дозволяти розрив елемента x[i+1]. Це необхідно для виконання обмеження завершення в пункті “4a” .

f) Перед вибором y[i] ми повинні перевірити, що з'єднання t[2i] з t[1] давало б приріст загального виграшу. Нехай y*[i] - це ребро, що зв'язує t[2i] з t[1], і нехай g*[i]=|y*[i]|-|x[i]|. Якщо D[i-1]+g*[i]>D*, встановимо D*=D[i-1]+g*[i]. D* – це максимальний виграш, знайдений раніше. D* >=0 і монотонно не спадає.

5. Завершуємо побудову множин x[i] та y[i] у кроках “2”-”4”, коли більше немає x[i] та y[i], що підходять під вимоги “4c”-”4e”, або D[i] <= D*.

Це наш критерій завершення списків замін. Зробимо в маршруті T заміну ребер x на ребра y і повторимо цей алгоритм з пункту "2".

6. Якщо D*=0, виконується бектрекінг, як показано далі:

a) Повторимо крок “4”-”5”, вибираючи y[2] як збільшення довжини до того часу, поки g[1]+g[2]>0. Якщо покращення знайдено, переходимо на крок “2”.

b) Якщо всі варіанти y[2] на кроці “4b” були перевірені і не дали виграшу, то повертаємось на крок “4a” та пробуємо інші варіанти для x[2].

c) Якщо і це не дало виграшу, повертаємося на крок “3”, де перевіряємо інші y[1] по мірі збільшення їх довжини.

d) Якщо всі варіанти y[1] не дають виграшу, ми перевіряємо альтернативний x[1] на кроці “2”.

e) Якщо і ця спроба закінчується невдачею, то вибирається новий t[1], і ми повторюємо крок “2”. Зазначимо, що бектрекінг виконується ТІЛЬКИ тоді, коли виграшу не вдається досягти, і ТІЛЬКИ лише на рівні 1 і 2 (i=1, i=2).

7. Процедура закінчується, коли n значень t[1] були випробувані без виграшу.

Слід зазначити, що розгляд альтернативного ребра x[2] є некоректним до роботи алгоритму. Це вводиться для підвищення якості рішення та організації бектрекінгу. Роботу алгоритму під час використання альтернативного x[2] необхідно реалізовувати окремо з розглядом окремих випадків на рівнях “2”, “3”. Для всіх наступних рівнів можна використовувати нормальну реалізацію алгоритму.

Алгоритм дерева.

Цей алгоритм використовує як початкову структуру мінімальний кістяк. Його можна побудувати відомими алгоритмами Пріма чи Крускала. Ребра кістяка подвоюються, і на отриманому графі знаходиться цикл Ейлера. Оскільки кожне ребро подвоєне, то такий цикл завжди буде існувати. В отриманому результаті залишається тільки видалити вершини, що повторюються. Трудомісткість роботи алгоритму $O(n^2)$.

Алгоритм 2опт

На початковому етапі роботи алгоритму 2опт шлях будується випадковим чином. Далі для кожної можливої пари міст цього шляху перевіряється, чи не виявиться шлях між ними коротшим, якщо проїхати його у зворотному напрямку. Якщо перевірка дає позитивний результат, то маршрут між парою повертається. Зазначимо, що перевірку можна здійснити за одиничний час. Той факт, що цей алгоритм не будує маршрут заново, а може використовувати шлях, побудований раніше, дозволяє застосовувати його для подальшої оптимізації знайденого рішення. Причому час роботи, як видно з результатів тестування, залежить від якості початкового шляху. Час роботи алгоритму O (n^2).

Локальна оптимізація

Локальна оптимізація раніше отриманого рішення реалізується повним перебором обмеженої послідовності міст з метою знайти варіант із мінімальною відстанню між ними [3,4]. Алгоритм перебору включає механізм відсікання безперспективних гілок пошуку, який повністю ідентичний механізму відсікання в алгоритмі для знаходження точного вирішення завдання комівояжера. Сама послідовність називається вікном оптимізації. Таку оптимізацію необхідно провести навколо всіх міст, що увійшли до рішення. Важливо правильно визначити розмір вікна. Великий його розмір призводить до швидкого зростання часу роботи. Практично виявляється зручним використовувати оптимізацію з вікном до 9 міст.

Приступимо до розгляду алгоритмів, що становлять основний об'єкт дослідження у цій роботі.

Алгоритм розрізання "загального" маршруту

Алгоритм розрізання "загального" маршруту є одним із найпростіших варіантів рішення ЗкК. Його суть полягає у знаходженні оптимального розрізання маршруту, попередньо побудованого на множині всіх міст шляхом рішення ЗК. Це можна зробити так:

1. Вирішити ЗК.

2. Знайти кількість міст для кожного комівояжера таким чином, щоб для будь-якої пари комівояжерів довжини маршрутів не відрізнялися б більш ніж на одиницю. Нехай N[i] - це кількість міст для об'їзду і-тим комівояжером.

3. У попередньо побудованому маршруті вибрати деяке місто як початкове.

4. Містами для об'їзду першим комівояжером будуть перші N[1] міст, починаючи з обраного початкового. Наступний відрізок маршруту з N[2] міст буде ділянкою для 2-го комівояжера і т.д. Внаслідок цього етапу ми отримаємо k незамкнутих послідовностей потрібної довжини.

5. До кожної такої послідовності додамо базу та замкнемо її.

6. Запам'ятаймо отриману довжину всіх маршрутів і повторимо операцію з кроку 4, але початкове місто змістимо на один щодо попереднього.

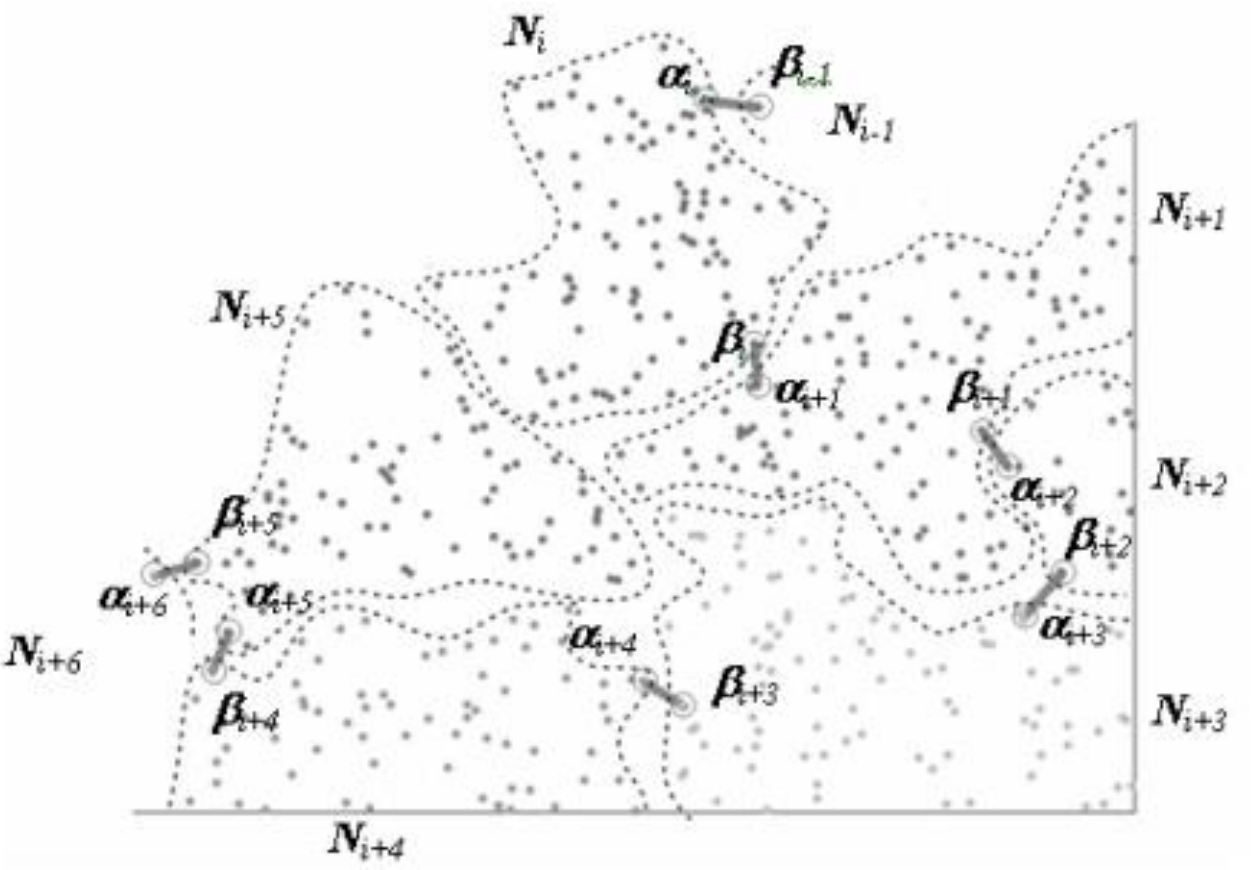

Рис.1.3. Алгоритм розрізання "загального" маршруту.

7. Крок 6 будемо повторювати Nmin раз, де Nmin - це число на один менше, ніж довжина найкоротшого шляху серед комівояжерів, отриманих на кроці 2. Трудомісткість такого пошуку буде Nmin * N, крім рішення ЗК, де N - це загальна кількість міст.

На рис.1.3 показано, як алгоритм розрізання покращує маршрут для вирішення задачі комівояжера. Вихідний маршрут проходить через усі вершини в

довільному порядку. Далі ілюструються місця розрізів (позначені пунктирними лініями) та етапи їх з'єднання для оптимізації шляху, що зменшує загальну довжину.

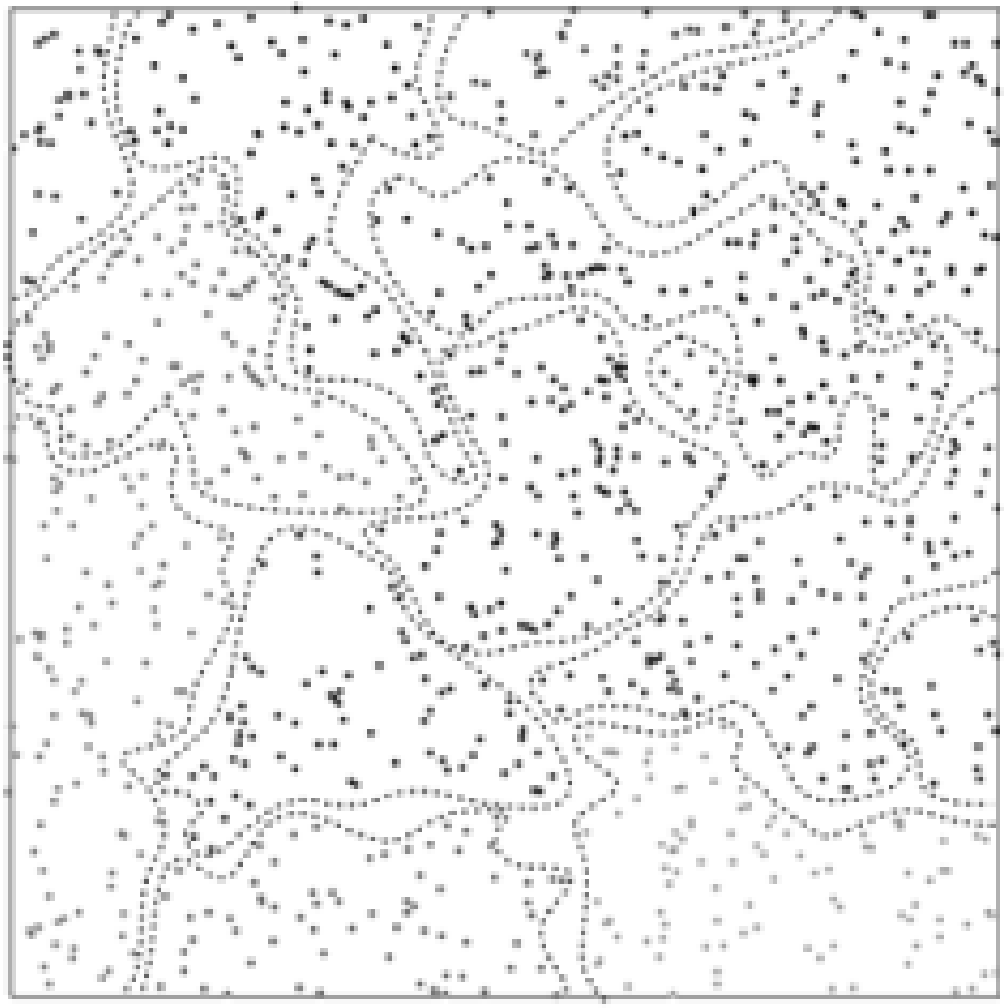

Рис.1.4. Алгоритм розрізання маршруту для задачі багатьох комівояжерів.

На рис.1.4 подано графічне зображення алгоритму розрізання для задачі багатьох комівояжерів. Воно ілюструє початкові маршрути для кількох комівояжерів, дії алгоритму з розподілу вузлів через розрізання та покращення, що веде до оптимальних маршрутів.

Метод попереднього поділу міст на групи для кожного комівояжера

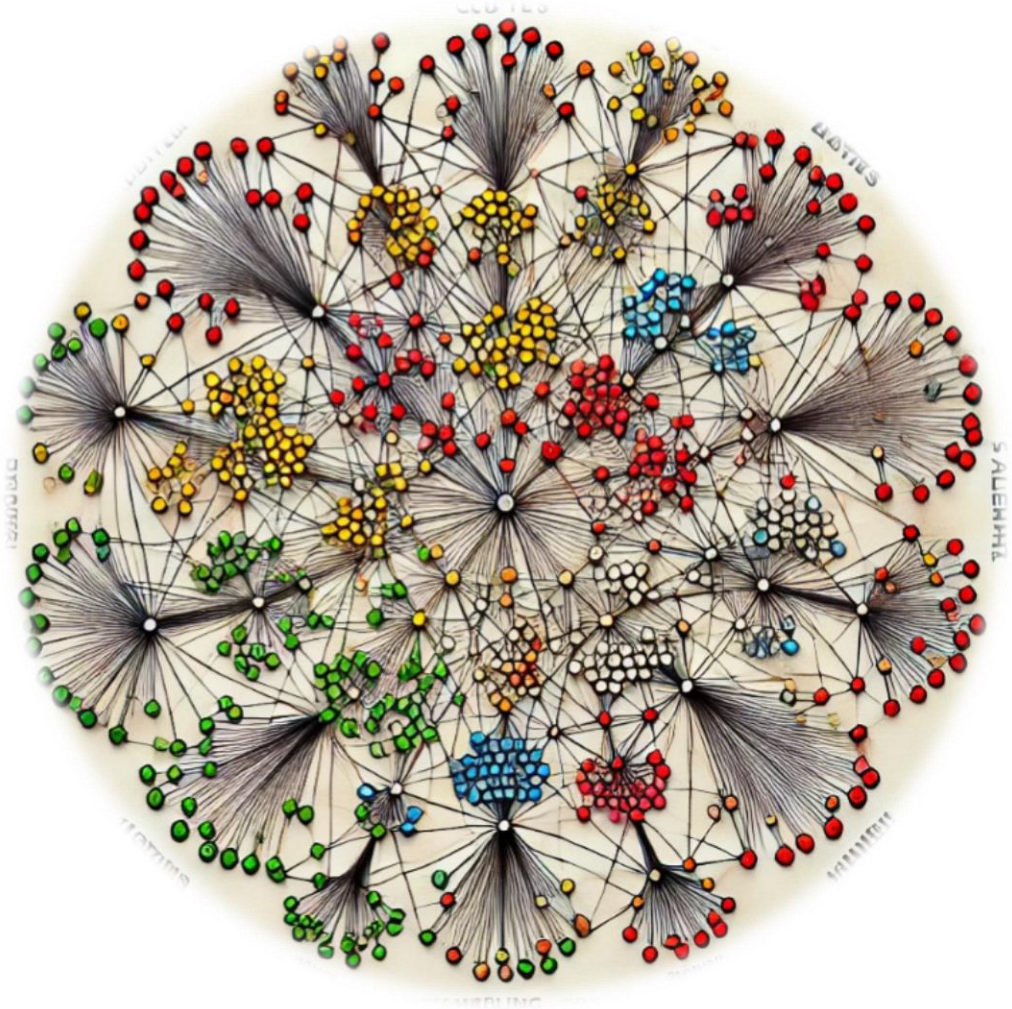

Рис.1.5. Графічне зображення методу попереднього поділу міст на групи для кожного комівояжера.

Цей підхід представляє собою ціле сімейство алгоритмів. Тут будуть описані як алгоритми готові до використання, так і деякі тенденції, що дають простір для подальшого дослідження. Як зазначалося раніше, в даному випадку ЗК вирішуватиметься для кожного комівояжера окремо, і головне завдання полягає в попередньому поділі міст на групи. В основі розглянутих методів лежить припущення, що результат буде тим краще, чим сильніше будуть "відокремлені" групи кожного комівояжера.

На рис.1.5 подано графічне зображення методу попереднього поділу міст на групи для кожного комівояжера. Міста розподілені на кластери, кожен із яких позначений своїм кольором і відповідає маршруту окремого комівояжера. Це дозволяє ефективно організувати розв'язання задачі.

Як показало тестування, алгоритм дихотомічного поділу є одним із найефективніших методів і можна виконати оптимізацію його результату. Розподіл міст на групи відбувається так:

1. Обчислюємо кількість міст для кожного комівояжера, як це описувалося раніше. Позначимо N[i] довжину маршруту для i-того комівояжера;

2. Поділимо міста на дві групи, виконуючи такі дії:

2.1. Насамперед потрібно визначитися, скільки міст буде в кожній з них. Міста, що підлягають поділу, повинні об'їхати k комівояжерів, тобто множина N[i] містить k елементів.

Визначимо порожнє множину M[i] і додаватимемо до неї елементи з N[i] до того часу, поки сума всіх M[i] не перевищить суму всіх N[i], що залишилися. Отримані суми елементів N[i] і M[i] будуть шуканими розмірами груп.

2.2. Серед міст вибирається пара, яка буде початковим вмістом майбутніх груп. Відстань між цією парою має бути якнайбільше.

2.3. Оскільки кількість міст у групах швидше за все не співпадатиме, то спочатку в групу, де міст має бути більше, виконаємо стільки додавань, скільки потрібно для вирівнювання. Існує 2 способи вибору чергового міста для додавання. Методи вибору будуть описані нижче.

2.4. Міста, що залишилися, будемо додавати по черзі в різні групи за тим же принципом.

3. У разі потреби, кожну групу піддамо подальшому поділу.

4. Алгоритм закінчить свою роботу, коли будуть отримані k груп.

Способи вибору чергового міста для додавання до групи міст

Випробовано два методи вибору міста для додавання до групи міст. В обох випадках з кожним не доданим містом пов'язані два міста у різних групах з мінімальними відстанями до нього. На початку роботи алгоритму в кожній групі знаходиться по одному місту, а решта міст зв'язуються з ними. У міру додавання міст ці зв'язки оновлюються, так як для не доданих міст нове місто у групі може виявитися ближчим, ніж місто до додавання. Відмінність полягає у критерії вибору міста для додавання. У першому випадку обирається місто з мінімальною

відстанню до міст, що знаходяться в групі для додавання. У другому випадку для кожного міста обчислюється різниця відстаней до міст, з якими місто, що додається, пов'язане в різних групах. Вибирається місто з мінімальною різницею відстаней.

Експеримент показав, що варіант з обчисленням різниці відстаней значно покращує якість алгоритму, не погіршуючи час його роботи.

При обчисленні різниці відстаней обмінна оптимізація, яка буде розглянута далі, дає лише незначне поліпшення.

Обмінна оптимізація поділу на дві групи

Пропонується оптимізація описаного вище методу, що дає помітне покращення кінцевого результату для невеликої кількості міст. Виконавши черговий поділ на дві групи, проведемо наступну перевірку:

1. Розглянемо всі можливі пари міст, де перше місто належатиме першій, а друге - другій групі.

2. Для кожної такої пари здійснимо пробний обмін містами, тобто перше місто потрапить до другої групи, а друге - до першої.

3. Виконавши обмін, підрахуємо суму довжин ребер мінімального кістяка в кожній групі, додавши довжину найдовшого ребра в ньому. Така оцінка буде використовуватися при тестуванні та дозволить знайти деяку нижню межу можливого рішення ЗК у цій групі.

4. Знайдемо оцінку з кроку 3 для всіх можливих пар. Якщо мінімальне значення такої оцінки для деякої пари менше, ніж у вихідних групах, здійснимо обмін. Очевидно, що ця оптимізація породжує багаторазове обчислення мінімальних кістяків, тому її можна використовувати тільки для невеликої кількості міст. Загалом сам алгоритм поділу на дві групи має квадратичну трудомісткість щодо кількості міст для поділу.

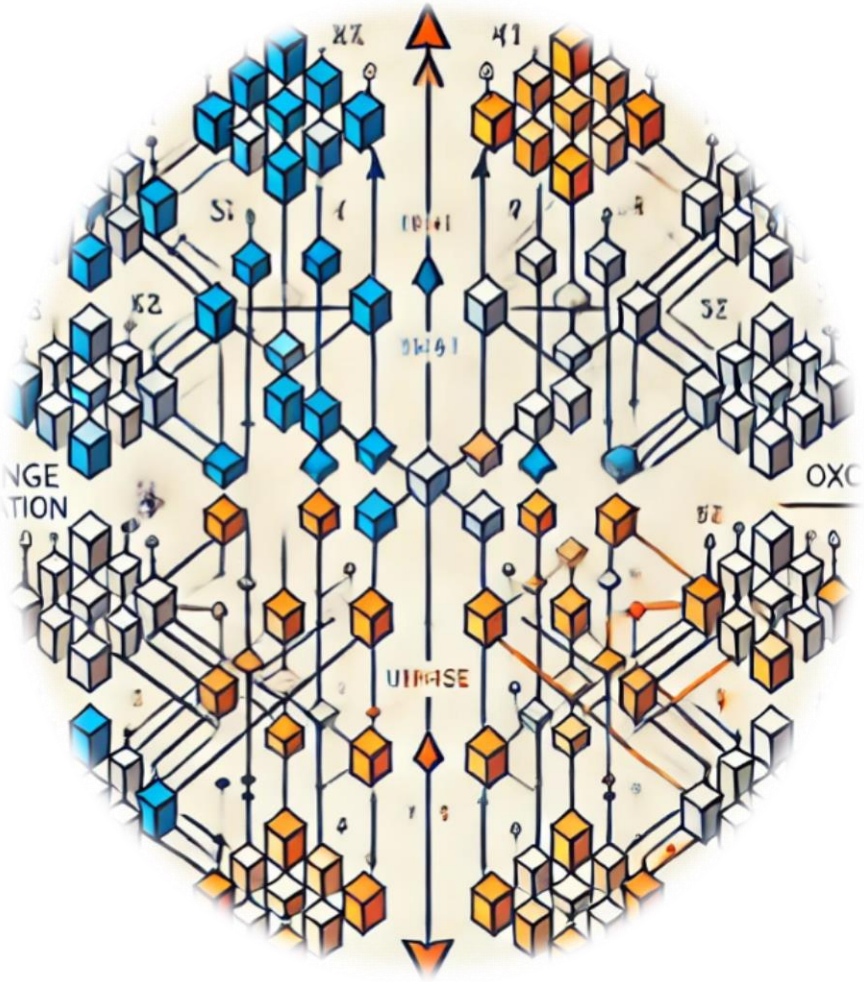

Рис.1.6. Графічне зображення, яке демонструє обмінну оптимізацію поділу міст на дві групи.

Для зменшення часу роботи такої оптимізації можна обмежити множину міст для перестановок у такий спосіб:

1. Знайдемо пару найближчих міст, де перше місто належатиме першій, а друге - другій групі.

2. У кожній групі знайдемо певну фіксовану кількість міст, найближчих до міст обраної пари.

3. Проводитимемо обмінну оптимізацію лише серед міст із множин, отриманих на попередньому кроці.

Таке обмеження множин для перебору погіршує загалом ефективність оптимізації, але й суттєво уповільнює зростання часу її роботи.

На рис.1.6 подано зображення, яке демонструє обмінну оптимізацію поділу міст на дві групи. Початкові групи відображені різними кольорами (наприклад, синім і

помаранчевим). Стрілки показують, як міста обмінюються між групами для досягнення кращого балансу або оптимізації.

Алгоритм поділу на 3 групи

Збільшення числа груп при розподілі не становить особливого інтересу, оскільки дає результати гірше, ніж при розподілі на дві. Проведено експериментальне тестування з розподілом на три групи, де було отримано погіршення якості роботи. Як і у варіанті з двома групами, спочатку знаходили трійку максимально віддалених один від одного міст, обчислювали кількість міст у кожній групі, а потім виконували додавання за таким принципом, що і при двох групах.

Кутове сортування міст на площині

Поділ за допомогою кутового сортування міст на площині є винятком серед методів, що розглядаються. Цей алгоритм використовує додаткову інформацію, не передбачену описаним завданням. У цих обчисленнях використовуються дані про кутах між містами і віссю X відносно бази у випадках їх розташування на площині. Така інформація може бути доступною при реальному застосуванні алгоритмів на практиці, і її можна використовувати для покращення результатів.

Алгоритм полягає в наступному:

1. До кожного міста обчислюються кути з-поміж них і віссю X за умови, що початок координат перебуває у базі.

2. Множина міст сортується за зростанням кутів, отриманих на попередньому етапі.

3. Обчислюється кількість міст кожної майбутньої групи. За результатами обчислень отримуємо масив Ni, де зберігаються знайдені значення.

4. Перші N1 міст із відсортованого списку віднесемо до першої групи, наступні N2 міст - до другої тощо, до кожної групи додамо по базі.

5. Для кожної отриманої групи обчислюється мінімальна оцінка можливого рішення шляхом знаходження мінімального кістяка.

6. Повторюємо дії на кроці 4, але при побудові відступаємо на одне місто та знаходимо оцінку, описану на кроці 5.

7. Такі обчислення проводимо Nmin разів, де Nmin - число на одиницю менше, ніж мінімум серед елементів масиву N.

8. Вибирається те розбиття, де мінімальна оцінка і виконується пошук остаточних маршрутів рішенням простої ЗК всередині кожної групи.

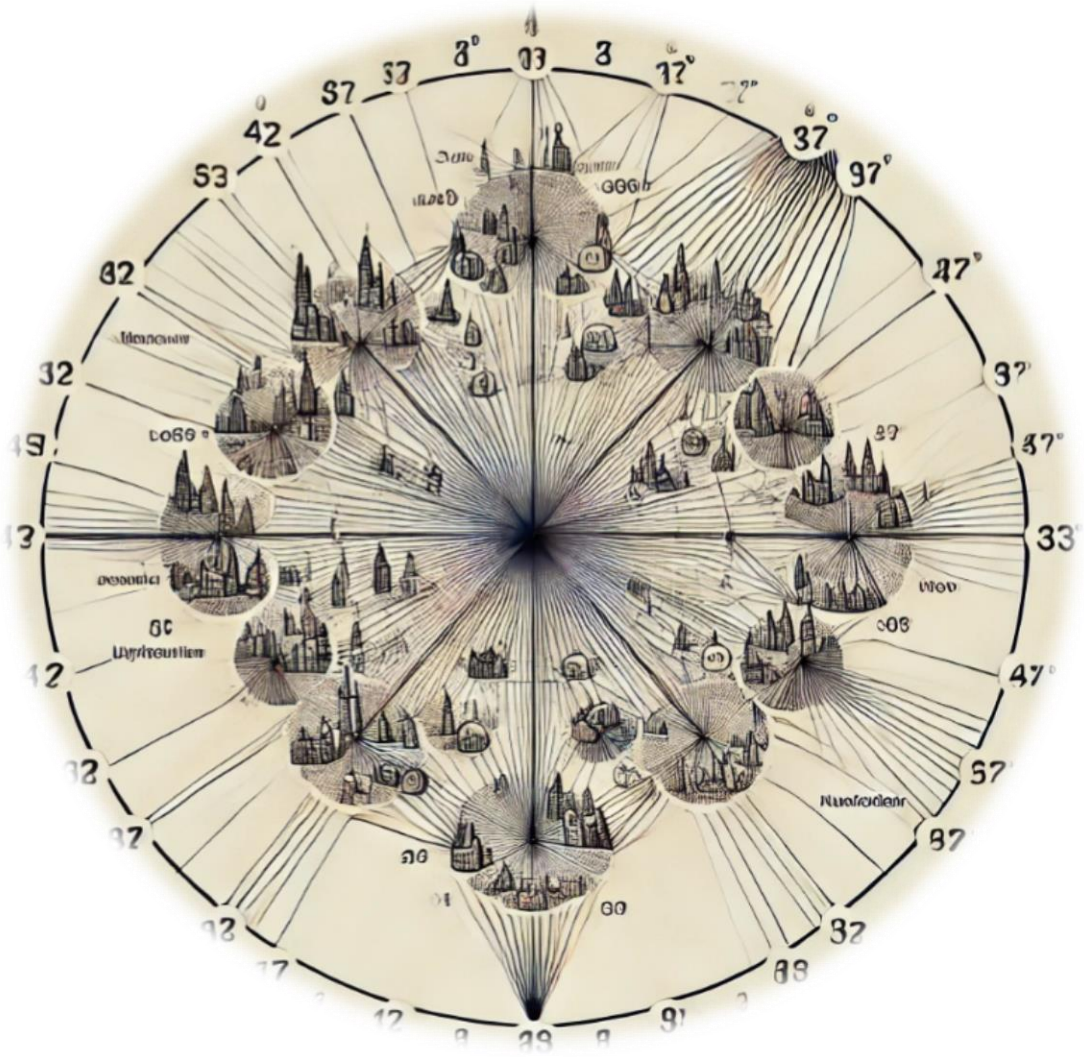

Рис.1.7. Графічне зображення кутового сортування міст на площині.

На рис.1.7 подано графічне зображення кутового сортування міст на площині. Вузли, що представляють міста, з'єднані лініями з центральною точкою, яка використовується як орієнтир. Міста впорядковані за їхнім кутовим положенням, а кольорові дуги або номери демонструють порядок.

1.4. Завдання кваліфікаційної роботи

Задача k-комівояжерів є узагальненням класичної задачі комівояжера, коли необхідно визначити оптимальні маршрути для k комівояжерів, які починають і закінчують свої маршрути у визначених точках (або в одній загальній точці). Завдання полягає у мінімізації загальної відстані або часу, що витрачається всіма комівояжерами. Для цього слід вирішити у роботі такі завдання:

➢ виконати формальну постановку задачі **k**-комівояжерів (З**k**К);

➢ дослідити аналіз алгоритмів вирішення задачі комівояжера;

➢ реалізувати програмний пакет вирішення задачі комівояжера, фундаментом для розробки пакету є: стандартна бібліотека C++, бібліотека функції рядкових перетворень Iconv, пакет для читання XML-файлів Xerces-C, бібліотека графічного виводу libgd2 та деякі функції операційної системи для роботи з файлами та вимірювання часу;

➢провести обчислювальний експеримент для порівняння можливих методів вирішення ЗkК та виявлення тенденцій для продовження досліджень;

➢ розробити заходи стосовно охорони праці та безпеки у надзвичайних ситуаціях;

➢ визначити економічну ефективність від розроблених програмних модулів.

РОЗДІЛ 2.
ОПИС РЕАЛІЗАЦІЇ АЛГОРИТМІВ ВИРІШЕННЯ ЗАВДАНЬ КОМІВОЯЖЕРА

2.1. Опис пакету TSP

До реалізації пред'являлися такі вимоги:

- алгоритми мають виконуватися ефективно;

- алгоритми не повинні бути прив'язані до конкретного типу вершин та просторів;

- реалізація має бути переносима з однієї операційної системи до іншої;

- реалізація повинна коректно обробляти неправильні дані та інші помилки, такі як нестача оперативної пам'яті та ін.

Ці вимоги необхідні використання описаних алгоритмів у реальних умовах практично.

У майбутньому передбачається, що реалізовані алгоритми повинні мати різні можливості для завантаження заздалегідь заготовлених даних, збереження результатів та перегляду проміжних структур обробки. Нині ці механізми реалізовані частково. Рекомендується підбирати формати таким чином, щоб при необхідності було легко організовувати взаємодію з іншими системами, наприклад, можна скористатися мовою розмітки XML. Її розробники передбачили всі деталі, пов'язані з різницею у поданні даних різних обчислювальних архітектурах, а функції її обробки є майже у всіх операційних системах.

Пакет TSP розроблено мовою C++. Структура вихідних кодів будується за принципом "листкового пиріжка". Пакет містить у собі кілька розділів, кожен із яких має своє призначення та середовище, де йому можна виконуватись. Під час розробки постійно відстежується можливість перенесення продукту з однієї платформи в іншу. В даний час пакет запускається в середовищі GNU/Linux під час використання компілятора gcc-4.1.0. Всі виклики операційної системи ізольовані

спеціальним шаром, тому для перенесення пакета, наприклад, Win32, потрібно створити окрему реалізацію такого шару, що містить в собі типові функції.

Усі строкові операції всередині пакету виконуються в UNICODE (UTF-32), що дозволяє уникнути проблем із обробкою національних символів. При збереженні завдань на диску використовується спеціальний формат, що базується на XML. Для ілюстрації представлення даних у файлі розглянемо кілька прикладів.

Приклад збереженого завдання одного комівояжера:

```
<?xml version='1.0' encoding='UTF-8'?>
<tsp type="simple">
     <node>
          <x>-429</x>
          <y>-369</y>
     </node>
     <node>
          <x>-234</x>
          <y>198</y>
      </node>
      <node>
           <x>646</x>
           <y>980</y>
      </node>
      <node>
           <x>860</x>
           <y>892</y>
      </node>
 </tsp>
```

Приклад вирішеної та збереженої задачі кількох комівояжерів:

```
<?xml version='1.0' encoding='UTF-8'?>
<tsp type="ksimple">
     <base>
          <x>0</x>
          <y>0</y>
     </base>
     <node>
          <x>-429</x>
          <y>-369</y>
     </node>
     <node>
          <x>-234</x>
          <y>198</y>
     </node>
     <node>
          <x>646</x>
          <y>980</y>
     </node>
     <node>
          <x>860</x>
          <y>892</y>
     </node>
     <node>
          <x>946</x>
          <y>27</y>
     </node>
```

```
        <tour>
            <item>0</item>
            <item>1</item>
            <item>base</item>
        </tour>
        <tour>
            <item>4</item>
            <item>3</item>
            <item>2</item>
            <item>base</item>
        </tour>
</tsp>
```

Усі об'єкти розробки поміщені в окремі простори імен мови C++. Це перешкоджає дублюванню ідентифікаторів. Проект має механізм автоматичної генерації документації на основі аналізу вихідних текстів за допомогою пакету Doxygen. Помилки часу виконання обробляються за допомогою генерації та перехоплення винятків.

Фундаментом для розробки пакету є:

- стандартна бібліотека C++;
- бібліотека функції рядкових перетворень Iconv;
- пакет для читання XML-файлів Xerces-C;
- бібліотека графічного виводу libgd2

та деякі функції операційної системи для роботи з файлами та вимірювання часу.

Для уникнення проблем з відмінністю реалізації стандартних класів мови C++ у проекті використовується бібліотека STLport, яка найповніше представляє стандарт мови у частині бібліотечних класів і здатна запускатися на кількох платформах. Ця бібліотека статично зв'язується з кодом проекту, що дозволяє уникнути зайвих залежностей у файлах, що запускаються.

Читання файлів XML здійснюється за допомогою бібліотеки Xerces-C проекту Apache. Вона реалізує SAX-інтерфейс та дозволяє прикладним програмам отримувати вміст XML-файлів у зручній формі. Функції для читання таких файлів у програмі ізольовані, тому за потреби можна безболісно змінити Xerces-C будь-яку іншу бібліотеку.

Пакет libgd2 використовується для створення графічних файлів у форматі PNG. API цієї бібліотеки являє собою набір функцій для малювання різних

графічних примітивів і збереження результату на диск. У бібліотеці TSP широко використовуються покажчики з підрахунком посилань. Цей механізм дозволяє організувати контроль за звільненням динамічної пам'яті та не допускає її витоків. Реалізація таких покажчиків береться із бібліотеки boost.

2.2. Архітектура пакету

Пакет TSP містить такі розділи:

- бібліотека Alant;
- бібліотека Arch;
- бібліотека Utils;
- бібліотека Tasks;
- бібліотека Programs.

Ці розділи розміщені в однойменних директоріях всередині дерева вихідних текстів.

Взаємозв'язки між розділами зображені рисунку 2.1.

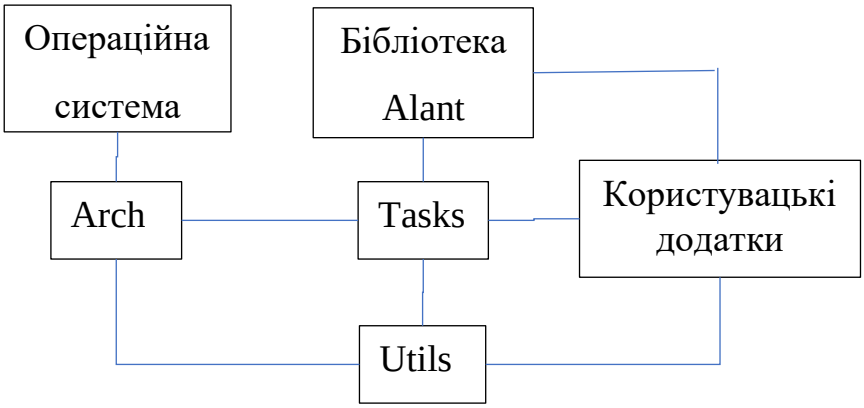

Рис.2.1. Взаємозв'язки між розділами пакету TSP.

Бібліотека Alant містить у собі реалізацію всіх досліджуваних алгоритмів. У ній не допускається використання жодних функцій та класів, крім власних чи стандартних класів мови C++. В основі цієї бібліотеки лежать реалізації кількох примітивів, таких як вершина (двовимірний варіант та індексований варіант),

орієнтоване та неорієнтоване ребро та простір. Ребра є шаблонами C++, для їх використання потрібно конкретизувати типом вершини, які вони з'єднують.

Об'єкт простору служить для отримання метрик. Всі алгоритми, які будь-яким чином обробляють відстані, повинні завжди користуватися ними для отримання метрик, тому що способи вимірювання відстаней можуть відрізнятися.

Примітиви мають різний набір операторів та методів, який залежить від їхньої природи. У загальному випадку гарантується лише коректна операція порівняння на рівність.

Поверх цих примітивів бібліотека Alant реалізує класи, що представляють графи та замкнуті маршрути. Обидва ці класи необхідно конкретизувати типом вершини, які вони містять. Об'єкт графа коректно обробляє дублювання ребер та незв’язність. Ребра зберігаються у вигляді пов'язаних списків, асоційованих з вершинами. Списки зберігаються в хеш-таблицях (клас std::hash_map). Це дозволяє швидко виконувати пошук ребер навколо певної вершини.

Для графа реалізовано два допоміжні класи - Alant::LinkWalking і Alant::LinkWalking1, що дозволяють здійснити обхід всіх ребер, враховуючи чи не враховуючи зв'язність графів.

Клас замкнутого маршруту своєю поведінкою нагадує масив std::vector. Крім того, простір індексів у нього необмежений, що й дозволяє досягти ефекту замкнутості.

Клас TourBuilder дозволяє виконувати операцію побудови замкнутого маршруту на основі списку ребер, що до нього входять.

Інший допоміжний клас, який часто використовується в бібліотеці Alant::FNum. Цей клас повторює поведінка числа з плаваючою точкою, але його операції порівняння виконуються з урахуванням деякого допуску. Він служить підвищення обчислювальної стійкості.

Серед іншого в бібліотеці Alant є реалізації загальних алгоритмів, що використовуються у рішенні ЗК і ЗкК, таких як сортування, знаходження ейлерового шляху, алгоритм Прима і т.д.

Поверх описаних класів реалізовані самі алгоритми розв'язання ЗК та ЗкК. Всі класи бібліотеки Alant знаходяться в окремому просторі імен. Детальна інформація про бібліотеку Alant є у додатковій HTML-документації.

У складі шару Arch реалізовані класи та функції для ізолювання операцій з операційною системою для досягнення переносимості. Серед них п'ять функцій, що виконують перетворення рядків (IO2String(), String2IO(), makeUTF8(), decodeUTF8() і decodeUTF8_()), клас TSP::Timer(), що виконує вимірювання інтервалів часу, і функція readTextFile(), що виконує читання текстового файлу. Виняток TSP::SystemException() також повинен бути уніфікований для всіх операційних систем. Це дозволяє писати програмні обробники помилок взаємодії з операційною системою єдиним чином.

У розділ Utils входять різноманітні допоміжні програми, серед яких класи:

TSP::String - клас, який організовує роботу з рядками;

TSP::Statistics - клас для обчислення різних статистичних оцінок за накопиченими даними тестування;

TSP::CmdArgsParser - клас для обробки параметрів командного рядка;

TSP::HTMLWriter - клас для організації збереження даних у HTML-файлі;

TSP::XMLWriter - клас для організації збереження у XML-файл.

Клас TSP::String успадковується від std::wstring і містить у собі додаткові функції для форматування часу, обробки чисел тощо.

Розділ Tasks містить у собі оболонку для використання алгоритмів у шарі Alant, зі спрощенням їх форми виклику і розширенням даних користувача.

Ключову роль тут грають інтерфейси TSP::Processable і TSP::KProcessable. Ці інтерфейси є уніфікацією різних об'єктів, які можна обробляти алгоритмами рішення ЗК і ЗкК. Завдяки їм немає необхідності робити різну реалізацію викликів алгоритмів окремо для проведення тестування та окремо для обробки конкретних даних користувача.

Інтерфейс TSP::KProcessable шлях кожного комівояжера представляє у вигляді TSP::Processable, що дає можливість легко обробити або покращити окремо знайдене рішення одного комівояжера. Крім базових визначень, розділ Tasks

містить реалізацію середовища тестування, виведення результатів у HTML-файл, побудова графічних уявлень тощо. На самому верху структури пакета TSP знаходиться шар додатків з вихідними текстами файлів, що запускаються. Усі вони будуються за однією схемою:

- ініціалізація пакета;
- обробка параметрів командного рядка;
- виконання корисних дій за допомогою розділів Tasks та Alant.

2.3. Пояснення користувачу пакету

Серед інструментального набору пакета TSP є утиліти, що виконують розв'язання задач для одного і декількох комівояжерів, що організовують порівняльне тестування різних методів розв'язання та оптимізації, створення графічних ілюстрацій рішень, що показують інформацію про завдання та виконують генерацію випадкових наборів даних для подальшого аналізу методів розв'язання ЗК та ЗkК.

Всі утиліти запускаються з командного рядка та призначені для роботи в середовищі GNU/Linux.

Опис порядку проведення тестування міститься у файлі, складеному за спеціальними правилами.

Набори даних для обробки та отримані рішення зберігаються у файлах спеціального формату з розширенням *.tsp. Усередині такого файлу міститься й інформація про набір міст та про розв'язання задачі, якщо воно було знайдено раніше.

Завдання бувають наступних типів:

• "SIMPLE" - завдання для одного комівояжера з містами, визначеними як двовимірні координати на площині;

• "KSIMPLE" - те саме, що і "SIMPLE", тільки для k комівояжерів.

Координати в таких завданнях можуть бути цілими. Відстань між містами обчислюється як евклідова відстань. Передбачається додати типи "MATRIX" і

"KMATRIX", де існуватиме матриця відстаней, і кожне місто описуватиметься індексом для такої матриці. Утиліта tsp-draw здатна обробляти лише завдання "SIMPLE" і "KSIMPLE", оскільки дані цих завдань можна зображати на площині.

2.4. Утиліта тестування

Утиліта tsp-testing виконує порівняльний аналіз алгоритмів рішення ЗК та ЗкК. Результати тестування виводяться на екран і, за бажанням користувача, зберігаються в HTML-файлі. Утиліта приймає такі параметри командного рядка:

• --help - показати довідку;

• --html - вказує на необхідність виведення результатів у html-файл;

• --ktsp - переключити утиліту в режим обробки ЗкК.

Після перерахування всіх ключів у командному рядку необхідно вказати ім'я файлу з описом тестів.

Вхідний файл обробляється послідовно від початку до кінця. Регістр літер та кількість символів пробілу не мають значення. Будь-який рядок можна вільно розбивати на кілька рядків. Рядки, що починаються із символу "#", є коментарями та в програмі ігноруються. Сприймаються лише літери англійського алфавіту та цифри. Розділові знаки та інші символи пропускаються.

Доступні такі команди:

• "ENABLE" – включити проведення оптимізації до всіх наступних тестів. Тип оптимізації вказується словом, що йде за цією командою. В даний час можна використовувати "LOCALOPT" - локальна оптимізація з вікном 9 і "2OPT" - наступний п'ятиразовий запуск алгоритму 2Opt. Порядок включення оптимізації значення не має. Локальна оптимізація завжди проводиться в останню чергу;

• "DISABLE" - команда аналогічна до попередньої, але виконує відключення оптимізацій у наступних тестах;

• "RESET" – виконує ініціалізацію середовища тестування та скидання всіх встановлених параметрів, крім прапорів проведення оптимізації. Список алгоритмів для тестування очищується. Вхідний файл повинен починатися із цієї

команди; "NODES" - встановлює кількість міст у тестах у значення, наступне за цією командою. Під час обробки команди "RESET" цей параметр встановлюється на 100.

• "TESTS" – встановлює кількість тестів для деякого набору алгоритмів у значення, наступне за цією командою Команда "RESET" скидає цей параметр в 10;

• "START" - встановлює початкове значення датчика випадкових чисел для отримання змісту тесту. Значення за промовчанням не визначено.

• "SALESMEN" - команда, яка встановлює кількість комівояжерів у значення, що йде за цією командою. Значення за промовчанням - 2.

Визначено набір команд, які додають певний алгоритм до списку для тестування. Ці команди:

• «LINKERN» - додає ітеративний алгоритм Ліна-Кернігана;

• «2OPT»- додає алгоритм 2опт;

• «BRANCHES»- одає метод гілок та границь;

• «DIVIDE» - додає стратегію "Поділяй і володарюй";

• «TOWN» - додає алгоритм найближчого міста;

• «BEST»- додає алгоритм знаходження точного рішення;

• «TREE» - додає алгоритм дерева;

• «MODTREE» - додає модифікований алгоритм дерева;

• «POST_SPLITTING» додає алгоритм із розрізанням готового маршруту;

• «DIVIDING» - додає алгоритм попереднього поділу міст на дві групи.

Команда "RUN" - запускає тест із встановленими перед цими параметрами.

Після завершення обробки цієї команди результати будуть відображені на екрані та, при необхідності, збережені у HTML-файлі.

Міста при тестуванні розташовуються у одиничному квадраті. База комівояжер завжди знаходиться на початку координат.

Результати тестування виводяться щодо мінімальної оцінки можливого рішення. Для будь-якого досліджуваного алгоритму програма видає середнє значення по всіх наборах даних, мінімальне та максимальне отримані та середньоквадратичне відхилення.

Крім якості алгоритмів утиліта видає результати вимірювання часу роботи алгоритмів з точністю більш ніж 0,001 сек. Всі повідомлення утиліти та рядки для оформлення HTML-файлу виводяться, їхній текст зберігається у зовнішньому файлі в кодуванні UTF-8. Можлива їхня зміна. Приклад вхідного файлу для ЗК.

```
# TSP testing example file.
ENABLE LOCALOPT
ENABLE 2OPT

RESET town linkern
     NODES 200
     TESTS 100
     START 1
RUN
# End of file.
```

Цей приклад порівнює алгоритм найближчого міста та ітеративний алгоритм Ліна-Кернігана на задачах у 200 міст 100 разів із включеними оптимізаціями.

Приклад вхідного файлу для ЗКК.

```
# TSP testing example file.

RESEt POST DIVIDING
     NODES 512
     TESTS 50
     START 1
     SALESMEn 4
RUN
# END OF FILE.
```

Цей файл порівнює наступне розрізання готового маршруту з розподілом міст на дві групи для 512 міст 50 разів. Поділ проводиться на 4 комівояжери.

2.5. Утиліта рішення ЗК та ЗкК

Утиліта tsp виконує рішення ЗК або ЗкК. Якість обробки визначається числом від нуля до десяти. При якості "0" обробка не здійснюється, а за "10" завжди виконується пошук точного рішення. Утиліта сама вибирає відповідний алгоритм та набір оптимізації на основі зазначеної якості та попередніх установок у тексті програми.

Користуватися якістю "10" можна лише для завдань із кількістю міст менше 30, інакше пошук рішення займає багато часу. Користувач має вказати ім'я вхідного файлу для обробки.

Утиліта автоматично визначає тип завдання і після виконання пошуку рішення збереже результат у тому самому файлі. За бажання його можна обробити ще раз з іншим зазначенням якості. Якщо файл вже містив розв'язання задачі, то обов'язкове поліпшення при подальших обробках не гарантується, так як багато алгоритмів будують рішення завжди наново.

Утиліта приймає такі параметри:

• "--quality" - встановлює бажану якість обробки;

• "--salesmen" – кількість комівояжерів, для яких потрібно знайти рішення, якщо завдання це має на увазі;

• "--help" - показати коротку довідку та інформацію про програму.

2.6. Утиліта створення графічних зображень ЗК та ЗкК

Tsp-draw – утиліта для створення графічних зображень, що показують результат обробки завдання. При відображенні шляху одного комівояжера на малюнок виводиться ламана, що з'єднує міста у потрібному порядку. Під час обробки завдання k комівояжерів база відображається червоною точкою, і з неї виходять шляхи всіх маршрутів.

Утиліта обробляє лише завдання типу "SIMPLE" та "KSIMPLE", т.е. до. Завдання інших типів неможливо зобразити на площині. Команда tsp-draw обробляє всі аргументи як імена вхідних файлів для зображення.

ип вхідних файлів визначається автоматично. Єдиний ключ, що сприймається програмою - "-help", який показує коротку інформацію про програму.

2.7. Інформаційна утиліта

Утиліта tsp-info є довідковою програмою, що відображає інформацію про завдання. Як аргумент для неї має бути вказано ім'я вхідного файлу. Утиліта самостійно визначить тип завдання та покаже інформацію про неї.

На екран виводиться така інформація:

- тип задачі;
- кількість міст у ній;
- загальна довжина шляху;
- нижня оцінка довжини можливого рішення;
- відношення поточної довжини маршруту до нижньої оцінки.

Якщо вхідний файл містить ЗкК, то також відображається кількість комівояжерів, для яких було знайдено рішення. Утиліта не вносить жодних змін у вхідний файл.

2.8. Утиліта генерації випадкових наборів даних

Утиліта tsp-gen виконує генерацію випадкового заповнення завдання. Це буває корисним, якщо необхідно провести експеримент для порівняння результатів роботи різних алгоритмів або оцінити час обробки завдання певної розмірності за заданої якості.

Для запуску цієї утиліти необхідно вказати тип завдання, що генерується (ключ "--ktsp", якщо це ЗкК), і кількість міст у новій задачі (ключ "--nodes"). При використанні ключа "-help" програма виводить коротку довідку. Останнім параметром під час виклику цієї програми має бути вказане ім'я вихідного файлу для збереження результату.

РОЗДІЛ 3.
РЕЗУЛЬТАТИ РОЗРОБКИ І ДОСЛІДЖЕННЯ АЛГОРИТМІВ ВИРІШЕННЯ ЗАВДАНЬ КОМІВОЯЖЕРА

3.1. Тестування алгоритмів вирішення завдань комівояжера

При тестуванні описаних методів виконувався багаторазовий запуск на випадково згенерованій множині міст. При генерації міста розташовувалися на площині в одиничному квадраті

Параметри тестування: Міст: 32; Комівояжерів: 2 ;

Тестів: 20 ; Початкове значення давача випадкових чисел: 7.

Таблиця 3.1

Результати тестування - значення функції мети

Алгоритм	Середнє	Середнє квадратичне відхилення	Мінімум	Максимум
Розрізання готового маршруту	1.58628	0.131008	1.3202	1.8771
Поділ по мінімуму	1.45338	0.0735101	1.33838	1.63047
Поділ по мінімуму (частковий обмін)	1.43961	0.0903995	1.29183	1.5724
Поділ по різниці	1.31379	0.0667745	1.19966	1.44576
Поділ по різниці (частковий обмін)	1.32542	0.0705355	1.19966	1.47539
Кутове сортування на площині	1.31385	0.066744	1.19966	1.45863

Таблиця 3.2

Результати тестування – час роботи

Алгоритм	Середнє	Середнє квадратичне відхилення	Мінімум	Максимум
Розрізання готового маршруту	0.017s	0.0170745	0.008s	0.039s
Поділ по мінімуму	0.015s	0.0154034	0.008s	0.028s
Поділ по мінімуму (частковий обмін)	0.052s	0.0528689	0.025s	0.116s
Поділ по різниці	0.013s	0.0133521	0.004s	0.029s
Поділ по різниці (частковий обмін)	0.049s	0.0494894	0.041s	0.074s
Кутове сортування на площині	0.016s	0.0163923	0.010s	0.025s

Такі є тестування було проведено для 4, 8, 16, 32, 64 комівояжерів та 32, 64, 128, 256, 512 та 1024 міст. На основі наведених даних можна проаналізувати результати розв’язання задачі комівояжера за різними алгоритмами. Подамо характеристику кожного методу за середньою ефективністю (значення функції мети) та часом роботи.

Розрізання готового маршруту: середнє значення - 1.58628 (найгірше серед усіх алгоритмів), середнє квадратичне відхилення - 0.131008 (велика варіативність результатів), час роботи - 0.017s (достатньо швидкий). Отже, алгоритм демонструє низьку ефективність і значну варіативність у результатах, хоча працює швидко.

Поділ по мінімуму: середнє значення - 1.45338 (краще, ніж розрізання маршруту), середнє квадратичне відхилення - 0.0735101 (більш стабільний), час роботи - 0.015s (швидкий). Отже, алгоритм дає кращі результати при низькому розкиді значень, що робить його ефективним і стабільним.

Поділ по мінімуму (частковий обмін): середнє значення - 1.43961 (краще, ніж базовий поділ по мінімуму), середнє квадратичне відхилення - 0.0903995 (трохи більший розкид, ніж у базового варіанту), час роботи - 0.052s (суттєво довше через додаткову операцію обміну). Отже, частковий обмін покращує якість рішень, але збільшує час роботи.

Поділ по різниці: середнє значення - 1.31379 (один із найкращих результатів), середнє квадратичне відхилення - 0.0667745 (дуже стабільний результат), час роботи - 0.013s (найшвидший серед усіх алгоритмів). Цей метод оптимальний за швидкістю та стабільністю результатів, що робить його одним із найкращих.

Поділ по різниці (частковий обмін): середнє значення - 1.32542 (трохи гірше, ніж базовий поділ по різниці), середнє квадратичне відхилення - 0.0705355 (розкид дещо більший), час роботи - 0.049s (повільніше через додаткову оптимізацію). Додавання часткового обміну дещо знижує стабільність і швидкість, але залишається ефективним.

Кутове сортування на площині: середнє значення - 1.31385 (результати на рівні поділу по різниці), середнє квадратичне відхилення - 0.066744 (дуже стабільний

результат), час роботи - 0.016s (трохи довше за базовий поділ по різниці). Алгоритм демонструє високу ефективність і стабільність, трохи поступаючись за часом найшвидшому варіанту.

Найкращі результати за середньою ефективністю: поділ по різниці (1.31379), кутове сортування на площині (1.31385).

Найшвидші алгоритми: поділ по різниці (0.013s), поділ по мінімуму (0.015s).

Алгоритми з найменшим розкидом результатів: поділ по різниці (0.0667745), кутове сортування на площині (0.066744).

Якщо потрібно враховувати як час роботи, так і ефективність, поділ по різниці є найкращим вибором. Для трохи більш складних сценаріїв кутове сортування є гідною альтернативою.

Результати наведено щодо нижньої оцінки точного рішення ЗК. Така оцінка обчислюється так:

1. Знаходиться мінімальний кістяк на повному графі, де вершини - це міста;

2. Підсумовується довжина всіх ребер мінімального кістяка;

3. До отриманої суми додається довжина найдовшого ребра цього кістяка.

Обмінна оптимізація проводилася тільки в частковому варіанті, оскільки багаторазове знаходження мінімального кістяка займає дуже багато часу.

Для вирішення простої ЗК застосовувалися послідовно алгоритм дерева, п'ятикратний запуск алгоритму 2опт і локальна оптимізація з вікном 9. Нижче наводяться зведені таблиці результатів тестування, де вказані середні значення якості обробки даних.

Таблиця 3.3

Алгоритм "розрізання готового маршруту"

Кількість міст	Число комівояжерів					
	2	4	8	16	32	64
32	1.58628	1.9754				
64	1.51489	1.79884	2.35193			
128	1.43254	1.63314	2.00798	2.77975		
256	1.38185	1.55467	1.83676	2.37177	3.48998	
512	1.33268	1.43715	1.64154	2.02108	2.80185	4.422
1024	1.31734	1.38206	1.51007	1.78692	2.33914	3.4615

Таблиця 3.4

Алгоритм "розподіл по різниці"

Кількість міст	Число комівояжерів					
	2	4	8	16	32	64
32	1.31379	1.59561				
64	1.28894	1.46649	1.93719			
128	1.2827	1.38236	1.69811	2.43513		
256	1.26398	1.32999	1.55063	2.05577	3.1428	
512	1.25287	1.28909	1.4444	1.78944	2.53635	4.11105
1024	1.24614	1.26619	1.36908	1.6167	2.13135	3.2194

Таблиця 3.5

Алгоритм "кутове сортування на площині"

Кількість міст	Число комівояжерів					
	2	4	8	16	32	64
32	1.31385	1.55664				
64	1.26437	1.40651	1.91918			
128	1.27502	1.34158	1.64633	2.50031		
256	1.26514	1.29434	1.47622	2.00292	3.37846	
512	1.24698	1.27611	1.36557	1.6627	2.59221	4.67948
1024	1.24465	1.26043	1.31751	1.48185	2.02404	3.50509

Таблиця 3.6

Алгоритм "розподіл мінімуму"

Кількість міст	Число комівояжерів					
	2	4	8	16	32	64
32	1.45338	1.81827				
64	1.39847	1.66598	2.19016			
128	1.34896	1.55229	1.9262	2.70399		
256	1.30677	1.46804	1.7709	2.34364	3.52239	
512	1.294	1.43086	1.67106	2.08204	2.91055	4.55168
1024	1.30013	1.39287	1.57881	1.91382	2.51509	3.67634

Таблиця 3.7

Алгоритм "розподіл мінімуму (частковий обмін)"

Кількість міст	Число комівояжерів					
	2	4	8	16	32	64
32	1.43961	1.79239				
64	1.38417	1.62723	2.15368			
128	1.34798	1.53341	1.91572	2.67497		
256	1.30478	1.46775	1.76577	2.31655	3.42333	
512	1.29003	1.41732	1.64759	2.06323	2.86557	4.4724
1024	1.30064	1.39836	1.56983	1.87618	2.46064	3.57944

3.2. Ілюстрація результатів роботи алгоритму та їх опрацювання

Для ілюстрації результати роботи алгоритму наведемо також кілька малюнків, де схематично зображені послідовності міст, які потрібно об'їхати

кожному комівояжеру. На кожному малюнку зображені результати роботи різних алгоритмів на одному й тому ж завданні з 64 випадково розташованих міст. Під час підготовки малюнків проста ЗК вирішувалася ітеративним алгоритмом Ліна-Кернігана.

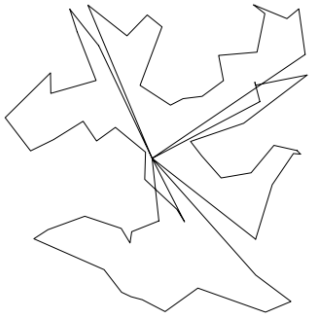

Рис. 3.1. Алгоритм "розрізання готового маршруту"

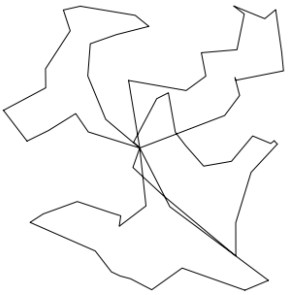

Рис.3.2 Алгоритм "розподіл по різниці"

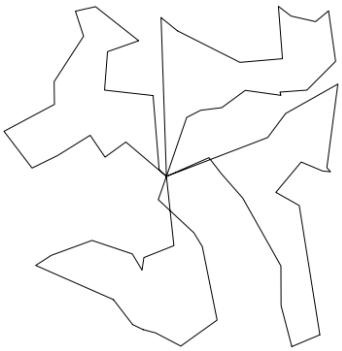

Рис. 3.3. Алгоритм "кутове сортування на площині"

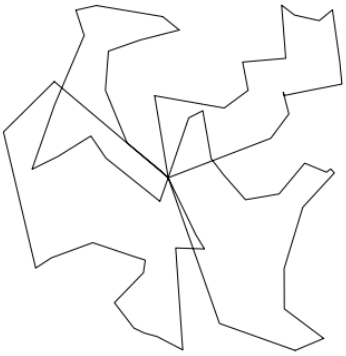

Рис.3.4. Алгоритм "розподіл мінімуму"

Для ухвалення рішення про те, який алгоритм є кращим чи гіршим у тій чи іншій ситуації скористаємося статистичним критерієм Вілкоксона [1], який дозволяє прийняти або відкинути гіпотези про порівняння якості роботи алгоритмів з деяким рівнем значущості.

Наведемо таблиці до різних наборів даних й у кожній таблиці попарно порівнюються різні алгоритми. У кожній клітинці записується рівень значущості, з яким можна стверджувати, що один алгоритм кращий за інший. Якщо значення наведено з негативним знаком, то вважається, що алгоритм, вказаний у рядку, є кращим, ніж алгоритм, вказаний у стовпці, і навпаки.

Таблиця 3.8

Результати тестування: параметри тестування: міст: 32; комівояжерів: 2; тестів: 20; початкове значення давача випадкових чисел: 7.

Алгоритм	Розрізання готового маршруту	Кутове сортування на площині	Поділ по мінімуму	Поділ по різниці
Розрізання готового маршруту	-	0.01	0.01	0.01
Кутове сортування на площині	- 0.01	-	- 0.01	приймається гіпотеза про рівність
Поділ по мінімуму	- 0.01	0.01	-	0.01
Поділ по різниці	- 0.01	приймається гіпотеза про рівність	- 0.01	-

Усі значення різниці між алгоритмами коливаються в межах -0.01 **і** 0.01**,** що вказує на незначні відмінності між їх результатами. Для деяких пар алгоритмів приймається гіпотеза про рівність результатів, що означає відсутність статистично значущої різниці між ними.

Розрізання готового маршруту має незначну перевагу або відставання (0.01 або -0.01) у порівнянні з іншими алгоритмами. Стабільність цього алгоритму є середньою, але він поступається іншим за ефективністю.

Кутове сортування на площині - приймається гіпотеза про рівність з поділом по різниці, що свідчить про їх подібну ефективність. У порівнянні з іншими методами результати дуже близькі, що підтверджується різницею у -0.01 або 0.01.

Поділ по мінімуму демонструє незначну перевагу (0.01) або відставання (-0.01) від інших методів, що вказує на близькість результатів. Є стабільним варіантом, але не показує явного лідерства.

Поділ по різниці - один із найбільш конкурентоспроможних методів. Результати практично ідентичні кутовому сортуванню, що підтверджується прийняттям гіпотези про рівність.

Значуща різниця між алгоритмами відсутня для більшості пар. У випадках, коли відмінності присутні, вони дуже незначні (-0.01 або 0.01).

Алгоритми з найкращими результатами - поділ по різниці та кутове сортування на площині демонструють подібну ефективність, що підтверджено гіпотезою про рівність результатів. Оскільки відмінності мінімальні, вибір алгоритму можна робити на основі інших критеріїв, таких як швидкість виконання або зручність реалізації.

Таблиця 3.9

Результати тестування: параметри тестування: міст: 32; комівояжерів: 4; тестів: 20; початкове значення давача випадкових чисел: 7.

Алгоритм	Розрізання готового маршруту	Кутове сортування на площині	Поділ по мінімуму	Поділ по різниці
Розрізання готового маршруту	-	0.01	0.01	0.01
Кутове сортування на площині	- 0.01	-	- 0.01	- 0.01
Поділ по мінімуму	- 0.01	0.01	-	0.01
Поділ по різниці	- 0.01	0.01	- 0.01	-

Таблиця 3.10

Результати тестування: параметри тестування: міст: 64; комівояжерів: 2; тестів: 20; початкове значення давача випадкових чисел: 7.

Алгоритм	Розрізання готового маршруту	Кутове сортування на площині	Поділ по мінімуму	Поділ по різниці
Розрізання готового маршруту	-	0.01	0.01	0.01
Кутове сортування на площині	- 0.01	-	- 0.01	- 0.05
Поділ по мінімуму	- 0.01	0.01	-	0.01
Поділ по різниці	- 0.01	0.05	- 0.01	-

Таблиця 3.11

Результати тестування: параметри тестування: міст: 64; комівояжерів: 4; тестів: 20; початкове значення давача випадкових чисел: 7.

Алгоритм	Розрізання готового маршруту	Кутове сортування на площині	Поділ по мінімуму	Поділ по різниці
Розрізання готового маршруту	-	0.01	0.01	0.01
Кутове сортування на площині	- 0.01	-	- 0.01	- 0.01
Поділ по мінімуму	- 0.01	0.01	-	0.01
Поділ по різниці	- 0.01	0.01	- 0.01	-

Порівняння виконувалося так:

1. Зберігалися значення двох вибірок x1:xn1 та y1:yn2, кожна з яких містить відповідно n1 та n2 елементів.

2. Елементи цих вибірок поєднувалися в одну впорядковану об'єднану вибірку.

3. Усередині об'єднаної вибірки кожному елементу призначається ранг, який відповідає його порядковому номеру. Обчислюються значення r1 та r2, рівні сумі рангів елементів, що належать першій та другій початковим вибіркам.

4. Обчислюється статистика u та ũ за формулами:

$$u = R_1 - \frac{n_1(n_1+1)}{2}$$

$$\tilde{u} = R_2 - \frac{n_2(n_2+1)}{2}$$

Знайдені значення порівнювали з критичними значеннями, знайденими за таблицями, що надаються в описі критерію Вілкоксона.

Якщо u менше, ніж критичне значення, то приймається гіпотеза про те, що перший алгоритм працює краще за другий. Якщо ũ менше, ніж критичне значення, то навпаки. Якщо і u, і ũ більше чи рівні критичного значення, то приймається гіпотеза про рівність алгоритмів.

Таблиці критерію Вілкоксона складені для різного рівня значущості, що дозволяє визначити його при порівнянні алгоритмів.

РОЗДІЛ 4.
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА У НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Аналіз небезпечних та шкідливих виробничих чинників під час роботи з комп’ютерною технікою

Однією із характерних особливостей сучасного розвитку суспільства є зростання сфер діяльності людини, в яких використовуються інформаційні технології. Широке розповсюдження отримали персональні комп'ютери. Однак їх використання загострило проблеми збереження власного та суспільного здоров'я, вимагає удосконалення існуючих та розробки нових підходів до організації робочих місць, проведення профілактичних заходів для запобігання розвитку негативних наслідків впливу ПК на здоров'я користувачів.

Заходи з охорони праці користувачів ПК необхідно розглядати в трьох основних аспектах:

1. соціальному;
2. психологічному;
3. та медичному.

У соціальному плані розв'язання цих проблем пов'язане з оптимізацією умов життя, праці, відпочинку, харчування, побуту, розвитком культури, транспорту.

Психологічний аспект. Значне місце у профілактиці розладів здоров'я належить психології праці. Тому заходи, пов'язані з формуванням раціональних виробничих колективів, у яких відсутня психологічна несумісність, сприяють зменшенню нервово-психічного перенапруження, підвищенню працездатності та ефективності праці. Особливої значущості у користувачів відеодисплейних терміналів (ВДТ) набуває психоемоційний стрес, який більшою або меншою мірою проявляється у кожного з них.

Медичний аспект. Значна роль у профілактиці захворювань користувачів ПК відводиться медицині. Існує перелік профілактичних заходів для користувачів ПК, що включає як складові первинної профілактики здоров'я (професійний відбір), так

і вторинної, яка направлена на зниження ймовірності розвитку перевтоми та перенапруження. Ці комплексні заходи спрямовані на відновлення функціонального стану зорового та опорно-рухового апарату.

Гігієнічні вимоги до організації та обладнання робочих місць з ПК Обладнання і організація робочого місця з ВДТ мають забезпечувати відповідність конструкції всіх елементів робочого місця та їх взаємного розташування ергономічним вимогам з урахуванням характеру і особливостей трудової діяльності. Конструкція робочого місця користувача ВДТ має забезпечити підтримання оптимальної робочої пози. Робочі місця з ВДТ слід так розташовувати відносно світлових прорізів, щоб природне світло падало збоку, переважно зліва. При розміщенні робочих столів з ВДТ слід дотримуватись таких відстаней: між бічними поверхнями ВДТ - 1,2 м; від тильної поверхні одного ВДТ до екрана іншого - 2,5 м. Екран ВДТ має розташовуватися на оптимальній відстані від очей користувача, що становить 600...700 мм, але не ближче ніж за 600 мм з урахуванням розміру літерно-цифрових знаків і символів. Розташування екрана ВДТ має забезпечувати зручність зорового спостереження у вертикальній площині під кутом +30° до нормальної лінії погляду працюючого. Клавіатуру слід розташовувати на поверхні столу на відстані 100...300 мм від краю, звернутого до працюючого. У конструкції клавіатури має передбачатися опорний пристрій (виготовлений із матеріалу з високим коефіцієнтом тертя, що перешкоджає мимовільному її зсуву), який дає змогу змінювати кут нахилу поверхні клавіатури у межах 5...15°.

Вимоги до режимів праці і відпочинку при роботі з ПК При організації праці, пов'язаної з використанням ЕОМ і ПЕОМ, для збереження здоров'я працюючих, запобігання професійним захворюванням і підтримки працездатності передбачаються внутрішньозмінні регламентовані перерви для відпочинку. Внутрішньозмінні режими праці і відпочинку містять додаткові нетривалі перерви в періоди, що передують появі об'єктивних і суб'єктивних ознак стомлення і зниження працездатності.

Впродовж робочої зміни мають передбачатися:

- перерви для відпочинку і вживання їжі (обідні перерви);

- перерви для відпочинку і особистих потреб (згідно з трудовими нормами);

- додаткові перерви, що вводяться для окремих професій з урахуванням особливостей трудової діяльності.

За характером трудової діяльності розрізняють три професійні групи, згідно з діючим класифікатором професій (ДК-003-95 і Зміна N1 до ДК-003- 95):1):

1. розробники програм (інженери-програмісти) виконують роботу з ЕОМ та документацією при необхідності інтенсивного обміну інформацією з ЕОМ і високою частотою прийняття рішень. Робота характеризується інтенсивною розумовою творчою працею з підвищеним напруженням зору, концентрацією уваги на фоні нервово-емоційного напруження, вимушеною робочою позою, загальною гіподинамією, періодичним навантаженням на кисті верхніх кінцівок. Робота виконується в режимі діалогу з ЕОМ у вільному темпі з періодичним пошуком помилок в умовах дефіциту часу;

2. оператори електронно-обчислювальних машин виконують роботу, пов'язану з обліком інформації, одержаної з ВДТ за попереднім запитом, або тієї, що надходить з нього, супроводжується перервами різної тривалості, пов'язана з виконанням іншої роботи і характеризується напруженням зору, невеликими фізичними зусиллями, нервовим напруженням середнього ступеня та виконується у вільному темпі;

3. оператор комп'ютерного набору виконує одноманітні за характером роботи з документацією та клавіатурою і нечастими нетривалими переключеннями погляду на екран дисплея, з введенням даних з високою швидкістю. Робота характеризується як фізична праця з підвищеним навантаженням на кисті верхніх кінцівок на фоні загальної гіподинамії з напруженням зору (фіксація зору переважно на документи), нервово-емоційним напруженням.

Правилами встановлюються такі внутрішньо змінні режими праці та відпочинку при роботі з ЕОМ при 8-годинній денній робочій зміні в залежності від характеру праці:

- для розробників програм із застосуванням ЕОМ слід призначати регламентовану перерву для відпочинку тривалістю 15 хвилин через кожну годину роботи за ВДТ;

- для операторів із застосуванням ЕОМ слід призначати регламентовані перерви для відпочинку тривалістю 15 хвилин через кожні дві години;

- для операторів комп'ютерного набору слід призначати регламентовані перерви для відпочинку тривалістю 10 хвилин після кожної години роботи за ВДТ.

У всіх випадках, коли виробничі обставини не дозволяють застосувати регламентовані перерви, тривалість безперервної роботи з ВДТ не повинна перевищувати 4 години. При 12-годинній робочій зміні регламентовані перерви повинні встановлюватися в перші 8 годин роботи аналогічно перервам при 8-годинній робочій зміні, а протягом останніх 4-х годин роботи, незалежно від характеру трудової діяльності, через кожну годину тривалістю 15 хвилин. Для зниження нервово-емоційного напруження, стомлення зорового аналізатора, поліпшення мозкового кровообігу, подолання несприятливих наслідків гіподинамії, запобігання втомі доцільно деякі перерви використовувати для виконання комплексу вправ, які наведені у Державних санітарних правилах і нормах роботи з ПК електронно-обчислювальних машин ДСанПІН 3.3.2.007-98.

Психофізіологічне розвантаження. При проведенні сеансів психофізіологічного розвантаження рекомендується використовувати деякі елементи методу аутогенного тренування, який ґрунтується на свідомому застосуванні комплексу взаємопов'язаних прийомів психічної саморегуляції й виконанні нескладних фізичних вправ із словесним самонавіюванням. Головна увага при цьому приділяється набуванню й закріпленню навичок м'язового розслаблення (релаксації). У рекомендованому сеансі, який має проводитися в кімнаті психофізіологічного розвантаження з відповідним інтер'єром та кольоровим оформленням, виділяються три періоди, що відповідають фазам відновлювального процесу.

Перший період - абстрагування працівників від виробничої обстановки - відповідає фазі залишкового збудження.

Другий період - заспокоєння - відповідає фазі відновлювального гальмування.

Третій період - активізація - відповідає фазі підвищеної збудженості.

Сеанси психологічного розвантаження можуть проводитись за єдиною програмою через індивідуальні навушники і складатись із двох періодів по 5 хвилин кожний:

1) повне розслаблення;

2) активізація працездатності.

У разі потреби, на фоні музичних програм можуть вимовлятися окремі фрази навіювання відпочинку, гарного самопочуття і, на заключному етапі, бадьорості. Після сеансів психофізіологічного розвантаження у працівників зменшується відчуття втоми, з'являються бадьорість, гарний настрій. Загальний стан відчутно поліпшується.

4.2. Моделювання процесу виникнення травм та аварій

Моделювання процесу виникнення травм та аварій під час роботи з комп’ютерною технікою може бути корисним для покращення безпеки та запобігання можливим негативним подіям. Нижче наведено загальний огляд етапів такого моделювання:

1. Визначення сценаріїв травм та аварій:
    - Вивчення історії попередніх інцидентів і аварій на робочих місцях.
    - Аналіз та ідентифікація потенційних сценаріїв травм та аварій.
2. Збір даних:
    - Збір статистичних даних про попередні інциденти.
    - Запис виявлених ризикових факторів та умов, що сприяють травмам і аваріям.
3. Визначення факторів ризику:
    - Аналіз впливу різних чинників на виникнення травм та аварій (наприклад, обладнання, люди, середовище).

- Визначення факторів, які можуть збільшити ризик інцидентів.

4. Розробка математичних моделей:
    - Створення математичних моделей, що відображають взаємозв'язки між різними чинниками та ймовірність виникнення травм та аварій.
    - Використання статистичних методів для аналізу даних та розрахунку ризиків.
5. Валідація моделей:
    - Перевірка правильності моделей на основі реальних інцидентів.
    - Коригування та уточнення моделей на основі нової інформації.
6. Визначення заходів безпеки:
    - Розробка та впровадження заходів безпеки на основі результатів моделювання.
    - Організація тренінгів та навчань з безпеки для персоналу.
7. Моніторинг та аналіз:
    - Систематичний моніторинг безпекових показників.
    - Постійний аналіз даних та моделей для вчасного виявлення нових ризикових факторів.

Моделювання такого роду допомагає вдосконалити безпекові стандарти та процедури, зменшити ймовірність травм та аварій та підвищити загальний рівень безпеки на робочому місці.

4.3. Розробка заходів щодо безпеки у надзвичайних ситуаціях

Забезпечення безпеки населення у надзвичайних ситуаціях включає в себе комплекс заходів та дій для запобігання, мінімізації та виправлення наслідків небезпеки чи кризової ситуації. Основні аспекти забезпечення безпеки під час загроз та надзвичайних ситуацій включають:

Попередження та інформування:

- Системи оповіщення: Розробка та впровадження систем оповіщення через різні канали (сирени, текстові повідомлення, соціальні мережі тощо) для швидкого повідомлення населення про небезпеку.
- Інформаційні кампанії: Проведення освітніх кампаній, які навчають населення діяти в надзвичайних ситуаціях та розпізнавати можливі загрози.

Евакуація та укриття:

- Плани евакуації: Розробка та публікація планів евакуації для населення, шкіл, медичних установ та інших важливих об'єктів.
- Створення укриттів: Розміщення та обладнання спеціальних укриттів для захисту населення від небезпеки.

Системи моніторингу та раннього попередження:

- Метеорологічний моніторинг: Постійне спостереження за погодними умовами для вчасного виявлення можливих стихійних лих та інших природних небезпек.
- Системи виявлення загроз: Використання сучасних технологій для виявлення потенційних загроз, таких як техногенні аварії, терористичні атаки тощо.

Підготовка та тренування:

- Навчання населення: Проведення регулярних навчань та тренувань для населення щодо дій в надзвичайних ситуаціях.
- Тренування служб безпеки: Підготовка та тренування відповідальних служб, включаючи рятувальників, медичний персонал та поліцію.

Комунікація та координація:

- Центри управління кризовими ситуаціями: Створення центрів управління, де представники різних служб можуть координувати свої дії та обмінюватися інформацією.
- Системи комунікації: Забезпечення ефективних систем зв'язку для оперативного обміну інформацією між відповідальними службами та населенням.

Системи медичної допомоги: Розвиток та удосконалення систем медичної допомоги для надання ефективної допомоги потерпілим в надзвичайних ситуаціях.

Забезпечення безпеки населення у надзвичайних ситуаціях вимагає інтегрованого підходу, співпраці різних служб та постійного оновлення планів та технічних рішень для ефективного вирішення різноманітних ситуацій загроз та надзвичайних подій.

РОЗДІЛ 5.
ВИЗНАЧЕННЯ ЕФЕКТИВНОСТІ ВІД ВИКОРИСТАННЯ АЛГОРИТМІВ ВИРІШЕННЯ ЗАВДАНЬ КОМІВОЯЖЕРА

Метою розробки програмного додатку є оптимізація маршрутів перевезення, зменшення витрат на логістику та підвищення продуктивності підприємства. Завдяки використанню алгоритмів для вирішення задачі комівояжера зменшуються витрати на паливо, час доставки та збільшується точність виконання логістичних операцій.

Економічні показники, які використовуються для оцінки ефективності впровадження розробки, дозволяють проаналізувати фінансову доцільність проєкту, обчислити вигоди та оцінити рентабельність інвестицій.

Етапи оцінки економічної ефективності

Витрати на розробку програмного додатку

1. Витрати на розробку ($В_{розробки}$) - це сукупність усіх витрат, пов’язаних із розробкою, впровадженням і технічним забезпеченням програмного додатку. До витрат на розробку входять:

- Заробітна плата програмістів — витрати на оплату праці розробників за час, протягом якого створюється програмний продукт.
- Витрати на технічне забезпечення — кошти на купівлю серверів, ліцензій на програмне забезпечення, оренду хмарних ресурсів тощо.
- Інші витрати — витрати на тестування, обслуговування, консультації, маркетинг або навчання персоналу.

2. Економія на паливі ($Е_{паливо}$)

Це зменшення витрат на паливо завдяки оптимізації маршрутів перевезень. Вона досягається за рахунок скорочення відстаней, які долають транспортні засоби. Розрахунок включає:

- Скорочення середньої відстані.
- Витрати пального на 1 км.
- Ціну пального.

- Кількість рейсів.

3. Економія на заробітній платі водіїв ($Е_{зарплата}$)

Це скорочення витрат на оплату праці водіїв завдяки зменшенню тривалості рейсів. Економія досягається за рахунок скорочення часу доставки вантажу та ефективнішого планування.

4. Підвищення продуктивності ($Е_{продуктивність}$)

Це фінансовий показник, який відображає збільшення доходу компанії завдяки виконанню більшої кількості перевезень за той самий час. Оптимізовані маршрути дозволяють транспорту виконувати більше замовлень, що призводить до підвищення ефективності логістичних операцій.

5. Рентабельність (R)

Рентабельність проєкту — це показник, який визначає співвідношення між вигодами від впровадження і витратами на розробку. Вона показує, наскільки прибутковим є проєкт у порівнянні з витратами.

R=(Ввигоди−Врозробки)·100%

- Ввигоди — загальна сума економії або доходів, отриманих від використання програмного продукту.
- Врозробки — загальна сума витрат на розробку.

6. Термін окупності (Токуп)

Термін окупності проєкту — це період часу, за який доходи або економія від впровадження програмного продукту компенсують витрати на його розробку.

ТокупВрозробкиВвигоди

- Врозробки — витрати на розробку програмного продукту.
- Ввигоди — економія або додатковий дохід, який отримується щомісячно завдяки впровадженню продукту.

7. Загальні вигоди (Ввигоди) - це сукупність усіх фінансових переваг, які отримує компанія завдяки використанню програмного додатку. Включає:

- Економію на паливі.
- Економію на заробітній платі водіїв.
- Підвищення продуктивності.

8. Чиста вигода (Вчиста) - це різниця між вигодами та витратами, яка показує, наскільки вигідним є проєкт у абсолютному значенні:

Вчиста=Ввигоди−Врозробки

1.1. Зарплата програмістів (ППП)

Це витрати на оплату праці команди розробників. Розраховується за формулою:

$$П = ЗП_{місяць} \cdot К_{працівників} \cdot К_{місяців}$$

- ЗПмісяць— середня заробітна плата програміста за місяць.
- Кпрацівників — кількість програмістів.
- Кмісяців — тривалість розробки в місяцях.

- Витрати на технічне забезпечення (Т) -це вартість серверів, ліцензійного програмного забезпечення, середовища розробки тощо.
- Інші витрати (І) - витрати на тестування, обслуговування, консультації тощо.

1.2. Витрати на технічне забезпечення (ТТТ)

Це витрати на сервери, ліцензії для програмного забезпечення, хмарне середовище, бази даних тощо.

Купівля сервера вартістю 8,000 USD і ліцензія на ПЗ за 2,000 USD:

Т=8,000+2,000=10,000 USD.

1.3. Інші витрати (ІІІ)

Це витрати на тестування, маркетинг, технічне обслуговування, навчання персоналу тощо.

Якщо тестування і обслуговування коштує 4,000 USD:

І=4,000 USD.

Загальні витрати на розробку:

$$В_{розробки} = П + Т + І$$

2.2. Очікувані вигоди від впровадження

2. Вигоди від впровадження (Ввигоди) - це сукупність усіх фінансових вигод, які отримує підприємство завдяки впровадженню програмного додатку. Складається з кількох основних компонентів.

2.1. Економія на паливі (ЕпаливоE_{\text{паливо}}Епаливо)

Оптимізація маршрутів дозволяє зменшити відстань, яку долають транспортні засоби, і, як наслідок, скоротити витрати на паливо.

$$Е_{\text{паливо}} = \Delta_{\text{відстань}} \cdot В_{\text{пального}} \cdot Ц_{\text{пального}} \cdot N_{\text{рейсів}}$$

- Δвідстань — скорочення відстані маршруту (км).
- Впального — витрати пального на 1 км (л/км).
- Цпального — ціна 1 літра пального (USD/л).
- Nрейсів — кількість рейсів за певний період.

Якщо зекономлено 20 км на маршрут, транспорт споживає 0.2 л/км, пальне коштує 1.5 USD/л, а кількість рейсів — 2000:

Епаливо=20·0.2·1.5·2000=12,000 USD/місяць.

- Зменшення витрат на паливо (Епаливо): Економія досягається за рахунок оптимізації маршрутів перевезення:

$$Е_{\text{паливо}} = \Delta_{\text{відстань}} \cdot В_{\text{пального}} \cdot Ц_{\text{пального}}$$

де:

- Δвідстань — скорочення середньої відстані маршрутів (км).
- Впального — середнє споживання пального (л/км).
- Цпального — вартість 1 літра пального.

Зменшення витрат на зарплати водіїв: Економія часу доставки дозволяє скоротити робочі години:

$$Езарплата = \Delta_{\text{час}} \cdot С_{\text{водій}}$$

- де:
  - Δчас — скорочення робочого часу водіїв (години).
  - Сводій — погодинна заробітна плата водія.

- Підвищення продуктивності (Епродуктивність). Завдяки оптимізації маршрутів можна виконати більше перевезень за той самий час.

2.2. Економія на заробітній платі водіїв (Езарплата)

Скорочення тривалості маршрутів зменшує робочий час водіїв.

Езарплата=Δчас·Своді́й·Nрейсів

- Δчас — скорочення часу маршруту (години).
- Своді́й — погодинна заробітна плата водія (USD/год).
- Nрейсів — кількість рейсів за певний період.

Якщо кожен маршрут скорочується на 2 години, водій отримує 10 USD/год, і кількість рейсів — 2000:

Езарплата=2·10·2000=40,000 USD/місяць.

2.3. Підвищення продуктивності (Епродуктивність)

Оптимізація дозволяє виконувати більше рейсів у той самий робочий час, збільшуючи доходи компанії.

Якщо додатково виконуються послуги вартістю 5,000 USD на місяць:

Епродуктивність=5,000 USD/місяць.

Сукупні вигоди:

$$В_{вигоди} = Е_{паливо} + Е_{зарплата} + Е_{продуктивність}$$

2.3. Розрахунок економічної ефективності

Рентабельність проєкту (R):

$$R = \frac{В_{вигоди} - В_{розробки}}{В_{розробки}} \cdot 100\%$$

Термін окупності (Токуп):

$$Т_{окуп} = \frac{В_{розробки}}{В_{вигоди}}$$

Вихідні дані:

- Витрати на розробку:

- $\text{ЗП}_{\text{місяць}} = 2000\,\text{USD}$, $\text{К}_{\text{працівників}} = 3$, $\text{К}_{\text{місяців}} = 6$:

$$\text{П} = 2000 \cdot 3 \cdot 6 = 36,000\,\text{USD}.$$

- $\text{Т} = 10,000\,\text{USD}$ (сервери, ПЗ).
- $\text{І} = 4,000\,\text{USD}$ (тестування).
- Загальні витрати: $\text{В}_{\text{розробки}} = 36,000 + 10,000 + 4,000 = 50,000\,\text{USD}.$

Очікувані вигоди

- $\Delta_{\text{відстань}} = 20\,\text{км/маршрут}$, $\text{В}_{\text{пального}} = 0.2\,\text{л/км}$, $\text{Ц}_{\text{пального}} = 1.5\,\text{USD/л}$, 2000 маршрутів на місяць:

$$\text{Е}_{\text{паливо}} = 20 \cdot 0.2 \cdot 1.5 \cdot 2000 = 12,000\,\text{USD/місяць}.$$

- $\Delta_{\text{час}} = 2\,\text{год/маршрут}$, $\text{С}_{\text{водій}} = 10\,\text{USD/год}$, 2000 маршрутів:

$$\text{Е}_{\text{зарплата}} = 2 \cdot 10 \cdot 2000 = 40,000\,\text{USD/місяць}.$$

- $\text{Е}_{\text{продуктивність}} = 5,000\,\text{USD/місяць}.$

Загальні вигоди:

$$\text{В}_{\text{вигоди}} = 12,000 + 40,000 + 5,000 = 57,000\,\text{USD/місяць}.$$

Рентабельність:

$$R = \frac{57,000 \cdot 12 - 50,000}{50,000} \cdot 100\% = 1268\%.$$

Термін окупності:

$$\text{Т}_{\text{окуп}} = \frac{50,000}{57,000} \approx 0.88\,\text{місяця}.$$

Розробка програмного додатку є економічно вигідною. Очікуваний термін окупності становить менше місяця, а рентабельність проєкту перевищує 1268%. Це свідчить про доцільність впровадження програмного забезпечення для оптимізації логістичних процесів.

ВИСНОВКИ І ПРОПОЗИЦІЇ

У ході досліджень та аналізу тенденцій були отримані алгоритми, здатні вирішувати задачу ЗкК та знаходити її наближені рішення та створено пакет, який при подальшому розвитку можна буде використовувати на практиці.

Отримані результати дозволяють виділити деякі закономірності:

1. У разі зростання розмірності завдання якість роботи різних алгоритмів зближується.

2. При зростанні числа комівояжерів якість помітно знижується, можливо, це викликано зростанням "загальної" роботи.

3. Алгоритм поділу на дві групи навіть без обмінної оптимізації дає кращий результат у переважній більшості випадків, виняток становлять варіанти з великою кількістю міст, де різниця в якості різних методів знижується.

4. Збільшення кількості груп під час поділу погіршує загальний результат.

5. Додавання міста до групи на основі різниці відстаней дає кращий результат, ніж додавання на основі мінімуму відстаней.

6. Обмінна оптимізація дає помітне покращення, але воно погіршується при зростанні кількості міст.

Цілком ясно, що подальший розгляд варіантів зі збільшенням кількості груп при розподілі не має жодного сенсу.

Однією з перспективних напрямів поліпшення результату роботи є розподіл на дві групи. Зокрема, цікаве питання про оцінку, яка використовується як критерій обміну при обмінній оптимізації. Обчислення мінімального кістяка занадто трудомістка операція.

Метод розрізання "загального" маршруту має сенс лише за великої кількості міст. Він значно простіший при реалізації, але не придатний для малої кількості міст.

Розробка програмного додатку є економічно вигідною. Очікуваний термін окупності становить менше місяця, а рентабельність проєкту перевищує 1268%. Це

свідчить про доцільність впровадження програмного забезпечення для оптимізації логістичних процесів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Андрейцев А.Ю., Клецька Т.С. Про розширення класу задач про кільцевий маршрут. Збірник матеріалів міжнародної науково-практичної конференції «Дніпровські читання-2020»). Київ: ДУІТ. 2020. С.172–175.

2. Біологічні основи мурашиних колоній – [Електронний ресурс]. URL: http://posibniki.com.ua/post-prikladni-sistemi-kolektivnogo-intelektu-swarm-intelligence (Дата звернення 15.10.2020).

3. Денисюк В.О. Обирання критеріїв для задачі комівояжера. The scientific heritage. https://www.tsh-journal.com/wpcontent/uploads/2020/09/VOL-8-No-46-46-2020.pdf

4. Денисюк В.О. Вибір графічного процесору для систем комп'ютерної графіки. The scientific heritage. https://www.tshjournal.com/wp-content/uploads/2020/09/VOL-1-No-47-47-2020.pdf

5. Задача комівояжера. https://uk.wikipedia.org/wiki/Задача_ комівояжера.

6. Красников С.О. Алгоритм налаштування параметрів алгоритму імітаційного відпалу для розв'язання задачі комівояжера // Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки. – 2018. – Том 29. - № 5. – С. 149-153.

7. Красников С.О. Алгоритм налаштування параметрів алгоритмів стохастичного локального пошуку для розв'язання задачі комівояжера // Збірник статей за XL міжнародної науково-практичної конференції 2 частина: «Розвиток науки в XXI столітті» від 13.10.2018. – Науково-інформаційний центр «Знання». – 2018. – частина 1. – С.45-54.

8. Математичні методи дослідження операцій: підручник/ Є. А. Лавров, Л. П. Перхун, В. В. Шендрик та ін. – Суми: Сумський державний університет, 2017. – 212 с.

9.Bektas T. The multiple traveling salesman problem: an overview of formulations and solution procedures // Omega. – Elsevier. – № 34. – 2006. – P. 209-219.

10.Boland N., Hewitt M., Minh Vu D., Savelsbergh M. Solving the Traveling Salesman Problem with Time Windows through Dynamically Generated Time-Expanded Networks // Integration of AI and OR Techniques in Constraint Programming. – 2017. – volume 10335. – P. 254-262.

11.1Cattaruzza D., Absi N., Feillet D., Vidal T. A memetic algorithm for the multi trip vehicle routing problem // European Journal of Operational Research. – Volume 236. – Issue 3. – 2014. – P.833-848

12.Divya M. A Comparison of Ant Colony Optimization Algorithms Applied to Distribution Network Reconfiguration// International Journal of Engineering Research & Technology, Volume 3, Issue 01, 2016. – pp. 1-4.

13.Geng X., Chen Zh., Yang W., Shi D., Zhao K. Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search // Applied Soft Computing. – Elsevier. – Volume 11. – 2011. – P.3680-3689.

14.Gutin G., Punnen A. P. The traveling salesman problem and its variations // Springer. – USA. – 2006. – 830 p.

15.Hahsler M., Hornik K. TSP – Infrastructure for the Traveling Salesperson Problem// Journal of Statistical Software, December 2007, Vol. 23, Issue 2, 2007. – pp. 1-21.

16.Hassan M. H. Mustafa, Ayoub Al-Hamadi, Mohamed Abdulrahman, Shahinaz Mahmoud, Mohammed O Sarhan On Comparative Analogy between Ant Colony Systems and Neural Networks Considering Behavioral Learning Performance// Journal of Computer Sciences and Applications. 2015, Vol. 3 No. 3, 79-89.

17. Kadri R.L., Boctor F.F. An efficient genetic algorithm to solve the resource-constrained project scheduling problem with transfer times: The single mode case // European Journal of Operational Research. – Volume 265. – Issue 2. – 2018. – P.454-462

18. Korte B., Vygen J. Combinatorial Optimization: Theory and Algorithms (Algorithms and Combinatorics) 6th ed., New York, 2018,455 p.

19. Meneses S., Cueva R., Tupia M., Guanira M. A genetic algorithm to solve 3D traveling salesman problem with initial population based on a GRASP algorithm // Journal of Computational Methods in Sciences and Engineering. – 2017 – vol. 17. – no. S1. – P. S1-S10.

20. Mills K.L., Filliben J.J., Haines A.L. Determining Relative Importance and Effective Settings for Genetic Algorithm Control Parameters // Evolutionary Computation. – 2015. – MIT Press. – Volume 23. – Issue 2. – P. 309-342.

21.Mu C.H., Xie J., Liu Y., Chen F., Liu Y., Jiao L.C. Memetic algorithm with simulated annealing strategy and tightness greedy optimization for community detection in networks // Applied Soft Computing. – Volume 34. – 2015. – P.485-501.

22.Naderi B., Ruiz R. A scatter search algorithm for the distributed permutation flowshop scheduling problem // European Journal of Operational Research. – Volume 239. – 2014. – P.323-334.

23.Nygard K.E., Yang C.H. Genetic algorithm for the traveling salesman problem with time windows // Computer Science and Operations Research: New Developments in their Interfaces. – Elsevier. – 2014. – P. 411-423

24. Mauricio Resende G.C., Ribeiro Celso C. A short tour of combinatorial optimization and computational complexity // Optimization by GRASP. – 2016. – P. 13-39.

25.Rukundo, O., Cao, H. Advances on image interpolation based on ant colony algorithm. SpringerPlus 5, 403 (2016) – [Електронний ресурс]. URL: https://springerplus.springeropen.com/articles/10.1186/s40064-016-2040-9

26. Sh. Chunxin, Zh. Xiaoxia, Ch. Hongyang, Y. Jiao,Wangpeng, Weiyu A Hybrid Scatter Search Algorithm for QoS Multicast Routing Problem // Chinese Control And Decision Conference. – 2018. – P.4875-4878.

27.Shao W., Salim F.D., Gu T., Dinh N.-T., Chan J. Traveling Officer Problem: Managing Car Parking Violations Efficiently Using Sensor Data // IEEE Internet of Things Journal. – April 2018 – Vol. 5. – Issue 2. – P. 802 – 810.

28. Taha H. Operations Research: An Introduction, 10th Edition. Boston: Princeton, 2017. 848 p.

29.TSPLIB [Електронний ресурс] – Режим доступу до ресурсу: https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

30.Wang Z., Xing H., Li T., Yang Y., Qu R., Pan Y. A modified ant colony optimization algorithm for network coding resource minimization // IEEE Transactions on Evolutionary Computation. – Volume 20. – Issue 3. – 2016. – P.325-342.