НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Механіко-машинобудівний інститут

Кафедра технології машинобудування

«На правах рукопису»
УДК 621.91_____

«До захисту допущено»
Завідувач кафедри
_____ О.А. Охріменко
(підпис)
“___”__ _____ 2022 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 131 – Прикладна механіка

(код і назва)

На тему: Оптимізація траєкторій руху осьового інструменту при обробленні отворів на верстатах з ЧПК.

Виконав (-ла): студент (-ка) _2_ курсу, групи __МТ-11мп__
(шифр групи)

Тарасов Богдан Артемович _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник доцент,к.т.н,Кореньков В.М._____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант - _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2022

ЗМІСТ

АННОТАЦІЯ

Тарасов Б.А. Оптимізація траєкторій руху осьового інструменту при обробленні отворів на верстатах з ЧПК.

Дисертація на здобуття наукового ступеня магістра за спеціальністю 131 – Прикладна механіка. Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського». – Київ, 2022.

На основі дослідів алгоритмів сортування, що виконуються для знаходження найкоротшого маршруту між заздалегідь відомими точками, в рамках дисертації ведеться дослідження по вдосконаленню та автоматизації прорахунку найкоротшого шляху осьового інструменту при переміщенні на холостих ходах.

В процесі роботи було розроблено алгоритм оптимізації руху осьового інструменту та програмне забезпечення для подальшого автоматизованого розрахунку проходження отворів для будь-якої деталі з CAD-систем.

Також було розроблено програмне забезпечення для CATIA v5 для автоматичного визначення геометрії отворів в моделі (такі як центри отворів з на вершині та на дні отвору та осьової лінії).

Ключові слова: Traveling Salesman Problem, Матриця , оптимальний маршрут, оптимізація переміщення, переміщення інструменту.

ABSTRACT

Tarasov B.A. Optimization of axial tool movement trajectories when machining holes on CNC machines.

Dissertation for obtaining a master's degree in specialty 131 - Applied mechanics. National Technical University of Ukraine "Ihor Sikorsky Kyiv Polytechnic Institute". - Kyiv, 2022.

Based on the experiments of sorting algorithms, which are performed to find the shortest route between previously known points, within the framework of the dissertation, research is being conducted on improving and automating the calculation of the shortest path of an axial tool when moving at idle speeds.

In the process of work, an algorithm for optimizing the movement of an axial tool and software for further automated calculation of the passage of holes for any part from CAD systems were developed.

Software was also developed for catia v5 to automatically determine the geometry of the holes in the model (such as hole centers with the top and bottom of the hole and center line).

Keywords: Traveling Salesman Problem, Matrix, optimal route, optimization of movement, tool movement.

РЕФЕРАТ

Магістерська дисертація на тему: « Оптимізація траєкторій руху осьового інструменту при обробленні отворів на верстатах з ЧПК», містить сторінки пояснювальної записки, рисунків – 31, таблиць – 13 , використаних джерел – 8, загальний об’єм роботи складає 65 сторінок.

Актуальність

Сучасні алгоритми знаходження найкоротшого шляху описують загальні алгоритми для оптимізації руху інструменту в основному для фрезерних верстатів з ЧПК та не мають відносно дієвих методів автоматизації цих розрахунків для різних деталей прямо з CAD-моделі, у свою чергу питанню оптимізації переміщення свердлильного інструменту присвячено не так багато робіт, в той час як ця тема дуже актуальна на виробництвах з деталями з великою кількістю отворів (як наприклад листові деталі корпусів або частини фюзеляжу літаків).

Мета дослідження

Зменшення часу технологічної підготовки виробництва шляхом оптимізації шляху холостих переміщення інструменту при обробленні отворів крупногабаритних деталей та автоматизації формування управляючої програми для верстатів з ЧПК по наявній тривимірній моделі деталі для осьового інструменту.

Задачі дослідження

1. Огляд можливих алгоритмів та визначення їх актуальності для вирішення задачі комівояжера в випадку оптимізації переміщення інструменту.

2. Розробка концептуального підходу щодо вирішення задачі комівояжера.

3.Практична реалізація.

4.Стартап проект.

Об'єкт дослідження

Об’єктом дослідження є траєкторні переміщення осьового інструменту при оброброблення отворів у крпногабаритних або листових деталях.

Предмет дослідження

Предметом дослідження є методи оптимізації холостих переміщень інструменту при обробленні на верстах з ЧПУ фрезерної групи.

Наукова новизна отриманих результатів

Використано мтод пошуку оптимальних маршрутів на графах до вирішення задачі оптимізації холостих переміщень при обході отворів при свердлінні крпногабаритних або листових деталей.

Практичне значення отриманих результатів

Розроблено програмне забезпечення на мовах програмування VBA та Pyhton, що дозволяє в автоматичному реживі розпізнавати геометрію отворів для довільних тривимірних моделей в середовищі CATIA v5, вирішувати задачу оптимізації обходу отворів та генерувати управляючу пограму для верстату з ЧПК.

Ключові слова : Traveling Salesman Problem, оптимальний маршрут, оптимізація переміщення, переміщення інструменту.

Вступ

Найважливішим економічним показником виробництва є необхідний час на завершення однієї технологічної операції (ТО). Час на одну технологічну операцію визначається багатьма факторами. Але найважливішим та найвагомішим критерієм часу ТО є переміщення інструмента, його зміна та підготовчий час. Задача зводиться не тільки до оптимізації переміщення інструменту та послідовність обробки отворів в загальному технологічному процесі, але і до автоматизації розрахунку траєкторії інструменту і порядку проходження отворів, за рахунок комп'ютерного ПО та написання скриптів до нього.

За допомогою сучасних графічних систем автоматизованого проектування та виготовлення виробів (CAD/CAM) значно спрощується створення керуючих програм для фрезерних верстатів з ЧПК. Автоматизоване програмування при вирішенні складних технологічних завдань дозволяє максимально ефективно використовувати дороге обладнання, використання якого дозволяє створювати вироби з обробкою будь-якої складності, що відповідає встановленим вимогам якості, з мінімальними витратами часу на їх обробку.

Суть руху різання полягає в точковому контакті між інструментом і деталлю, що обробляється. Час, витрачений на цей рух інструменту, займає значну частину часу операції обробки деталі і вважається основним часом. Холостий хід верстата - це сума холостих ходів верстата при зміні різального інструменту і при зміні обробленої поверхні. Час простою інструменту є допоміжним часом. При переміщені інструменту кожний пройдений не оптимальний шлях може призвести до непродуктивних витрат в часі. Скорочення непродуктивних витрат при обробці деталей значно підвищує ефективність використання технологічного обладнання, особливо дорогих універсальних верстатів.

Варто зазначити що сучасні CAD/CAM системи не мають модулів для автоматичного розрахунку послідовність проходження отворів, як показує практика Інженер на виробництві визначає послідовність проходження отвору на власному досвіді, в той час як на простих деталях, шлях буде визначено оптимально, на складно профільних деталях вірогідність втрат в часі, за рахунок неоптимального шляху інструменту, дуже висока. На Рис.1.1 вказані результати експерименту [1] на вибірці людей які намагались знайти найкоротший шлях між двома точками в місті(червоні точки), та результати багатьох розрахунків алгоритму який вбудований в GPS (зелені точки). Як видно з цього експерименту при виконанні нетривіальних задач по знаходженню найкоротшого шляху з великою кількістю змінних ручний

вибір маршруту не стоїть в пріоритеті якщо наша задача стосується мінімізації переміщень інструменту.

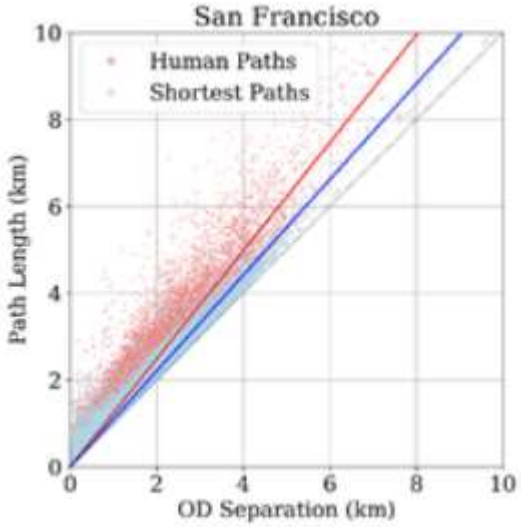


Рис. 1.1 Порівняння результатів знаходження найкоротшого шляху [1]

Тому так як час роботи верстату відносно дорогий в рамках огляду виробництва було вирішено провести оптимізацію переміщення інструменту через всі точки за найкоротшим шляхом. Крім написання автоматизованого алгоритму розрахунку оптимального проходження отворів, задачею цієї роботи була також автоматизація та інтегрування цих алгоритмів для прямої роботи з CAD/CAM системами.

1. АНАЛІЗ СТАНУ ПИТАННЯ

1.1 Алгоритми пошуку найкоротших шляхів

Завдяки широкому застосуванню теорія найкоротших маршрутів останнім часом інтенсивно розвивалася і використовувалась в різних сферах діяльності, наприклад як, знайти оптимальний маршрут між двома точками на місцевості (найкоротший шлях від дому до робочого місця), знайти оптимальний маршрут під час вирішення логістичних задач, в системах автопілотування, або в системах передачі інформаційних пакетів у мережі Інтернет тощо. В випадку цієї роботи для переміщення між точками свердління. Найкоротший шлях можливо розрахувати за допомогою комбінаторних графів і потокових математичних моделей. Існують наступні види алгоритмів пошуку оптимального шляху на основі методу графів:

- Алгоритм Дейкстри (використовується для визначення найкоротшого маршруту тільки між двома вершинами);

- Алгоритм Флойда (використовується для визначення найкоротшого маршруту між усіма парами вершин на визначеному графі);

- Алгоритм Беллмана-Форда (для визначення оптимального шляху від вершини графа до кожної точки окремо не прокладаючи маршрут між всіма точками);

- Алгоритм A* (використовується для знаходження маршруту з найменшим значенням від вершини (початку) до інших (остаточних) з використанням алгоритму пошуку першої точки до найкращого знайденого розв’язку на графі)

- Алгоритм Джонсона (використовується для визначення оптимального шляху між усіма парами вузлів зваженого орієнтованого графа);

- Хвильовий алгоритм (алгоритм сортування, який характеризується методом пошуку в ширину) знаходить шлях між вершинами графа, який містить мінімальну кількість проміжних ребер;

- Алгоритм Габова (використовується для визначення найкоротшого шляху шляхом масштабування);

- Алгоритм Карпа (використовується для визначення циклу з найменшою загальною вагою).

Наведені алгоритми легко виконати з невеликою кількістю вершин у графі. Зі збільшенням кількості цих вершин завдання пошуку оптимального шляху ускладняється. Існують також наступні види алгоритмів:

- Мурашиний (Суть розрахунку полягає в використанні типової поведінки мурах, що шукають та прокладають дороги від основної колонії до їжі.)

- генетичний алгоритм або алгоритм динамічного програмування (в основі лежить подібний алгоритм до біологічної еволюції в природі, що використовується для розрахунку задач оптимізації і моделювання знайдених шляхів, методом послідовного підбору, поєднання і вибірці необхідних параметрів з розрахунками, що нагадують біологічну еволюцію.

- алгоритм перебору грубою силою (вирішує проблему через проходження всіх можливих маршрутів: він переглядає всі можливі варіанти, до тих пір поки не знайде рішення. Часова складність алгоритму грубої сили часто пропорційна розміру вхідних даних.

- Метод гілок та кордонів (Branch and bound) метод гілок та кордонів складається з систематичного перебору варіантів рішень за допомогою пошуку розгалужень шляхів з відокремленням неоптимальних гілок дерева. Алгоритм розглядає гілки цього дерева, які являються собою підмножини набору рішень.

Розглянемо детальніше існуючі алгоритми пошуку найкоротшого шляху, перераховані вище.

1.1.1 Алгоритм Дейкстри

Алгоритм Дейкстри - графовий алгоритм, винайдений голландським вченим Е. Дейкстрою в 1959 році. Алгоритм дозволяє знайти найкоротшу відстань від однієї з вершин графа до решти.

Алгоритм було описано в роботах [2] [7].

Алгоритм працює тільки для графів без негативних вагових ребер і широко використовується в програмуванні логічних елементів, він використовується протоколом OSPF, наприклад, для усунення кільцевих маршрутів. Алгоритм Дейкстри вирішує проблему найкоротшого шляху від вузла для зваженого орієнтованого графа G=(V,E) з вихідним вузлом s, у якому ваги всіх ребер є невід'ємними ($\omega(u, v) \geq 0$ для всіх $(u , v) \in E$, де V — непорожній набір вершин або вузлів, E — набір пар (у випадку неорієнтованого невпорядкованого графа) вершин, які називаються ребрами.

Під час роботи алгоритм Дейкстри підтримує набір S $\in V$, що складається з вершин v, для яких δ(s, v) уже знайдено (тобто a $d[v] = \delta$(s, v). Алгоритм вибирає вузол u $\in$ V \ S із найменшим $d[u]$, додає u до набору S та проводить релаксацію всіх ребер, що походять від u, після чого цикл повторюється. Вершини не в S зберігаються в черзі Q з пріоритетами, визначеними значеннями функції d. Передбачається, що граф заданий за допомогою списків суміжних вершин. Кожному вузлу V присвоюється мітка - відома мінімальна відстань від цього вузла до α. Алгоритм працює крок за кроком - на кожному кроці він «відвідує» вершину і намагається зменшити помічені маршрути. Алгоритм завершується, коли відвідано всі вершини.

1.1.2 Алгоритм Флойда-Воршелла

Алгоритм Флойда-Воршелла — динамічний алгоритм для знаходження найкоротших відстаней між усіма вершинами зваженого орієнтованого графа. Розроблено Робертом Флойдом і Стівеном Воршеллом у 1962 році.

Алгоритм було описано в роботах [4] [7].

Короткий опис алгоритму. Нехай вузли графа G=(V,E), V=n пронумеровано від 1 до n і введемо позначення довжини d_{ij}^k найкоротшого шляху від i до j, який, крім самих вузлів i, j, проходить лише через вершини 1…k. Очевидно d_{ij}^0 є довжиною (вагою) ребра (i, j), якщо воно присутнє (інакше його довжину можна позначити як ∞), як було описано в [7]:

Є два варіанти значень d_{ij}^k, k ∈ (1, ... ,n):

1. Найкоротший шлях між i, j не проходить через вершину k, тоді $d_{ij}^k = d_{ij}^{k-1}$.

2. Існує коротший шлях між i, j, який проходить через k, потім він йде на початку від i до k а потім від k до j. В даному випадку очевидно що $d_{ij}^k = d_{ij}^{k-1} + d_{kj}^{k-1}$.

Отже, щоб знайти значення функції, достатньо вибрати найменше з двох позначених значень.

Тоді повторювана формула для d_{ij}^k має вигляд:

d_{ij}^{0} — довжина ребра (i, j)

d_{ij}^{k} = min ($d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}$).

Отримані значення d_{ij}^{n} є довжинами найкоротших шляхів між вершинами i, j.

1.1.3 Алгоритм Беллмана-Форда

Запропоновано незалежно Річардом Беллманом і Лестером Фордом. Алгоритм Беллмана-Форда — це алгоритм пошуку найкоротшого шляху у зваженому графі. Алгоритм знаходить найкоротші шляхи від вершини графа до решти. На відміну від алгоритму Дейкстри, алгоритм Беллмана-Форда передбачає ребра з від'ємною вагою.

Алгоритм було описано в роботах [4] [7].

Опишемо алгоритм крок за кроком.

Крок 0. Пронумеруємо всі вузли G = (A, B) таким чином, щоб A = {1, 2,....p} і при цьому число «1» має саме той вузол, з якого Найкоротші шляхи ведуть до всіх інших вершин. Далі будуємо матрицю M = (m_{ij}), i,j = 1,2, ... p,

$$m_{ij} = \begin{cases} f\big((i,j)\big), \text{якщо } i,j \in B \\ \text{комірка залишається порожньою, якщо } (i,\text{j}) \notin B \end{cases}$$

Крок 1. Поставимо цифрову позначку «0» біля першого рядка матриці M (ліворуч від матриці) і таку ж позначку поставимо над першим стовпцем матриці. Потім ми дивимося на виділений рядок ліворуч і кожного разу, коли ми натрапляємо на клітинку з числом, ми додаємо це число до мітки рядка та пишемо загальну суму в стовпці, у якому знаходиться ця клітинка. Потім ми відображаємо стовпець мітки відносно головної діагоналі. З'являються позначені лінії. Ми збираємося зробити те саме з кожним із виділених рядків: ми будемо дивитися на виділений рядок зліва направо і кожного разу, коли ми зустрічатимемо клітинку з числом, ми будемо додавати це число до мітку рядка та введіть підсумок як мітку над стовпцем, у якому знаходиться ця клітинка стоїть. При цьому ми будемо дотримуватись принципу «не змінювати наявну мітку». Потім показуємо підписи стовпців по відношенню до головної діагоналі і знову робимо те ж саме з позначеними рядками. І так до тих пір, поки їх не будуть позначені всі рядки та всі стовпці.

Крок 2. Розглянемо рядки таблиці в порядку зростання їх номерів. Клітини видно в кожному рядку ліворуч і щоразу, коли зустрічається число, воно додається до маркера рядка, а сума порівнюється позначивши стовпець, у якому знаходиться знайдене число. Якщо сума менша позначки в графі, ця позначка в графі замінюється зазначеною сумою. якщо сума більша або дорівнює позначці, то нічого не змінюється. Після цього перегляду всіх рядків з'явилися нові примітки. Колонки дзеркально відображаються відносно головної діагоналі і процес повторюється. І так до тих пір, поки не будуть припинені всі зміни в позначках.

Крок 3. Тепер ви можна використовувати маркери для створення найкоротших шляхів від першої вершини до решти. Фіксуємо довільний вузол k (k=2,3,....p) і опишемо найкоротший шлях від першого вузла до вузла k. По-перше, довжина цього найкоротшого шляху дорівнює мітці λ_k, над у якому стовпчику записаний номер k. По-друге, передостанній вузол розташовується на найкоротшому шляху від першого вузла до вузла k таким чином: у стовпчику номер k знаходимо число, сума якого з позначкою рядка, в якому воно знаходиться. Знайдене число дорівнює λ_k. Номер рядка, в якому знайдено знайдене число, дорівнює I. Тоді передостаннім вузлом на найкоротшому шляху від 1 до k є вершина I. Вершина, яка є попередньою до вершини I, повинна бути отримана як попередня на найкоротшому шляху від 1 до I і так далі.

1.1.4 Алгоритм A*

Алгоритм A* використовується для проходження маршруту з найменшою вартістю від вершини (початкової) до другої (останньої) за допомогою алгоритму пошуку першого найкращого збігу на графіку.

Він використовує оцінку вузлів, об’єднуючи в собі g (n) - вартість проходження до конкретного вузла і h (n) - вартість переміщення від конкретного вузла до місця призначення [7]:

$$f(n) = g(n) + h(n)$$

Оскільки функція g(n) дозволяє визначити вартість шляху від початкового вузла до вузла n, а функція h (n) визначає оцінку вартості найдорожчого шляху від вузла до цілі, то справедлива формула:

f (n) = оцінка вартості найдорожчого шляху рішення, що проходить через вузол n.

Тому, намагаючись знайти найдешевше рішення, потрібно спочатку спробувати перевірити вузол з найменшим значенням: якщо евристична функція h (n) задовольняє деякі умови, пошук A* буде повним і оптимальним.

Аналіз оптимальності A*-пошуку є простим, якщо цей метод використовується в поєднанні з алгоритмом Tree-Search. У цьому випадку пошук A* є оптимальним, якщо h (n) представляє можливу евристичну функцію, тобто за умови, що вона ніколи не переоцінює вартість досягнення цілі. А оскільки g(n) – це точна вартість досягнення вузла n безпосередньо з нього Звідси випливає, що функція f (n) ніколи не переоцінює справжню вартість досягнення рішення над вузлом n.

Останнє спостереження полягає в тому, що серед оптимальних алгоритмів цього типу (алгоритмів, які розширюють шляхи пошуку від кореня), пошук A* є оптимально ефективним для будь-якої заданої евристичної функції. Це означає, що використання іншого оптимального алгоритму не гарантує використання меншої кількості вузлів, ніж пошук A*.

Ці міркування показують, що пошук A*, як один із усіх цих алгоритмів, справді є ідеальним оптимумом і оптимально ефективним не означає, що пошук A* може служити відповіддю на всі пошукові запити. Складність полягає в тому, що при вирішенні більшості задач кількість вузлів у контурі цільового простору станів все ще експоненціально залежить від довжини розв’язку.

Експоненціальне зростання відбувається, коли похибка евристичної функції не зростає швидше, ніж логарифм фактичної вартості шляху. У математичній нотації умова субекспоненціального зростання виглядає так:

$$|h(n) - h^*(n)| \leq 0\ (\log h^*(n))$$

де h (n) – справжня вартість досягнення мети вузла n. Майже для всіх практичних евристичних функцій ця помилка принаймні пропорційна вартості шляху, і результуюче експоненціальне зростання зрештою перевищує можливості будь-якого комп’ютера. З цієї причини прагнення до

оптимального рішення часто не виправдане на практиці. Іноді замість цього краще використовувати параметри пошуку A*, які можуть допомогти вам швидко знайти неоптимальні рішення, а іноді розробити точнішу, але не обов'язково юридичну евристику.

1.1.5 Алгоритм Джонсона

Алгоритм було описано в роботах [2] [7].

Алгоритм Джонсона знаходить найкоротші шляхи для всіх пар вузлів за час O (V^2 log V + VE) і базується на ідеї зміни ваг (reweighting). Якщо ваги ребер графа невід'ємні, то ви можете знайти найкоротший шлях між усіма парами вершин, застосувавши алгоритм Дейкстри до кожної вершини. У цьому випадку цей алгоритм або повертає матрицю ваг найкоротшого шляху, або повідомляє, що граф має цикл із від'ємними вагами, ребра з від'ємними вагами, тоді ми можемо спробувати звести проблему до випадку невід'ємних ваг, і замінимо вагову функцію ω новою функцією ω', і наступні властивості повинні бути збережені.

а) Найкоротші шляхи не змінилися: для будь-якої пари вузлів u,$v \in V$ найкоротший шлях до u та від V у вигляді вагової функції ω також є найкоротшим шляхом з точки зору ω' і навпаки.

б) Усі нові вагові функції ω' (u, v) є невід'ємними.

Це означає, що нова вагова функція ω' має такі ж властивості і може бути побудована за час O(VE).

1.1.6 Алгоритм Габова

Алгоритм було описано в роботі [7].

Розглянемо зважений орієнтований граф G = (V, E), де ваги ребер є невід'ємними цілими числами, що не перевищують W . Визначимо, як знайти найкоротший шлях від вузла за час O (E log W).

Нехай k= [log (W + 1)] — кількість бітів у двійковому представленні числа W. Для i=1, 2,… K ми встановлюємо $\omega_i(u,v) = [\omega_i(u,v)/2^{k-i}]$ (іншими словами, $\omega_i(u,v)$ випливає з $\omega(u,v)$ шляхом відкидання k-і найменших бітів у двійковому представленні числа $\omega(u,v)$).

Наприклад, якщо k=5 і $\omega(u,v) = 25 = \langle 11001 \rangle$ то з $\omega_3(u,v) = \langle 110 \rangle = 6$. Зокрема, ω_1 приймає тільки значення 0 і 1,які визначаються старшим розрядом, $\omega_k = \omega$.

Нехай δ(u, v) буде вагою найкоротшого шляху від u до v відносно вагової функції ω_i (зокрема δ_k(u, v) = δ(u, v)). Алгоритм спочатку знаходить кожні δ_1(s, v) (s — початкова вершина), потім кожні δ_2(u, v) і так далі, поки не досягне δ_k(u, v) = δ(u, v). Крім того, ми припускаємо, що |E| > |V| - 1.

Вартість знаходження δ_i при відомому δ_{i-1} дорівнює O (E), тому алгоритм виконується за час, який дорівнює O (kE)= O (E log W).

Одним із таких планів вирішення проблеми є заміна вихідних даних їхніми двійковими наближеннями шляхом послідовних уточнень називається масштабуванням (scaling).

1.1.7 Алгоритм Карпа

Алгоритм було описано в роботі [7].

Нехай G = (V, E) — орієнтований граф із ваговою функцією ω: E→ R, і нехай n = |V|. Середня вага (mean weight) циклу c=$\langle e_1, e_2, \dots e_k \rangle$, де e_j ребро графа, назвемо число

$$\mu(c) = \frac{1}{k}\sum_{i=1}^{k} \omega(e_i)$$

Нехай μ`= min $\mu(c)$, де c пробігає всі (орієнтовані) цикли. Без обмеження спільності (оскільки можна додати додатковий початковий вузол), ми припускаємо, що кожен вузол $v \in V$ доступний для вершини s. Через δ(s, v) позначимо вагу найкоротшого шляху від s до v; нехай δ_k(s, v) буде вагою найкоротшого шляху від s до v, який складається рівно з k ребер (якщо такого шляху немає, розглядаємо δ_k(s, v)=∞).

1.1.8 Мурашиний алгоритм

Алгоритм було описано в роботах [8] [6] [4].

Алгоритм заснований на поведінці колонії мурах, які позначають хороші дороги великою кількістю феромонів. Робота починається з розміщення мурах по кутах діаграми (точок цілей), потім починається рух мурашок - напрямок визначається ймовірнісним методом, виходячи з формули:

$$\boldsymbol{P_i} = \frac{\boldsymbol{l_i^q * f_i^p}}{\sum_{\boldsymbol{k=0}}^{\boldsymbol{N}} \boldsymbol{l_k^q * f_k^p}}$$

$\boldsymbol{P_i}$- ймовірність перетину шляху

l_i- довжина і-го переходу,

f_i - кількість феромонів на і-му переході,

q – величина, що визначає «жадібність» алгоритму,

p — значення, яке визначає «стадність» алгоритму і q + p = 1

Результат не є точним і навіть може бути одним із найгірших, але через ймовірність рішення повторення алгоритму може дати (досить) точний результат.

Розглянемо розрахунок кількості «феромону» наступне рівняння показує кількість феромону для одного ребра для шляху мурахи k. Q - const.

$$\triangle \tau_{ij}^{k}(t) = \frac{Q}{L^{k}(t)}$$

Результатом рівняння являється певний шлях вимірювання, де короткий шлях характеризується високою концентрацією феромонів, а довший – меншою. Далі отриманий результат використовується в наступному рівнянні, щоб збільшити кількість феромону вздовж кожного краю пройденого шляху мурахи. Константа p приймає значення від 0 до 1.

$$\tau_{ij}(t) = \triangle \tau_{ij}(t) + (\tau_{ij}^{k} \times \rho)$$

Основна проблема алгоритму заключається в тому що алгортим чекає повернення кожної мурахи в початкову точку щоб продовжити дослідження шляху що сповільнює подальші розрахунки.

1.1.9 Генетичний алгоритм

Алгоритм було описано в роботах [8] [4].

Генетичний алгоритм — це еволюційний алгоритм пошуку, який використовується для вирішення проблем оптимізації та моделювання шляхом послідовного вибору, комбінування та варіювання параметрів пошуку за допомогою механізмів, що подібні до біологічної еволюції. Особливістю генетичного алгоритму є акцент на використанні оператора «crossover», який виконує операцію рекомбінації розчинів-кандидатів, роль якого подібна до ролі кросинговеру в живій природі. «Батьком-засновником» генетичних алгоритмів є Джон Холланд, книга якого «Адаптація в природних і штучних системах» є фундаментальною для цієї галузі досліджень. Принцип генетичного алгоритму заснований на еволюційному процесі: - Створення первинного покоління; - цикл трансформації (мутації); - цикл

генерації нових особин (кросинговер); - вибір (вибір). Через кілька поколінь залишаються лише найкращі екземпляри, що означає найкращі рішення в контексті проблем, які потрібно вирішити. Задача закодована таким чином, що її розв’язок можна представити у вигляді масиву. У масиві випадковим чином створюється певна кількість початкових елементів із «Людей» або початкової популяції.

Індивідууми оцінюються за допомогою функції придатності, за допомогою якої кожній людині призначається конкретна оцінка придатності, яка визначає її шанси на виживання. Після цього, використовуючи отримані значення, відбираються люди, яким дозволено переходити (відбір). «Генетичні оператори» застосовуються до особин (у більшості випадків це оператор кросинговеру та оператор мутації), створюючи наступне покоління особин. Їх також оцінюють за допомогою генетичних операторів і проводять відбір і мутацію. Генетичні алгоритми зазвичай використовуються для пошуку рішень у великих і складних системах.

1.1.10 Алгоритм перебору грубою силою (Brute Force Algorithm)

Алгоритм було описано в роботах [1] [3] [7].

В інформатиці пошук грубою силою або вичерпний пошук, також званий генерувальним та тестувальним, являється дуже загальною технікою вирішення проблем з алгоритмічною парадигмою, яка складається з систематичного перерахування всіх можливих варіантів вирішення задачі та первірки, чи задовольняє кожен розв’язок постановці проблеми. .

Алгоритм грубої сили, який знаходить дільники натурального числа n, перераховує всі цілі числа від 1 до n і перевіряє, чи кожне з них ділить n без залишку. Наприклад підхід грубої сили до головоломки з вісьмома ферзями полягав би у перевірці всіх можливих розташувань 8 фігур на дошці розміром 64 квадрата та перевірці для кожної розстановки, чи може кожна фігура (ферзь) атакувати іншу.

У той час як пошук грубою силою легко реалізувати і завжди знаходить рішення, якщо воно існує, зусилля щодо реалізації пропорційні кількості можливих рішень, яка для багатьох практичних завдань має тенденцію до швидкого зростання зі збільшенням розміру проблеми (комбінаторний вибух). Таким чином, пошук грубою силою зазвичай використовується, коли розмір проблеми обмежений або коли існують специфічні для проблеми евристики, які можна використовувати для зменшення набору рішень до керованого розміру. Техніка також використовується, коли простота реалізації важливіша за швидкість**.**

Основним недоліком методу грубої сили є те, що для багатьох реальних проблем кількість натуральних варіантів рішень є непомірно великою. Наприклад, якщо ми шукаємо дільники числа, як описано вище, кількість протестованих кандидатів дорівнює заданому числу n. Так, наприклад, якщо n має шістнадцять знаків після коми, для пошуку знадобиться щонайменше 10^{15} комп'ютерних інструкцій, що займе кілька днів на типовому ПК.

1.1.11 Метод Гілок та кордонів (Branch and bound)

Алгоритм було описано в роботах [8] [7].

Метою алгоритму розгалуження та межі є знаходження значення x, яке максимізує або мінімізує значення дійсно значної функції f(x), яка називається цільовою функцією, серед набору S можливих або кандидатських рішень. Множину S називають простором пошуку або допустимим діапазоном. Залишок цього розділу передбачає, що бажана мінімізація f(x); це припущення функціонує без втрати загальності, оскільки можна знайти максимальне значення f(x), знайшовши мінімум g(x) = −f(x). Алгоритм Розгалуженого дерева рішень з обмеженнями працює за двома принципами:

- Він рекурсивно ділить простір пошуку на менші простори, а потім мінімізує f(x) на цих менших просторах;
- Розщеплення означає розгалуження. Лише розгалуження означало б грубе перерахування можливих рішень і тестування їх усіх. Щоб підвищити ефективність пошуку грубою силою, алгоритм Розгалуженого дерева рішень з обмеженнями відстежує обмеження по мінімуму, яке він намагається знайти, і використовує ці обмеження для «обрізання» простору пошуку, усуваючи параметри рішення, які, як доведено для алгоритму, не містять найкращого рішення.

Використання цих принципів в конкретному алгоритмі для конкретної задачі оптимізації вимагає певної структури даних, що представляє собою набори можливих рішень. Таке представлення називається *екземпляром* проблеми. Позначимо множину кандидатів у рішення екземпляра I через S_I. Представлення екземпляра має складатися з трьох операцій:

- обмеження (I) створює два або більше екземпляри, кожен з яких представляє підмножину S_I. (Зазвичай підмножини є непересічними, щоб запобігти повторному відвідуванню алгоритмом одного і того ж рішення-кандидата, але це не є обов'язковим. Проте оптимальне рішення за S_I. має бути принаймні в одній із підмножин.)

- обмеження (I) обчислює нижню межу будь-якого рішення-кандидата в просторі, представленому I, тобто обмеження (I) ≤ f(x) для всіх x у S_I.

- Рішення (I) визначає, чи є I єдиним варіантом рішення. (Якщо це не так, операція може додатково повернути для повторної перевірки можливий розв'язок S_I.) Якщо рішення (I) повертає розв'язок, тоді f(розв'язок (I)) забезпечує верхню межу для оптимального цільового значення. на всьому просторі можливих рішень.

Використовуючи ці операції, алгоритм Розгалуженого дерева рішень з обмеженнями виконує рекурсивний пошук зверху вниз по дереву екземплярів, утвореному операцією розгалуження. Після відвідування екземпляра I він перевіряє, чи обмеження (I) дорівнює або перевищує поточну верхню межу; Якщо так, цю гілку можна сміливо виключити з пошуку, і рекурсія завершиться. Цей крок скорочення зазвичай реалізується шляхом підтримки глобальної змінної, яка запам'ятовує мінімальну верхню межу, що спостерігається серед усіх екземплярів, досліджених до цього моменту.

1.2 Визначення необхідного алгоритму розрахунку найкоротшого шляху

Порівняння алгоритмів також було проведено в книзі [7] яке було взято за основу цього порівняння при реалізації оптимізаційного процесу різання.

Класичні алгоритми знаходження найкоротшого шляху такі як: Алгоритм Дейкстри, Алгоритм Флойда, Алгоритм Беллмана-Форда, Алгоритм A*, Алгоритм Джонсона, Хвильовий алгоритм, Алгоритм Габова, Алгоритм Карпа, більшість з них швидко визначають найкоротший шлях але час розрахунку на прикладах графів більше ніж з 30 точками вказує на те що вони не дуже підходять для розрахунку графів з великою кількістю вершин, також їх реалізація для автоматизації цієї конкретної задачі зазвичай надто складна в реалізації тому було вирішено розглянути більш легкі в реалізації на мовах програмування алгоритми.

Швидкі алгоритми такі як мурашиний, генетичний алгоритм (алгоритм динамічного програмування), через сортування тільки найкоротших шляхів на локальних ділянках з подальшим ігноруванням можливих рішень на пройдених ділянках не перевіряючи в кінці можливість використання цих маршрутів, робить ці алгоритми менш точними при розрахунку.

Алгоритм перебору грубою силою (Brute force) він ідеально точний, завдяки розгляданню абсолютно всіх можливих маршрутів, але для вирішення NP-повної задачі не підходить через занадто великий час розв’язку який в випадку розглянутого завдання з 146 точками буде (n-1)!/2 Маршрутів або 145!/2.

Тому під цю задачу підходить метод гілок та кордонів (Branch and Bound) він не швидший за алгоритми генетичного підбору або мурашиної колонії але набагато більш точний за рахунок самоперевірки відкинутих гілок дерева, також він не настільки точний але час вирішення цього завдання буде на декілька порядків нижче що дозволить розрахувати майже будь-який маршрут для виробництва деталі в часовому обмеженні одного робочого дня на звичайному комп’ютері.

2. Вирішення задачі комівояжера для обробки отворів

2.1 Вирішення задачі комівояжера методом Branch and Bound

Вступ. Вершини графів в цій частині репрезентують вершини отворів між якими пересувається свердло на прискорених подачах, а маршрути відповідно траєкторії переміщення інструментів між центрами кожного отвору.

Дискретні завдання оптимізації над кінцевими множинами мають кінцеве безліч допустимих рішень, які можна перераховувати і вибрати з них найкраще, що забезпечує отримання екстремуму цільової функції (ЦФ). Предметна область таких завдань – процеси дослідження операцій (ДО), теорія якої формується вже кілька десятиліть. Методи вирішення завдань, які теорія має сьогодні - це, насамперед, некласичні методи математичного програмування, лінійне, нелінійне та динамічне програмування, цілечисленне і булеве, геометричне та опукле програмування, принципи оптимальності, мінімаксні та максимінні методи та багато інших. Але серед цього великого арсеналу засобів отримання оптимальних рішень можна назвати один метод, який вніс оригінальний погляд проблеми оптимізації і дозволив інакше сприймати сенс оптимальності рішень.

Насправді, всі дискретні завдання, що розглядаються, завжди мають кінцеве рішення, яке можна отримати простим перебором варіантів. Інше питання, у що це виллється за витратами та за часом. Тому переважна більшість методів ДО орієнтовано скорочення переборів, де це можливо. У різних методах авторам вдавалося значно скорочувати такий перебір, або просто усікати безліч варіантів без їхнього детального розгляду. Тут передбачається зайнятися конкретним методом гілок та кордонів (МГК), розглянути його можливості та переваги, вказати на наявні недоліки.

У процесі перебору варіантів завжди бажано розглядати не всі з них, а лише ті, які слід вважати перспективними та відкидати безперспективні. Як же здійснювати такий вибір? Де шукати оптимальне рішення? Частково відповіді на ці питання були надані в 1960 авторами Алісою Ленд і Елісон Дойг в роботі [2]. Ця публікація відкрила можливість застосування запропонованого ними підходу до пошуку оптимальних рішень різноманітних завдань комбінаторної та дискретної оптимізації, що пізніше стали пов'язувати вже з роботами Літтла, Мурті, Суїні та Керела [3], присвяченими специфічній задачі комівояжера.

Підхід отримав назву метод гілок та кордонів (англ. branch and bound) - як загальний алгоритмічний метод оптимізації. МГК пов'язують із деревом пошуку оптимального (T_{opt}) рішення, яке будується у процесі обробки

вихідних даних завдання. Звідси назва корінь, якому в дереві приписують усі можливі завдання, рішення-гілки, що з'єднують вузли дерева, Використання поняття кордонів та їх розрахунок стимулює чи гальмує зростання гілок у такому дереві. Важливу роль грає процедура розбиття на вузли області допустимих рішень (ОДР) вихідного завдання, тобто. на менші непересічні підмножини та їх оцінювання. Інша процедура, названа процедурою розгалуження, реалізує розбиття на безліч допустимих значень змінної x на підобласті менших розмірів. Ще один важливий елемент МГК - процедура обчислення оцінок, яка полягає у пошуку значень меж ЦФ для вирішення задачі. Обчислення нижньої межі ЦФ (НМЦФ) є найважливішим, ключовим елементом запропонованої схеми. Таким чином, в основі методу гілок і кордонів лежить ідея послідовного розбиття безлічі допустимих рішень на підмножини (стратегія "поділяй і володарюй") та оцінювання одержуваних при розбитті частин. Кожен крок алгоритму розбиття супроводжується перевіркою умови того, чи конкретне підмножина містить оптимальне рішення чи ні.

Математична модель завдання комівояжера

Сформульована задача - завдання цілочислленне. Розглядається n точок для обробки, пов'язаних дорожньою мережею. Нехай $x_{ij} = 1$, якщо інструмент переїжджає з i-ї точки до j-ї та $x_{ij} = 0$, якщо j-та точка не відвідується. Умовно запровадимо (n+1) точка, поєднане з 1-шою точкою, тобто. відстані від (n+1)-ої точки до будь-якої іншої, відмінної від першої, дорівнюють відстаням від першої точки. При цьому, якщо з першої точки можна лише вийти, то в (n+1) точку можна лише прийти. Введемо додаткові цілі змінні, що рівні номеру відвідування цієї точки на шляху. u1=0, un+1=n. Для того, щоб уникнути замкнутих шляхів, вийти з першої точки і повернутися до (n+1) введемо додаткові обмеження, що зв'язують змінні xij та змінні ui (ui цілі негативні числа).

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

При обмежнях:

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad 1 \le j \le n,$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad 1 \le i \le n,$$

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \ge 1, \qquad S \ne \emptyset, S \subset \{1, \dots, n\},$$

$$x_{ij} \in \{0,1\}, \qquad 1 \le i, j \le n.$$

ui-uj+nxij ≤ n-1, j=2..n+1, i=1..n, i≠j, при i=1 j≠n+10≤ui≤n, xin+1=xi1, i=2..n

Верхня подвійна сума - цільова функція завдання, на яку найменше значення слід шукати. Співвідношення нижче-обмеження, що накладаються на змінні. Одноразові суми - вказують на вимогу для заняття одиничними елементами плану-рішення X єдиної позиції в кожному рядку (1≤ i ≤ n) та в кожному стовпці (1 ≤ j ≤ n).

Загальна характеристика завдання. Комівояжер «фр. commis voyageur» — бродячий торговець відвідує населені пункти (n), пов'язані розгалуженою дорожньою мережею, проїзд якою оплачується окремо між i-м, j-м пунктами мережі. Наслідуючи уздовж маршруту (туру), побувати необхідно в кожному з n пунктів (одноразово) і повернутися звідки вийшов. Це - формулювання замкнутого завдання, можна не повертатися до пункту відправлення, і завдання стає незамкненим. Симетричне завдання. Симетрична проблема комівояжера «TSP - traveling salesman problem» виникає, коли вартість (Cij = Cji) в обидва кінці між i-м, j-м пунктами однакова. Вибір маршруту диктується витратами, які мінімізуються торговцем. Асиметрична проблема комівояжера (ATSP) допускає несиметричність матриці Cji ≠Cij. У ще більш загальному випадку, шляхи між деякими містами можуть бути відсутніми, а щоб вони не вибиралися ним вписують у матриці Cji = ∞ нескінченну довжину). Завдання з частковим упорядкуванням «SOP = sequential ordering problem», що вимагає, щоб певне місто i було відвідане до міста j (таких умов може бути кілька). Пошук циклу Гамільтона (HCP = hamiltonian cycle problem) - Виявлення в довільному графі замкнутих шляхів, що проходять через кожну вершину точно один раз. У TSP (симетричній задачі комівояжера) шлях замкнутий і стартувати можна з будь-якого міста (і в будь-який бік). Для n міст існує (n - 1)!/2 різних шляхів. Факторіал росте дуже швидко: n! ~ nn і простір у якому шукається оптимальне рішення виявляється величезним. Наприклад, для 15 міст існує 43 мільярди маршрутів

та для 18 міст вже 177 трильйонів. Саме тому завдання комівояжера цікаве і для тестування різних алгоритмів.

1)Точні методи як знаходять деяке рішення, а й під час закінчення своєї роботи доводять, що це рішення - найкраще. Зазначимо такі: Повний перебір перестановок n-1 чисел (стартова точка фіксована). Практично неможливий для прорахунку при n > 15. Спрямований пошук із поверненнями - перебір варіантів "щодо" деякого рішення з відсіканням шляхів, що мають довжину більшу, ніж найкращий до поточного моменту шлях. Метод гілок та кордонів - найбільш ефективний з відомих метод відсікання "неперспективних" вузлів, за рахунок аналізу матриці відстаней. При пошуку оптимального рішення будується бінарне дерево (у кожному вузлі породжуються 2 гілки: комівояжер йде в деяку точку або не йде до неї). Лінійне програмування застосовується для мінімізації з обмеженнями лінійної форми d x, де x - бінарний вектор розмірності n(n-1)/2, компоненти якого xi рівні 1 або 0, залежно від того, входить i-я дуга в шлях або ні. Вектор d (та сама розмірність) дорівнює довжинам дуг з мережі доріг.

2) Евристичні методи: Жадібний алгоритм; Метод шнурка; Ковзаючий перебір;

3)Імовірнісні методи: метод відпалу; генетичний алгоритм.

Модельний повно зв'язковий n-вершинний орієнтований граф (орграф) задачі є таким, що між кожною (i, j) парою вершин існує 2 дуги з різною вартістю переміщення та Cij ≠ C ji (односторонній рух).

Таким чином, розв'язання задачі комівояжера полягає у відшуканні на нашому орграфі маршруту, що проходить одноразово через усі (n) вершини, при найменшій його вартості. Всі такі існуючі ормаршрути в орграфах називають Гамільтоновими циклами, які задаються одноцикловими перестановками на безлічі вершин графа, а для n міст завжди існує ½ (n - 1)! різних маршрутів.

Визначення 1. Гамільтоновим циклом називається маршрут орієнтованого графа, що включає по одному разу кожну його вершину, виключаючи вихідну.

Визначення 2. Підстановна матриця - (0, 1) - матриця, що відповідає перестановці, цифра позиції якої відповідає рядку, а порядковий номер

позиції стовпцю. Такі матриці називають діагональними, вони реалізують плани TSP.

Визначення 3. Одноцикловою називається перестановка – елемент ступеня n симетричної групи Sn, якому відповідає один цикл.

Простий приклад для 4-х точок для обробки. На цьому прикладі покажемо та опишемо основи МГК, не вирішуючи його. У всіх контрольних положеннях МГК у прикладі є відповіді для самоперевірки. Спочатку покажемо сутність процесу рішення ЗК з усіма елементами методу гілок і кордонів (МГК), з поняттями і позначеннями, що використовуються, але без деталей МГК. На Рисунку 2.1 позначені: корінь дерева пошуку оптимального рішення: корінь містить усі 6 рішень – одноциклових перестановок симетричної групи Sn; інші вузли дерева позначені символами X та Y; X завжди використовується як ім'я поточного вузла, що обробляється; дочірні вузли X (результати розгалуження) позитивний Y і негативний Y - вузол (з підкресленням зверху чи знизу), у якому алгоритм відмовляється здійснювати розгалуження (у ньому цьому кроці маршрут не нарощується новими гілкою і вузлом).

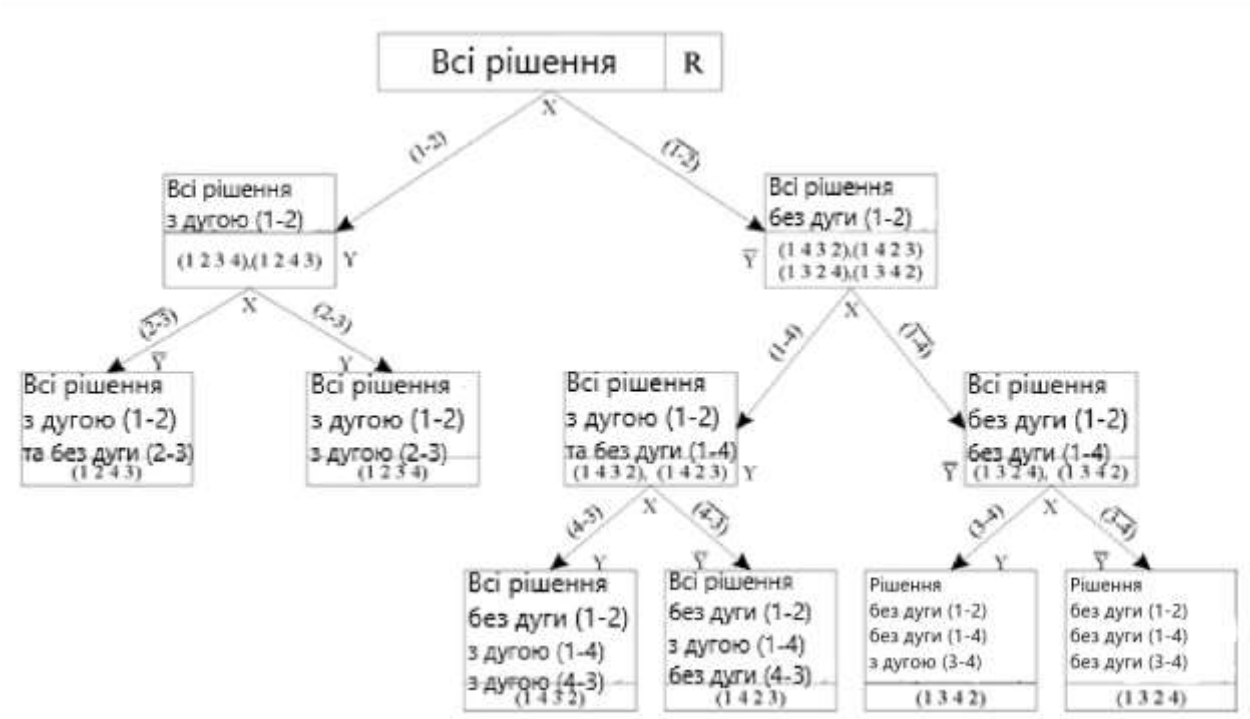


Рис. 2.1 Повне дерево пошуку оптимального рішення задачі комівояжера

Продемонструємо перебіг рішення на невеликому прикладі простим перебором маршрутів. Цей приклад покликаний ілюструвати загальний

принцип та поняття МГК. Деталі будуть подані нижче з вичерпними подробицями. З цією метою сформуємо зведену таблицю для дорожньої мережі із 4-х вузлів права клітинка таблиці.

Рис 2.2 Вихідні дані та приведення матриці вартості переміщення (ij)

Формується безліч S {(i, j)} клітин наведеної матриці $C_{[4]}$ з нульовими значеннями. Починаємо будувати маршрут (Рис 2.1) з вершини 1, включаючи в нього гілку (1-2), негативну гілку (1-2) забороняємо включати до маршруту присвоєнням елементу $C_{12} = \infty$ великої вартості. При цьому безліч рішень в корені дерева розпадається спочатку на два підмножини Y і Y, потім кожне з отриманих підмножин розчленується на дрібніші і процес дроблення підмножин рішень може продовжуватися, поки в кожному з підмножин залишиться по одному рішенню. Але найчастіше до цього не доходить. У процес втручаються інші процедури та розвиток процесу відбувається в інших напрямках. Іноді розпочатий маршрут доводиться кидати не завершивши, і повертатися до раніше перерваного та зупиненого. Річ у тім, що з нарощуванні маршруту відстежується його якість. Якщо якість маршруту можна покращити, повертаючись до перерваного раніше маршруту, то продовжувати неякісний маршрут не варто. Зауважимо, що розв'язання задачі моделюються перестановками n вершин (точок обробки), тобто, елементами симетричної алгебраїчної групи Sn ступеня n. Не всі перестановки є допустимими рішеннями і, звісно, далеко ще не всі серед допустимих будуть оптимальними. У симетричній групі підстановок реалізується розбиття на класи, що не перетинаються, сполучених елементів. Один такий клас складається з безлічі допустимих рішень - це клас одноциклових підстановок, тобто. підстановок утворюють цикл, що включає всі елементи.

Таблиця 2.1 - Подання безлічі рішень ЗК (n = 4) з їх характеристиками

№	перестановки-рішення ЗК	Циклічне представлення	Значення цільової функції ЦФ	Нижня межа ЦФ для всіх рішенн
1	1234	(1234)	9	

2	1243	(1)(2)(34)	8	
3	1324	(1)(23)(4)	10	
4	1342	(1)(234)	11	
5	1423	(1)(243)		
6	1432	(1)(24)3)	7	
7	2134	(12)(3)(4)		
8	2143	(12)(34)		29
9	2314	(123)(4)		
10	2341	**(1234)**	**5+6+11+8=30**	
11	2413	**(1243)**	5+9+7+10=31	
12	2431	(124)(3)		
13	3124	(132)(4)		
14	3142	**(1342)**	16+11+15+13=55	
15	3214	(13)(2)(4)	15	
16	3241	(1342)	13	
17	3412	(13)(24)		
18	3421	**(1324)**	16+12+9+ 8= 45	
19	4123	**(1432)**	14+7+12+13=46	
20	4132	(142)(3)		
21	4213	(143)(2)	6	
22	4231	(14)(2)(3)		
23	4312	**(1423)**	14+15+6+10=45	
24	4321	(14)(23)	12	

Аналіз множини рішень ЗК. У таблицю А включені всі n!=4! =24 можливих перестановок - рішень (Планів Х) завдання, вони впорядковані за їх лексикографічним номером. Для кожної перестановки включено її подання циклами (колонка 3). Наприклад, перестановці (12)(34) у рядку 8 відповідають парі дуг 1→ 2 та 1← 2; 3→ 4 та 3 ← 4, які не реалізують маршруту (єдиного циклу). А перестановці (1234) з рядка 10 відповідає ланцюжок 1→2→3→4→1, тобто. вийшов замкнутий цикл, до якого включаються всі вершини графа доріг, але одноразово. Вибору дуг цього циклу відповідають елементи матриці: C_{12} = 5, C_{23} = 6, C_{34} = 11, C_{41} = 8. Їх сума - значення ЦФ маршруту ЦФ = 5 +6 +11 +8 = 30. З n! = 4! =24 перестановок номерів вершин графа шляхів циклічними (одноцикловими) є лише шість (n-1)! = (4-1)! =3!= 6 (їхні номери 10,11.14, 18,19,23). Маршрути в мережі (переміщення) - допустимі обмеження моделі рішення наведені в таблиці (3-я колонка виділені жирним шрифтом). Серед них можна здійснити вибір кращого туру. Для кожного з шести маршрутів підраховано значення цільової функції (4-я колонка ЦФ: 30,31,55,45,46,45). Серед цих значень найменше дорівнює 30. Хоча воно і перевищує нижню межу ЦФ рівну 29.

Саме це рішення відповідно до дерева пошуку рішень на підставі наведеної матриці вартостей є оптимальним T_{opt}= 30.

Виконуємо наведення матриці цін. Знаходимо коефіцієнти приведення кожного рядка hi (виписуємо їх у рядках праворуч від матриці) і після цього кожного стовпця hj потім підсумовуючи всі h отримуємо константи зведення всієї матриці.

$$H = \sum_{i-1}^{n} h_i + \sum_{j-1}^{n} h_j = 5+6+10+7+1 = 29 = НГЦФ$$

Надалі при пошуку оптимального рішення використовується наведена C [n] матриця вартості. Сума H констант приведення є загальною частиною, яка відповідає всім рішенням ЗК; саме, виходячи з цього, вона і отримана відніманням констант hi приведення всіх ліній матриці та подальшого підсумовування констант один з одним. З викладеного далі випливає, що за жодному турі (допустимому рішенні) це значення може бути зменшено. Навіть T_{opt} перевищує НГЦФ = 29 хоча лише на одиницю.

Справді, порівняння H з табличними, отриманими методом перебору оцінками ЦФ всім турів показує, що M менше їх усіх. З цієї причини H називають нижньою границею цільової функції (НГЦФ), оцінкою ЦФ для безлічі рішень задачі, тобто для кореневого вузла дерева (Рис. 2.1).

2.2 Приклад вирішення задачі методом гілок та кордонів

Вихідні дані для завдання комівояжера.

Число точок n = 5. Граф дорожньої мережі, що зв'язує точки, повний без петель, орієнтований, несиметричний звичайний, задається матрицею C{5} розміру 5×5, дуги графа зважені; рядки матриці відповідають пунктам відправлення, стовпці – пунктам прибуття. Позначимо початкову (вихідну) нижню межу цільової функції символом із значенням Q*=∞ – НГЦФ; вона служить порівняння з нею поточних оцінок. Як тільки вдається сформувати повний маршрут і обчислити для нього ЦФ, значення Q*=∞ замінюється на нове, ним стає обчислена ЦФ, з якою далі порівнюються та оцінюються інші маршрути, Hc= 1+5+6+7+8+3+1 = 31 – значення (оцінка) НГЦФ отримано як сума констант приведення рядків і стовпців матриці вартості. Вага дуг (вартість проїзду по дузі) задається елементами Cij квадратної матриці. Ця матриця цін C[5] - основний об'єкт перетворень у задачі. Усі діагональні

елементи матриці C{5} нічого не повинні включатися в T маршрут (в переміщення), оскільки відповідні їм дуги - це петлі графа і тому їм приписані дуже великі вартості (∞). Інші значення клітинах заповнені поспіль наступними натуральними числами (у прикладі спірально за годинниковою стрілкою від клітини C_{11}=1 до центру матриці). Робота з пошуку маршруту проводиться на постійному графі переміщеннь (з вершинами і дугами) і на дереві, що послідовно вирощується, (з вузлами і гілками) пошуку рішень. Поточному вузлу дерева пошуку рішення на кожному кроці алгоритму присвоюється ім'я X, а вузлам-результатам розгалуження ім'я Y-позитивному, що включається в тур, і ім'я Y-негативному вузлу, що не включається до T тур (маршрут).

Багатокроковий алгоритм пошуку на дереві рішень

Успішне розв'язання задачі досягається при забезпеченні наявності хоча б одного нуля в кожному рядку і кожному стовпчику матриці C{5} за їх "діагональному" розташуванні. В цьому випадку маршрут можна прокласти через клітини з нулями і вартість маршруту буде найменшою. Далі ми побачимо, що таке положення нулів у матриці забезпечується багаторазовим застосуванням процедури приведення матриці {5} та її перетвореннями, що скорочують розмірність до dim C{5} = 2×2 . У момент досягнення матрицею розмірності 2×2 вона забезпечує однозначне визначення вершин, що включаються до маршруту. Розглядаються характеристики (di, dj) дуг графа доріг у рядку di і в стовпці dj матриці, що змінюється C{n} вартостей. Вузли X дерева розгалужуються на "позитивний" та "негативний". Безліч висячих «відкладених» вузлів (листя) у дереві рішень позначається символом, а початковому значенню нижньої межі цільової функції задається Q*= ∞. Кроки алгоритму бувають двох типів: звичайні та з поверненням у відкладений вузол; другий тип реалізує процедуру "back tracking":

На кожному кроці алгоритму розв'язання задачі комівояжера виконуються (визначаються):

- Приведення матриці C[i,j] для отримання нульових елементів у кожних її рядку та стовпці;

- Формування множини клітин S з нульовими значеннями в матриці та їх характеристик di, dj, ∑ij у формі плоскої таблиці;

- Обчислюються max ∑ij та визначається клітинка (k, l) з max яка не включається до маршруту;

- Модифікується матриця: з матриці видаляються k рядок і l-й стовпець клітини (k, l);

- Обчислюються оцінки НГЦФ для позитивного та негативного вузлів дерева пошуку рішення;

- У маршрут включається елемент (k, l) з кращою (меншою) оцінкою ω(k, l) НГЦФ

- Порівнюються оцінки між собою вихідна Q* та обчислена H на поточному кроці;

- Обчислюється розмірність dimC[n] матриці та перевіряється рівність цієї розмірності з dim 2×2;

- Формується безліч висячих вузлів дерева рішень з їх оцінками НГЦФ.

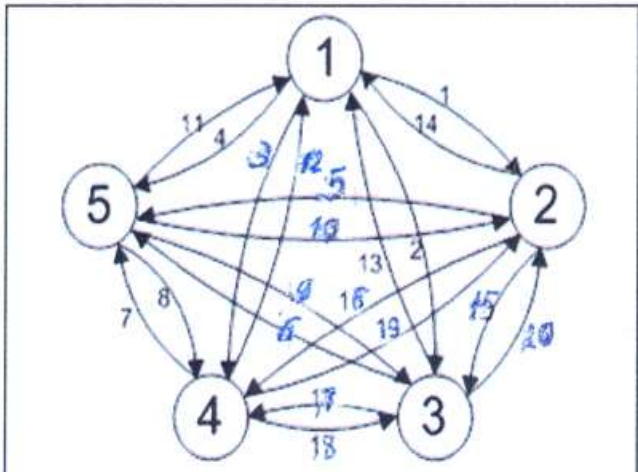
- Наводиться фрагмент дерева пошуку рішень для наступного кроку.

Ітераційні кроки: Формування туру T починається вибором 1-ї дуги (k, l) графа доріг включення їх у маршрут T, оформлюваний як послідовність дуг T = {(a,b)(c,d)(ef)(hg). ....}.

КРОК 1. Процедура приведення вихідної матриці C{n}(отримання нулів у ній). У кожному i-му рядку C{n} знаходимо найменший елемент і віднімаємо його з усіх елементів рядка. Проходимо по всіх рядках матриці. Для такої зміненої матриці в кожному j-му стовпці знову знаходимо найменший елемент і віднімаємо його з усіх елементів кожного j-го стовпця. Проходимо по всіх стовпцях. Знайдені найменші елементи називають константами приведення рядків та стовпців (ліній матриці) та позначають hi, i =1(1)2n. Константою Hc приведення всієї матриці називають суму ∑hi

констант ліній C{n}. Ця сума Нс (C{5}) = ∑hi = 1+5+6+7+8+3+1= 31 є оцінкою НГЦФ всіх розв'язків задачі. Приведення поточних матриць C{n}на кроках алгоритму виконується лише у разі відсутності нулів у її деяких рядках та/або стовпцях, і нові константи визначаються тільки для таких ліній.

Нижче в таблицях показані карта переміщень, дії та їх результати щодо приведення матриці завдання.



$C_{[5]}$ =

	1	2	3	4	5	Min h_i
1	∞	1	2	3	4	1
2	14	∞	15	16	5	5
3	13	20	∞	17	6	6
4	12	19	18	∞	7	7
5	11	10	9	8	∞	8
$Minh_i$	3	0	1	0	0	31

Матриця вартості дуг графу C

Зведення матриці C{n}
Вибір min елементів hi ліній по рядкам і стовбцям матриці C{n}

Рис.2.3 карта переміщень та зведення матриці

Для наведеної матриці рядкові константи hi приписані в стовпці праворуч, а стовпцеві - у рядку знизу матриці {5}. Константа приведення матриці (C{5}) = ∑hi = 31 обчислюється один раз.

$C_{[5]}$ =

	1	2	3	4	5	
1	∞	0	1	2	3	1
2	9	∞	10	11	0	5
3	7	14	∞	11	0	6
4	5	12	11	∞	0	7
5	3	2	1	0	∞	8

Зведення матриці по рядкам

$C_{[5]}$ =

	1	2	3	4	5	
1	∞	0	0	2	3	1
2	6	∞	9	11	0	5
3	4	14	∞	11	0	6
4	2	12	10	∞	0	7
5	0	2	0	0	∞	8
	3	0	1	0	0	

Зведення матриці по стовбцям

Рис.2.4 Матриця вартості з константами приведення рядками та стовпцями

Тепер нулі в клітинках матриці є у всіх рядках і стовпцях і навіть можливо більш ніж по одному. На жаль позиції, зайняті нулями, не утворюють діагонального плану X. Інакше рішенням завдання був би якраз цей план і вартість відповідного йому маршруту дорівнювала оцінці НГЦФ вихідної C{5} матриці, тобто. H = 31 = 1 +5 +6 +7 +8 +3 +1.

Формування множини S нульових клітин та їх характеристик. Перед створенням таблиці S = {(i, j) | Cij = 0} нульових клітин виконано приведення вихідної матриці цін C{5}.

- Формуємо безліч S із клітин матриці C{5} з нульовими значеннями (верхній рядок S={(i,j)}) визначаємо і вписуємо в таблицю значення характеристик у рядках di і в стовпцях dj дуг-претендентів на включення в тур. Для кожної Cij = 0 клітини наведеної матриці C{5} знаходимо di - найменше значення у рядку і dj - найменше значення у стовпці, виключаючи саму Cij = 0 клітинку. Наприклад, для клітини таблиці S(i, j) = (2, 5)= 0 у матриці C{5} маємо: у другому рядку (i =2) менший елемент $C_{2,1}$= 6, тобто. di = 6. У (j=5) п'ятому стовпці три елементи нулі (крім верхнього), тобто. dj = 0. На кожному кроці визначається множина клітин S = {(i, j) | Cij = 0} у наведеній матриці C{5} вартості – безліч претендентів-дуг для включення першої в оптимальний тур. Загальні правила заповнення таблиці S набувають вигляду:

$$S = [(ij)|C_{ij} = 0], di = \min_r[C_{ir} - C_{ij}|C_{ij} = 0], dj = \min_r[Crj - Cij|Cij = 0]$$

Знаходимо в множині клітин матриці C{5} з нульовими значеннями ту клітинку, яка інші перевершує за сумою ∑ij = 6 = di + dj . Ця клітина argmax ∑ij = (k, l) = (2, 5);

Таблиця 2.2 Таблиця S={(i, j)} нульових клітинок Cij = 0 матриці C{5} першого кроку

Для S={I,j}	(1,2)	(1,3)	(2,5)	(3,5)	(4,5)	(5,1)	(5,3)	(5,4)
di	0	0	6	4	2	0	0	0
dj	2	0	0	0	0	2	0	2
∑ij	2	0	6	4	2	2	0	2
argmax ∑ij			6					

Обчислюються max ∑ij та визначається клітина (k, l) з max яка включається до маршруту;

Ці значення, наприклад, для клітинки (1,2) у першому стовпці ∑ij = di + dj = 0+2 = 2 сумуються. І для всіх нульових клітин S для наведеної матриці C{5} обробляються аналогічно. Серед таких стовпцевих сум знаходиться найбільша та визначається відповідна максимальній сумі (max∑ij = 6) клітина

(k,l), argmax ∑ij = arg 6 =(k, l) = (2, 5). Після того, як визначено координати клітини, модифікується матриця C{5}.

Модифікація матриці C{5} - з матриці видаляються k-й рядок і l-й стовпець;

У процесі реалізації багатокрокового алгоритму відбувається обробка вихідних даних завдання, мета обробки – формування маршруту одноразового обходу вершин графа, що моделює дорожню карту місцевості, за найменшої вартості маршруту. Маршрут формується послідовним вибором дуг графа включення до нього. Кожна чергова гілка дерева рішень знаходиться (вибирається) за певними правилами-обмеженнями і при задоволенні всіх обмежень завдання дуга включається до поточного маршруту. Крім того, реалізується так званий «back tracking – пошук з поверненням». Нарощування довжини маршруту виконується з контролем значення цільової функції, що обчислюється, і умови не перевищення нею раніше відкладених проміжних «залишених на потім» значень. У разі перевищення ЦФ такого значення алгоритм переходить до продовження такого відкладеного варіанту (наприклад, на кроці 4), відновлюючи поточний стан процесу (матриці) пошуку рішення, на момент відкладення рішення. Виконується акт повернення "вгору" по дереву рішень до відкладеного раніше варіанту (вузла).

- Перетворимо матрицю C{5}. Видаляємо рядок з номером k = 2 та стовпець з номером l = 5; при цьому розмірність матриці цін змінилася і стала рівною (5-1)×(5-1)=4×4; З матриці при видаленні ліній частина клітин з нульовими значеннями може бути видаленими; їх доводиться одержувати знову;

- Запобігаємо зациклюванню, тобто. забороняємо наступний вибір і включення клітини (p, q) до маршруту, де q = k = 2, p = l = 5, а в позицію (l, k) = (5, 2) вписуємо велике значення Cpq =∞, роблячи її забороненою для вибору та включення в маршрут. Результати цих дій відображені у правій матриці {4} нижче.

$C_{[4]}$ =

	1	2	3	4	h_i
1	∞	0	0	2	0
3	4	14	∞	11	4
4	2	12	10	∞	2
5	0	∞	0	0	0

Зведення матриці по рядкам

$C_{[4]}$ =

	1	2	3	4
1	∞	0	0	2
3	0	10	∞	7
4	0	10	8	∞
5	0	∞	0	0
h_i	0	0	0	0

Зведення матриці по стовбцям

Рис.2.5 Модифіковані матриці вартості 4×4 першого кроку

- Обчислюємо нову константу приведення матриці розмірністю 4×4, H =∑hi=4+2=6;

- Обчислюються оцінки НГЦФ для позитивного та негативного вузлів дерева пошуку рішення; Обчислення НГЦФ для негативного вузла ω(Y) = ω(25) = ω(X) +∑(kl) =31 +6 = 37 та обчислення НГЦФ для позитивного вузла ω(Y) = ω(25) = ω(X) + H = 31 +6 = 37; Формули обчислення розрізняються, проте оцінки іноді збігаються, порівняння оцінок показує, що вони в цій ситуації рівні. Як зробити, який вузол вибрати? Якщо формуємо маршрут T, то включаємо до нього позитивну вершину (2, 5), тобто. тоді T = {(2,5)(..)…}. Задаємо співвідношенням екстремальності обраному для розгалуження вузлу (2, 5) нове ім'я X = argmin { ω (Y = (25)) = 37, ω (Y = (25)) = 37} = (2, 5) або X = Y (k, l) = Y(2, 5). Цей вузол розгалужуватиметься далі.

- Порівняння поточного значення НГЦФ із вихідним значенням (Q*=∞)≤ ω(25)=37 ? НІ. Обчислюється розмірність матриці (в результаті перетворення) та перевіряється її рівність розмірності 2×2 = 4×4? Відповідь НІ.

- У безліч висячих вузлів дерева пошуку рішень включаємо вузол (Y)=(2,5) з його оцінкою НГЦФ =ω (25) = 37.

- Фрагмент дерева рішень для першого кроку набуває вигляду:

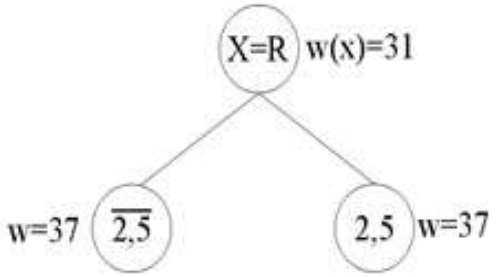


Рис.2.6 Фрагмент дерева пошуку рішень після 1-го кроку

КРОК 2. Оскільки відповідь питання розмірності матриці C{4} на КРОКУ 1 негативна (НІ) переходимо до вибору чергового вузла (k, l) включення в маршрут. Матриця цін C{4} вже сформована та наведена (її константа приведення H = 6, а її розмірність dim C{4} = 4×4). Усі міркування 1-го кроку залишаються справедливими й у 2-го кроку.

Формуємо безліч S із клітин матриці C{4} з нульовими значеннями, визначаємо та вписуємо в таблицю (як і раніше) значення характеристик у рядках di і в стовпцях dj дуг-претендентів на включення в тур. Формули обробки елементів матриці {4} залишаються без зміни.

Таблиця 2.3 Таблиця S{(i,j)} нульових клітин Cij = 0 матриці C{4} другого кроку

Для S={I,j}	(1,2)	(1,3)	(3,1)	(4,1)	(5,1)	(5,3)	(5,4)
di	0	0	7	8	0	0	0
dj	10	0	0	0	0	0	2
∑ij	10	0	7	8	0	0	2
argmax ∑ij	10						

Знаходимо в множині клітинок матриці C{4} з нульовими значеннями те значення, яке перевищує інші за сумою max∑ij = di + dj =10. Аргументом цієї суми є клітина argmax∑ij = (k, l) = (1, 2);

Перетворимо матрицю C{4}: видаляємо рядок з номером k = 1 і стовпець з номером l = 2; при цьому розмірність матриці цін змінилася і стала рівною (4-1)×(4-1)=3×3; нульові клітини в лініях матриці {3} збереглися. Приведення матриці не потрібно, H = 0

Запобігаємо зациклюванню, тобто. забороняємо вибір клітини (p, q), де q = k = 1, p = l = 5, а в позицію (l, k) = (5, 1) вписуємо велике значення Cpq = C51 =

∞, роблячи її забороненою для вибору та включення в маршрут. Ці дії відображені у правій матриці {3} нижче:

Зведення матриці по рядкам	Зведення матриці по стовпцям
C[3] =	C[3] =

	1	3	4
3	0	∞	7
4	0	8	∞
5	∞	0	0

Рис.2.7 Модифіковані матриці вартості 3×3 другого кроку

- Обчислюємо константу приведення скороченої матриці 3×3, H=∑hi=0+0 =0;

- Обчислюються оцінки НГЦФ для позитивного та негативного вузлів дерева пошуку рішення; Обчислення НГЦФ для негативного вузла ω(Y) = ω(12) = ω(X) +∑(kl) =37 +10 = 47 та обчислення НГЦФ для позитивного вузла ω(Y) = ω(1,2) = ω(X) + H = 37 + 0 = 37; формули обчислення розрізняються, оцінки на КРОКУ 2 не збігаються, порівняння оцінок показує, що значення у цій ситуації не рівні. Який вузол вибрати? Ми формуємо маршрут T, то включаємо до нього позитивну вершину (1,2), з меншою НГЦФT = {(1,2)(2,5)…}. Задаємо співвідношенням екстремальності обраному для розгалуження вузлу (1, 2) дерева нове ім'я X = argmin { ω (Y = (1, 2)) = 47, ω (Y = (1, 2)) = 37} = (1, 2) або X = Y(k, l) = Y(1, 2).

- Порівняння поточного значення НГЦФ з вихідним значенням (Q*=∞)≤ ω(1,2)=37? **НІ**.

- Для обчисленої розмірності матриці перевіряється рівність її розмірності 2×2=3×3? **НІ.**

- У безліч висячих вузлів включаємо вузол (Y)=(1,2) з його оцінкою НГЦФ = ω(12)=47 B = {(25) (1,2)...}

- Фрагмент дерева рішень для другого кроку набуває вигляду:

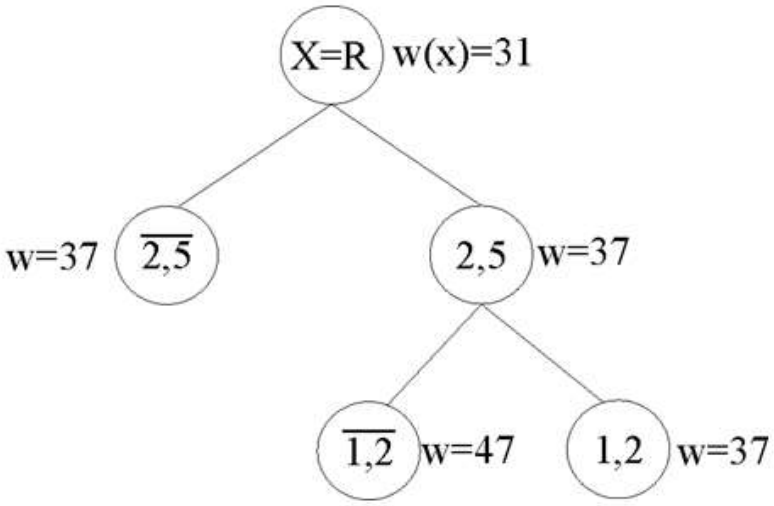


Рис.2.8 Фрагмент дерева пошуку рішення після 2-го кроку

КРОК 3. Оскільки відповідь питання розмірності матриці C{3} на КРОКУ 2 негативна (НІ) переходимо до вибору чергового вузла (k, l) включення в маршрут. Матриця цін C{3} вже сформована і наведена (константа приведення H = 0) (її розмірність 3×3).

Формуємо безліч S із клітин матриці C{3} з нульовими значеннями, визначаємо та вписуємо в таблицю (як і раніше) значення характеристик у рядках di і в стовпцях dj дуг-претендентів на включення в тур. Формули обробки елементів матриці C{3} залишаються без зміни.

Таблиця 2.4 Таблиця S{(i,j)} нульових клітинок Cij = 0 матриці C{3}

Для S={I,j}	(1,2)	(1,3)	(3,1)	(4,1)
di	7	8	0	0
dj	0	0	8	7
∑ij	7	8	8	7
argmax ∑ij		8		

- Знаходимо в множині клітинок матриці C{3} з нульовими значеннями ту клітинку, яка інші перевищує за сумою max∑ij = 8 =di + dj . Аргументом цієї суми є клітина argmax ∑ij = 8 = (k, l) = (4, 1); у таблиці два стовпці зі значенням 8, беремо перше;

- Перетворимо матрицю C{3}: видаляємо рядок з номером k = 4 і стовпець з номером l = 1; при цьому розмірність матриці цін змінилася і стала рівною (3-1)×(3-1)=2×2;

- Запобігаємо зациклюванню, тобто. забороняємо вибір клітини (p, q), де q = k = 4, p = l = 1, а у позицію (l, k) = (5, 4) вписуємо велике Cpq = C_{54} = ∞ значення, роблячи її забороненою для вибору та включення в маршрут. Ці дії відображені у правій матриці C{2} нижче

C[2] =	C[2] =
3 4 h 3 ∞ 7 7 5 0 ∞	3 4 3 ∞ 0 5 0 ∞ 0 0
Зведення матриці по рядкам	Зведення матриці по стовпцям

Рис.2.9 Модифіковані матриці вартості 2×2 третього кроку

- Обчислюємо константу приведення скороченої матриці 2×2, H=∑hi=7+0=7

- Обчислюються оцінки НГЦФ для позитивного та негативного вузлів дерева пошуку рішення; обчислення НГЦФ для негативного вузла ω(Y) = ω(4, 1) =ω(X) +∑(kl)=37+8=45 та обчислення НГЦФ для позитивного вузла ω(Y)=ω(4, 1)= ω(X) + H = 37 +7 = 44; формули обчислення оцінки різні, порівняння оцінок показує, що у цій ситуації менше значення НГЦФ має гілка ω(1,2)=44. Тому цей вузол обраний черговим для маршруту T = {(4, 1) (1, 2) (2, 5) ... }. Задаємо співвідношенням екстремальності вибраному вузлу (4, 1) нове ім'я X = argmin { ω (Y = (4, 1)) = 45, ω (Y = (4, 1)) = 44} = (4,1) або X = Y (k, l) = Y (4, 1).

- Порівняння поточного значення НГЦФ з вихідним значенням (Q*=∞)≤ ω(1,2)=44 ? **НІ.**

- Для обчисленої розмірності матриці перевіряється її рівність розмірності 2×2=2×2? **ТАК.**

- У безліч висячих вузлів включаємо вузол (Y)=(4,1) з його оцінкою НГЦФ =ω(4 1)=47 B={(2,5) (1,2)(4, 1) ...} А далі цей КРОК має відмінність від попередніх.

- Матриця C{2} вартості з розміром 2×2 дозволяє завершити побудову маршруту. Вибраними можуть бути тільки дві дуги (3,4) вагою 7 і (5,3) вагою 0 їх і включаємо в маршрут, завершуючи його, T={(4, 1)(1, 2)(2, 5)(5 , 3)(3, 4)} цільова функція цього маршруту визначається сумою ЦФ = 12+1+5+9+17 = 44. Для інших маршрутів цей реально прорахований маршрут може використовуватися як новий критерій ефективності (критерію порівняння) та підставою для вибору найкращого маршруту.

- Рішення прикладу дозволяє замінити вихідну оцінку НГЦФ Q*=∞ на більш реальну оцінку Q*=ЦФ = 44.

- Фрагмент дерева рішень для третього кроку набуває вигляду:

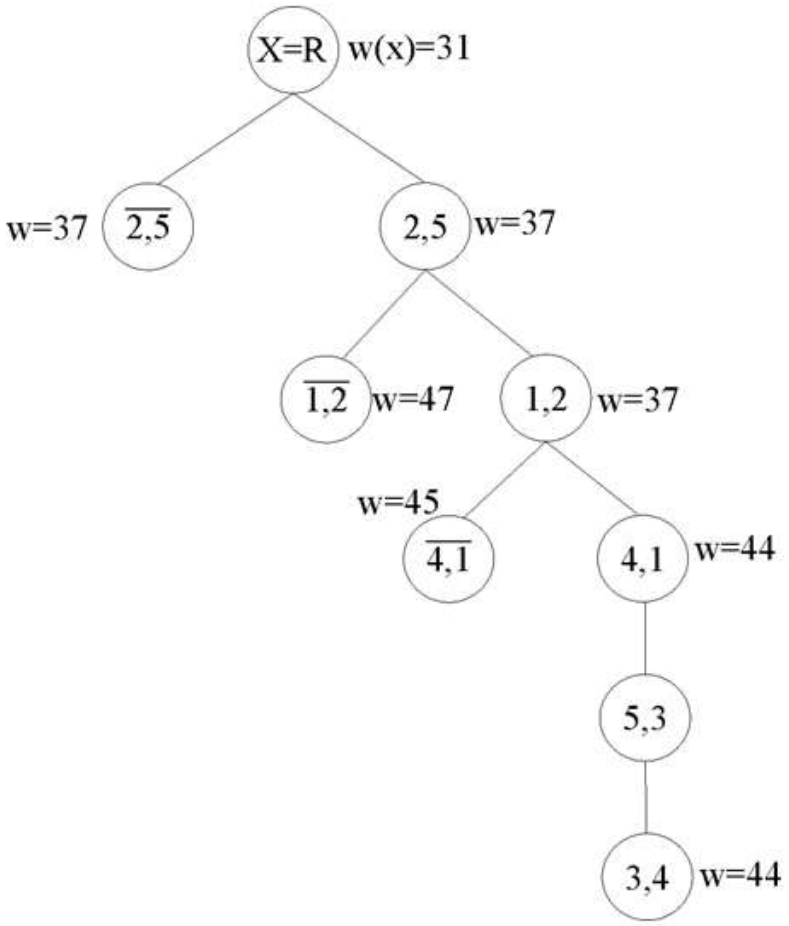


Рис.2.10 Фрагмент дерева пошуку рішень після 3 кроку

КРОК 4. Оскільки відповідь на питання про розмірність матриці C{3} на КРОКУ 3 позитивна (ТАК) в алгоритмі розв'язання задачі відбувається низка змін. Завершився пошук одного з варіантів (1-го) маршруту з оцінкою НГЦФ, яка стала критеріальною (Q*=44). Переходимо до відповіді на запитання чи слід шукати інші маршрути і чи виявляться вони кращими за знайдене? Відповідь на питання допомагає вказати безліч відкладених рішень з оцінками НГЦФ і збереженими станами. Якщо серед відкладених є оцінка НГЦФ менша знайденої ω=44, то, можливо, продовжуючи розрахунок в цій частині дерева, в результаті отримаємо інший маршрут, краще вже знайденого, а при аналізі всіх відкладених рішень буде знайдено оптимальний (невдосконалений) маршрут. Діємо далі так. У множині серед відкладених є оцінка w = 37 < 44, яка відповідала відмові на КРОКУ 1 від вибору вузла (k, l) = (2,5) для включення в маршрут. Матриця цін у ситуації була наведена C{5}.Ми повертаємо в матрицю дугу (2,5), але забороняємо її вибір, вважаючи $C_{25} = \infty$, при якому в рядку 2 матриці {5} пропадає нульова клітинка. Її повернення оцінюється як hi = 6 для 2-го рядка, що збільшує колишню оцінку НГЦФ. Інші нулі в C{5} зберігаються. Послідовність

маршруту – порожня, не містить жодної дуги. Константа приведення матриці H = 6.

	1	2	3	4	5	h_i
1	∞	0	0	2	3	0
2	6	∞	9	11	∞	6
3	4	14	∞	11	0	0
4	2	12	10	∞	0	0
5	0	2	0	0	∞	0

	1	2	3	4	5
1	∞	0	0	2	3
2	0	∞	3	5	∞
3	4	14	∞	11	0
4	2	12	10	∞	0
5	0	2	0	0	∞

Рис.2.11 Відновлені стани завдання при поверненні до відкладених рішень. H = 6 четвертого кроку

Формуємо безліч S із клітин матриці C{5} з нульовими значеннями, визначаємо та вписуємо в таблицю (як і раніше) значення характеристик у рядках di і в стовпцях dj дуг-претендентів на включення в тур. Формули обробки елементів матриці {5} залишаються без зміни.

Таблиця 2.5 Таблиця S{(i, j)} нульових клітин Cij = 0 матриці C{5} вартості кроку 4

Формуємо множину S по матриці C{n}								
Для S={I,j}	(1,2)	(1,3)	(2,5)	(3,5)	(4,5)	(5,1)	(5,3)	(5,4)
di	0	0	6	4	2	0	0	0
dj	2	0	0	0	0	2	0	2
∑ij	2	0	6	4	2	2	0	2
argmax ∑ij				4				

- Знаходимо в множині клітин матриці C{5} з нульовими значеннями ту клітинку, яка інші перевищує за сумою max∑ij = 4 = di + dj . Аргументом цієї суми є клітинка argmax ∑ij = 4 = (k, l) = (3, 5);

- Перетворимо матрицю C{5}: видаляємо рядок з номером k = 3 та стовпець з номером l = 5; при цьому розмірність матриці цін змінилася і стала рівною (5-1)×(5-1)=4×4; при цьому зникла нульова клітинка в 4-му рядку.

- Запобігаємо зациклюванню, тобто. забороняємо вибір клітинки (p, q), де q = k = 5, p = I = 3, а у позицію (l, k) = (5, 3) вписуємо велике Cpq =

$C_{53} = \infty$ значення, роблячи її забороненою для вибору та включення в маршрут. Ці дії відображені у правій матриці C{4} нижче

	1	2	3	4	
1	∞	0	0	2	0
2	0	∞	3	5	0
4	2	12	0	∞	2
5	0	2	∞	0	0

H=2

	1	2	3	4
1	∞	0	0	2
2	0	∞	3	5
4	0	10	8	∞
5	0	2	∞	0

Рис.2.12 Модифіковані матриці вартості 4×4 четвертого кроку 2

- Обчислюємо константу приведення скороченої матриці 4×4, H=∑hi=2+0=2;

- Обчислюються оцінки НГЦФ для позитивного та негативного вузлів дерева пошуку рішення; обчислення НГЦФ для негативного вузла ω(Y) = ω(3, 5) =ω(X) +∑(kl)=37+4=41 та обчислення НГЦФ для позитивного вузла ω(Y)=ω(3, 5)= ω(X) + H = 37 +2 = 39; порівняння оцінок показує, що у цій ситуації менше значення НГЦФ має гілка ω(3,5)=39. Тому цей вузол обраний черговим для включення в маршрут T = {(3, 5) .... } Задаємо співвідношенням екстремальності обраному для розгалуження вузлу (3, 5); нове ім'я X = argmin { ω (Y = (3, 5)) = 41, ω (Y = (3, 5)) = 39} = (3,5) або X = Y (k, l) = Y (3, 5).

- Порівняння поточного значення НГЦФ із вихідним значенням (Q*=44)≤ ω(3,5)=39? НІ.

- Обчислюється розмірність матриці та перевіряється рівність її розмірності 2×2=4×4? НІ.

- У безліч висячих вузлів включаємо вузол (Y)=(3, 5) з його оцінкою НГЦФ =ω(3 5)=41 B={(35) ...}

- Фрагмент дерева рішень для четвертого кроку набуває вигляду:

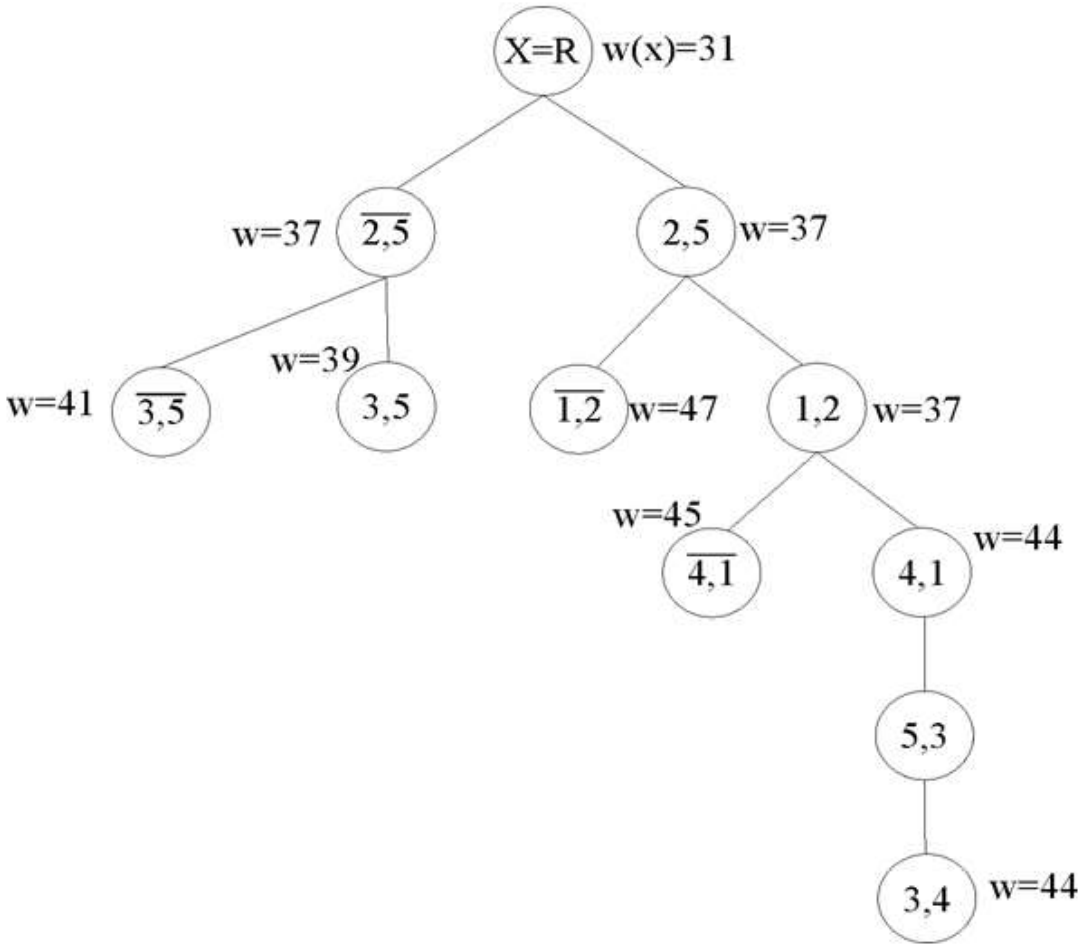


Рис.2.13 Фрагмент дерева пошуку рішення після 4 кроку

КРОК 5. Оскільки відповідь питання розмірності матриці C{4} на КРОКУ 4 негативна (**НІ**) переходимо до вибору чергового вузла (k, l) включення в маршрут. Матриця цін C{4} вже сформована та наведена (коефіцієнт приведення H = 2) (її розмірність 4×4).

Формуємо безліч S із клітин матриці C{4} з нульовими значеннями, визначаємо та вписуємо в таблицю (як і раніше) значення характеристик у рядках di і в стовпцях dj дуг-претендентів на включення в тур. Формули обробки елементів матриці C{3} залишаються без зміни.

Таблиця 2.6 S {(i, j)} нульових клітинок Cij = 0 матриці C{4} вартості 5-го кроку

Для S={I,j}	(1,2)	(1,3)	(3,1)	(4,1)	(5,1)	(5,3)	(5,4)
di	0	0	7	8	0	0	0
dj	10	0	0	0	0	0	2
∑ij	10	0	7	8	0	0	2
argmax ∑ij	10						

- Знаходимо в множині клітин матриці C{4} з нульовими значеннями ту клітинку, яка інші перевершує за сумою max∑ij = 8=di + dj . Аргументом цієї суми є клітинка argmax ∑ij = (k, l) = (4, 1);

- Перетворимо матрицю C{4}: видаляємо рядок з номером k = 4 та стовпець з номером I = 1; при цьому розмірність матриці цін змінилася і стала рівною (4-1)×(4-1)=3×3; зникла клітина з нульовим значенням у 2-му рядку

- Запобігаємо зациклюванню, тобто. забороняємо вибір клітини (p, q), де q = k = 1, p = l = 4, а у позицію (l, k) = (4, 1) вписуємо велике Cpq = C_{41} = ∞ значення, роблячи її забороненою для вибору та включення в маршрут. Ці дії відображені у правій матриці C{3} нижче

	2	3	4	h_i
1	0	0	∞	
2	∞	3	5	3
5	2	∞	0	

	2	3	4
1	0	0	∞
2	∞	0	2
5	2	∞	0

Рис.2.14 Модифіковані матриці вартості 3×3 п'ятого кроку

- Обчислюємо константу приведення скороченої матриці 3×3, H = ∑hi = 3 + 0 = 3;

- Обчислюються оцінки НГЦФ для позитивного та негативного вузлів дерева пошуку рішення; обчислення НГЦФ для негативного вузла ω(Y) = ω(4,1) = ω(X) +∑(kl) =39 +8 = 47 та обчислення НГЦФ для позитивного вузла ω(Y) = ω(4,1) = ω(X) + H =39 +3 = 42; формули обчислення розрізняються, оцінки на КРОКУ 5 не збігаються, порівняння оцінок показує, що вони у цій ситуації не рівні. Як вчинити, який вузол вибрати? Ми формуємо маршрут T, що включає в нього позитивну вершину (4,1), з меншою НГЦФ T = {(4,1)(1,2)…}. Задаємо співвідношенням екстремальності обраному для розгалуження вузлу (4,1) нове ім'я X = argmin { ω(Y=(4,1)) =47, ω(Y=(4, 1))=42}=(4, 1) або X = Y (k, l) = Y (4, 1).

- Порівняння поточного значення НГЦФ з вихідним значенням (Q * = 44) ≤ ω (4,1) = 42? **НІ**.

- Обчислюється розмірність матриці та перевіряється її рівність розмірності 2×2=3×3? **НІ**.

- У безліч висячих вузлів включаємо вузол (Y)=(4,1) з його оцінкою НГЦФ =ω(41)=47 B = {(35) (4,1)...}

- Фрагмент дерева рішень для п'ятого кроку набуває вигляду:

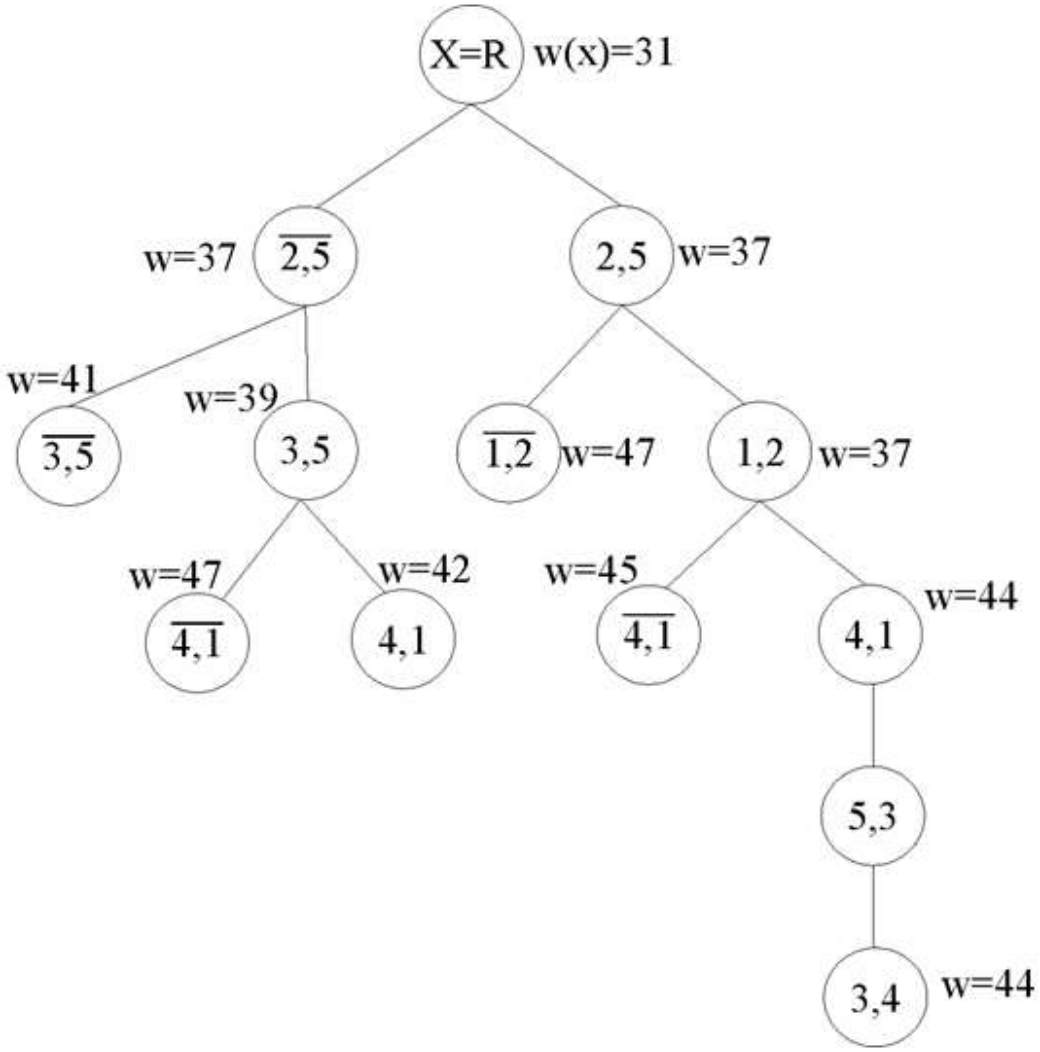


Рис.2.15 Фрагмент дерева пошуку рішення після 5 кроку

КРОК 6. Оскільки відповідь питання розмірності матриці C{4} на КРОКУ 5 негативна (НІ) переходимо до вибору чергового вузла (k, l) включення в маршрут. Матриця цін C{3} вже сформована і наведена (H = 3) (її розмірність 3×3).

Формуємо безліч S із клітин матриці C{3} з нульовими значеннями, визначаємо та вписуємо в таблицю (як і раніше) значення характеристик у рядках di і в стовпцях dj дуг-претендентів на включення в тур. Формули обробки елементів матриці C{3} залишаються без зміни.

Таблиця 2.7 S{(i, j)} нульових клітинок Cij = 0 матриці C{3} вартості 6-го кроку

Для S={I,j}	(1,2)	(1,3)	(2,3)	(5,4)
di	0	0	2	2
dj	2	0	0	2
∑ij	2	0	2	4
argmax ∑ij				4

- Знаходимо в множині клітин матриці C{3} з нульовими значеннями ту клітину, яка інші перевищує за сумою max∑ij = 4 = di + dj . Аргументом цієї суми є клітина argmax∑ij = (k, l) = (5, 4);

- Перетворимо матрицю C{3}: видаляємо рядок з номером k = 5 і стовпець з номером l = 4; при цьому розмірність матриці цін змінилася і стала рівною (3-1)×(3-1)=2×2;

- Запобігаємо зациклюванню, тобто. забороняємо вибір клітини (p, q), де q = k = 4, p = I = 5, а у позицію (l, k) = (5, 4) вписуємо велике Cpq = C_{54} = ∞ значення, роблячи її забороненою для вибору та включення в маршрут. Ці дії відображені у матриці C{2} нижче:

	2	3
1	0	∞
2	∞	0

Рис.2.16 Модифікована матриця вартості 2×2 6-го кроку

- Обчислюємо константу приведення скороченої матриці 2×2, H = ∑hi = 0;

- Обчислюються оцінки НГЦФ для позитивного та негативного вузлів дерева пошуку рішення; обчислення НГЦФ для негативного вузла ω(Y) = ω(54) = ω(X) +∑(kl) =42 +4 = 46 та обчислення НГЦФ для

позитивного вузла ω(Y) = ω(5,4) = ω(X) + H = 42 + 0 = 42; формули обчислення розрізняються, оцінки на КРОКУ 2 не збігаються, порівняння оцінок показує, що у цій ситуації не рівні. Як вчинити, який вузол вибрати? Ми формуємо маршрут Т, то включаємо до нього позитивну вершину (5, 4), з меншою НГЦФ Т = {(3, 5)(5, 4)(4,1)....}. Задаємо співвідношенням екстремальності обраному вузлу (5, 4) нове ім'я X = argmin { ω (Y = (5, 4)) = 46, ω (Y = (5, 4)) = 42} = (5, 4) або X = Y(k, l) = Y(5, 4).

- Порівняння поточного значення НГЦФ з вихідним значенням (Q*=44)≤ ω(5, 4)=42 ? **НІ**.

- Обчислюється розмірність матриці та перевіряється її рівність розмірності 2×2=2×2? **ТАК**.

- У безліч висячих вузлів включаємо вузол (Y)=(5, 4) з його оцінкою НГЦФ =ω(5, 4) = 46 В = {(3, 5) (4, 1) (5, 4)... }. А далі цей КРОК має відмінність від попередніх.

- Матриця С{2} вартості з розміром 2×2 дозволяє завершити побудову маршруту. Вибраними можуть бути тільки дві дуги (1, 2) вагою 0 та (2, 3) вагою 0 їх і включаємо в маршрут, завершуючи його. Т = {(3, 5)(5, 4)(4,1)(1,2)(2, 3)} цільова функція цього маршруту визначається сумою ЦФ = 6 + 8 + 12 + 1 +15 = 42. інших маршрутів цей реально прорахований маршрут, яке НГЦФ = 42 може використовуватися як новий критерій ефективності (критерію порівняння) і основою вибору кращого маршруту.

- Рішення прикладу дозволяє замінити попередню оцінку НГЦФ Q*=44 кращу оцінку, більш реальну Q*=ЦФ = 42.

- Фрагмент дерева рішень для шостого кроку набуває вигляду:

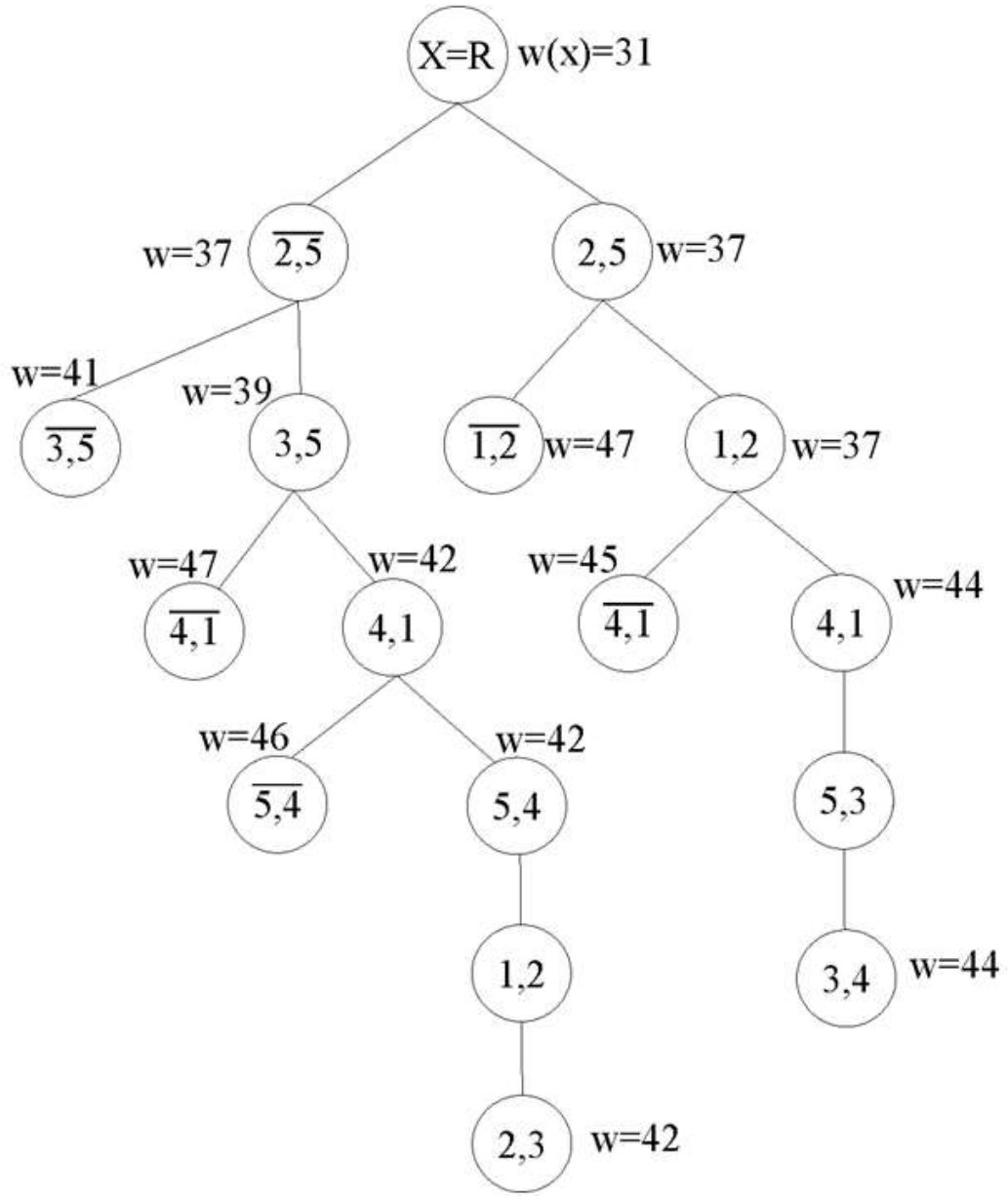


Рис.2.17 Фрагмент дерева пошуку рішення після 6 кроку

КРОК 7. Оскільки відповідь на питання про розмірність матриці C{4} на КРОКУ 6 позитивний (ТАК) в алгоритмі завдання відбувається ряд змін. Завершився пошук другого варіанта повного маршруту з оцінкою НГЦФ (Q * = ЦФ = 42), яка стає критеріальною. Переходимо до відповіді на запитання чи слід шукати інші маршрути і чи виявляться вони кращими за знайдене? Відповідь на це питання допомагає вказати безліч відкладених рішень з оцінками НГЦФ і збереженими станами процесів обробки. Якщо серед відкладених є оцінка менша знайденої Q*= 42, то, можливо, продовжуючи розрахунок по цій галузі дерева, в результаті отримаємо інший (третій)

маршрут, краще вже двох знайдених, а при аналізі всіх відкладених рішень (з меншою оцінкою ЦФ) буде знайдено оптимальний (невдосконалений) маршрут. Діємо далі так: У множині є оцінка w = 41 < 42, яка відповідала відмові на КРОКУ 4 від вибору вузла (k, l) = (3, 5) для включення в маршрут. Матриця цін у ситуації була C{5}, оскільки у маршрут була включена жодна з вершин графа шляхів. Ми повертаємо в матрицю дугу (3,5), але забороняємо вибір дуги $C_{35} = \infty$, при якому в рядку 3 матриці {5} пропадає нульова клітинка. Її повернення оцінюється як hi =4, для 3-го рядка, що підвищує колишню оцінку НГЦФ. До цього вже було повернуто дугу (2,5) із забороною її включення до маршруту призначенням C25 =∞. Але то вже історія. Інші нулі в C{5} зберігаються. Послідовність маршруту – порожня, не містить жодної дуги. У третьому рядку зник нуль, тому його наводимо з константою h =6.

	1	2	3	4	5	h_i
1	∞	0	0	2	3	0
2	0	∞	3	5	∞	0
3	4	14	∞	11	∞	4
4	2	12	10	∞	0	0
5	0	2	0	0	∞	0

	1	2	3	4	5
1	∞	0	0	2	3
2	0	∞	3	5	∞
3	0	10	∞	7	∞
4	2	12	10	∞	0
5	0	2	0	0	∞

Рис.2.18 Відновлені стани завдання при поверненні до відкладеного рішення після 6 кроку H=4

- Формуємо безліч S із клітин матриці C{5} з нульовими значеннями, визначаємо та вписуємо в таблицю (як і раніше) значення характеристик у рядках di і в стовпцях dj дуг-претендентів на включення в тур. Формули обробки елементів матриці {5} залишаються без зміни.

Таблиця 2.8 Множина клітинок S {(i,j)} з нульовими значеннями Cij = 0 у відновленій матриці C{5} вартості 7-го кроку

Для S={I,j}	(1,2)	(1,3)	(2,1)	(3,1)	(4,5)	(5,1)	(5,3)	(5,4)
di	0	0	3	7	2	0	0	0
dj	2	0	0	0	3	0	0	2
∑ij	2	0	3	7	5	0	0	2
argmax ∑ij				7				

- Знаходимо в множині клітин матриці C[5] з нульовими значеннями ту клітину, яка інші перевищує за сумою max∑ij = 7=di + dj .Аргументом цієї суми є клітина argmax∑ij = (k, l) = (3, 1);

- Перетворимо матрицю C{5}: видаляємо рядок з номером k = 3 та стовпець з номером l = 1; при цьому розмірність матриці цін змінилася і стала рівною (5-1)×(5-1)=4×4;

- Запобігаємо зациклюванню, тобто. забороняємо вибір клітини (p, q), де q = k = 1, p = l = 3, а у позицію (l, k) = (3, 1) вписуємо велике Cpq = C_{31} = ∞ значення, роблячи її забороненою для вибору та включення в маршрут. Ці дії відображені у правій матриці C{4} нижче:

	2	3	4	5
1	0	∞	2	3
2	∞	0	2	∞
4	12	10	∞	0
5	2	0	0	∞

	2	3	4	5	
1	0	∞	2	3	0
2	∞	3	5	∞	3
4	12	10	∞	0	0
5	2	0	0	∞	0

H=3

Рис.2.19 Модифіковані матриці цін 4×4 7-го кроку

- Обчислюємо константу приведення скороченої матриці 4×4, H = ∑hi = 3 + 0 = 3;

- Обчислюються оцінки НГЦФ для позитивного та негативного вузлів дерева пошуку рішення; обчислення НГЦФ для негативного вузла ω(Y) = ω(3,1) = ω(X) +∑(kl) =41 +7 = 48 та обчислення НГЦФ для позитивного вузла ω(Y) = ω(3,1) = ω(X) + H = 41 +3 = 44; формули обчислення розрізняються, оцінки на КРОКУ 7 не збігаються, порівняння оцінок показує, що вони у цій ситуації не рівні. Як вчинити, який вузол вибрати? Ми формуємо маршрут T, то включаємо до нього позитивну вершину (3,1), з меншою НГЦФ T = {(3,1)…}. Задаємо співвідношенням екстремальності вибраному вузлу (3,1) нове ім'я X = argmin { ω(Y=(3,1)) =48, ω(Y=(3, 1))=44}=(3, 1) або X = Y (k, l) = Y (3, 1).

- Порівняння поточного значення НГЦФ з вихідним значенням (Q * = 42) ≤ ω (3,1) = 44? **ТАК**. Поточна оцінка перевищує вихідну, а отже, оптимальне рішення вже знайдено.

- Дерево рішень для сьомого кроку набуває вигляду:

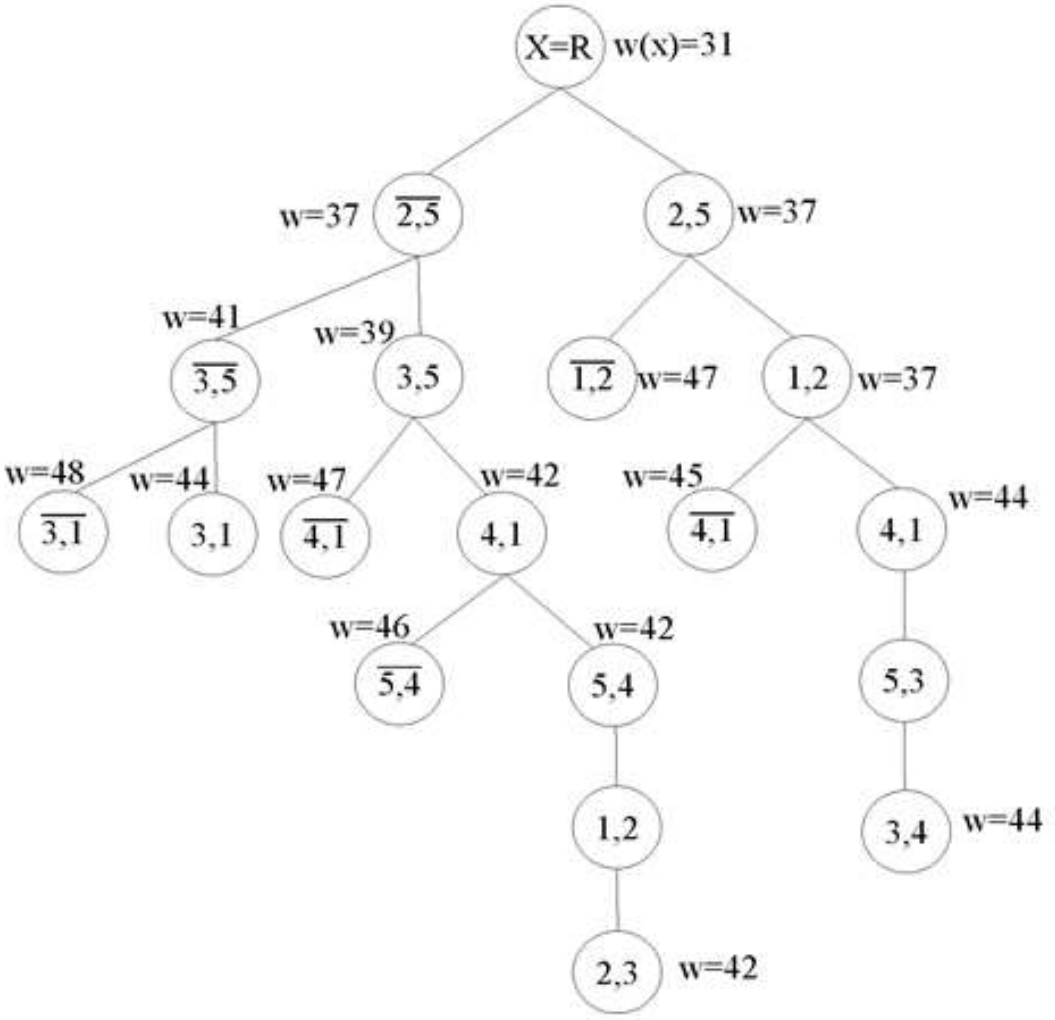


Рис.2.20 Повне дерево пошуку рішення після 7 кроку

Отже, щоб знайти оптимальний маршрут для нашого прикладу виявилося достатнім побудувати лише 2 повних маршрути і визначити значення цільової функції, досяжної на маршрутах. За остаточний T_{opt} маршрут приймається маршрут, отриманий на 6-му кроці, Topt = {(3, 5)(5, 4)(4,1)(1,2)(2, 3)}, йому відповідає вартість переміщень між точками ЦФ (Topt) = 6 + 8 + 12 + 1 + 15 = 42.

3. Практична реалізація пошуку оптимального маршруту

3.1 Визначення способу автоматизації розрахунку пошуку мінімального шляху переміщення інструменту.

В першу чергу для автоматизації процесу необхідно було провести автоматичне виділення геометрії отвору CAD деталі з іншого програмного забезпечення. Для відповідних розрахунків було обрано програмний продукт CATIA V5 яка містить в собі вбудовані скрипти на VBA та також має вбудований CAM модуль для подальшого генерування G-Коду для верстата з ЧПК на основі вже отриманої геометрії.

Наступним кроком було визначення способу опрацювання отриманих данних, для чого був написаний додатковий скрипт що копіює та переносить дані координат точок центрів отворів в Excel файл з яким в подальшому і буда проведенна остаточна робота по оптимізації та повернення координат в відсортованому порядку, проходження інструменту при обробці.

Останнім кроком було визначення мови програмування для написання коду для сортування точок оброблення в оптимальному порядку та повернення цих значеннь в таблиці Excel, для цього було обрана мова програмування Python завдяки тому що вона легка в освоєнні що вигідно в рамках вирішення однієї конкретної задачі, та також має безліч вбудованих простих модулів які мають можливість напряму працювати з файлами Ecxel з розширеннями *.csv.

```
import csv
with open('Part1.csv', 'r') as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)
```

Рис 3.1 Приклад коду для роботи з .csv файлами Excel

```
import pandas as pd
import requests
import json
import time
from pandas.io.json import json_normalize

df = pd.read_csv('Global_country_populations_2013.csv')
```

Рис 3.2 Приклад коду використовуючи додатковий інстальований модуль Pandas

Для вирішення цієї задачі було обрано вбудований модуль для роботи з Excel файлами через те, що функціоналу вбудованого модулю більш ніж достатньо для вирішення цієї задачі.

Після створення наступного алгоритму опрацювання завдання і зваження всіх опцій було вирішено використовувати саме цей набір програмного забезпечення при практичній реалізації цієї роботи.

3.2 Результати практичної релізації на тестовій деталі.

Для початку було відпрацьовано скрипт для VBA для CATIA V5 і перевірено його працездатність на деталі з невеликою кількістю отворів.

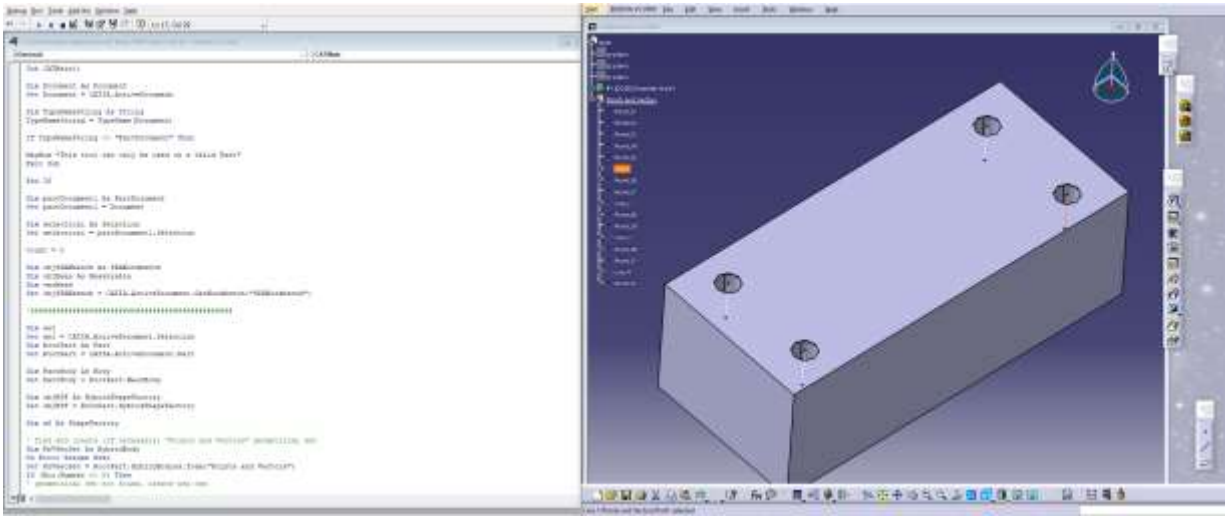


Рис.3.3 Результати випробуваннь автоматичного генерування основної геометрії отворів деталі.

Також було протестовано роботу коду для сортування на данних тестової деталі

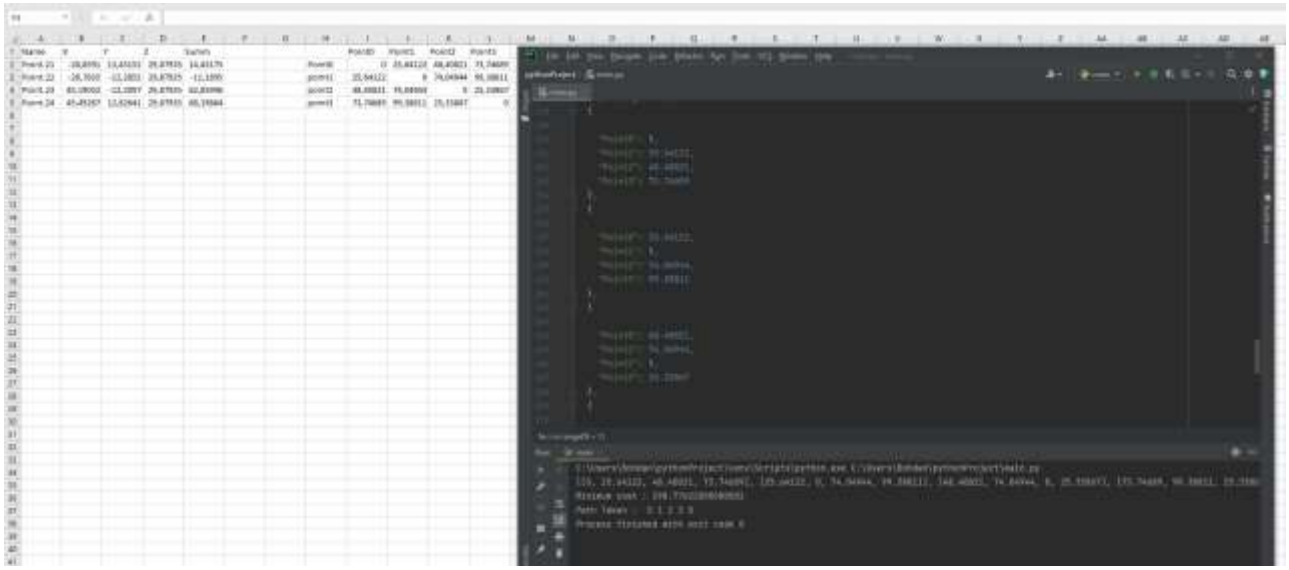


Рис.3.4 Сортування оптимального шляху інструменту методом гілок та кордонів

3.3 Результати практичної релізації.

Після перевірки функцонування наступних додатків було вирішено перейти до визначення оптимального шляху на основній проектній деталі. На Рис 3.5-7 Зображені результати роботи алгоритму автоматичного виділення геометрії.

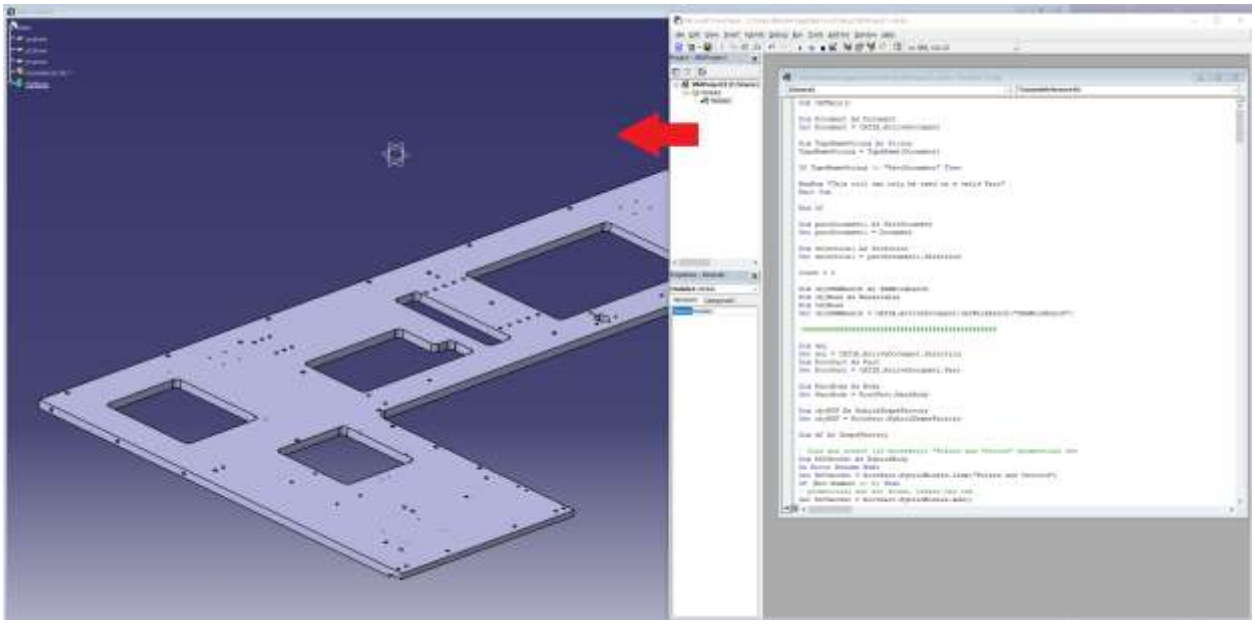


Рис 3.5 Приклад роботи коду VBA для заданої деталі для CATIA V5

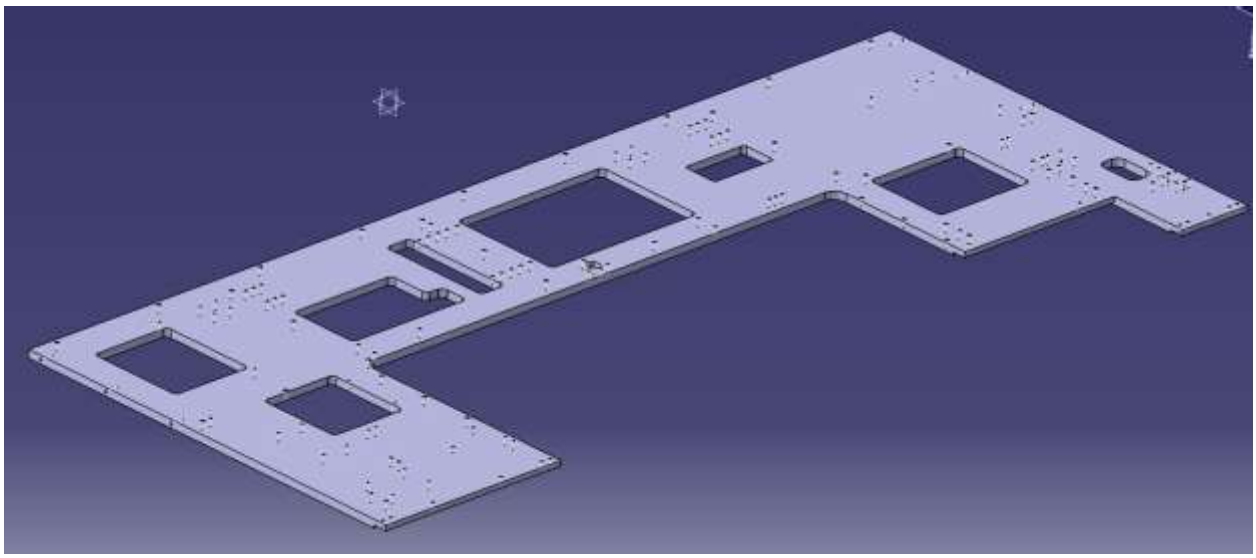


Рис.3.6 Результат автоматичного виділення геометрії отворів

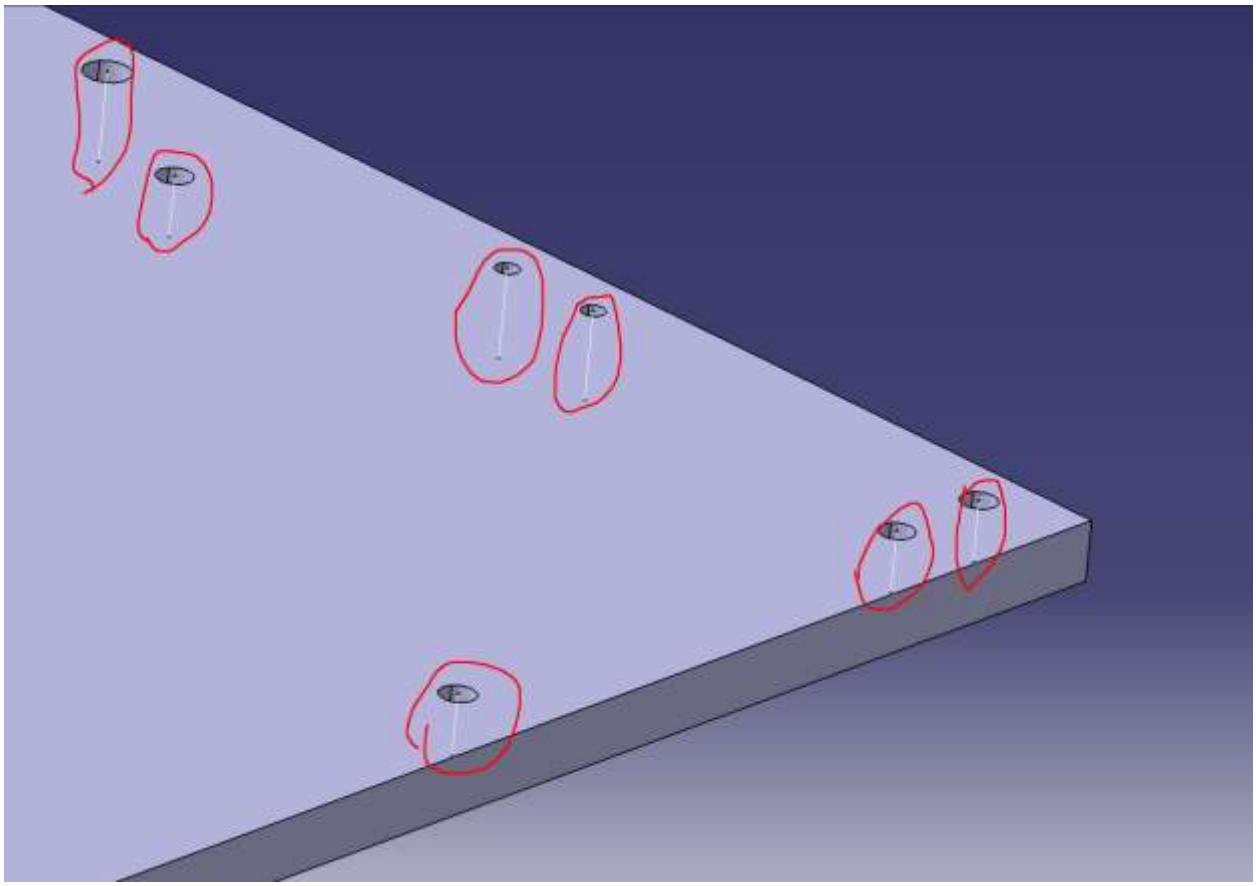


Рис.3.7 Макро-репрезентація частини деталі.

Час визначення всіх точок склав 4.5 секунд. Що значно менше ніж опрацювання цієї геометрії вручну.

Після визначення геометрії отворів було виділено тільки точки вершин отворів, інші були тимчасово приховані та перенесені в книгу Excel.

Name	X	Y	Z	Coordinats sum
Point.1	1910	577	322.5	2809.5
Point.2	1910	337	322.5	2569.5
Point.3	1910	97	322.5	2329.5
Point.4	-1395	577	322.5	-495.5
Point.5	-1395	337	322.5	-735.5
Point.6	-1395	97	322.5	-975.5
Point.7	1805.0000	1117.6606	310	3232.660626
Point.8	1855	1109.4997	310	3274.499781
Point.9	1900	1159.4997	310	3369.499781
Point.10	1655.0000	1052	310	3017.000001
Point.11	1655.0000	1017	310	2982.000001
Point.12	1655.0000	967	310	2932.000001
Point.13	1427.5	67	310	1804.5
Point.14	214.49997	586.99853	310	1111.498507
Point.15	-20.00017	202	310	491.999829
Point.16	42	152	310	504
Point.17	-20.00017	472	310	761.999829
Point.18	-1218	802	310	-106
Point.19	-1128	802	310	-16
Point.20	-1218	967	310	59
Point.21	-1128	967	310	149
Point.22	-1085	1117	310	342
Point.23	-1085	1087	310	312
Point.24	-1286.5	1263	310	286.5
Point.25	-1286.5	1208	310	231.5
Point.26	-1350	1342	310	302
Point.27	-770	1342	310	882
Point.28	-900	1139.5	310	549.5
Point.29	-770	1089.5	310	629.5
Point.30	1130	1089.5	310	2529.5
Point.31	1555	1089.5	310	2954.5
Point.32	1705	1342	310	3357
Point.33	1865	1342	310	3517
Point.34	1637	897	310	2844
Point.35	1637	837	310	2784
Point.36	1677	837	310	2824
Point.37	1677	897	310	2884
Point.38	1169.9996	1052	310	2531.999638
Point.39	1169.9996	1017	310	2496.999638
Point.40	1169.9996	967	310	2446.999638
Point.41	1817.5	632	310	2759.5
Point.42	-1302.5	632	310	-360.5
Point.43	52.499829	472	310	834.499829
Point.44	52.499829	202	310	564.499829
Point.45	1855	1159.4997	310	3324.499781
Point.46	-740	1177	310	747
Point.47	-740	1207	310	777
Point.48	1745	887	310	2942
Point.49	1745	837	310	2892
Point.50	1695	837	310	2842
Point.51	1695	887	310	2892
Point.52	1900	1109.4997	310	3319.499781
Point.53	1885	1034	310	3229
Point.54	1885	1064	310	3259
Point.55	-1105	1157	310	362

Рис.3.8 Результати автоматичного заповнення таблиці Excel.

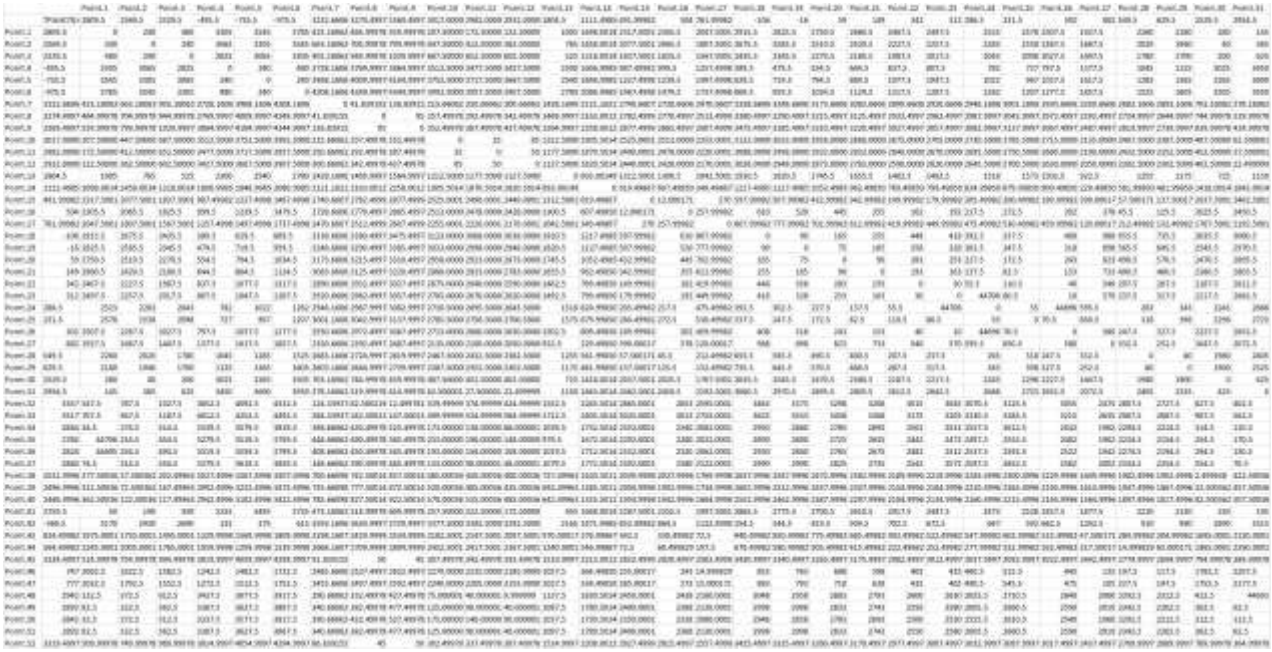


Рис 3.9 Обрахована матриця для виконанння алгоритму оптимізації.

Наступним кроком було обрахування попередньо отриманих данних в вигляді матриці та додати ці данні в алгоритм сортування. Розрахунок деталі для цієї задачі алгоритмом склав ***2 години 46 хвилин***. Результати розрахунку репрезентовані на Рис 3.10-11.

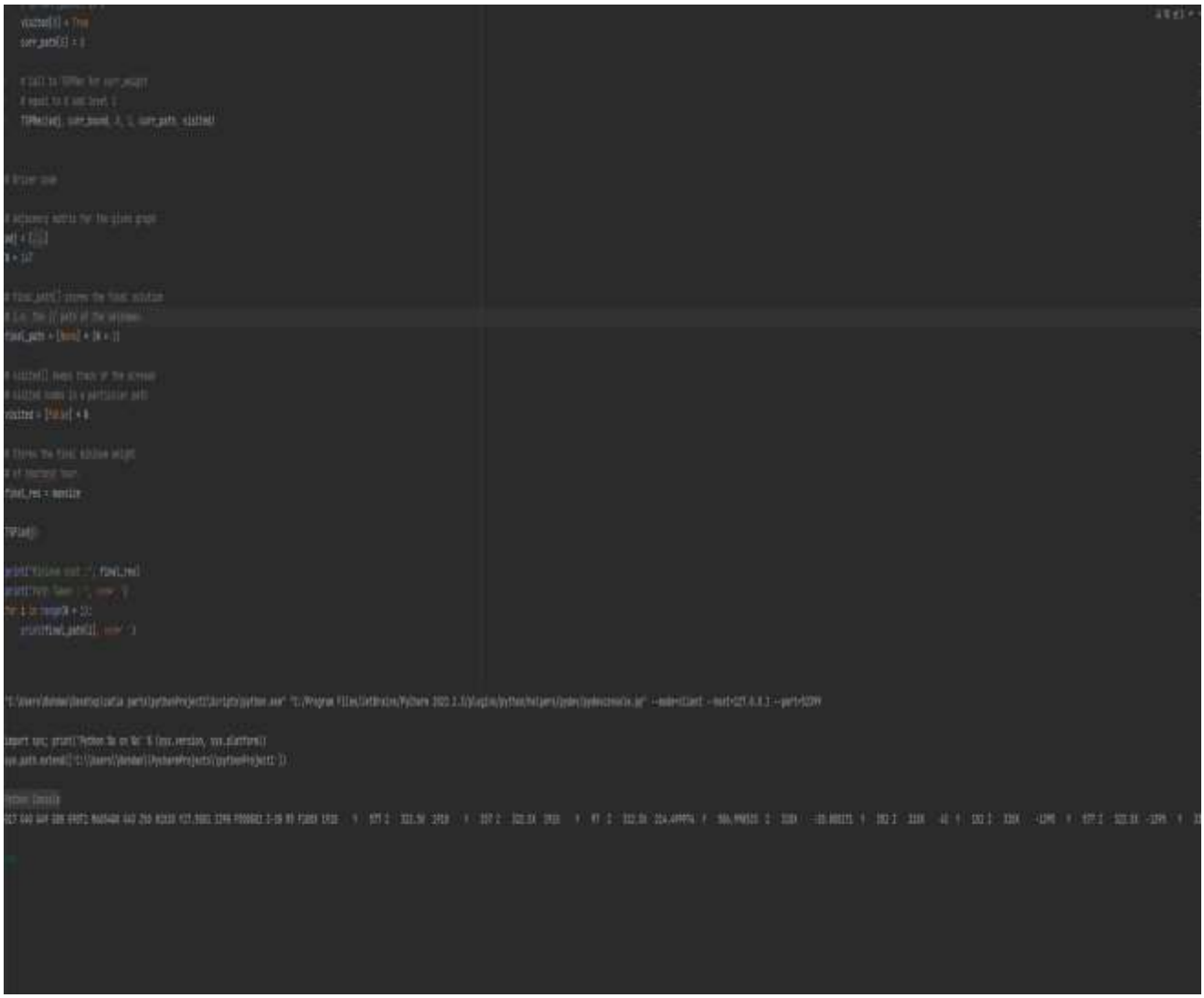


Рис 3.10 Результат виведення програми сортування

Висновок розділу 3.

В цьому розділі було представлено Розроблене Програмне забезпечення(ПЗ) для автоматизації, розрахунків з оптимізації переміщення інструменту. Дане розроблене ПЗ не вирішує задачі переміщення фрезерного інструменту або проходження обробки великим сортаментом свердл різних діаметрів, але може бути використана для свердління великої кількості типових отворів на одній площині деталі. Це ПЗ також може бути покращене в майбутньому з підвищенням швидкості роботи алгоритму з модулем розбиття зон обробки отворів з додатковим сортуванням наступних та з додаванням розподілення для обробки різними свердлами.

Наступна розроблена програма вирішує задачу оптимізації переміщення інструменту сумарно за ***2 години 52 хвилини***, коли у працівника на вирішення цієї задачі може піти не один робочий день.

Результат роботи програми:

```
O0001
G17 G40 G49
G80 G90 G54
T1 M6
G43 Z250 H1X20 Y380
G1 Z290 F500
G81 X1910 Y577 Z-30 R2 F100
X1910 Y337
X1910 Y97
X214.49 Y586.99
X-20.1 Y202
X42 Y152
X-1395 Y577
X-1395 Y337
X-1395 Y97
X1805 Y1117.66
X1855 Y1109.49
X1900 Y1159.49
X1655 Y1052
X1655 Y1017
X1655 Y967
X1427.5 Y67
X-20.1 Y472
X-1218 Y802
X-1128 Y802
...
```

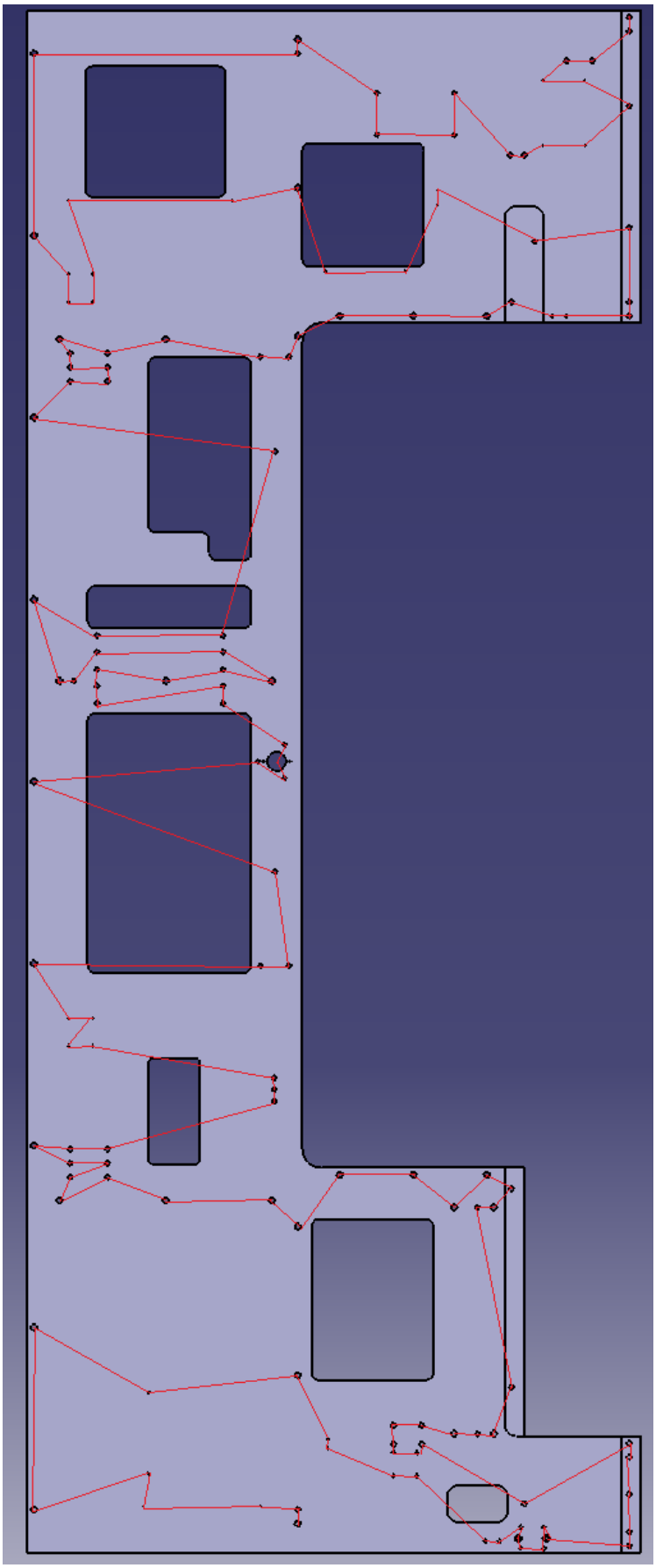


Рис 3.11 Візуальна репрезентація отриманого оптимального шляху з алгоритму.

4. Стартап проект

Таблиця 4.1 Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача ПЗ
Основною ідеєю цього стартап-проекту є автоматична побудова шляху проходження інструменту за максимально короткий час роботи верстату.	Виробництва з цехами металообробки	Автоматизація Складання порядку проходження точок при формуванні Технологічної операції свердління
		Підвищення ефективності виробництва за рахунко значного скорочення проектувального часу та часу оброблення отворів на верстаті

Таблиця 4.2 Статут проекту

Дата створення		№ документу	
Організація (замовник)			
Назва проекту	Створення програмного забезпечення для автоматизованого розрахунку порядку проходження точок обробки інструментом.		
Автор документу	Тарасов Богдан Артемович		
Місце знаходження	Київ		
Точність наведеної інформації	100%		
1. ***Причини ініціалізації проекту*** Забезпечити впровадження автоматизації процесу розрахунку оптимального шляху при обробці деталей свердлінням на підприємствах.			
2. ***Сутність запропонованої ідеї*** Основна ідея проекту – створення програмного забезпечення автоматизованого розрахунку проходження інструменту та скорочення як виробничого так і підготовчого часу складання маршрутних карт на виробництві.			
3. ***Мета проекту*** Автоматизувати процес розрахунку оптимального шляху свердлильного			

інструменту на верстаті з ЧПК.
4. ***Очікувані вигоди проекту*** За рахунок великого попиту на програмний продукт перейти міжнародний ринок
5. ***Обмеження проекту*** Для реалізації проекту буде необхідно залучити чотирьох досвідчених розробників ПЗ та 2 400 000 грн інвестиційних коштів на необхідне технічне обладнання, заробітну платню, оренду офісного приміщення.
6. ***Завдання та продукти проекту, критерії їхнього досягнення*** Цілі проекту: центр розробки ПЗ орієнтований на автоматизацію та оптимізацію виробничих процесів пов’язану з обробленням деталей в металообробних цехах. Опис продукту проекту: програмний продукт для мінімізації підготовчого часу та машинного часу обробки деталей. Критерії оцінки досягнення запланованих результатів: встановлення поточних контрактів на розробку ПЗ згідно технічних вимог проекту.
7. ***Зміст (межі) проекту*** Основним етапом створення проектів є формування робочої команди розробників ПЗ та створення основного програмного забезпечення яке буде мінімально виконувати необхідні функції.
8. ***Допущення та ризики проекту*** Основним ризиком є недостатність бюджету для роботи необхідної кількості робочого персоналу проекту та недостатня компетенція працівників HR-менеджменту в металообробних цехах, в галузі IT що може спровокувати затримки в виконанні роботи та можливе підвищення бюджету проекту.

Таблиця. 4.3 – Матриця відповідальності проекту

Роботи	Виконавці							
	Менеджер з підбору персоналу	Персонал групи контролю якості	Персонал групи по закупівлям	Персонал групи планування	Менеджер проекту	Персонал логістичної групи	Персонал конструкторсько-технологічної групи	Адм.-фін. група
Визначення організаційної структури				*R*	+			*A*
Визначення функціональних обов'язків	*R*				+			
Розробка посадових інструкцій	+				*A*			*R*
Узгодження функцій відділу з іншими відділами	*R*	*R*	*R*	*R*	+	*R*	*R*	*A*
Подання замовлення до служби зайнятості	+				*A*			
Проведення співбесід з претендентами	+				*A*			
Укладання договорів про роботу	+				*A*			
Контроль якості		+		*A*	*R*			
Закупівля основних матеріалів			+	*A*	*A*	*R*	*A*	*A*
Адміністративне управління					+			+

проектом								
Транспортування				*A*	*R*	+		
Розробка технології виготовлення				*A*	*R*		+	
Розробка конструкції виробу				*A*	*R*		+	
Визначення строків виконання робіт				+				

A — первинна відповідальність;

R — вторинна відповідальність.

Таблиця. 4.4 - Календар проектних робіт

Код Роботи	**Зміст роботи (назва)**	**Тривалість тиж.**
A.01	Залучення інвесторів	3
A.02	Визначення строків проекту	1
A.03	Набір персоналу	3
A.04	Укладення договорів	2
A.05	Визначення функцій груп	1
A.06	Визначення кошторису проекту	2
B.01	Розробка архітектури програмного продукту	4
B.02	Розробка ядра програмного продукту	5
B.03	Розробка окремих модулів	2
B.04	Закупівля тестового обладнання	2
C.01	Проведення тестування програмного продукту на цільовому обладнанні	1

C.02	Виправлення помилок на етапі тестування	12
C.03	Проведення налагоджувальних робіт по інтеграції програмного продукту	7
D.01	Реліз бета версії програмного продукту	4
D.02	Впровадження програмного продукту на підприємстві	3
D.03	Проведення аналізу роботи програмного продукту в умовах реального виробництва	1
Загальна тривалість		53

Таблиця. 4.5 - SWOT-аналіз проекту виробництва листових деталей з великою кількістю отворів для виготовлення фюзеляжу літаків.

Сильні сторони	**Слабкі сторони**
- Легке інтегрування в виробничий процесс, завдяки з сумісністю ПО з популярними CAx системами. - Моментальний зворотній відгук з виробництва та просте виправлення та модифікація функціоналу ПО «на ходу».	- Високі видатки на заробітню платню Персоналу розробки програмного забезпечення. - Велика кількість часу на опрацювання та підтвердження результатів ПО на практиці.
Можливості	**Загрози**
- Відносно легка типізація деталей з отворами на класи в виробництві завдяки електронній базі данних попередніх шляхів обробок(наявність порівняння) - Відносно невеликі терміни розробки ПО, що дає змогу швидко реалізувати продукцію. - Завдяки програмній оптимізації розрахунку данних з деталі та їх автоматичний перенос в ПО ЧПК верстату, дає можливість економити	Недостатнє фінансування відділу розробки ПО для забезпечення неперервної роботи.

значну кількість часу на надходженні кожного нового креслення деталі для обробки на виробництві. - За рахунок модульності програми можливі модифікації програми на інші види металообробки.	

Висновки

В даній роботі було розроблено програимне забезпечення, яке вирішує задачу оптимізації траєкторії обходу отворів при свердлінні. Для вирішення

задачі були розглянуті алгоритми пошуку найкоротших маршрутів на графах, було проведено їх аналіз та порівняння.

Огляд літературних джерел, дав зрозуміти не тільки можливі види формалізації задачі комівояжера але і напрямки існуючих робіт, серед яких не було оптимізації проходження отворів для свердління. За основу були також взяті зарубіжні статті та джерела з напрацьованою інформацією стосовно використання задачі комівояжера при обробці металевих деталей фрезеруванням.

В результаті опрацювання теоретичної бази було створено програмний продукт який не тільки прораховує найкоротший шлях переміщення між центрами отворів але і додаткові допоміжні програмні продукти які автоматично визначають геометрію отворів, перенос цих даних в табличні їх подальше автоматичне сортування алгоритмом та повернення відсортованих даних назад в табличний вигляд для простого перенесення в G-код для верстату з ЧПК.

Основний програмний продукт був створений на мові програмування Python, допоміжні програмні продукти були створені на Catia V5 VBA та Excel VBA.

Література та використані джерела

[1] VECTOR-BARED AEDERTRIAN NAVIGATION IN CITIER ChriRtian Bongiorno, Yulun Zhou, Marta Kryven, David Theurel,AleRRandro Rizzo, Aaolo Ranti, JoRhua Tenenbaum, Carlo Ratti. March 23, 2021.

[2] Land A.H., and Doig A.G. An automatic method of Solving discrete Programming Problem. Econometrica. v28 (1960), ст. 497-520

[3] Little J.D.C., Murty K.G., Rweeney D.W., and Karel C. An algorithm for the traveling RaleRman Aroblem. OAerationR ReRearch. v11 (1963), ст. 972-989.

[4] Application of precedence constrainedtravelling salesman problem model for tool path optimization in CNC milling machines. Ilker Kucukoglu,Tulin Gunduz, Fatma Balkancioglu, Emine Chousein TopalOznur Sayim, ISSN: 2146-0957; 28.05.2019

[5] А. О. Скоркін, О. Л. Кондратюк, О. П. Старченко «Теоретичні основи оптимізації холостих переміщень інструмента при фрезеруванні складних поверхонь» Сучасний стан наукових досліджень та технологій в промисловості. 2018. №4 (6) ISSN 2254-2296.

[6] Karla L. Hoffman , Manfred Padberg, Giovanni Rinaldi Traveling salesman problem. Kluwer Academic Publishers 2001. № 10.1007/1-4020-0611-X_1068

[7] The Traveling Salesman Problem David L. Applegate , Robert E. Bixby , Vašek Chvátal and William J. Cook Chapter 1,4 https://doi.org/10.1515/9781400841103

[8] Estimating the size of branch-and-bound trees. Gregor Hendel , Daniel Anderson , Pierre le Bodic , Marc e. Pfetsch. ISSN 2192-7782; April 2020

Додатки

Додаток А

Код алгоритму сортування:

```
# Python3 program to solve
# Traveling Salesman Problem using
# Branch and Bound.
import math

maxsize = float('inf')


# Function to copy temporary solution
# to the final solution
def copyToFinal(curr_path):
    final_path[:N + 1] = curr_path[:]
    final_path[N] = curr_path[0]


# Function to find the minimum edge cost
# having an end at the vertex i
def firstMin(adj, i):
    min = maxsize
    for k in range(N):
        if adj[i][k] < min and i != k:
            min = adj[i][k]

    return min


# function to find the second minimum edge
# cost having an end at the vertex i
def secondMin(adj, i):
    first, second = maxsize, maxsize
    for j in range(N):
        if i == j:
            continue
        if adj[i][j] <= first:
            second = first
            first = adj[i][j]

        elif (adj[i][j] <= second and
              adj[i][j] != first):
            second = adj[i][j]

    return second


# function that takes as arguments:
# curr_bound -> lower bound of the root node
# curr_weight-> stores the weight of the path so far
# level-> current level while moving
# in the search space tree
# curr_path[] -> where the solution is being stored
# which would later be copied to final_path[]
def TSPRec(adj, curr_bound, curr_weight,
           level, curr_path, visited):
    global final_res
```

```
# base case is when we have reached level N
# which means we have covered all the nodes once
if level == N:

    # check if there is an edge from
    # last vertex in path back to the first vertex
    if adj[curr_path[level - 1]][curr_path[0]] != 0:

        # curr_res has the total weight
        # of the solution we got
        curr_res = curr_weight + adj[curr_path[level - 1]] \
            [curr_path[0]]
        if curr_res < final_res:
            copyToFinal(curr_path)
            final_res = curr_res
    return

# for any other level iterate for all vertices
# to build the search space tree recursively
for i in range(N):

    # Consider next vertex if it is not same
    # (diagonal entry in adjacency matrix and
    #  not visited already)
    if (adj[curr_path[level - 1]][i] != 0 and
            visited[i] == False):
        temp = curr_bound
        curr_weight += adj[curr_path[level - 1]][i]

        # different computation of curr_bound
        # for level 2 from the other levels
        if level == 1:
            curr_bound -= ((firstMin(adj, curr_path[level - 1]) +
                            firstMin(adj, i)) / 2)
        else:
            curr_bound -= ((secondMin(adj, curr_path[level - 1]) +
                            firstMin(adj, i)) / 2)

        # curr_bound + curr_weight is the actual lower bound
        # for the node that we have arrived on.
        # If current lower bound < final_res,
        # we need to explore the node further
        if curr_bound + curr_weight < final_res:
            curr_path[level] = i
            visited[i] = True

            # call TSPRec for the next level
            TSPRec(adj, curr_bound, curr_weight,
                   level + 1, curr_path, visited)

        # Else we have to prune the node by resetting
        # all changes to curr_weight and curr_bound
        curr_weight -= adj[curr_path[level - 1]][i]
        curr_bound = temp
```

```
            # Also reset the visited array
            visited = [False] * len(visited)
            for j in range(level):
                if curr_path[j] != -1:
                    visited[curr_path[j]] = True


# This function sets up final_path
def TSP(adj):
    # Calculate initial lower bound for the root node
    # using the formula 1/2 * (sum of first min +
    # second min) for all edges. Also initialize the
    # curr_path and visited array
    curr_bound = 0
    curr_path = [-1] * (N + 1)
    visited = [False] * N

    # Compute initial bound
    for i in range(N):
        curr_bound += (firstMin(adj, i) +
                       secondMin(adj, i))

    # Rounding off the lower bound to an integer
    curr_bound = math.ceil(curr_bound / 2)

    # We start at vertex 1 so the first vertex
    # in curr_path[] is 0
    visited[0] = True
    curr_path[0] = 0

    # Call to TSPRec for curr_weight
    # equal to 0 and level 1
    TSPRec(adj, curr_bound, 0, 1, curr_path, visited)


# Driver code

# Adjacency matrix for the given graph
adj = [...]
N = 147

# final_path[] stores the final solution
# i.e. the // path of the salesman.
final_path = [None] * (N + 1)

# visited[] keeps track of the already
# visited nodes in a particular path
visited = [False] * N

# Stores the final minimum weight
# of shortest tour.
final_res = maxsize
```

```
TSP(adj)

print("Minimum cost :", final_res)
print("Path Taken : ", end=' ')
for i in range(N + 1):
    print(final_path[i], end=' ')
```