МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Запорізька політехніка»

Кафедра системного аналізу та обчислювальної математики

ПОЯСНЮВАЛЬНА ЗАПИСКА ДО КУРСОВОГО ПРОЄКТУ

з дисципліни «Методи оптимізації та дослідження операцій»

на тему: Транспортна задача

Розробив: студент групи КНТ-811 Погодаєв Д.В.

Керівник: к.ф.-м.н., доцент Терещенко Е.В.

2024р.

ЗАВДАННЯ ДО КУРСОВОГО ПРОЄКТУ

Дослідження методів розв’язання транспортної задач з допомогою різних бібліотек Python. Порівняння їх ефективності та обґрунтування вибору оптимального методу.

РЕФЕРАТ

ПЗ: 36 c., 7 формул, 6 таблиці, 6 рисунків, 5 додатків, 5 джерел.

Об’єкт дослідження – об’єктом дослідження є транспортні системи, які включають сукупність постачальників, споживачів та транспортних шляхів між ними. Це можуть бути логістичні мережі, системи дистрибуції товарів або будь-які інші системи, що передбачають переміщення ресурсів з одного місця в інше.

Предмет дослідження – предметом дослідження є різні бібліотеки та способи написання коду для розв’язання задач лінійного програмування, а саме транспортних задач.

Мета роботи – метою дослідження є розробка і аналіз різних програм для розв’язання транспортних задач.

Метод дослідження – аналітичний.

Ключові слова – ТРАНСПОРТНА ЗАДАЧА, ЛІНІЙНЕ ПРОГРАМУВАННЯ, PYTHON, ОТПИМІЗАЦІЯ ТРАНСПОРТНИХ ЗАДАЧ.

ЗМІСТ

[ВСТУП](#_Toc41803) 5

РОЗДІЛ 1. ТЕОРЕТИЧНА ЧАСТИНА 6

[1.1 Модель транспортної задачі](#_Toc41806) 6

[1.2 Побудова опорного плану](#_Toc41806) 7

[1.3 Методи оптимізації опорного плану. Метод потенціалів 8](#_Toc41806)

[РОЗДІЛ 2. ПРАКТИЧНА ЧАСТИНА](#_Toc41815) 10

2.1 Ручний розрахунок 10

2.2 Рішення за допомогою Excel 13

2.3 Реалізація програмного коду з допомогою бібліотеки PuLP 16

2.4 Реалізація програмного коду з допомогою бібліотеки SciPy 17

2.5 Реалізація програмного коду з допомогою Ortool 17

2.6 Порівняння швидкості виконання програм різних бібліотек 18

2.7 Код без використання бібліотек лінійного програмування та порівняння його ефективності 20

[ВИСНОВКИ](#_Toc41828) 21

[ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ](#_Toc41829) 22

[Додаток А. pulpprogram.py](#_Toc41830) 23

[Додаток B. scipy\_program.py](#_Toc41830) 25

[Додаток C. ortools\_program.py](#_Toc41830) 27

[Додаток D. timecheck.py](#_Toc41830) 29

[Додаток E. mycode.py](#_Toc41830) 32

ВСТУП

Транспортна задача є однією з ключових задач оптимізації у логістиці та управлінні ресурсами. Вона виникає у ситуаціях, коли необхідно оптимально розподілити обмежені ресурси між постачальниками та місцями споживання. Зазвичай, ця задача ставиться з метою мінімізації загальних витрат або максимізації прибутку при врахуванні обмежень, що виникають у процесі транспортування.

Хоча багато досліджень було проведено у сфері транспортної оптимізації, більшість з них базується на базовій формулюванні задачі без урахування певних обмежень, які можуть виникнути в реальних ситуаціях. Однак у багатьох практичних випадках існують обмеження, такі як обмеження на кількість доступних ресурсів або обмеження на максимальну кількість одиниць, які можуть бути перевезені між певними джерелами та місцями призначення.

Цей курсовий проєкт спрямований на вивчення транспортної задачі з обмеженнями, а саме на теорію, методи розв’язання та її практичні застосування. Він розгляне сучасні інструменти оптимізації та аналізу даних: [pulp, scipy, ortools](#_Toc41830), ця робота може зробити внесок у розуміння та вирішення складних проблем логістики та розподілу ресурсів.

1. ТЕОРЕТИЧНА ЧАСТИНА
   1. Модель транспортної задачі

Для створення моделі транспортної задачі введемо деякі формули. Задача полягає в мінімізації цільової функції (формула 1.1) за деяких обмежень (формули 1.2-4), де – кількість продукції що перевозиться від i-того постачальника до j-ого споживача; – вартість перевезення одиниці продукції від i-того постачальника до j-ого споживача; – запас продукції i-того постачальника; – потреби j-ого споживача.

В задачах транспортного типу розділяють два види задач: закритого (або збалансованого) типу та відкритого (тобто незбалансованого) типу. Для першої виконується умова (формула 1.5), для другого вона не виконується.

Умови прописані в формулах 1.2-3 виконуються лише для збалансованої задачі, для задач закритого типу вони відрізняються і мають вигляд формул 1.6-7. За умови не виконання цих обмежень розв’язку задачі не існує.

Оптимальним планом транспортної задачі називають матрицю , яка задовольняє обмеження та умови задачі і для якої цільова функція (1.1) набуває найменшого значення.

* 1. Побудова опорного плану

Для побудови початкового опорного плану транспортної задачі існує декілька методів: північно-західного кута, мінімальної вартості, апроксимації Фогеля та інші. Побудову цього плану зручно представляти у вигляді таблиці, в якій постачальники продукції є рядками, а споживачі – стовпчиками.

**Метод північно-західного кута.**

Побудову цього метода починають з верхнього лівого кута () та беруть менше з обмежень , . Далі переходять до наступної клітини по рядку чи стовпцю(там де ще не вичерпано обмеження) та заповнюють так до кінця. Після цього задачу можна оптимізувати для знаходження оптимального рішення.

**Метод мінімальної вартості.**

Ідея цього методу полягає в тому, що на кожному кроці заповнюємо ту клітину таблиці, яка має найменшу вартість перевезення одинці продукції. Ці дії повторюються доти, доки не буде розподілено всю продукцію між постачальниками та споживачами.

**Метод апроксимації Фогеля.**

Цей метод враховує не тільки мінімальні витрати, але й різницю між мінімальними і наступними за величиною витратами в кожному рядку і стовпчику. Ця різниця називається штрафом, і вона дозволяє краще враховувати важливість вибору тієї чи іншої комірки для постачання. Алгоритм виходить наступний:

Для кожного рядка і стовпчика обчислюються штрафи як різниця між найменшою і наступною за величиною вартістю в цьому рядку або стовпчику. Вибирається рядок або стовпчик з максимальним штрафом. Якщо таких декілька, вибирається будь-який з них. У вибраному рядку або стовпчику вибирається клітина з мінімальною вартістю. Якщо пропозиція або попит задовільнені, відповідний рядок або стовпчик виключається з подальшого розгляду. Алгоритм повторюється, поки не залишиться один стовпець або рядок, тоді переходимо до методу мінімального елемента.

* 1. Методи оптимізації опорного плану. Метод потенціалів.

Для оптимізації опорного плану методом потенціалів введемо псевдовартість з такими правилами: нехай – ціна яку платить i-тий пункт видачі при перевезенні, а – ціна яку платить j-тий пункт прийому. При цьому при . Тобто псевдовартість буде рівна вартості для тих пар, де перевезення між пунктами існує.

Для всіх пустих клітинок розрахуємо їх потенціал . Якщо виявляється, що всі , то план оптимальний.

Процес оптимізації включає кілька кроків. Спочатку вибирається пуста клітина з максимальним потенціалом . Далі використовується цикл для перенесення перевезень таким чином, щоб зберегти умови балансу. Вибирається маршрут, який починається і закінчується в обраній клітині, чергуючи додавання і віднімання перевезень. Після оновлення плану перевіряються знову значення потенціалів для пустих клітин. Якщо знову виявляються клітини з процес оптимізації повторюється.

Метод потенціалів є ефективним інструментом для оптимізації опорного плану транспортної задачі. Він дозволяє знайти оптимальне рішення, мінімізуючи загальні витрати на транспортування. Застосування методу в практичних умовах вимагає ретельних розрахунків, але забезпечує високий рівень точності і надійності результатів.

1. ПРАКТИЧНА ЧАСТИНА
   1. Ручний розрахунок

Розглянемо побудову опорного плану різними методи для однієї задачі, умови якої можна побачити в таблиці 2.1.1.

Таблиця 2.1.1 Збалансована транспортна задача

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| речовина | ціна за перевезення | | | | виробництво |
| 1 | 2 | 3 | 4 |
| 1 | 8 | 7 | 5 | 10 | 240 |
| 2 | 13 | 8 | 10 | 7 | 360 |
| 3 | 12 | 4 | 11 | 9 | 180 |
| потреба | 230 | 220 | 130 | 170 |  |

**Метод північно-західного кута.**

Використовуючи метод північно-західного кута починаємо заповнення з = min (230, 240) = 230. Далі, так як обмеження по стовпчику вичерпано, переходимо до клітини = min (10, 220) = 10. Далі обмеження по рядку вичерпано, переходимо до клітини 210, 330) = 210. Далі вичерпав обмеження другого стовпця переходимо до клітини = min (130, 120) = 120. = min (10, 180) = 10. = min (170, 170) = 170.

Результат цього методу можна побачити в таблиці 2.1.2, а цільова функція = 6430

Таблиця 2.1.2 опорний план методом північно-західного кута

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| речовина | Обсяг перевезення | | | | виробництво |
| 1 | 2 | 3 | 4 |
| 1 | 230 | 10 |  |  | 240 |
| 2 |  | 210 | 120 |  | 330 |
| 3 |  |  | 10 | 170 | 180 |
| потреба | 230 | 220 | 130 | 170 |  |

**Метод мінімальної вартості.**

За цим методом знаходимо мінімальну вартість перевезення між двома пунктами, для початкової таблиці це = 4, значить = min (220, 180) = 180. Далі виключаємо 3 рядок і дивимось на залишкові вартості і шукаємо мінімальну серед них: = 5, значить = min (130, 240) = 130. Далі вилучаємо 3 стовпець і дивимось на вартості що залишились: = 7, значить = min (40, 110) = 40. Виключаємо 2 стовпець: = 7, значить = min (170, 330) = 170. Виключаємо 4 стовпець: = 8, значить = min (70, 230) = 70. Виключаємо 1 рядок: = 13, значить = min (160, 160) = 160.

Результат цього методу можна побачити в таблиці 2.1.3, а цільова функція = 5480.

Таблиця 2.1.3 опорний план методом мінімальної вартості

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| речовина | Обсяг перевезення | | | | виробництво |
| 1 | 2 | 3 | 4 |
| 1 | 70 | 40 | 130 |  | 240 |
| 2 | 160 |  |  | 170 | 330 |
| 3 |  | 180 |  |  | 180 |
| потреба | 230 | 220 | 130 | 170 |  |

**Метод апроксимації Фогеля.**

Розглянемо спочатку «штраф» для кожного рядку:

Рядок1 = 7-5=2

Рядок2 = 8-7=1

Рядок3 = 4-9 = 5

Тепер для стовпців:

Стовпець1= 12-8 = 4

Стовпець2 = 7-4 = 3

Стовпець3 = 10-5 = 5

Стовпець4 = 9-7 = 2

Шукаємо найбільший штраф та заповнюємо відповідну клітину з мінімальною вартістю: = min (220, 180) = 180. Розглядаємо штрафи не рахуючи 3 рядок:

Рядок1 = 7-5=2

Рядок2 = 8-7=1

Стовпець1= 13-8 = 5

Стовпець2 = 8-7 = 1

Стовпець3 = 10-5 = 5

Стовпець4 = 10-7 = 3

Найбільший штраф у стовпці 1 та 3, візьмемо 3 з них та заповнимо відповідну клітину = min (130, 240) = 130. Виключаємо третій стовпець:

Рядок1 = 8-7=1

Рядок2 = 8-7=1

Стовпець1= 13-8 = 5

Стовпець2 = 8-7 = 1

Стовпець4 = 10-7 = 3

= min(230, 110) = 110

Перший рядок виключили, залишився один, значить тепер переходимо до методу мінімального елементу. Результат можна побачити в таблиці 2.1.4 і результат цільової функції = 5320, що дало мінімальний можливий результат і не потребую подальшої оптимізації.

Таблиця 2.1.4 опорний план методом апроксимації Фогеля

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| речовина | Обсяг перевезення | | | | виробництво |
| 1 | 2 | 3 | 4 |
| 1 | 110 |  | 130 |  | 240 |
| 2 | 120 | 40 |  | 170 | 330 |
| 3 |  | 180 |  |  | 180 |
| потреба | 230 | 220 | 130 | 170 |  |

* 1. Рішення за допомогою Excel

Один зі способів розв’язання таких задач є Microsoft Excel. Розглянемо наступну задачу(табл. 2.2.1). Для її розв’язання використаємо розв’зувач.

Таблиця 2.2.1 Збалансована транспортна задача

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| речовина | ціна за перевезення | | | | виробництво |
| 1 | 2 | 3 | 4 |
| 1 | 8 | 7 | 5 | 10 | 240 |
| 2 | 13 | 8 | 10 | 7 | 330 |
| 3 | 12 | 4 | 11 | 9 | 180 |
| потреба | 230 | 220 | 130 | 170 |  |

"Розв’язувач" – це надбудова Microsoft Excel, яка використовується для аналіз "what-if". За її допомогою можна знайти оптимальне (максимальне або мінімальне) значення для формула в одній клітинці (так званій клітинці цільової функції), що обмежується значеннями формул в інших клітинках аркуша. Надбудова "Розв’язувач" працює із групою клітинок (які називаються клітинками змінних рішення або просто клітинками змінних), що використовуються для обчислення формул у цільових функціях і клітинках обмежень. Надбудова регулює значення у клітинках змінних відповідно до меж у клітинках обмежень і виводить потрібний результат у клітинці цільової функції[1].

Для розв’язання такоїї задачі треба побудувати ще одну таблицю(табл. 2.2.2).

Табл. 2.2.2 – Допоміжна таблиця для розв’язку

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| речовина | кількість перевезення | | | | виробництво |
|  | 1 | 2 | 3 | 4 |  |
| 1 |  |  |  |  | 0 |
| 2 |  |  |  |  | 0 |
| 3 |  |  |  |  | 0 |
| потреба | 0 | 0 | 0 | 0 |  |

Тепер перейдемо до Розв’язувачу і введемо наші обмеження. До обмежень має входити наступне: всі перевезення цілі, всі перевезення не від’ємні, сума перевезень по стовпчикам не перевищує потреби стовпчика, сума перевезень рядків не перевищує потреби рядків. Цільова функція – це сума добутків цін та перевезень і має бути мінімальна(рис. 2.2.1)

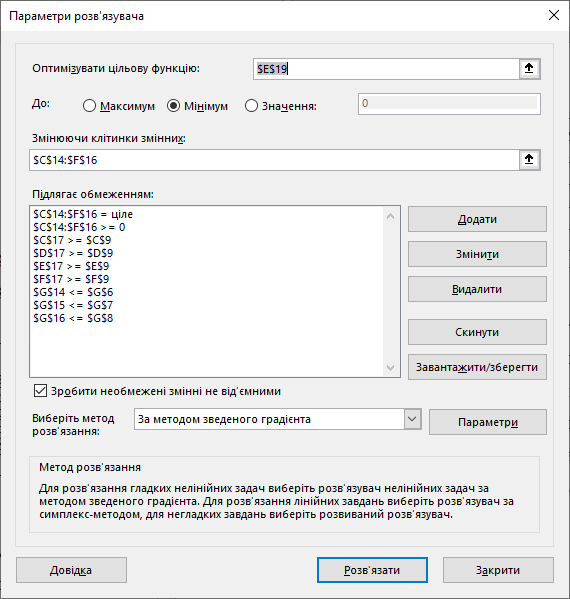


Рис. 2.2.1 – Розв’язувач

Як результат цього Розв’язувача можемо побачити адекватний розв’язок який виконує всі обмеження транспортної задачі(рис. 2.2.2)

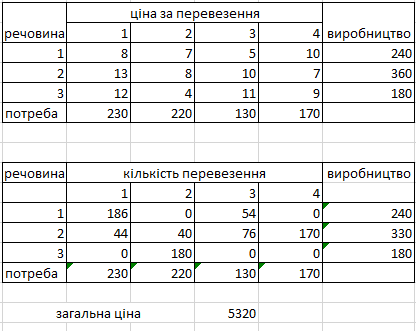


Рис. 2.2.2 – результат роботи Розв’язувача

* 1. Реалізація програмного коду з допомогою бібліотеки PuLP

Для написання програмного коду на Python використаємо бібліотеку PuLP[2] для розв’язання задач лінійного програмування. З цієї бібліотеки нам знадобляться такі функції:

from pulp import LpProblem, LpMinimize, LpVariable, lpSum, value

Тепер можемо реалізувати код(додаток А) для знаходження оптимального рішення знаючи вартість, обмеження виробництва та потреб:

def solve\_transportation\_problem(cost, supply, demand).

Якщо введемо ті самі дані як і в Excel, то отримаємо трохи іншу матрицю розв’язку, але з таким самим значенням цільової функції:

cost = np.array([

[8, 7, 5, 10],

[13, 8, 10, 7],

[12, 4, 11, 9] ])

supply = [240, 330, 180]

demand = [230, 220, 130, 170]

Optimized solution using PuLP:

[[230. 0. 10. 0.]

[ 0. 40. 120. 170.]

[ 0. 180. 0. 0.]]

Objective function value (total transportation cost):

5320.0

* 1. Реалізація програмного коду з допомогою бібліотеки SciPy

Для використання іншого програмного коду можемо використати наступну функцію з бібліотеки SciPy[3]:

from scipy.optimize import linprog

Для тих самих вхідних даних ця програма(додаток В) видає іншу таблицю, але з тим самим значенням цільової функції:

Optimized solution using SciPy:

[[110. 0. 130. 0.]

[120. 40. 0. 170.]

[ 0. 180. 0. 0.]]

Total transportation cost:

5320.0

* 1. Реалізація програмного коду з допомогою Ortools

Наступний програмний код напишемо з допомогою бібліотеки OR-Tools [4]. Основним вирішувачем в OR-Tools для завдань такого типу є вирішувач лінійної оптимізації, який насправді є оболонкою для різних бібліотек для лінійної і змішано-цілочисельної оптимізації, включаючи сторонні бібліотеки. З цієї бібліотеки нам знадобиться наступна функція:

from ortools.linear\_solver import pywraplp

Код написаний з допомогою цієї функції можна знайти в Додатку С дає наступний результат:

[[230. 0. 10. 0.]

[ 0. 40. 120. 170.]

[ 0. 180. 0. 0.]]

Objective function value (total transportation cost):

5320.0

Як можна побачити результат цільової функції не відрізняється від попередніх методів, а матриця збігається з одним з них.

* 1. Порівняння швидкості виконання програм різних бібліотек

За допомогою бібліотеки time можна вирахувати скільки часу займає виконання тої чи іншої функції. Використовуючи код з додатку D вирахуємо який час займає кожний код. Для початку перевіримо це для тих даних які у нас були вище(рис 2.6.1)

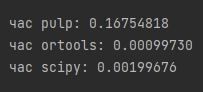


Рис. 2.6.1 – результат виконання коду першої матриці

Тепер візьмемо більші дані. Для цього напишемо функцію яка сама генерує дані(рис. 2.6.2)

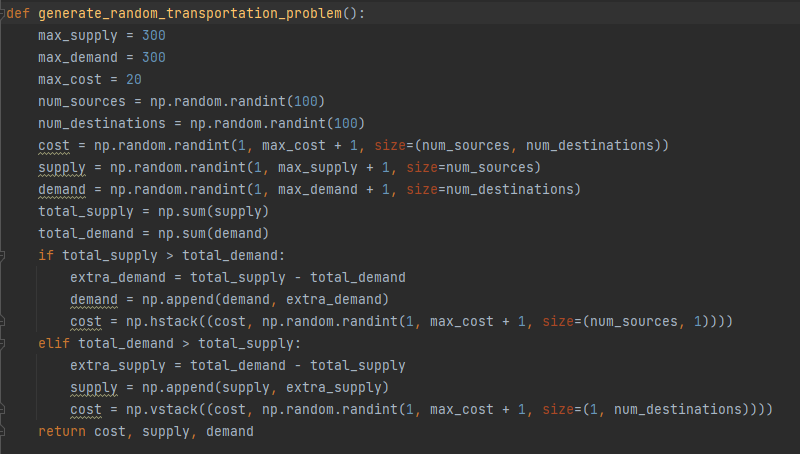


Рис. 2.6.2 – функція для генерування даних задачі транспортування

Виконаємо це для 100 різних наборів і подивимось на середній час виконання кожного з них(рис. 2.6.3). Як можна помітити найбільше часу займає код з бібліотекою PuLP. Так відбувається тому, що ця бібліотека робить короткий вивід інформації для кожної задачі, що займає додатковий час. Найшвидше виконується код з SciPy, що говорить про те, що це найкращий варіант для розв’язку таких задач.

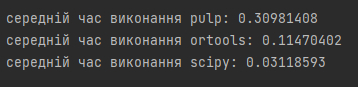


Рис. 2.6.3 – середній час виконання для 100 випадкових задач

* 1. Код без використання бібліотек лінійного програмування та порівняння його ефективності

Тепер напишемо власний код для порівняння його ефективності в порівнянні з готовими бібліотеками лінійного програмування. Для цього код, який наведено в додатку Е на прикладі коду з додатку D порівняємо їх час виконання(рис. 2.7.1). Як можемо бачити найкращий час виконання Scipy. Для написання своєї програми використовували метод північно-західного кута для побудови опорного плану та метод потенціалів для його оптимізації.

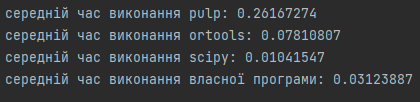


Рис. 2.7.1 – порівняння виконання різних програм

**ВИСНОВКИ**

Оптимізація транспортної задачі є досить непростою задачею, яка має безліч методів розв’язку та готових бібліотек з функціями для цього, що робить їх досить простими.

Розглянувши лише кілька прикладів можна сказати, що SciPy забезпечує високу точність та швидкість розв’язання задачі, що робить його відмінним інструментом для практичного застосування. Водночас Microsoft Excel залишається зручним для користувачів, які не мають навичок програмування, але потребують швидких та точних розв’язків.

Подальше вдосконалення методів може включати інтеграцію з машинним навчанням для більш точного прогнозування попиту та постачання, а також розвиток більш ефективних алгоритмів. Такі дослідження можуть значно розширити можливості оптимізації транспортної задачі в майбутньому.

Вивчення транспортної задачі також сприяє розвитку навичок у математичному моделюванні, оптимізації та програмуванні, що є важливими компетенціями для технічних спеціальностей.

**СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ**

1. Розв’язувач Microsoft Excel [електронний ресурс] – <https://support.microsoft.com/uk-ua/office/%D0%B2%D0%B8%D0%B7%D0%BD%D0%B0%D1%87%D0%B5%D0%BD%D0%BD%D1%8F-%D1%82%D0%B0-%D0%B2%D0%B8%D1%80%D1%96%D1%88%D0%B5%D0%BD%D0%BD%D1%8F-%D0%B7%D0%B0%D0%B4%D0%B0%D1%87-%D0%B7%D0%B0-%D0%B4%D0%BE%D0%BF%D0%BE%D0%BC%D0%BE%D0%B3%D0%BE%D1%8E-%D0%BD%D0%B0%D0%B4%D0%B1%D1%83%D0%B4%D0%BE%D0%B2%D0%B8-%D0%BF%D0%BE%D1%88%D1%83%D0%BA-%D1%80%D0%BE%D0%B7%D0%B2-%D1%8F%D0%B7%D0%B0%D0%BD%D0%BD%D1%8F-5d1a388f-079d-43ac-a7eb-f63e45925040>
2. Бібліотека PuLP [електронний ресурс] – <https://pypi.org/project/PuLP/>
3. Бібліотека SciPy [електронний ресурс] – <https://scipy.org/>
4. Бібліотека Or-tools [електронний ресурс] – https://developers.google.com/optimization/introduction/python?hl=ua

**ДОДАТОК А**

Файл: pulpprogram.py

import numpy as np

from pulp import LpProblem, LpMinimize, LpVariable, lpSum, value

def solve\_transportation\_problem(cost, supply, demand):

m, n = cost.shape

prob = LpProblem("Transportation\_Problem", LpMinimize)

# Define decision variables

x = [[LpVariable(f'x\_{i}\_{j}', lowBound=0, cat='Continuous') for j in range(n)] for i in range(m)]

# Objective function

prob += lpSum(cost[i][j] \* x[i][j] for i in range(m) for j in range(n))

# Supply constraints

for i in range(m):

prob += lpSum(x[i][j] for j in range(n)) == supply[i]

# Demand constraints

for j in range(n):

prob += lpSum(x[i][j] for i in range(m)) == demand[j]

# Solve the problem

prob.solve()

# Extract the solution

solution = np.array([[value(x[i][j]) for j in range(n)] for i in range(m)])

return solution, value(prob.objective)

# Example usage

cost = np.array([

[8, 7, 5, 10],

[13, 8, 10, 7],

[12, 4, 11, 9]

])

supply = [240, 330, 180]

demand = [230, 220, 130, 170]

optimized\_solution, objective\_value = solve\_transportation\_problem(cost, supply, demand)

print("Optimized solution using PuLP:")

print(optimized\_solution)

print("Objective function value (total transportation cost):")

print(objective\_value)

**ДОДАТОК В**

Файл: scipy\_program.py

import numpy as np

from scipy.optimize import linprog

def solve\_transportation\_problem\_scipy(cost, supply, demand):

m, n = cost.shape

c = cost.flatten()

# Create equality constraints for supply

A\_eq = np.zeros((m + n, m \* n))

b\_eq = np.concatenate((supply, demand))

for i in range(m):

for j in range(n):

A\_eq[i, i \* n + j] = 1

# Create equality constraints for demand

for j in range(n):

for i in range(m):

A\_eq[m + j, i \* n + j] = 1

bounds = [(0, None) for \_ in range(m \* n)]

# Solve the linear programming problem

result = linprog(c, A\_eq=A\_eq, b\_eq=b\_eq, bounds=bounds, method='highs')

if result.success:

solution = result.x.reshape(m, n)

return solution, result.fun

else:

raise ValueError("Linear programming did not converge")

# Example usage

cost = np.array([

[8, 7, 5, 10],

[13, 8, 10, 7],

[12, 4, 11, 9]

])

supply = [240, 330, 180]

demand = [230, 220, 130, 170]

optimized\_solution, objective\_value = solve\_transportation\_problem\_scipy(cost, supply, demand)

print("Optimized solution using SciPy:")

print(optimized\_solution)

print("Total transportation cost:")

print(objective\_value)

**ДОДАТОК С**

Файл: ortools\_program.py

from ortools.linear\_solver import pywraplp

import numpy as np

def solve\_transportation\_problem\_ortools(cost, supply, demand):

m, n = cost.shape

solver = pywraplp.Solver.CreateSolver('GLOP')

x = {}

for i in range(m):

for j in range(n):

x[i, j] = solver.NumVar(0, solver.infinity(), f'x\_{i}\_{j}')

# Supply constraints

for i in range(m):

solver.Add(solver.Sum(x[i, j] for j in range(n)) == supply[i])

# Demand constraints

for j in range(n):

solver.Add(solver.Sum(x[i, j] for i in range(m)) == demand[j])

# Objective function

objective = solver.Sum(cost[i, j] \* x[i, j] for i in range(m) for j in range(n))

solver.Minimize(objective)

status = solver.Solve()

if status == pywraplp.Solver.OPTIMAL:

solution = np.array([[x[i, j].solution\_value() for j in range(n)] for i in range(m)])

return solution, solver.Objective().Value()

else:

raise ValueError("The problem does not have an optimal solution.")

# Example usage

cost = np.array([

[8, 7, 5, 10],

[13, 8, 10, 7],

[12, 4, 11, 9]

])

supply = [240, 330, 180]

demand = [230, 220, 130, 170]

optimized\_solution, objective\_value = solve\_transportation\_problem\_ortools(cost, supply, demand)

print("Optimized solution using OR-Tools:")

print(optimized\_solution)

print("Objective function value (total transportation cost):")

print(objective\_value)

**ДОДАТОК D**

Файл: timecheck.py

import time

import numpy as np

from pulpprogram import solve\_transportation\_problem

from scipy\_program import solve\_transportation\_problem\_scipy

from ortools\_program import solve\_transportation\_problem\_ortools

def generate\_random\_transportation\_problem():

max\_supply = 300

max\_demand = 300

max\_cost = 20

num\_sources = np.random.randint(100)

num\_destinations = np.random.randint(100)

# Генеруємо вартості транспортування

cost = np.random.randint(1, max\_cost + 1, size=(num\_sources, num\_destinations))

# Генеруємо обсяги постачання

supply = np.random.randint(1, max\_supply + 1, size=num\_sources)

# Генеруємо обсяги попиту

demand = np.random.randint(1, max\_demand + 1, size=num\_destinations)

# Балансуємо попит і постачання

total\_supply = np.sum(supply)

total\_demand = np.sum(demand)

if total\_supply > total\_demand:

# Додаємо додатковий пункт призначення для балансування

extra\_demand = total\_supply - total\_demand

demand = np.append(demand, extra\_demand)

cost = np.hstack((cost, np.random.randint(1, max\_cost + 1, size=(num\_sources, 1))))

elif total\_demand > total\_supply:

# Додаємо додаткове джерело для балансування

extra\_supply = total\_demand - total\_supply

supply = np.append(supply, extra\_supply)

cost = np.vstack((cost, np.random.randint(1, max\_cost + 1, size=(1, num\_destinations))))

return cost, supply, demand

time1 = []

time2 = []

time3 = []

for i in range(100):

cost, supply, demand = generate\_random\_transportation\_problem()

tic1 = time.time()

solve\_transportation\_problem(cost, supply, demand)

toc1 = time.time()

time1.append(toc1 - tic1)

tic2 = time.time()

solve\_transportation\_problem\_ortools(cost, supply, demand)

toc2 = time.time()

time2.append(toc2 - tic2)

tic3 = time.time()

solve\_transportation\_problem\_scipy(cost, supply, demand)

toc3 = time.time()

time3.append(toc3 - tic3)

print(f'середній час виконання pulp: {np.mean(time1):.8f}')

print(f'середній час виконання ortools: {np.mean(time2):.8f}')

print(f'середній час виконання scipy: {np.mean(time3):.8f}')

**ДОДАТОК Е**

Файл: mycode.py

import numpy as np

def northwest\_corner\_method(cost, supply, demand):

m, n = cost.shape

x = np.zeros((m, n))

i = j = 0

while i < m and j < n:

min\_val = min(supply[i], demand[j])

x[i, j] = min\_val

supply[i] -= min\_val

demand[j] -= min\_val

if supply[i] == 0:

i += 1

if demand[j] == 0:

j += 1

return x

def calculate\_potentials(cost, x):

m, n = cost.shape

u = np.full(m, np.nan)

v = np.full(n, np.nan)

u[0] = 0

while np.any(np.isnan(u)) or np.any(np.isnan(v)):

for i in range(m):

for j in range(n):

if x[i, j] > 0:

if not np.isnan(u[i]) and np.isnan(v[j]):

v[j] = cost[i, j] - u[i]

elif np.isnan(u[i]) and not np.isnan(v[j]):

u[i] = cost[i, j] - v[j]

return u, v

def find\_entering\_variable(cost, u, v):

m, n = cost.shape

delta = np.full((m, n), np.inf)

for i in range(m):

for j in range(n):

if not np.isnan(u[i]) and not np.isnan(v[j]):

delta[i, j] = cost[i, j] - (u[i] + v[j])

min\_val = np.min(delta)

if min\_val >= 0:

return None

return np.unravel\_index(np.argmin(delta), delta.shape)

def find\_loop(x, start):

m, n = x.shape

u, v = calculate\_potentials(cost, x)

delta = np.full((m, n), np.inf)

for i in range(m):

for j in range(n):

if not np.isnan(u[i]) and not np.isnan(v[j]):

delta[i, j] = cost[i, j] - (u[i] + v[j])

for i in range(m):

for j in range(n):

if start[0] != i and start[1] != j:

if delta[i, j] <= 0 and delta[start[0], j] == 0 and delta[i, start[1]] == 0:

loop = [(start[0], j), (i, j), (i, start[1]), (start[0], start[1])]

return loop

return False

def adjust\_solution(x, loop):

if not loop:

return

loop\_positions = loop[::2]

loop\_values = [x[i, j] for i, j in loop\_positions]

min\_val = min(loop\_values)

for k, (i, j) in enumerate(loop):

if k % 2 == 0:

x[i, j] -= min\_val

else:

x[i, j] += min\_val

def optimize\_transportation(cost, supply, demand):

x = northwest\_corner\_method(cost, supply, demand)

while True:

u, v = calculate\_potentials(cost, x)

entering = find\_entering\_variable(cost, u, v)

if entering is None:

break

loop = find\_loop(x, entering)

if not loop:

break

adjust\_solution(x, loop)

return x

cost = np.array([

[8, 7, 5, 10],

[13, 8, 10, 7],

[12, 4, 11, 9]

])

supply = [240, 330, 180]

demand = [230, 220, 130, 170]

optimized\_solution = optimize\_transportation(cost, supply, demand)

print("Оптимізоване рішення:")

print(optimized\_solution)

print("Загальна вартість транспортування:")

print(np.sum(optimized\_solution \* cost))