Київський національний університет імені Тараса Шевченка
Факультет комп’ютерних наук та кібернетики
Кафедра дослідження операцій

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА
за спеціальністю 113 «Прикладна математика»
на тему:

Методи розв’язання транспортної задачі та область її практичного використання

студентки 4 курсу
Сушко Єлизавети Олександрівни

Науковий керівник:
доцент, доктор фізико–математичних наук
Самойленко І.В.

Робота заслухана на засіданні кафедри дослідження операцій та рекомендована до захисту в ЕК, протокол № 9 від 23.05.2023 р.

Завідувач кафедри ДО проф. Іксанов О.М.

Київ – 2023

ЗМІСТ

Вступ……………………………………………………………………...….3

Розділ 1. Методи знаходження базисного розв'язку…………………………..4
1.1 Метод північно-західного кута……………………………………………...4
1.2 Метод мінімального елемента………………………………………………9
1.3 Метод Фогеля……………………………………………………………..10
1.4 Угорський метод…………………………………………………………14

Розділ 2. Метод потенціалів…………………………………………………17
2.1 Двоїстий критерій оптимальності…………………………………………17
2.2 Алгоритм методу потенціалів……………………………………………...18

Розділ 3. Практичне застосування транспортної задачі……………………...23

Висновки……………………………………………………………………...29
Література…………………………………………………………………....30

Вступ

Транспортні моделі, як задачі, є спеціальним класом лінійного програмування. Вони широко використовуються для опису переміщення або перевезення різних товарів від пункту відправлення (наприклад, місце виробництва) до пункту призначення (складу, магазину, вантажосховища). Головна мета транспортних задач полягає у визначенні оптимальних обсягів перевезень з пунктів відправлення до пунктів призначення з мінімальною сумарною вартістю перевезень. При цьому необхідно враховувати обмеження, що стосуються доступних обсягів вантажу в пунктах відправлення (пропозиція), а також обмеження, які визначають потребу у вантажі у пунктах призначення (попит). В рамках транспортної моделі припускається, що вартість перевезення по будь-якому маршруту прямо пропорційна обсягу вантажу, який перевозиться цим маршрутом.

Математична модель у координатній формі ТЗЛП має вигляд:

$$L(\boldsymbol{x}) = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \to \min,$$

$$\sum_{j=1}^{n} x_{ij} = a_i, \quad i = 1, \ldots, m,$$

$$\sum_{i=1}^{m} x_{ij} = b_j, \quad j = 1, \ldots, n,$$

$$x_{ij} \geq 0, \quad i = 1, \ldots, m, \; j = 1, \ldots, n,$$

$$\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$$

Слід зауважити, що остання умова визначає збалансовану модель ТЗЛП.
У матричній формі ТЗЛП описується так:

$$L(\boldsymbol{x}) = \boldsymbol{c}\,\boldsymbol{x} \to \min, \quad \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}, \quad \boldsymbol{x} \geq \boldsymbol{0}$$

Розділ 1. Методи знаходження базисного розв’язку

У загальній транспортній моделі, яка має m пунктів відправлення і n пунктів призначення, встановлюється m + n незалежних обмежень у вигляді рівнянь, по одному на кожний пункт відправлення та призначення. Оскільки транспортна модель завжди є збалансованою (сума пропозицій дорівнює сумі попиту), одне з цих рівнянь повинно бути надмірним. Тому транспортна модель має m + n - 1 незалежних обмежень, що означає, що початкове базисне рішення складається з m + n - 1 базисних змінних.

Ця спеціальна структура транспортної моделі дозволяє використовувати певні методи для побудови початкового рішення замість використання штучних змінних, як це робиться у симплекс-методі. Деякі з таких методів включають:

1. Метод північно-західного кута.
2. Метод найменшої вартості.
3. Метод Фогеля.

Ці методи відрізняються "якістю" початкового рішення, тобто наскільки близько воно знаходиться до оптимального. Загалом, метод Фогеля надає найкращі результати, а метод північно-західного кута - найгірші. Однак метод північно-західного кута вимагає меншої кількості обчислень.

1.1 Метод північно-західного кута

Процес виконання починається з верхнього лівого кута транспортної таблиці, який називається північно-західним кутом.

Крок 1: Початкові значення присвоюються змінній x11, враховуючи максимальні допустимі обмеження, які визначені попитом і пропозицією.

Крок 2: Викреслюється рядок або стовпець, що повністю задовольняється (завершується) пропозицією (попитом). Це означає, що в цьому рядку (стовпці) не будуть призначені значення жодній іншій змінній, крім змінної, що була визначена на першому кроці. Якщо як попит, так і пропозиція задовольняються одночасно, то викреслюється лише рядок або лише стовпець.

Крок 3: Якщо залишився не викресленим лише один рядок або лише один стовпець, процес зупиняється. У протилежному випадку, переходимо до комірки праворуч, якщо був викреслений стовпець, або до комірки нижче, якщо був викреслений рядок. Потім повертаємось до першого кроку.

Таким чином, процес продовжується до досягнення оптимального розв'язку транспортної задачі.

Нижче наведено код на мові програмування Python, що реалізує метод північно-західного кута за допомогою бібліотеки NumPy.

```
import numpy as np

def get_balanced_transportation_problem(supply, demand, costs):
    total_supply = sum(supply)
    total_demand = sum(demand)

    if total_supply < total_demand:
        new_supply = supply + [total_demand - total_supply]
        new_costs = costs
        return new_supply, demand, new_costs
    if total_supply > total_demand:
        new_demand = demand + [total_supply - total_demand]
        new_costs = costs + [[0 for _ in demand]]
        return supply, new_demand, new_costs
    return supply, demand, costs

def userCostInput():
    sp_row = int(input('Enter length row value: '))
    dm_column = int(input('Enter length column value: '))
    userCostInput.costs = [[int(input("Enter Cost value by row: ")) for x in
range(dm_column)] for y in range(sp_row)]

def userSupplyAndDemand():
    supply = int(input("Enter the length of supply values: "))
    demand = int(input("Enter the length of demand values: "))
    userSupplyAndDemand.Supply = [int(input("Enter Supply values: ")) for x in
range(supply)]
    userSupplyAndDemand.Demand = [int(input("Enter Demand values: ")) for x in
range(demand)]

# ## NorthWest Corner Method
def north_west_corner(model_supply, model_demand):
    supply_copy = model_supply.copy()
    demand_copy = model_demand.copy()
    i = 0
    j = 0
    bfs = []
    while len(bfs) < len(model_supply) + len(model_demand) - 1:
        s = supply_copy[i]
        d = demand_copy[j]
        v = min(s, d)
        supply_copy[i] -= v
```

```
        demand_copy[j] -= v
        bfs.append(((i, j), v))
        if supply_copy[i] == 0 and i < len(model_supply) - 1:
            i += 1
        elif demand_copy[j] == 0 and j < len(model_demand) - 1:
            j += 1
    return bfs

# ## Stepping Stone
def get_us_and_vs(bfs, costs):
    us = [None] * len(costs)
    vs = [None] * len(costs[0])
    us[0] = 0
    bfs_copy = bfs.copy()
    while len(bfs_copy) > 0:
        for index, bv in enumerate(bfs_copy):
            i, j = bv[0]
            if us[i] is None and vs[j] is None: continue

            cost = costs[i][j]
            if us[i] is None:
                us[i] = cost - vs[j]
            else:
                vs[j] = cost - us[i]
            bfs_copy.pop(index)
            break

    return us, vs

def get_ws(bfs, costs, us, vs):
    ws = []
    for i, row in enumerate(costs):
        for j, cost in enumerate(row):
            non_basic = all([p[0] != i or p[1] != j for p, v in bfs])
            if non_basic:
                ws.append(((i, j), us[i] + vs[j] - cost))

    return ws

def can_be_improved(ws):
    for p, v in ws:
        if v > 0: return True
    return False

def get_entering_variable_position(ws):
```

```
    ws_copy = ws.copy()
    ws_copy.sort(key=lambda w: w[1])
    return ws_copy[-1][0]

def get_possible_next_nodes(loop, not_visited):
    last_node = loop[-1]
    nodes_in_row = [n for n in not_visited if n[0] == last_node[0]]
    nodes_in_column = [n for n in not_visited if n[1] == last_node[1]]
    if len(loop) < 2:
        return nodes_in_row + nodes_in_column
    else:
        prev_node = loop[-2]
        row_move = prev_node[0] == last_node[0]
        if row_move: return nodes_in_column
        return nodes_in_row

def get_loop(bv_positions, ev_position):
    def inner(loop):
        if len(loop) > 3:
            can_be_closed = len(get_possible_next_nodes(loop, [ev_position])) == 1
            if can_be_closed: return loop

        not_visited = list(set(bv_positions) - set(loop))
        possible_next_nodes = get_possible_next_nodes(loop, not_visited)
        for next_node in possible_next_nodes:
            new_loop = inner(loop + [next_node])
            if new_loop: return new_loop

    return inner([ev_position])

def loop_pivoting(bfs, loop):
    even_cells = loop[0::2]
    odd_cells = loop[1::2]
    get_bv = lambda pos: next(v for p, v in bfs if p == pos)
    leaving_position = sorted(odd_cells, key=get_bv)[0]
    leaving_value = get_bv(leaving_position)

    new_bfs = []
for p, v in [bv for bv in bfs if bv[0] != leaving_position] + [(loop[0], 0)]:
        if p in even_cells:
            v += leaving_value
        elif p in odd_cells:
            v -= leaving_value
        new_bfs.append((p, v))
```

```
    return new_bfs

# Final computation
def transportation_simplex_method(supply, demand, costs):
    balanced_supply, balanced_demand, balanced_costs =
get_balanced_transportation_problem(
        model_supply, model_demand, costs
    )
    def inner(bfs):
        us, vs = get_us_and_vs(bfs, balanced_costs)
        ws = get_ws(bfs, balanced_costs, us, vs)
        if can_be_improved(ws):
            ev_position = get_entering_variable_position(ws)
            loop = get_loop([p for p, v in bfs], ev_position)
            return inner(loop_pivoting(bfs, loop))
        return bfs

    basic_variables = inner(north_west_corner(balanced_supply, balanced_demand))
    solution = np.zeros((len(costs), len(costs[0])))
    for (i, j), v in basic_variables:
        solution[i][j] = v

    return solution

def get_total_cost(costs, solution):
    total_cost = 0
    for i, row in enumerate(costs):
        for j, cost in enumerate(row):
            total_cost += cost * solution[i][j]
    return total_cost

if name == "__main__":
    userCostInput()

   costs = userCostInput.costs
    print(f'Cost Values: \n {costs}')

    userSupplyAndDemand()

    model_supply = userSupplyAndDemand.Supply
    model_demand = userSupplyAndDemand.Demand
    print(f'\n Supply: {model_supply}')
    print(f'Demand: {model_demand}')

    bfs = north_west_corner(model_supply, model_demand)
```

```
print(f'\nBasic Feasible Solution:\n {np.array(bfs, dtype=object)}')

get_balanced_transportation_problem(model_supply,model_demand,costs)
solution = transportation_simplex_method(model_supply, model_demand, costs)
print(f"\n Final Tableau:\n {solution}")
print(f'Optimal Solution: {get_total_cost(costs, solution)}')
```

1.2 Метод мінімального елемента

Цей метод, на відміну від методу північно-західного кута, знаходить краще початкове рішення шляхом вибору змінних з найменшою вартістю. Його кроки виконання описуються наступним чином:

1. Пошук найменшої вартості: Спочатку проглядається вся транспортна таблиця для знаходження комірки з найменшою вартістю.

2. Присвоєння значень: Змінній, яка відповідає цій комірці, присвоюється найбільше можливе значення, враховуючи обмеження попиту та пропозиції. Якщо є кілька таких змінних, можна обрати будь-яку з них.

3. Викреслення рядка або стовпця: Після цього відповідний стовпець або рядок викреслюються, а значення попиту та пропозиції відповідно коригуються. Якщо і попит, і пропозиція задовольняються одночасно, викреслюється лише рядок або лише стовпець, так само, як у методі північно-західного кута.

4. Пошук нового осередку: Потім переглядаються невикреслені комірки і вибирається новий осередок з найменшою вартістю.

5. Повторення процесу: Описаний процес продовжується до тих пір, поки не залишиться лише один невикреслений рядок або стовпець.

Таким чином, метод найменшої вартості забезпечує краще початкове рішення шляхом вибору змінних з найменшою вартістю і послідовного викреслення рядків або стовпців до досягнення оптимального розв'язку транспортної задачі.

Нижче наведено код на мові програмування Python, що реалізує метод найменшої вартості за допомогою модулю PuLP (Python Linear programming).

```
from pulp import *
import time
start = time.time()
```

```
a1 = pulp.LpVariable("a1", lowBound=0)
a2 = pulp.LpVariable("a2", lowBound=0)
a3 = pulp.LpVariable("a3", lowBound=0)
a4 = pulp.LpVariable("a4", lowBound=0)
a5 = pulp.LpVariable("a5", lowBound=0)
a6 = pulp.LpVariable("a6", lowBound=0)
a7 = pulp.LpVariable("a7", lowBound=0)
a8 = pulp.LpVariable("a8", lowBound=0)
a9 = pulp.LpVariable("a9", lowBound=0)
problem = pulp.LpProblem('0',pulp.LpMaximize)
problem += 5*a1 + 13*a2 - 2*a3 + 7*a4 - 10*a5 + 3*a6 - 35*a7 + 6*a8 - 19*a9,
"Функція цілі"
problem += a1 + a5 + a7 <= 39,"1"
problem += a2 + a4 + a6 == 12, "2"
problem += a3 + a8 +  a9 == 45, "3"
problem += a4+ a6+ a8 == 70, "4"
problem += a5 + a7 + a3 == 27, "5"
problem += a6 + a3 + a9 == 9, "6"
problem.solve()
print ("Результат:")
for variable in problem.variables():
   print (variable.name, "=", variable.varValue)
print ("Вартість доставки:")
print (abs(value(problem.objective)))
stop = time.time()
print ("Час :")

print(stop - start)
```

1.3 Метод Фогеля

Цей метод є варіацією методу найменшої вартості і в загальному випадку забезпечує краще початкове рішення. Опис алгоритму цього методу включає наступні кроки:

1. Обчислення штрафу: Для кожного рядка або стовпця з позитивною пропозицією (попитом) визначається штраф шляхом віднімання найменшої вартості від наступної за величиною в даному рядку або стовпці.

2. Вибір рядка або стовпця з найбільшим штрафом: Виділяється рядок або стовпець з найбільшим штрафом. Якщо є декілька таких рядків або стовпців, можна обрати будь-який з них. З цього виділеного рядка або стовпця вибирається змінна, якій відповідає мінімальна вартість, і їй присвоюється найбільше можливе значення з урахуванням обмежень. Потім змінюються

значення невикресленого попиту та пропозиції відповідно до присвоєного значення змінної. Рядок або стовпець, що відповідають виконаному обмеженню, викреслюються з таблиці. Якщо і попит, і пропозиція одночасно задовольняються, викреслюється тільки рядок або стовпець, і викресленому рядку або стовпцю приписується нульова пропозиція або попит.

3. Перевірка закінчення обчислень:

a) Якщо не залишилося лише одного невикресленого рядка або стовпця з нульовим попитом або пропозицією, обчислення завершуються.

b) Якщо залишився лише один невикреслений рядок (стовпець) з позитивною пропозицією (попитом), в цьому рядку (стовпці) знаходяться базисні змінні за допомогою методу найменшої вартості, і обчислення завершуються.

c) Якщо всі невикреслені рядки і стовпці відповідають нульовим обсягам пропозиції та попиту, знаходяться нульові базисні змінні за допомогою методу найменшої вартості, і обчислення завершуються.

d) У всіх інших випадках потрібно повернутися до кроку 1.

Тож, цей метод виконує ітерації, поки не буде досягнуто оптимального розв'язку, змінюючи виділені рядки або стовпці та присвоюючи їм значення з найменшою вартістю.

```
# попит - A,B,C,D

# W,X,Y - пропозиція


costs  = {'W': {'A': 8, 'B': 12, 'C': 7, 'D': 4},

         'X': {'A': 5, 'B': 13, 'C': 9, 'D': 1},

         'Y': {'A': 2, 'B': 3, 'C': 15, 'D': 19}}

demand = {'A': 250, 'B': 275, 'C': 225, 'D': 200}

cols = sorted(demand.iterkeys())

supply = {'W': 300, 'X': 400, 'Y': 270}

res = dict((k, defaultdict(int)) for k in costs)
```

```
g = {}

for x in supply:
    g[x] = sorted(costs[x].iterkeys(), key=lambda g: costs[x][g])
for x in demand:
    g[x] = sorted(costs.iterkeys(), key=lambda g: costs[g][x])

while g:
    d = {}
    for x in demand:
        d[x] = (costs[g[x][1]][x] - costs[g[x][0]][x]) if len(g[x]) > 1 else
costs[g[x][0]][x]
    s = {}
    for x in supply:
        s[x] = (costs[x][g[x][1]] - costs[x][g[x][0]]) if len(g[x]) > 1 else
costs[x][g[x][0]]
    f = max(d, key=lambda n: d[n])
    t = max(s, key=lambda n: s[n])
    t, f = (f, g[f][0]) if d[f] > s[t] else (g[t][0], t)
    v = min(supply[f], demand[t])
    res[f][t] += v
    demand[t] -= v
    if demand[t] == 0:
        for k, n in supply.iteritems():
```

```
            if n != 0:
                g[k].remove(t)
        del g[t]
        del demand[t]
    supply[f] -= v
    if supply[f] == 0:
        for k, n in demand.iteritems():
            if n != 0:
                g[k].remove(f)
        del g[f]
        del supply[f]

for n in cols:
    print "\t", n,
print
cost = 0
for g in sorted(costs):
    print g, "\t",
    for n in cols:
        y = res[g][n]
        if y != 0:
            print y,
        cost += y * costs[g][n]
        print "\t",
```

```
    print
print "\n\nTotal Cost = ", cost
```

1.4 Угорський метод

Задача про призначення полягає у виборі найкращого працівника для виконання конкретної роботи. В цій задачі потрібно розподілити працівників на різні роботи, де кожен працівник може виконувати будь-яку роботу з різним рівнем вмінь. Якщо працівник, який має необхідну кваліфікацію, призначений на певну роботу, то вартість виконання цієї роботи буде нижчою, ніж у випадку, коли працівник з непідходящою кваліфікацією виконує цю роботу. Метою задачі є знайти оптимальний розподіл працівників по всіх заявлених роботах з мінімальною вартістю.

У цій задачі важливо враховувати відповідність між кваліфікацією працівників і вимогами робіт. Оптимальне рішення полягає в призначенні працівників на роботи, які найкраще відповідають їхнім навичкам, що призводить до зниження вартості виконання робіт.

У задачі про призначення коефіцієнт $c_{i\,j}$ представляє вартість призначення працівника i на роботу j (i, j = 1, 2,..., n). Кількість працівників, яка дорівнює кількості робіт, не є обмеженням, оскільки завжди можна ввести фіктивних працівників або роботи. Задача про призначення є окремим випадком транспортної задачі, де працівники відповідають пунктам відправлення, а роботи - пунктам призначення. В даному випадку всі величини попиту та пропозиції рівні 1. Вартість "транспортування" працівника і на роботу j позначається як $c_{i\,j}$. Задачу про призначення можна ефективно розв'язати так само, як і транспортну задачу. Однак, той факт, що всі величини попиту рівні 1, призводить до розробки спрощеного алгоритму розв'язання, який називається угорським методом. Хоча цей метод не має прямого відношення до транспортної задачі, він, подібно до методу потенціалів, ґрунтується на симплекс-методі.

Алгоритм угорського методу для розв'язання задачі про призначення складається з наступних кроків:

Створення початкової матриці вартостей.

Крок 1: Визначення найменшого елемента в кожному рядку матриці і віднімання цього значення від усіх елементів рядка.

Крок 2: Визначення найменшого елемента в кожному стовпці матриці і віднімання цього значення від усіх елементів стовпця.

Крок 3: Знаходження оптимального призначення, де нульові елементи матриці відповідають оптимальним призначенням працівників на роботи.

Перевірка умови закінчення: Якщо кількість оптимальних призначень дорівнює кількості робіт, або кількості працівників, то розв'язок знайдено. В іншому випадку переходимо до наступного кроку.

Крок 4: Здійснення модифікації матриці вартостей для пошуку нових оптимальних призначень.

Повторення кроків 2-6 до досягнення оптимального розв'язку.

Цей алгоритм дозволяє знайти оптимальне призначення працівників на роботи, мінімізуючи вартість виконання всіх робіт.

У деяких випадках нульові елементи, отримані на першому та другому кроках угорського методу, не дозволяють безпосередньо отримати допустиме рішення. Тоді необхідні додаткові дії для знаходження оптимального (припустимого) рішення.

Якщо після виконання першого та другого кроків описаного алгоритму не отримано допустиме рішення, виконайте наведені нижче дії.

i) в останній матриці проведіть мінімальне число горизонтальних і вертикальних прямих по рядках і стовпцях, щоб викреслити у матриці всі нульові елементи.

ii) Знайдіть найменший невикреслений елемент і відніміть його з інших невикреслених елементів і додайте до елементів, що стоять на перетині проведених на попередньому кроці прямих

iii) Якщо новий розподіл нульових елементів не дозволяє побудувати допустиме рішення, повторіть крок 2,a. Інакше перейдіть до третього кроку алгоритму.

Нижче наведено код на мові програмування Python, що реалізує угорський метод за допомогою бібліотеки NumPy, що дозволяє працювати з масивами даних, а також модулю munkers.

```
import numpy as np
from munkres import Munkres, print_matrix
matrix = [[2,10,9,7],
        [15,4,14,8],
        [13,14,16,11],
        [4,15,13,19]]
m = Munkres()
indexes = m.compute(matrix)
print_matrix(matrix, msg='Lowest cost through this matrix:')
total = 0
for row, column in indexes:
   value = matrix[row][column]
   total += value
   print('(%d, %d) -> %d' % (row, column, value))
print('total cost: %d' % total)
print('-------------now step by step------------------')
print('-------------minus in column ------------------')
matrix = np.array([[2,10,9,7],
        [15,4,14,8],
        [13,14,16,11],
        [4,15,13,19]])
for i in range(4):
   matrix[:,i] = (np.transpose(matrix[:,i] - min(matrix[:,i])))
print(matrix)
print('-------------minus in raw ------------------')
for i in range(5):
   matrix[i,:] = (np.transpose(matrix[i,:] - min(matrix[i,:])))
print(matrix)
```

Розділ 2. Метод Потенціалів

Для розв'язання задачі лінійного програмування методом потенціалів використовується двоїстий критерій оптимальності.

2.1 Двоїстий критерій оптимальності

Задача лінійного програмування (ЗЛП) може бути записана у формі:

$$\boldsymbol{c\,x} \;\rightarrow\; \min\,, \quad \boldsymbol{A\;x} = \boldsymbol{b}\,, \quad \boldsymbol{x} \geq \boldsymbol{0}$$

де c - вектор витрат, x - вектор змінних рішення, A - матриця коефіцієнтів обмежень, b - вектор обмежень.

Двоїстою задачею до цієї ЗЛП є:

$$\boldsymbol{u\,b} \;\rightarrow\; \max\,, \quad \boldsymbol{u\;A} \;\leq \boldsymbol{c}\,,$$

де u - вектор двоїстих змінних, що має розмірність (m+n), де m - кількість обмежень типу Ax = b, n - кількість обмежень типу x ≥ 0.

Визначимо компоненти вектора u таким чином:

$$u = (-u_1, \dots, -u_m, v_1, \dots, v_n,).$$

Двоїста змінна $-u_i$ відповідає обмеженню

$$\sum_{j=1}^{n} x_{i\,j} = \; a_i\,, \quad \text{для} \quad i = 1, \dots, m\,,$$

а двоїста змінна v_j відповідає обмеженню

$$\sum_{i=1}^{m} x_{i\,j} = \; b_j\,, \quad j = 1, \dots, n\,.$$

Базисний розв'язок $x = \; \|x_{i\,j}\|\,, i = 1, \dots, m,\;\; j = 1, \dots, n\,,$ є оптимальним розв'язком ЗЛП тоді і тільки тоді, коли існують потенціали $u_{\,i}\,,\; i = 1, \dots, m,\;\; j = 1, \dots, n\,,$ такі, що

$v_j - u_i \; = c_{i\,j},$ якщо $x_{i\,j}$ є базисним елементом ($\Delta_{\,i\,j} \; = 0$ для базисних клітинок),

$v_j - u_i \; = c_{i\,j}$, якщо $x_{i\,j}$ є небазисним елементом ($\Delta_{\,i\,j} \;\geq 0$ для небазисних клітинок).

2.2 Алгоритм методу потенціалів

1. Знаходимо базисний початковий невироджений розв'язок $\boldsymbol{x}$ ЗЛП за допомогою одного з відомих методів.

2. Знаходимо потенціали u_i, $i = 1, \dots, m,$ $j = 1, \dots, n$, таким чином, щоб в кожній базисній клітинці стверджувалось співвідношення $\Delta_{i\,j} = 0$, або $v_j - u_i = c_{i\,j},$ (i, j) — базисні клітинки.

3. Обчислюємо значення $\Delta_{i\,j}$ для небазисних клітинок. Базисний розв'язок $\boldsymbol{x}$ є оптимальним, якщо всі значення є невід'ємними. В іншому випадку, його можна поліпшити, введеною до базисних однією з клітинок, де $\Delta_{i\,j} < 0$ (зазвичай, клітинка $(i_{0,}\ j_0)$, для якої $\Delta_{i_0 j_0} = min\,\Delta_{i\,j}$).

4. За допомогою процедури перерозподілу перевезень вздовж циклу **C**, клітинка ($i_{0,}\ j_0$) (змінна $x_{i_0\,j_0}$) вводиться до базисних, а клітинка (k, l), де

$$x_{k\ l} = \theta = \min_{i,j:\,(i,j)\in\ C^-} x_{ij}$$

виводиться з базисних. Повертаємося до кроку 2 алгоритму.

Метод потенціалів для розв'язання ЗЛП є модифікованим симплекс-методом, застосованим до ЗЛП.

Нижче наведено код на мові програмування C#, що реалізує метод потенціалів:

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApplication11

{

    class Program

    {
```

```
struct Element
{
    public int Delivery { get; set; }
    public int Value { get; set; }
    public static int FindMinElement(int a, int b)
    {
        if (a > b) return b;
        if (a == b) { return a; }
        else return a;
    }
}

static void Main(string[] args)
{
    int i = 0;
    int j = 0;
    int n;
    Console.WriteLine("Введіть кількість A");
    n = Convert.ToInt32(Console.ReadLine());
    int[] a = new int[n];
    Console.WriteLine("Введіть кількість B");
    int m = Convert.ToInt32(Console.ReadLine());
    int[] b = new int[m];
    Element[,] C = new Element[n, m];
```

```
Console.WriteLine("Введіть a[i]");
for (i = 0; i < a.Length; i++)
{
    a[i] = Convert.ToInt32(Console.ReadLine());
}
Console.WriteLine("Введіть b[i]");
for (j = 0; j < b.Length; j++)
{
    b[j] = Convert.ToInt32(Console.ReadLine());
}
Console.ForegroundColor = ConsoleColor.White;
Console.WriteLine("Введите C[i][j]");
for (i = 0; i < n; i++)
{
    for (j = 0; j < m; j++)
    {
        Console.Write("a[{0},{1}] = ", i, j);
        Console.ForegroundColor = ConsoleColor.Red;
        C[i, j].Value = Convert.ToInt32(Console.ReadLine());
        Console.ResetColor();
    }
}
i = j = 0;
```

```
while (i < n && j < m)
{
    try
    {
        if (a[i] == 0) { i++; }
        if (b[j] == 0) { j++; }
        if (a[i] == 0 && b[j] == 0) { i++; j++; }
        C[i, j].Delivery = Element.FindMinElement(a[i], b[j]);
        a[i] -= C[i, j].Delivery;
        b[j] -= C[i, j].Delivery;
    }
    catch { }
}
for (i = 0; i < n; i++)
{
    for (j = 0; j < m; j++)
    {
        if (C[i, j].Delivery != 0)
        {
            Console.ForegroundColor = ConsoleColor.Blue;
            Console.Write("{0}", C[i, j].Value);
            Console.ForegroundColor = ConsoleColor.Red;
            Console.Write("({0})", C[i, j].Delivery); Console.ResetColor();
        }
```

```
        else
            Console.Write("{0}({1})", C[i, j].Value, C[i, j].Delivery);
    }
    Console.WriteLine();
}
int ResultFunction = 0;

for (i = 0; i < n; i++)
{
    for (j = 0; j < m; j++)
    {
        ResultFunction += (C[i, j].Value * C[i, j].Delivery);
    }
}
Console.WriteLine(" Result = {0}", ResultFunction);
Console.ReadLine();
        }
    }
}
```

Розділ 3. Практичне застосування транспортної задачі

Математичне моделювання грає велику роль у вирішенні різних економічних проблем, дозволяючи визначити цілі та типи їх вирішення, забезпечуючи структуру для цілісного аналізу. За допомогою кількісних моделей можливе докладніше вивчення отриманих даних, тому економіко-математичне моделювання є невід'ємною частиною будь-якого дослідження галузі економіки. Зважаючи на складність економіки для її модельного опису, використовуються різні підходи, одним з яких є лінійне програмування.

Частиною лінійного програмування є транспортні задачі, які грають особливу роль у зменшенні транспортних витрат підприємства. Це актуальне питання за умов ринкової економіки, коли будь-які витрати мають бути мінімізовані, адже тоді витрати покриваються меншою частиною прибутку, а також дозволяють знизити собівартість продукції на ринку, що робить підприємство більш конкурентоспроможним.

Транспортна задача є однією з найважливіших тем в галузі оптимізації та управління логістикою. Вона полягає у пошуку оптимального способу перевезення товарів з джерел постачання до пунктів споживання з урахуванням обмежень та цілей. Також, у деяких випадках транспортну модель можна використовуватиме для опису ситуацій, щодо управління запасами, управління рухом капіталів, призначення персоналу і так далі.

Основна важливість теми "Методи розв'язання транспортної задачі та область її застосування" полягає в наступних аспектах:

1. Економічна ефективність: Розв'язання транспортної задачі дозволяє знайти оптимальний спосіб розподілу ресурсів, таких як сировина, продукція або інші матеріальні блага. Це допомагає знизити витрати на транспортування, зберегти час та ресурси, оптимізувати логістичні процеси і, отже, збільшити ефективність бізнесу.

2. Планування маршрутів та розкладів: Транспортна задача допомагає вирішити проблему планування оптимальних маршрутів та розкладів транспортних засобів, що має велике значення для компаній, які займаються доставкою товарів або наданням послуг у сфері транспорту.

3. Управління логістикою: Розв'язання транспортної задачі дозволяє управляти логістичними потоками та розподіляти ресурси оптимальним чином.

Це важливо для підтримки постійного постачання товарів та задоволення потреб споживачів.

4. Граничні ситуації та кризові ситуації: У разі виникнення непередбачуваних обставин, таких як природні катастрофи або інші надзвичайні події, методи розв'язання транспортної задачі можуть допомогти вирішити екстрені ситуації шляхом реструктуризації транспортних потоків та перерозподілу ресурсів.

5. Застосування в інших галузях: Методи розв'язання транспортної задачі мають широкий спектр застосувань в різних галузях, таких як електроенергетика, телекомунікації, медицина, соціальне планування тощо. Вони дозволяють оптимізувати розподіл ресурсів та покращувати якість надання послуг у цих галузях.

Підсумовуючи, вивчення методів розв'язання транспортної задачі та їх застосування є важливим засобом у подальшому вирішенні багатьох економічних проблем, що виникають перед підприємствами. З їхньою допомогою можливе як раціональне планування шляхів, планування логістичних процесів в різних галузях діяльності, так і усунення далеких, повторних перевезень. Це веде до швидшої доставки товарів, скорочення витрат виробництва палива, ремонту машин, тобто до скорочення транспортних витрат.

Нижче наведено код програми, що розв'язує практичну задачу про пошук оптимальної моделі для постояльця готеля до найближчого таксі. Були обрані 2 датасети (New York.csv – 392 вибрані точки з координатами, назвами та адресами готелів; Taxi.csv - датасет місць, де відбувалися ДТП на вулицях Манхеттена за 2016 рік, дані 392 локації вважаються місцями знаходження таксі).

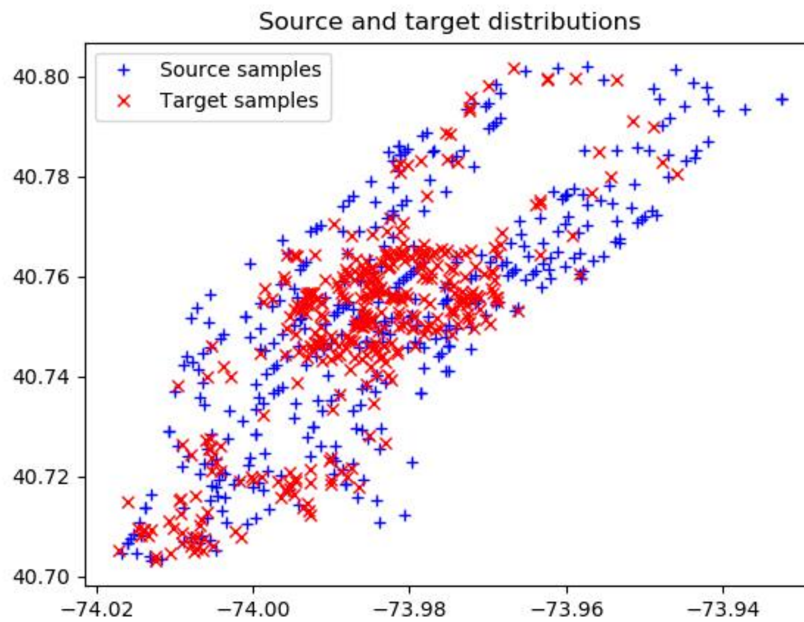
За умову взято: необхідно відправити кожну машину до одного готелю так, щоб усі маршрути були оптимальними, тобто сума всіх відстаней була мінімальною. Для спрощення завдання не треба враховувати геометрію вулиць, а просто знайти відстані між готелями та поточними положеннями таксі.

Завдання вирішено за допомогою мови програмування Python із використанням бібліотеки POT: Python Optimal Transport.

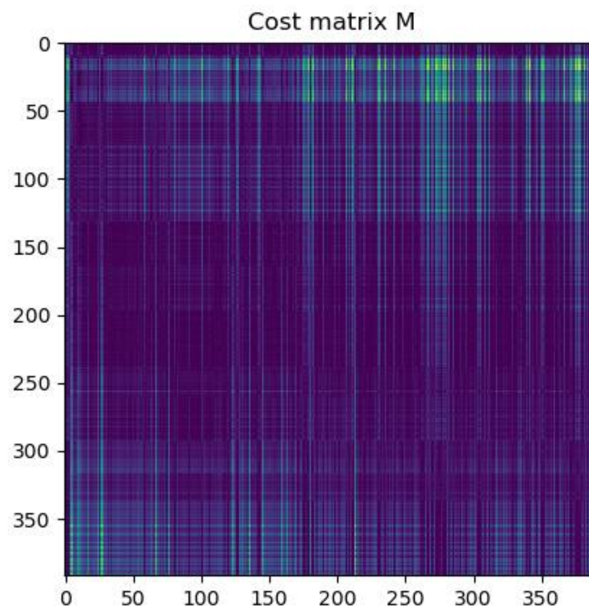
```
import numpy as np
import pandas as pd
import csv
import matplotlib.pylab as pl
import ot.plot

n = 392 # nb samples

data1 = pd.read_csv
('/Users/liza/PycharmProjects/pythonProject1/Motor_Vehicle_Collisions_-
_Crashes.csv', encoding="ISO-8859-1", nrows=n)
data2 = pd.read_csv
('/Users/liza/PycharmProjects/pythonProject1/new_york_hotels.csv',
encoding="ISO-8859-1", nrows=n)

a1 =pd.DataFrame (data1, columns=["LONGITUDE", "LATITUDE"])
a2 =pd.DataFrame (data2, columns=["longitude", "latitude"])

xs = np.array(a1)

xt = np.array(a2)

print(a1)
print(a2)

a, b = np.ones((n,)) / n, np.ones((n,)) / n  # uniform distribution on samples

# loss matrix
M = ot.dist(xs, xt)
M /= M.max()

pl.figure(1)
pl.plot(xs[:, 0], xs[:, 1], '+b', label='Source samples')
pl.plot(xt[:, 0], xt[:, 1], 'xr', label='Target samples')
pl.legend(loc=0)
pl.title('Source and target distributions')

pl.figure(2)
pl.imshow(M, interpolation='nearest')
pl.title('Cost matrix M')

G0 = ot.emd(a, b, M)

pl.figure(3)
pl.imshow(G0, interpolation='nearest')
```

```
pl.title('OT matrix G0')

pl.figure(4)
ot.plot.plot2D_samples_mat(xs, xt, G0, c=[.5, .5, 1])
pl.plot(xs[:, 0], xs[:, 1], '+b', label='Source samples')
pl.plot(xt[:, 0], xt[:, 1], 'xr', label='Target samples')
pl.legend(loc=0)
pl.title('OT matrix with samples')

table=[]table2=[]i = 0
j = 0
while j < n:
   while i < n:
      if G0[i,j]>0:
         table.append([i,j])
      i = i + 1
   table2.append(table)
   i = 0
   j = j + 1

print(type(table2))

pl.show()

with open('/Users/liza/PycharmProjects/pythonProject1/transition_matrix.csv', "w",
newline="") as f:
   writer = csv.writer(f)
   writer.writerows(table)
```

У процесі виконання програми отримуються точки таксі і готелей:

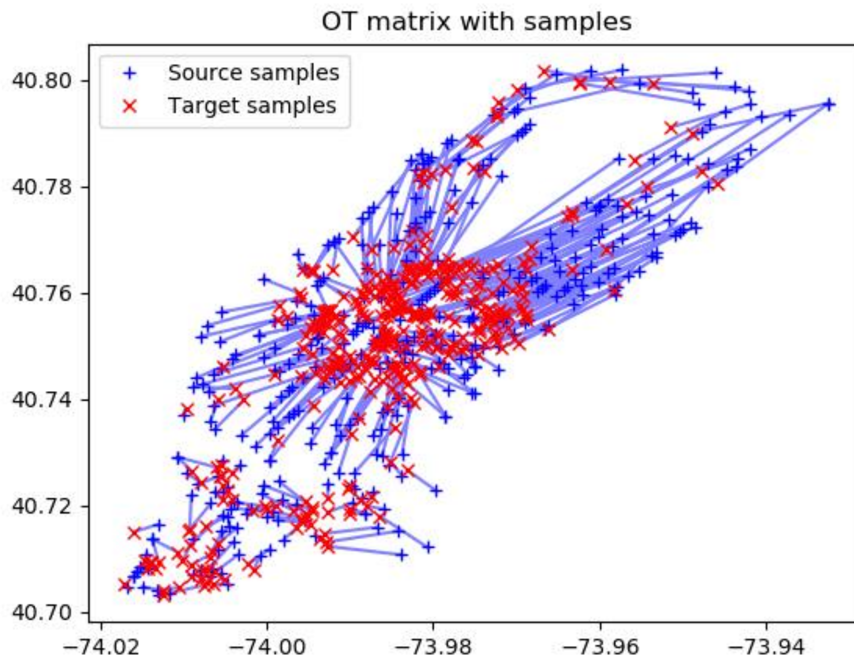


Далі обчислюються всі відстані між групами Source і Target і будується матриця відстаней (витрат на переміщення):



Чим світліша точка, тим менша відстань між точками.

Після цього виходячи з матриці відстаней будується матриця оптимального транспорту. І далі на карті будується те, що по суті вимагається в умові – оптимальний набір маршрутів таксі до готелів:



І, врешті-решт, код записує ці співвідношення у файл з розширенням .csv.

Висновки

Транспортна задача є особливим випадком загальної задачі лінійного програмування, яка має специфічну структуру і може бути ефективно розв'язана за допомогою методів, заснованих на теорії двоїстості. Серед основних методів для розв'язання транспортної задачі є метод північно-західного кута, метод найменшої вартості, метод Фогеля, угорський метод та метод потенціалів. Ці методи знаходять широке застосування не тільки у задачах транспортування, але й у задачах управління запасами та виробничого планування.

Транспортні моделі входять до підкласу загальних мережевих моделей і використовуються для моделювання різних проблем. А також є важливим засобом вирішення багатьох економічних проблем, що виникають перед підприємствами. Зокрема, її можна використовуватиме для опису ситуацій, щодо управління запасами, управління рухом капіталів, призначення персоналу.

Література

[1] Ю.Д. Попов В.І. Тюптя В.І. Шевченко. Методи оптимізації. Навчальний електронний посібник для студентів. Київ, 2003

[2] Хемді А. Таха. Вступ до дослідження операцій. Університет Арканзаса, 2001

[3] Васил'єв О. Програмування мовою Python. 2019

[4] Д. М. Козаченко. Основи дослідження операцій у транспортних системах. Дніпропетровськ, 2015

[5] І. Н. Ляшенко. Лінійне і нелінійне програмування. 1978

[6] LINEAR PROGRAMMING OUTPUT SUMMARY [Електронний ресурс]. *um.co.ua*. Режим доступу до ресурсу: http://um.co.ua/4/4-9/4-99545.html.

[7] Транспортні задачі. [Електронний ресурс]. *StudFiles*. Режим доступу до ресурсу: https://studfile.net/preview/9338109/page:3/.