Розрахунок верхньої межі вихідної валентності з в транспортній задачі з додатковими обмеженнями

Студент: Панасюк Роман Олегович

Керівник: Швай Надія Олександрівна

Мотивація та актуальність

Тему було обрано з огляду на її актуальність у моїй роботі:

- Я працюю в компанії, що займається розробкою додатку для знайомств, що з математичного погляду є мережею, яку можна інтерпретувати як двочастковий граф, який показує доступних для взаємного пошуку користувачів.
- Хоча конкретні деталі внутрішньої логіки продукту доведеться залишити конфіденційними, загальна ідея проблеми зводиться до «урівняння конкуренції» між користувачами на платформі.
- Водночас подібні підходи до балансування навантаження та обмеження ресурсів мають застосування і в інших галузях.

Мотивація та актуальність

Інші галузі в яких актуальне застосування задачі, що розглядається:

- Розподіл товарів між складами й точками продажу з обмеженою пропускною здатністю та маршрутами.
- Балансування навантаження в енергетичних або телекомунікаційних мережах
- Розподіл обчислювальної потужності в хмарних сховищах
- Планування маршрутів та графіків із урахуванням пропускної здатності ліній та пасажирського попиту у громадському транспорті

Враховуючи, що розглянутий мною набір обмежень ще не був досліджений, задача має цінність, а розроблений алгоритм має потенціал для застосування в реальному житті

Постановка задачі

Вхідні дані:

$A_1, \ldots, A_N$ — виробники (безлімітний запас);

$B_1, \ldots, B_M$ — споживачі з попитом b_j;

K — максимальна кількість товару від одного виробника конкретному споживачу;

$R = [r_{ij}]$ - матриця допустимості, $r_{ij} = \begin{cases} 1, & \text{існує звязок} \\ 0, & \text{зв'язку немає} \end{cases}$

Постачання:

$x_{ij} \geq 0$ — кількість товару від A_i до B_j.

Обмеження:

$$x_{ij} \leq K\, r_{ij}, \quad \forall i, j;$$

$$\sum_{j=1}^{M} x_{ij} \leq L, \quad \forall i;$$

$$\sum_{i=1}^{N} x_{ij} = b_j, \quad \forall j.$$

Цільова функція:

$\min\ L$ — знайти найменший L, за якого усі виробники задоволені, та виконуються обмеження

Реалізований алгоритм

Інтерпретуємо нашу задачу як задачу максимальної пропускної здатності у спрямованому графі $G = (V, E)$, де:

- s — витік,
- t — стік.

Ємкості ребер задані так:

$$s \to A_i : \ L, A_i \to B_j : \ K\, r_{ij}, B_j \to t : \ b_j.$$

1. **Побудова двочасткового графа** Вершини: s, A_i, B_j, t. Ємкості ребер наведені вище.
2. **Визначення початкових меж для** L

$$L_{\min} = \left\lceil \frac{\sum_{j=1}^{M} b_j}{N} \right\rceil, \quad L_{\max} = \sum_{j=1}^{M} b_j.$$

3. **Перевірка розв'язності та виділення підграфа**
   (а) Здійснити алгоритм Edmonds–Karp для обчислення max-flow f^*.
   (б) Якщо $f^* < \sum_j b_j$, виконати пошук у ширину (Breadth-First Search) у залишковій мережі від s і виділити підграф із досяжними споживачами.

4. **Бінарний пошук мінімального L**

   (a) Обчислити $L_{\text{mid}} = \lfloor (L_{\min} + L_{\max})/2 \rfloor$.

   (б) Оновити ємності ребер $s \to A_i$ до L_{mid}.

   (в) Знову виконати алгоритм Edmonds–Karp на поточному підграфі і отримати нове f^*.

   (г) Якщо $f^* = \sum_j b_j$, призначити $L_{\max} := L_{\text{mid}}$, інакше — $L_{\min} := L_{\text{mid}} + 1$.

   (д) Повторювати кроки (a)–(d) до збіжності $L_{\min} = L_{\max}$.

5. **Оптимізація оновлення залишкової мережі**

   (a) Зберігати структуру попередньої залишкової мережі.

   (б) Інкрементально оновлювати лише ємності ребер $s \to A_i$ при зміні L.

Реалізований алгоритм

Реалізований алгоритм

У результаті знаходимо найменше L = L_min = L_max, що дозволяє задовольнити весь сумарний попит.

Реалізований алгоритм

Крім цього було реалізовано аналогічні алгоритми, тільки з іншими допоміжними алгоритмом, що перевіряють наявність розв'язку при фіксованому L:

- Preflow Push;
- Shortest Augmenting Path (SAP);

А також 2 версії, де ми перевіряємо наявність розв'язку задачі як задачі змішаного лінійного програмування, використано такі допоміжні алгоритми:

- Branch and cut
- Solving Constraint Integer Programs Algorithm

Результати

Приклад 1

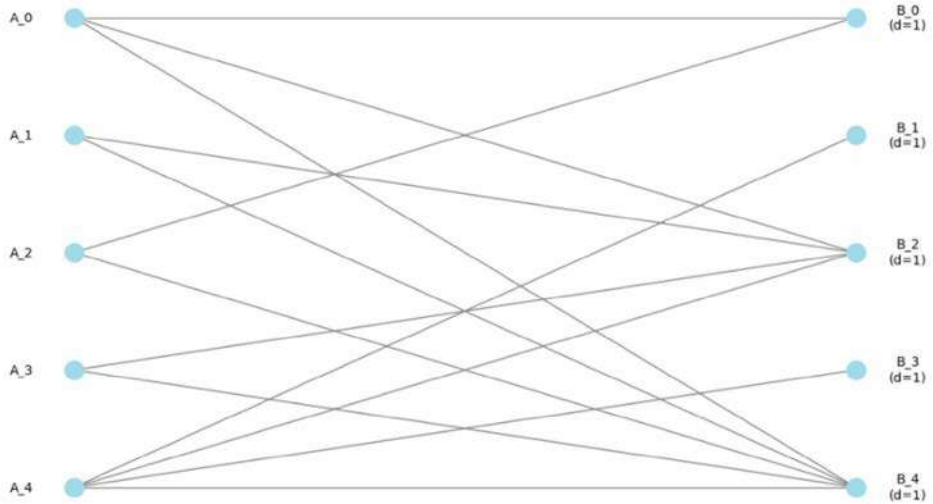
📌 Матриця з'єднаності (R):

	B_0	B_1	B_2	B_3	B_4
A_0:	1	0	1	0	1
A_1:	0	0	1	0	1
A_2:	1	0	0	0	1
A_3:	0	0	1	0	1
A_4:	0	1	1	1	1

📌 Вектор попитів (demand):

B_0: 1
B_1: 1
B_2: 1
B_3: 1
B_4: 1

K=1



========== РЕЗУЛЬТАТ ==========

Розв'язність повної задачі: ✅

Мінімальне L: 2

Матриця постачань:

	B_0	B_1	B_2	B_3	B_4
A_0	1	0	1	0	0
A_1	0	0	0	0	1
A_2	0	0	0	0	0
A_3	0	0	0	0	0
A_4	0	1	0	1	0

⏱ Час виконання алгоритму: 0.0180 секунд

Приклад 2

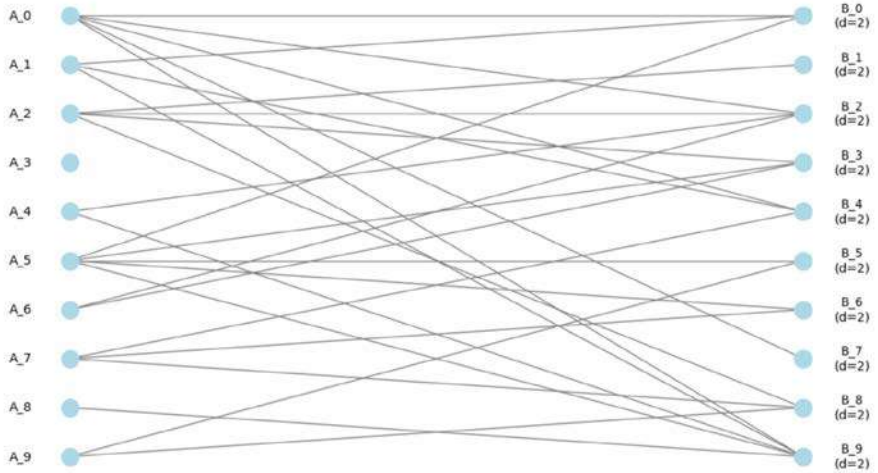
📌 Матриця з'єднаності (R):

	B_0	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9
A_0:	1	0	1	0	1	0	0	1	0	1
A_1:	1	0	0	0	1	0	0	0	0	1
A_2:	0	1	1	1	0	0	0	0	1	0
A_3:	0	0	0	0	0	0	0	0	0	0
A_4:	0	0	1	0	0	0	0	0	0	1
A_5:	1	0	0	1	0	1	1	0	0	1
A_6:	0	0	1	1	0	0	0	0	0	0
A_7:	0	0	0	0	1	0	1	0	1	0
A_8:	0	0	0	0	0	0	0	0	0	1
A_9:	0	0	0	0	0	1	0	0	1	0

📌 Вектор попитів (demand):

B_0: 2
B_1: 2
B_2: 2
B_3: 2
B_4: 2
B_5: 2
B_6: 2
B_7: 2
B_8: 2
B_9: 2

K=1



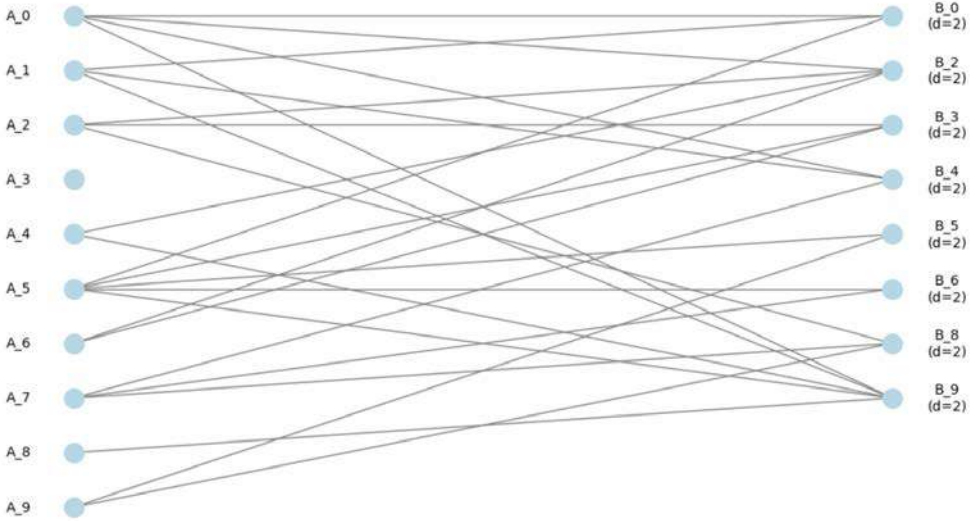
Приклад 2

📌 Матриця з'єднаності (R):

	B_0	B_2	B_3	B_4	B_5	B_6	B_8	B_9
A_0:	1	1	0	1	0	0	0	1
A_1:	1	0	0	1	0	0	0	1
A_2:	0	1	1	0	0	0	1	0
A_3:	0	0	0	0	0	0	0	0
A_4:	0	1	0	0	0	0	0	1
A_5:	1	0	1	0	1	1	0	1
A_6:	0	1	1	0	0	0	0	0
A_7:	0	0	0	1	0	1	1	0
A_8:	0	0	0	0	0	0	0	1
A_9:	0	0	0	0	1	0	1	0

📌 Вектор попитів (demand):

B_0: 2
B_2: 2
B_3: 2
B_4: 2
B_5: 2
B_6: 2
B_8: 2
B_9: 2



оскільки повний при повному графі нема розв'язку, знайшли підграф на якому розв'язок є

Приклад 2

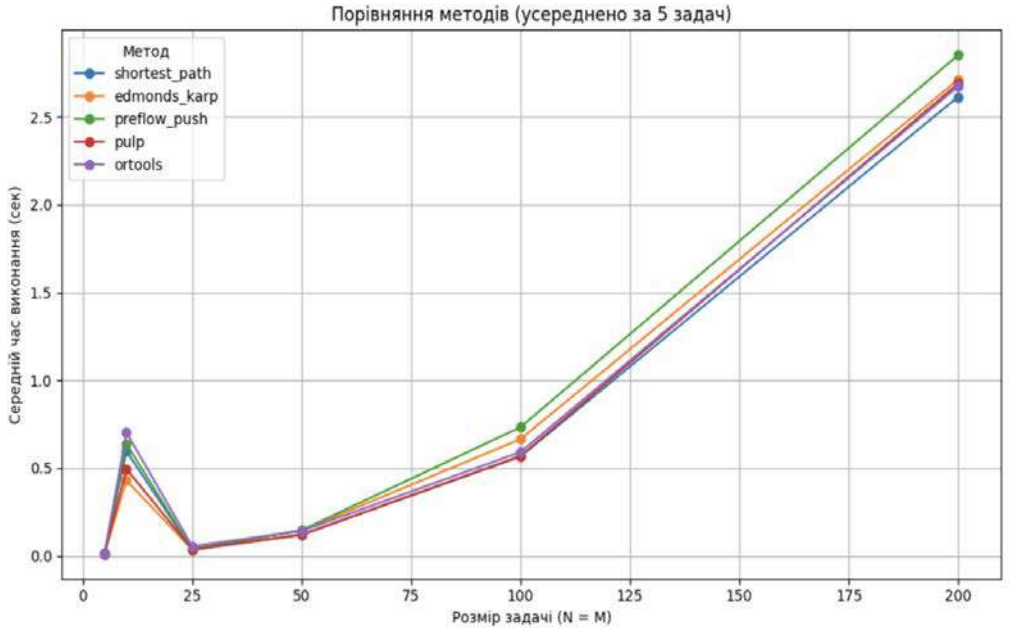
```
==========  РЕЗУЛЬТАТ  ==========

Розв'язність повної задачі: ❌
Знайдено підграф на якому можемо розв'язати: ✅

Мінімальне L: 2

Матриця постачань:
     B_0  B_2  B_3  B_4  B_5  B_6  B_8  B_9
A_0    1    1    0    0    0    0    0    0
A_1    1    0    0    1    0    0    0    0
A_2    0    0    1    0    0    0    1    0
A_3    0    0    0    0    0    0    0    0
A_4    0    1    0    0    0    0    0    1
A_5    0    0    0    0    1    1    0    0
A_6    0    0    1    0    0    0    0    0
A_7    0    0    0    1    0    1    0    0
A_8    0    0    0    0    0    0    0    1
A_9    0    0    0    0    1    0    1    0

Задоволений підграф споживачів: [0, 2, 3, 4, 5, 6, 8, 9]
Невдоволені споживачі: [1, 7]

⏱ Час виконання алгоритму: 0.8536 секунд
```

Приклад 3

Порівняння допоміжних алгоритмів перевірки існування матриці перевезень, задовольняючи усіх споживачів

📊 Усереднені результати:

	Розмір (N=M)	Метод	Середній час (сек)	Середнє L_min	Розв'язано повністю (%)
0	5	shortest_path	0.0158	1.5	40%
1	5	edmonds_karp	0.0113	1.5	40%
2	5	preflow_push	0.0167	1.5	40%
3	5	pulp	0.0129	1.5	40%
4	5	ortools	0.0131	1.5	40%
5	10	shortest_path	0.5964	2.4	40%
6	10	edmonds_karp	0.4292	2.4	40%
7	10	preflow_push	0.6373	2.4	40%
8	10	pulp	0.4949	2.4	40%
9	10	ortools	0.7021	2.4	40%
10	25	shortest_path	0.0344	2.2	100%
11	25	edmonds_karp	0.0365	2.2	100%
12	25	preflow_push	0.0385	2.2	100%
13	25	pulp	0.0373	2.2	100%
14	25	ortools	0.0554	2.2	100%
15	50	shortest_path	0.1199	2.2	100%
16	50	edmonds_karp	0.1443	2.2	100%
17	50	preflow_push	0.1456	2.2	100%
18	50	pulp	0.1200	2.2	100%
19	50	ortools	0.1415	2.2	100%



Висновки

- Вдалося розробити метод для мінімізації L за наявних обмежень у вигляді максимуму перевезення від конкретного виробника конкретному споживачу
- На проведених експериментах не вдалося побачити дуже суттєвої відмінності часу виконання алгоритму від методу, що використовувався для перевірки існування плану постачань при наявних обмеженнях, але можу зазначити, що для різних розмірів матриці
- Дуже сильно впливає на час виконання висока розрідженість матриці допустимості, а також випадки, коли для повного графу не існує розв'язку, через що доводиться шукати підграф для якого розв'язок існує
- Враховуючи наведені вище пункти бачу сенс продовжувати дослідження в сторону попередньої класифікації задачі більш швидким та ефективним алгоритмом, щоб наперед розуміти, чи вдасться розв'язати задачу, щоб мінімізувати кількість холостих застосувань, адже саме вони збільшують час

Висновки

- Для подальших дослідження вважаю за доцільне:
  - Провести експерименти на дуже великих матрицях
  - Покращити алгоритм для випадків розрідженої матриці допустимості
  - Застосувати нейронні мережі для одного з етапів задачі
  - Поєднати перевірку допустимості розв’язку, та сам алгоритм, щоб одразу роз’язок будувався лише для тієї частини графа, де це можливо

Дякую за увагу!