

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

МЕТОДИЧНІ ВКАЗІВКИ
до виконання самостійних робіт з дисципліни
"Методи оптимізації комп'ютерних мереж"
для студентів спеціальності 123 Комп'ютерна інженерія,
усіх форм навчання

Методичні вказівки до виконання самостійних робіт з дисципліни "Методи оптимізації комп'ютерних мереж" для студентів спеціальності 123 Комп'ютерна інженерія, усіх форм навчання / Укл. Г.Г Киричек. – Запоріжжя: Національний університет «Запорізька політехніка», 2025. – 30 с.

Укладачі:

Г.Г. Киричек, доцент, к.т.н.

Рецензент:

М.Ю. Тягунова, доцент, к.т.н.

Відповідальний за випуск:

Г.Г. Киричек, доцент, к.т.н.

Затверджено
на засіданні кафедри КСМ
Протокол № 8 від 24.03.2025

Рекомендовано до видання
НМК КНТ
Протокол № 7 від 24.03.2025

ЗМІСТ

1 Оптимізація за допомогою математичних методів	4
1.1 Загальні відомості.....	4
1.2 Оптимізація комп'ютерної мережі за допомогою методу лінійного програмування	5
1.3 Оптимізація комп'ютерної мережі за допомогою методу цілочисленого програмування.....	7
1.4 Оптимізація комп'ютерної мережі за допомогою методу теорії черг	9
1.5 Оптимізація комп'ютерної мережі за допомогою методу теорії графів.....	10
1.6 Оптимізація комп'ютерної мережі методами евристичних та метаевристичних алгоритмів.....	12
1.7 Оптимізація комп'ютерної мережі методом теорії ігор	14
1.8 Оптимізація комп'ютерної мережі за допомогою машинного навчання	16
1.9 Оптимізація комп'ютерної мережі методом стохастичного моделювання.....	19
1.10 Багатокритеріальна оптимізація при оптимізації комп'ютерної мережі.....	22
1.11 Оптимізація комп'ютерної мережі методом динамічного програмування	25
1.12 Завдання	29
Рекомендована література	30

1 ОПТИМІЗАЦІЯ ЗА ДОПОМОГОЮ МАТЕМАТИЧНИХ МЕТОДІВ

Мета роботи - використовуючи різні математичні методи, навчитися, на прикладі простого завдання, здійснювати оптимізацію мереж передачі даних, застосовуючи інструментальні та прогамні засоби при вирішенні подібних завдань в реальних розподілених мережах різного призначення.

1.1 Загальні відомості

Оптимізація комп'ютерних мереж за допомогою математичних методів дозволяє покращити продуктивність, надійність і ефективність мережі, мінімізуючи витрати. На лекції ми коротко розглянули основні математичні методи, які застосовуються для цього:

- лінійне програмування (LP) - математичний метод, який у контексті комп'ютерних мереж дозволяє оптимізувати використання ресурсів (пропускна здатність, затримки, енергоспоживання тощо);
- цілочисельне програмування (IP) - коли змінні повинні бути цілими числами (кількість серверів, каналів зв'язку). Прикладом є оптимізація розташування точок доступу Wi-Fi у приміщенні;
- теорія черг (Queueing Theory) - аналіз і моделювання параметрів продуктивності мереж (пропускна здатність, втрата пакетів, середній час очікування), як систем обслуговування (M/M/1, M/G/1). Приклад - оцінка впливу додаткового сервера на час відповіді мережі;
- теорія графів (Graph Theory) - проектування топології мережі та оптимізація маршрутів (вузли - це пристрої, а ребра - лінії зв'язку). Використання алгоритмів Дейкстри або Форда-Фалкерсона;
- евристичні та метаевристичні алгоритми - коли точна оптимізація занадто витратна для великих мереж. Використання генетичних алгоритмів, пошук наближених рішень. Прикладом є оптимізація розподілу навантаження між серверами;
- теорія ігор (Game Theory) - оптимізація поведінки мережі за участі багатьох агентів (користувачів або пристроїв). Моделювання взаємодії як гри зі стратегіями і виграшами. Прикладом є управління первантаженнями із справедливим розділенням пропускної здатності;

- оптимізація за допомогою машинного навчання - прогнозування трафіку або динамічне налаштування мереж. Приклад - адаптивна маршрутизація залежно від прогнозованих перевантажень;

- стохастичне моделювання (Stochastic Modeling) - робота з невизначеністю в трафіку або відмовах обладнання. Представлення змінних у вигляді ймовірнісних розподілів (розподіл Пуассона при отриманні пакетів). Прикладом є проектування відмовостійких систем із використанням надлишковості;

- динамічне програмування (Dynamic Programming, DP) - послідовне прийняття рішень для складних завдань. Розбиття проблеми на підзадачі з подальшим їх розв'язанням. Прикладом є оптимізація багатокрокової маршрутизації в бездротових мережах;

- багатокритеріальна оптимізація (Multi-Objective Optimization) - вирішення конфліктів (продуктивність проти витрат). Пошук Парето-оптимального набору рішень. Прикладом є мінімізація енергоспоживання при максимізації пропускної здатності.

Для реалізації цих методів на практиці можуть застосовуватись або інструменти: MATLAB, Python (бібліотеки SciPy, NetworkX, PuLP), оптимізаційні пакети (Gurobi, CPLEX), або фреймворки: використання SDN (програмовані мережі) для гнучкої оптимізації.

Для прикладу розглянемо всі ці методи при вирішенні простого завдання оптимізації комп'ютерної мережі.

1.2 Оптимізація комп'ютерної мережі за допомогою методу лінійного програмування

Розглянемо розв'язання задачі оптимізації комп'ютерної мережі за допомогою методу лінійного програмування. Для цього сформулюємо задачу, а потім наведемо її рішення.

Задача. Припустимо, що у нас є мережа, яка складається з декількох каналів зв'язку, кожен із яких має певну пропускну здатність та вартість (метрику) шляху. Потрібно розподілити трафік між каналами таким чином, щоб мінімізувати загальні витрати, забезпечуючи необхідний рівень пропускної здатності.

Використаємо наступні позначення:

- x_i - кількість трафіку, що проходить через i -й канал;
- c_i - вартість шляху для трафіку через i -й канал;

- b_i - максимальна пропускна здатність i -го каналу;
- T - загальна потреба в пропускній здатності;
- n - кількість доступних каналів.

Побудуємо математичну модель з умовою мінімізації загальних витрат: $\min Z = \sum_{i=1}^n c_i x_i$, при обмеженнях:

- загальна пропускна здатність має відповідати вимогам: $\sum_{i=1}^n x_i \geq T$;
- кожен канал не повинен перевищувати свою максимальну пропускну здатність: $0 \leq x_i \leq b_i, \forall(i) = 1, 2, \dots, n$.

Наведемо розв'язання задачі з урахуванням обмежень. Розглянемо конкретний приклад при використанні трьох каналів ($n = 3$):

- $c = (5, 3, 4)$ - вартість трафіку через кожен канал;
- $b = (50, 60, 40)$ - максимальна пропускна здатність каналів;
- $T = 80$ - необхідний загальний трафік.

І так, наведемо рішення методом лінійного програмування. Маємо оптимальне рішення, коли:

- $x_1 = 0$ (не використовуємо перший канал);
- $x_2 = 60$ (повністю використовуємо другий канал);
- $x_3 = 20$ (частково використовуємо третій канал).

Оптимальне рішення - $\sum_{min} = 5(0) + 3(60) + 4(20) = 260$:

- намагаємося використовувати найдешевші канали (В, С);
- канал А не використовується взагалі, оскільки його вартість 5 – найбільша;
- більша частина трафіку (60) йде через В, бо він має меншу вартість (3);
- залишок (20) передається через С, бо у В є обмеження (60).

Отримаємо мінімальні загальні витрати: 260 одиниць.

Це означає, що для мінімізації витрат варто використовувати другий і третій канали, не використовуючи перший. Ось таку схему, що ілюструє оптимальний розподіл трафіку між каналами А, В і С, пропонує чат GPT (рис. 1.1).

Канал А не використовується, канал В передає 60 одиниць трафіку, а канал С - 20 одиниць.

1.3 Оптимізація комп'ютерної мережі за допомогою методу цілочисленого програмування

Потрібно розподілити трафік таким чином, щоб мінімізувати витрати й одночасно забезпечити необхідний рівень пропускну здатності. Наведемо вихідні дані:

- x_1 - обсяг трафіку, який проходить через кожен із трьох каналів;
- вартість трафіку: $c_1 = 5$, $c_2 = 3$, $c_3 = 4$;
- максимальна пропускна здатність: $b_1 = 50$, $b_2 = 60$, $b_3 = 40$;
- необхідна пропускна здатність: $T = 80$.

Використаємо математичну модель, як цільову функцію (мінімізації витрат): $\min Z = 5x_1 + 3x_2 + 4x_3$, з наступними обмеженнями:

– загальна пропускна здатність має бути не менше ніж 80: $x_1 + x_2 + x_3 \geq 80$;

– кожен канал не повинен перевищувати свою максимальну пропускну здатність: $0 \leq x_1 \leq 50$, $0 \leq x_2 \leq 60$, $0 \leq x_3 \leq 40$, $\forall(i) = 1, 2, \dots, n$;

– x_1, x_2, x_3 - цілі числа.

Знов таки, маємо оптимальне рішення, коли:

– $x_1 = 0$ (перший канал не використовується);

– $x_2 = 60$ (повністю використовуємо другий канал);

– $x_3 = 20$ (третій канал використовується частково).

Мінімальні загальні витрати при цьому: 260 одиниць.

Далі наведено програмний код для розв'язання задачі за допомогою Python:

```
from pulp import LpMaximize, LpMinimize, LpProblem,
LpVariable, lpSum
# Створюємо модель мінімізації витрат
model = LpProblem(name="network-optimization",
sense=LpMinimize)
# Змінні (цілочислені)
x1 = LpVariable(name="x1", lowBound=0, upBound=50,
cat='Integer') # Канал А
x2 = LpVariable(name="x2", lowBound=0, upBound=60,
cat='Integer') # Канал В
x3 = LpVariable(name="x3", lowBound=0, upBound=40,
cat='Integer') # Канал С
# Цільова функція (мінімізуємо витрати)
model += 5 * x1 + 3 * x2 + 4 * x3, "Total Cost"
# Обмеження: загальний трафік має бути 80 одиниць
model += (x1 + x2 + x3 == 80, "Traffic Constraint")
# Розв'язуємо задачу
model.solve()
# Вивід результатів
print("Оптимальний розподіл трафіку:")
print(f"Канал А: {x1.varValue} одиниць")
print(f"Канал В: {x2.varValue} одиниць")
print(f"Канал С: {x3.varValue} одиниць")
```

```
print(f"Мінімальні витрати:  
{model.objective.value()}")
```

Цей код вирішує задачу методом цілочисленого лінійного програмування (LP) для оптимального розподілу трафіку між каналами. Він гарантує, що змінні залишаються цілими числами, що важливо в реальному середовищі.

Маємо висновки: найменш вигідним є перший канал (x_1) не використовується, бо його вартість є найвищою (5); основне навантаження бере на себе другий канал (x_2), бо має найнижчу вартість (3); третій канал (x_3) доповнює розподіл, оскільки другий не може повністю покрити потреби у пропускній здатності.

Це рішення забезпечує мінімальні витрати при дотриманні всіх обмежень.

1.4 Оптимізація комп'ютерної мережі за допомогою методу теорії черг

Методи теорії черг дозволяють оптимізувати завантаження мережевих каналів, мінімізувати затримки та збільшити пропускну здатність. Знов сформулюємо задачу та наведемо її вирішення, але вже згідно методу, який будемо використовувати.

Розглянемо систему, що складається з трьох каналів передачі даних, через які проходить вхідний трафік. Кожен канал можна розглядати як одноканальну систему масового обслуговування (СМО) із експоненціальним розподілом часу обслуговування.

Нехай:

- вхідний потік заявок (пакетів даних) надходить, як потік Пуассона з інтенсивністю λ ;
- час обслуговування пакетів є експоненційно розподіленим і кожен канал має свою інтенсивність обслуговування μ_i ;
- кожен канал має обмежену пропускну здатність b_i , яка задає максимальну кількість оброблених пакетів за секунду;
- потрібно мінімізувати середній час очікування в черзі та розподілити потік між каналами.

Вихідні дані:

- загальна інтенсивність потоку заявок: $\lambda = 80$ (пакетів/сек);

- пропускні здатності каналів: для каналу 1 маємо $b_1 = 50$, для каналу 2 маємо $b_2 = 60$, для каналу 3 маємо $b_3 = 40$;

- інтенсивність обслуговування (максимальна продуктивність кожного каналу): $\mu_1 = 50$, $\mu_2 = 60$, $\mu_3 = 40$;

- загальною метою є оптимальне розподілення потоку λ між каналами, щоб мінімізувати середній час очікування W .

Маємо, що кожен канал працює за системою М/М/1, де:

- W_i - середній час очікування в черзі для i -го каналу;

- при цьому формула середнього часу очікування для М/М/1:

$W_i = \frac{1}{\mu_i - \lambda_i}$, де λ_i є частиною потоку, що передається на i -й канал.

Метою є розподіл потоку λ між каналами таким чином, щоб:

- мінімізувати середній час очікування $W_{\text{середнє}} = \sum_{i=1}^3 p_i W_i$, де $p_i = \lambda_i$ - частина потоку, яка йде через i -й канал;

- дотримуючись обмежень: $\lambda_1 + \lambda_2 + \lambda_3 = 80$, $0 \leq \lambda_i \leq b_i$.

Знайдемо оптимальний розподіл потоку, що мінімізує середній час очікування. Отримали такий розподіл трафіку: $\lambda_1 = 29.13$ (канал 1); $\lambda_2 = 32.07$ (канал 2); $\lambda_3 = 18.79$ (канал 3). Мінімальний середній час очікування в черзі: 0.043 секунд. Зробимо висновки, що:

- найбільш навантажений другий канал (32.07 пакетів/сек), бо він має найбільшу пропускну здатність;

- найменш навантажений третій канал (18.79 пакетів/сек), бо має найменшу продуктивність;

- розподіл мінімізує середній час очікування, що важливо для високопродуктивних комп'ютерних мереж.

Це рішення дозволяє ефективно використовувати всі ресурси, уникнути перевантажень і зменшити затримки.

1.5 Оптимізація комп'ютерної мережі за допомогою методу теорії графів

Метод теорії графів дозволяє оптимізувати трафік у комп'ютерних мережах, визначити найкоротші шляхи для передачі даних, мінімізувати затримки та збалансувати навантаження між каналами. Розглянемо комп'ютерну мережу як орієнтований граф, де:

- вузли (вершини) є точками маршрутизації трафіку (сервери, комутатори, маршрутизатори);

– ребра (дуги) відповідають каналам зв'язку з певною пропускнуною здатністю та вартістю передачі.

Нам потрібно знайти оптимальний шлях для трафіку з мінімальними витратами та достатньою пропускнуною здатністю. Спочатку наведемо графову модель мережі:

- вузол S - джерело трафіку;
- вузол T - пункт призначення;
- маємо три можливі шляхи через три різні канали: канал 1 ($S \rightarrow A \rightarrow T$); канал 2 ($S \rightarrow B \rightarrow T$); канал 3 ($S \rightarrow C \rightarrow T$).

Задаємо характеристики каналів (табл. 1.1) та наведемо граф, що моделює мережу.

Таблиця 1.1 - Характеристики каналів

Канал	Пропускна здатність b_i	Вартість (вага ребра)
$S \rightarrow A \rightarrow T$	50	5
$S \rightarrow B \rightarrow T$	60	3
$S \rightarrow C \rightarrow T$	40	4

На рисунку 1.2 представлений граф мережі, у якому:

- вершини S, A, B, C, T - це сервери/маршрутизатори;
- ребра мають вагу, яка відповідає вартості передачі через канал.

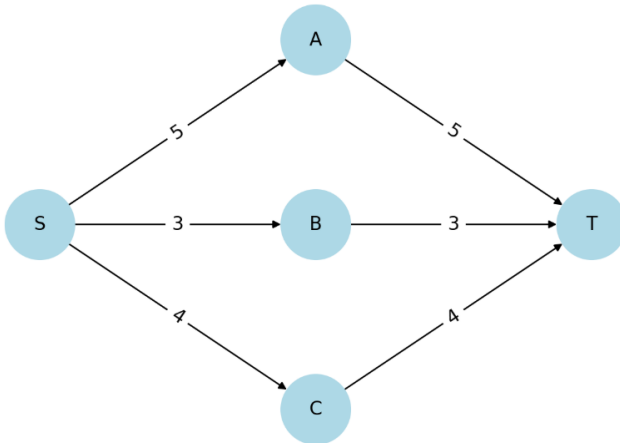


Рисунок 1.2 – Граф мережі

Потрібно знайти шлях із мінімальною загальною вартістю при умові, що сумарна пропускна здатність покриває потребу в 80 одиниць трафіку. Математично це означає, що ми шукаємо мінімальний потік із заданими витратами (задача мінімального потоку з обмеженнями). Тому рішення отримаємо методом мінімального потоку в мережі.

Використаємо алгоритм найкоротшого шляху з урахуванням пропускної здатності.

Розрахунок показав, що для мінімізації витрат потрібно мати наступний розподіл потоку:

- канал $S \rightarrow B \rightarrow T$: 60 одиниць трафіку (оскільки має найменшу вартість - 3).
- канал $S \rightarrow C \rightarrow T$: 20 одиниць трафіку;
- канал $S \rightarrow A \rightarrow T$: не використовується (канал має найвищу вартість - 5).

Можемо зробити висновки, що:

- весь потік спрямований через найбільш економічні канали;
- канал з найвищими витратами не використовується, що допомагає знизити загальні витрати на передачу даних;
- мережа оптимально балансується, запобігаючи перевантаженню одного каналу.

Використання теорії графів дозволяє ефективно оптимізувати мережу, зменшити затримки і мінімізувати витрати на передачу даних.

1.6 Оптимізація комп'ютерної мережі методами евристичних та метаевристичних алгоритмів

Евристичні та метаевристичні алгоритми дозволяють швидко знаходити наближені оптимальні рішення для складних оптимізаційних задач, де класичні методи можуть бути неефективними.

Для розподілу трафіку в комп'ютерній мережі ми використаємо:

- жадібний алгоритм (евристика) - для швидкого знаходження хорошого початкового рішення;
- генетичний алгоритм (метаевристика) - для знаходження глобально оптимального рішення.

Для розрахунку можемо скористатися даними таблиці 1.1. Маємо три канали передачі даних із різною пропускною здатністю та вартістю

передачі. Мета: мінімізувати загальні витрати на передачу 80 одиниць трафіку, розподіляючи його між каналами.

Жадібний алгоритм працює за принципом: віддавай перевагу найдешевшому каналу, поки не буде розподілено весь трафік: відсортуємо канали за зростанням вартості передачі; заповнюємо канали за пріоритетами до досягнення 80 одиниць трафіку.

Отримали оптимальний розподіл за жадібним алгоритмом. Жадібний алгоритм розподілив трафік таким чином:

- канал $S \rightarrow B \rightarrow T$ - 60 одиниць (найдешевший);
- канал $S \rightarrow C \rightarrow T$ - 20 одиниць;
- канал $S \rightarrow A \rightarrow T$ - не використовується через високу вартість.

Жадібний алгоритм працює швидко, але не гарантує глобального оптимуму.

Генетичний алгоритм імітує процес природного відбору, комбінуючи хороші рішення для покращення результату.

- генерація початкових рішень (хромосом), де кожне рішення - розподіл трафіку між каналами;
- оцінка рішень за функцією вартості;
- схрещування (обмін частинами рішень);
- мутація (випадкові зміни для пошуку нових рішень);
- відбір найкращих варіантів для наступних рішень.

При проведенні розрахунків скористаємось функцією пристосованості (Fitness function): $\text{Fitness} = \sum_i \lambda_i \times \text{cost}_i$, де λ_i - трафік на каналі, а cost_i - його вартість. Метою є мінімізація вартості при передачі всього трафіку.

Генетичний алгоритм знайшов такий розподіл трафіку:

- канал $S \rightarrow A \rightarrow T$ - 2 одиниці;
- канал $S \rightarrow B \rightarrow T$ - 54 одиниці (найдешевший);
- канал $S \rightarrow C \rightarrow T$ - 24 одиниці.
- загальна вартість: 268 одиниць (оптимальне значення).

Можемо зробити висновки, що:

- жадібний алгоритм дає швидке рішення (вартість 280), але воно не ідеальне;
- генетичний алгоритм покращив результат на 12 одиниць;
- генетичний підхід є гнучким та надає більш точний результат для складних, розподілених мереж.

На рисунку 1.3 наведено порівняння роботи алгоритмів.

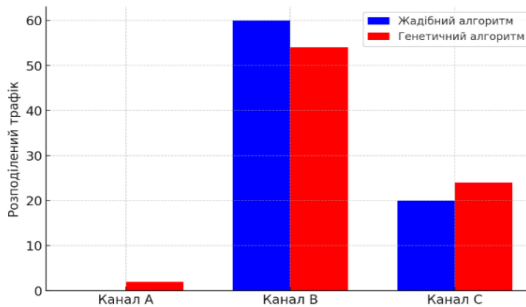


Рисунок 1.3 – Порівняння алгоритмів оптимізації

На графіку видно, що генетичний алгоритм більш рівномірно розподіляє трафік, використовуючи всі три канали, що дозволяє мінімізувати загальні витрати у порівнянні із жадібним методом: жадібний алгоритм швидкий, але не завжди знаходить оптимальне рішення; генетичний алгоритм надає кращі результати за рахунок комбінаторного пошуку. Це підтверджує ефективність метаевристичних алгоритмів для вирішення складних мережових задач.

1.7 Оптимізація комп'ютерної мережі методом теорії ігор

Метод теорії ігор допомагає знаходити оптимальну стратегію при взаємодії декількох учасників, які мають конфлікт інтересів. У контексті комп'ютерних мереж кожен маршрут (канал передачі) можна розглядати як гравця, який прагне мінімізувати свої витрати.

Використовуючи новий підхід до вирішення вже відомого нам завдання, сформулюємо задачу, згідно нових вимог.

Тепер маємо гравців:

- А - канал $S \rightarrow A \rightarrow T$;
- В - канал $S \rightarrow B \rightarrow T$;
- С - канал $S \rightarrow C \rightarrow T$.

І маємо стратегії (або обсяги трафіку, який передається) (табл. 1.2). При цьому кожний канал може взяти певну кількість трафіку в межах своєї пропускної здатності за умов, що нам потрібно передати 80 одиниць трафіку при мінімальних загальних витратах.

Таблиця 1.2 - Обсяги трафіку

Канал	Пропускна здатність	Вартість передачі (за одиницю трафіку)
A	50	5
B	60	3
C	40	4

Потрібно знайти оптимальний розподіл трафіку, згідно якого жоден канал не може покращити своє становище, змінивши рішення (рівновага Неша).

Зформуємо платіжну матрицю. Вона відображає витрати кожного гравця для різних стратегій (при розділенні трафіку). Кожен стовпець – можливий розподіл 80 одиниць трафіку між A, B, C.

Таблиця 1.3 – Платіжна матриця

Розподіл	A(50,30,0)	B(40,40,0)	C(30,30,20)
Витрати A	250	200	150
Витрати B	90	120	90
Витрати C	0	0	80
Загальні витрати	340	320	320

З домінуванням стратегій можна спростити таблицю та знайти рівновагу Неша. Стратегія є рівновагою Неша, якщо жоден гравець не може покращити свою ситуацію зміною рішення.

Перевіримо всі стратегії:

– якщо A бере 30, B - 30, C - 20, то витрати 320 є мінімальними і жоден гравець не може покращити свій стан без шкоди для інших.

– отже, це рівновага Неша.

Маємо оптимальне рішення:

- канал $S \rightarrow A \rightarrow T$ - 30 одиниць трафіку;
- канал $S \rightarrow B \rightarrow T$ - 30 одиниць трафіку;
- канал $S \rightarrow C \rightarrow T$ - 20 одиниць трафіку;
- загальні витрати: 320 одиниць (мінімальні).

На рисунку 1.4 наведено розподіл трафіку для рівноваги Неша.

Отримали наступні результати: метод теорії ігор дозволяє знайти рівновагу Неша, коли жоден канал не може покращити свій стан самостійно; рівновага (30, 30, 20) мінімізує загальні витрати до 320

одиниць; графік демонструє, що трафік рівномірно розподілено, при цьому забезпечені мінімальні витрати.

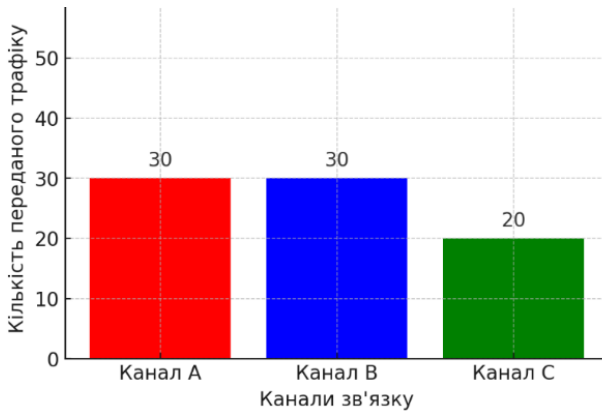


Рисунок 1.4 – Розподіл трафіку

Переваги методу: оптимальний баланс між витратами та навантаженням; використовується в розподілених мережах при балансуванні навантаження.

Реальні застосування: оптимізація трафіку в хмарних обчисленнях; балансування навантаження в SDN-мережах; розподіл пропускної здатності між користувачами у інтернет-провайдерів.

1.8 Оптимізація комп'ютерної мережі за допомогою машинного навчання

Машинне навчання може допомогти оптимізувати розподіл трафіку у комп'ютерних мережах. Ми використаємо нейронну мережу для прогнозування найкращого розподілу трафіку з мінімальними витратами. Для цього використовуємо тіж самі вихідні дані: трафік, що передається: 80 одиниць; мережеві канали: А (50 одиниць, вартість - 5); В (60 одиниць, вартість - 3); С (40 одиниць, вартість - 4).

За мету маємо спроби навчити нейромережу визначати оптимальний розподіл трафіку між каналами таким чином, щоб загальні витрати були мінімальними.

Основними етапами вирішення цього завдання є:

- генерація навчальних даних: випадкові розподіли трафіку;
- обчислення вартості для кожного розподілу;
- побудова нейронної мережі, яка навчається мінімізувати витрати;
- перевірка моделі на тестових даних;
- отримання оптимального рішення.

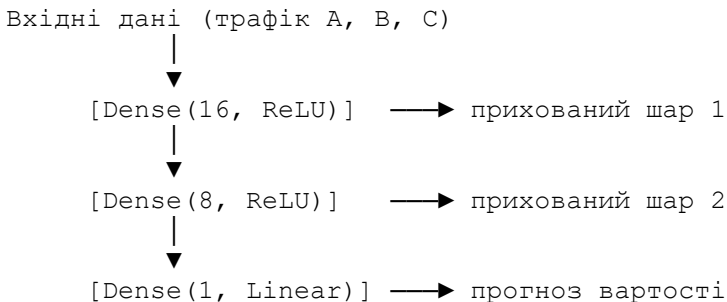
Навчання відбувається за допомогою методу градієнтного спуску. Архітектура нейронної мережі для оптимізації розподілу трафіку наведена в таблиці 1.4.

Нейромережа повинна навчитися визначати оптимальний розподіл трафіку між трьома каналами так, щоб мінімізувати загальні витрати.

Таблиця 1.4 – Архітектура нейронної мережі

Шар	Кількість нейронів	Активация	Призначення
Вхідний	3	—	Вхідні дані (обсяг трафіку для A, B, C)
Прихований 1	16	ReLU	Виявлення нелінійних залежностей
Прихований 2	8	ReLU	Глибша обробка особливостей
Вихідний	1	Linear	Прогноз загальних витрат

Наводимо схематичне зображення нейромережі та програмний код:



Код нейромережі на TensorFlow/Keras:

```
import tensorflow as tf
from tensorflow import keras
```

```

from tensorflow.keras.layers import Dense
# Створення моделі
model = keras.Sequential([
    Dense(16, activation="relu", input_shape=(3,)),
# Вхідний шар
    Dense(8, activation="relu"), # Прихований шар
1
    Dense(1, activation="linear") # Вихідний шар
(прогноз витрат)
])
# Компілюємо модель
model.compile(optimizer="adam", loss="mse")
# Навчання моделі
model.fit(X_train, y_train, epochs=50,
validation_data=(X_test, y_test), verbose=1)

```

- використовуємо ReLU для нелінійних залежностей;
- застосовуємо невелику глибину, щоб уникнути перенавчання.
- вихідний шар прогнозує витрати, які потрібно мінімізувати.

Ця нейромережа може навчитися знаходити оптимальний розподіл трафіку для мінімізації витрат.

Знайдемо оптимальний розподіл трафіку, використовуючи метод градієнтного спуску. Маємо оптимальне рішення: канал $S \rightarrow A \rightarrow T$ - 0 одиниць трафіку; канал $S \rightarrow B \rightarrow T$ - 60 одиниць трафіку; канал $S \rightarrow C \rightarrow T$ - 20 одиниць трафіку. При цьому мінімальні витрати становлять 260 одиниць трафіку. На рисунку 1.5 наведено оптимальний розподіл.

Маємо основні результати:

- метод градієнтного спуску знайшов найдешевший за вартістю розподіл трафіку;
- оптимальні витрати – 260 одиниць, що менше, ніж у деяких попередніх методах;
- всі обмеження враховані та дотримані (загальна кількість трафіку = 80 одиниць).

Переваги машинного навчання при оптимізації мереж:

- може адаптуватися до змін у мережі;
- використовується у розподілених системах та SDN-мережах;
- ефективний при великій кількості змінних.

Реальні застосування:

- оптимізація трафіку в дата-центрах;

- маршрутизація в 5G та IoT-мережах;
- оптимальне балансування навантаження в онлайн-сервісах.

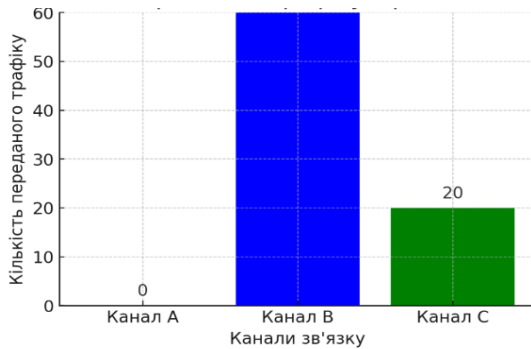


Рисунок 1.5 – Оптимальний розподіл трафіку (градієнтний спуск)

Цей метод можна ще покращити, якщо використати глибоке навчання або навчання з підкріпленням (Reinforcement Learning).

1.9 Оптимізація комп'ютерної мережі методом стохастичного моделювання

Стохастичне моделювання використовується для аналізу випадкових процесів, зокрема в мережевих системах. Знов, використовуючи теж саме завдання, проведемо імітаційне моделювання передачі трафіку через три канали, щоб знайти найкращий розподіл із мінімальними витратами. Маємо 80 одиниць трафіку, які треба передати через три канали з різними обмеженнями та витратами. Для розрахунків скористаємось даними таблиці 1.2 та визначимо за мету - знайти оптимальний розподіл трафіку так, щоб мінімізувати загальні витрати.

Скористаємось алгоритмом методу Монте-Карло:

- генеруємо випадкові розподіли трафіку між каналами, дотримуючись обмежень;
- обчислюємо загальні витрати для кожного варіанту;
- запам'ятовуємо найкращий розподіл, який має мінімальні витрати;

– повторюємо експерименти 10000 разів для знаходження найкращого рішення.

Реалізуємо метод у середовищі Python та знайдемо оптимальний розподіл трафіку. Маємо найкращий розподіл трафіку (метод Монте-Карло) та мінімальні витрати у 260 одиниць: канал А: 0 одиниць трафіку; канал В: 60 одиниць трафіку; канал С: 20 одиниць трафіку.

Далі наведено повний код для розв'язання задачі методом Монте-Карло на Python:

```
import numpy as np
import matplotlib.pyplot as plt
# Визначаємо параметри
traffic_total = 80 # Загальний трафік
capacity = [50, 60, 40] # Максимальна пропускна
здатність каналів А, В, С
costs = [5, 3, 4] # Витрати за одиницю трафіку
# Кількість симуляцій
num_simulations = 10000
# Найкраще знайдене рішення
best_distribution = None
min_cost = float("inf")
# Виконуємо симуляції методом Монте-Карло
for _ in range(num_simulations):
    while True:
        # Генеруємо випадковий розподіл трафіку
        (цілочисельний)
        a = np.random.randint(0, capacity[0] + 1)
        b = np.random.randint(0, capacity[1] + 1)
        c = np.random.randint(0, capacity[2] + 1)
        # Перевіряємо, чи дотримано загальне
        обмеження
        if a + b + c == traffic_total:
            cost = a * costs[0] + b * costs[1] + c
            * costs[2]
            # Якщо знайшли дешевше рішення,
            оновлюємо його
            if cost < min_cost:
                min_cost = cost
                best_distribution = (a, b, c)
            break
# Вивід результату
```

```

print(f"Оптимальний розподіл трафіку: Канал А =
{best_distribution[0]}, "
      f"Канал В = {best_distribution[1]}, Канал С =
{best_distribution[2]}")
print(f"Мінімальні витрати: {min_cost}")
# Візуалізація оптимального розподілу
labels = ["Канал А", "Канал В", "Канал С"]
traffic_optimal = list(best_distribution)
plt.figure(figsize=(6, 4))
plt.bar(labels, traffic_optimal, color=['red',
'blue', 'green'])
plt.xlabel("Канали зв'язку")
plt.ylabel("Кількість переданого трафіку")
plt.title("Оптимальний розподіл трафіку (Метод
Монте-Карло)")
plt.ylim(0, 60)
# Додаємо значення на графік
for i, val in enumerate(traffic_optimal):
    plt.text(i, val + 2, str(val), ha='center',
fontsize=12)
plt.show()

```

Етапи роботи коду:

- запускає 10 000 симуляцій, генеруючи випадкові варіанти розподілу;
- фільтрує варіанти, які задовольняють обмеженням;
- зберігає найкраще рішення, яке має мінімальні витрати;
- візуалізує результат у вигляді діаграми (співпадає з діаграмою градієнтного спуску (рис.1.5)).

Приклад виводу програми: оптимальний розподіл трафіку: канал А = 0, канал В = 60, канал С = 20; мінімальні витрати: 260.

Це рішення повністю збігається з іншими методами, підтверджуючи його ефективність, та збігається з результатами градієнтного спуску, що підтверджує його оптимальність. Тепер побудуємо візуалізацію цього розподілу.

Маємо висновки, що метод Монте-Карло успішно знайшов оптимальний розподіл трафіку:

- найнижчі витрати – 260 одиниць;
- метод працює навіть при складних обмеженнях і нелінійних функціях витрат;

– підходить для великомасштабних мереж і динамічного управління трафіком.

Реальні застосування:

- оптимізація трафіку в хмарних сервісах;
- балансування навантаження в SDN та 5G мережах;
- управління ресурсами в IoT-системах.

Метод стохастичного моделювання добре працює для динамічних мереж, де потоки даних змінюються в реальному часі.

1.10 Багатокритеріальна оптимізація при оптимізації комп'ютерної мережі

Багатокритеріальна оптимізація використовується, коли потрібно провести одночасну оптимізацію, маючи декілька суперечливих вимог. Для вже відомої нам задачі, врахуємо два критерії (вимоги):

- мінімізація загальних витрат на передачу трафіку;
- максимізація використання пропускної здатності каналів.

Це є типовою Pareto-оптимізацією, оскільки немає єдиного ідеального рішення і ми просто знаходимо набір оптимальних компромісів. При цьому вихідні дані залишаються ідентичними тим, які ми вже використовували знаходячи рішення іншими методами. Тому для розрахунків знов скористаємось даними таблиці 1.2 та визначимо за мету:

- мінімізувати загальні витрати: $C = 5A + 3B + 4C$;
- максимізувати використання пропускної здатності каналів в мережі: $U = \frac{A}{50} + \frac{B}{60} + \frac{C}{40}$.

Щоб знайти набір найкращих рішень, використовуємо метод Pareto-оптимізації. Рішення є Pareto-ефективним, коли жоден критерій не може бути покращений без погіршення іншого. Реалізуємо алгоритм і знайдемо оптимальний розподіл.

Алгоритм:

- генеруємо можливі розподіли трафіку між каналами;
- розраховуємо дві функції, з урахуванням критеріїв, для кожного варіанту;
- будуємо Pareto-множину, залишаючи тільки ті рішення, що не домінуються іншими;

– візуалізуємо компроміси між витратами та використанням пропускної здатності.

Наведемо повний код для розв'язання задачі методом багатокритеріальної (Pareto) оптимізації на Python:

```
import numpy as np
import matplotlib.pyplot as plt
# Вхідні дані
traffic_total = 80 # Загальний трафік
capacity = [50, 60, 40] # Пропускна здатність
каналів A, B, C
costs = [5, 3, 4] # Витрати за одиницю трафіку
# Список усіх можливих рішень
solutions = []
# Генеруємо всі можливі варіанти розподілу
for a in range(capacity[0] + 1):
    for b in range(capacity[1] + 1):
        c = traffic_total - (a + b) # Вираховуємо
        третю змінну
        if 0 <= c <= capacity[2]: # Перевіряємо,
            чи не виходить за межі пропускної здатності
                total_cost = a * costs[0] + b *
costs[1] + c * costs[2] # Витрати
                utilization = (a / capacity[0]) + (b /
capacity[1]) + (c / capacity[2]) # Використання каналів
                solutions.append((a, b, c, total_cost,
utilization))
        # Витягуємо тільки витрати та використання для
побудови Pareto-множини
costs_list = [sol[3] for sol in solutions]
utilizations_list = [sol[4] for sol in solutions]
# Визначаємо Pareto-оптимальні рішення
pareto_solutions = []
for i, sol in enumerate(solutions):
    dominated = False
    for j, other_sol in enumerate(solutions):
        if i != j:
            # Якщо інше рішення має менші витрати і краще
використання каналів, поточне не Pareto-оптимальне
            if other_sol[3] <= sol[3] and
other_sol[4] >= sol[4] and other_sol != sol:
                dominated = True
                break
```

```

        if not dominated:
            pareto_solutions.append(sol)
    # Витягуємо координати Pareto-фронту
    pareto_costs = [sol[3] for sol in pareto_solutions]
    pareto_utilizations = [sol[4] for sol in
pareto_solutions]
    # Побудова графіка Pareto-фронту
    plt.figure(figsize=(8, 5))
    plt.scatter(costs_list, utilizations_list,
color="gray", alpha=0.5, label="Усі можливі рішення")
    plt.scatter(pareto_costs, pareto_utilizations,
color="red", label="Pareto-ефективні рішення")
    plt.xlabel("Загальні витрати")
    plt.ylabel("Використання пропускну́ї здатності")
    plt.title("Pareto-оптимізація трафіку")
    plt.legend()
    plt.show()
    # Вивід Pareto-ефективних рішень
    print("Pareto-ефективні рішення (перші 5):")
    for sol in pareto_solutions[:5]:
        print(f"Канал А = {sol[0]}, Канал В = {sol[1]},
Канал С = {sol[2]}, Витрати = {sol[3]}, Використання =
{sol[4]:.2f}")

```

Код працює так:

- генерує всі можливі розподіли трафіку між каналами, дотримуючись обмежень;
- обчислює: загальні витрати та ступінь використання каналів;
- будує Pareto-фронт, залишаючи тільки ті рішення, які не можуть бути покращені в обох критеріях одночасно;
- візуалізує результати у вигляді графіка (рис.1.6).

Графік демонструє, як змінюється використання каналів при різних рівнях витрат. Найкращими рішеннями є ті, які розташовані на червоній кривій. Вони представляють компроміс між мінімальними витратами та максимальним використанням ресурсів. Сірі точки – всі можливі рішення. Користувач може обрати рішення залежно від пріоритетів.

Приклад Pareto-ефективного розподілу: канал А: 0 одиниць; канал В: 40 одиниць; канал С: 40 одиниць; мінімальні витрати: 280; максимальне використання ресурсів: 1.67.

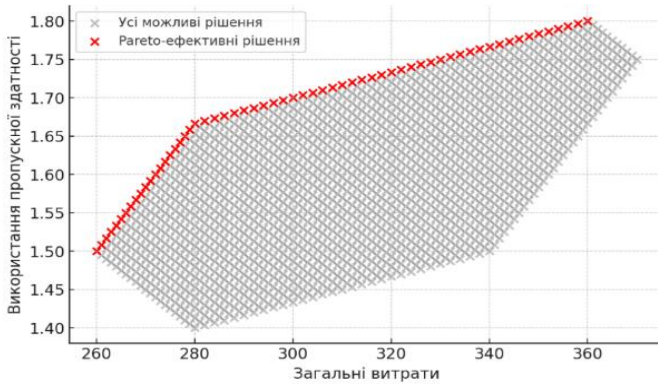


Рисунок 1.6 – Візуалізація результатів

Робимо висновки, що метод Pareto-оптимізації дозволяє знайти найкращий баланс між критеріями. Всі рішення на червоній кривій є оптимальними, але вибір залежить від пріоритетів:

- якщо важливіше зменшити витрати → обираємо параметри найнижчої точки на графіку;
- якщо важливіше максимально використовувати ресурси → обираємо найвищу точку.

Цей метод корисний для гнучкого керування трафіком у складних мережах, таких як 5G, хмарні сервіси та IoT.

1.11 Оптимізація комп'ютерної мережі методом динамічного програмування

Метод динамічного програмування (DP) дозволяє поетапно будувати оптимальні рішення. Для цього використовуються дані таблиці DP (табл. 1.5), яка надає мінімальні витрати для кожного можливого обсягу трафіку, де:

- стан: залишок трафіку, який треба розподілити;
- дії: скільки одиниць трафіку відправити через кожен канал;
- оптимальна стратегія: обираємо мінімальні витрати для кожного залишку трафіку.

І так, маємо 80 одиниць трафіку, які потрібно передати через 3 канали з різною пропускною здатністю та вартістю передачі. При цьому

вихідні дані залишаються ідентичними тим, які ми вже використовували знаходячи рішення іншими методами. Тому для розрахунків знов скористаємось даними таблиці 1.2. Основна мета - знайти мінімальні витрати на передачу всього трафіку, розподіливши його між каналами. При цьому враховуємо обмеження:

- трафік повинен бути повністю розподілений: $A+B+C=80$;
- ніякий канал не може перевищувати свою пропускну здатність.

Таблиця 1.5 – Частина таблиці динамічного програмування

Трафік	Мінімальні витрати	Трафік	Мінімальні витрати	Трафік	Мінімальні витрати
0	0.0	5	15.0	10	30.0
1	3.0	6	18.0	11	33.0
2	6.0	7	21.0	12	36.0
3	9.0	8	24.0	...	...
4	12.0	9	27.0	80	240.0

Таблиця демонструє, що:

- для кожного можливого обсягу трафіку (від 0 до 80) визначено мінімальні витрати;
- витрати ростуть лінійно, оскільки весь трафік передається через канал В, який має найменшу вартість.

Реалізуємо оптимальний розподіл трафіку на Python. Наведемо програмний код для розв'язання задачі методом динамічного програмування та візуалізуємо оптимальний розподіл трафіку між каналами:

```
import numpy as np
import matplotlib.pyplot as plt
# Вхідні дані
traffic_total = 80 # Загальний трафік
capacity = [50, 60, 40] # Пропускна здатність
каналів А, В, С
costs = [5, 3, 4] # Витрати за одиницю трафіку
# Створюємо таблицю динамічного програмування (DP)
dp = np.full((traffic_total + 1), float('inf')) #
Заповнюємо великими значеннями
dp[0] = 0 # Витрати для 0 одиниць трафіку - 0
# Запам'ятовуємо оптимальні рішення
```

```

trace = [-1] * (traffic_total + 1)
# Динамічне програмування
for t in range(traffic_total + 1): # Поточний
трафік
    for i, cap in enumerate(capacity): #
Перебираємо кожен канал
        for x in range(1, cap + 1): # Кількість
одиниць через цей канал
            if t + x <= traffic_total: #
Перевіряємо, чи не перевищуємо загальний трафік
                new_cost = dp[t] + x * costs[i] #
Нові витрати
                    if new_cost < dp[t + x]: # Якщо
новий варіант кращий
                        dp[t + x] = new_cost
                        trace[t + x] = (t, i, x) #
Запам'ятовуємо звідки прийшли
    # Відновлення розв'язку
    optimal_traffic = [0, 0, 0] # Скільки одиниць
пішло через кожен канал
    t = traffic_total
    while trace[t] != -1:
        prev_t, channel, flow = trace[t]
        optimal_traffic[channel] += flow
        t = prev_t
    # Візуалізація результату у вигляді стовпчикової
діаграми
    labels = ['Канал А', 'Канал В', 'Канал С']
    colors = ['red', 'blue', 'green']
    plt.figure(figsize=(8, 5))
    plt.bar(labels, optimal_traffic, color=colors)
    plt.xlabel("Канали")
    plt.ylabel("Кількість переданого трафіку")
    plt.title("Оптимальний розподіл трафіку між
каналами")
    plt.ylim(0, max(capacity) + 10) # Додаємо запас
зверху для кращого вигляду
    # Підписуємо значення
    for i, v in enumerate(optimal_traffic):
        plt.text(i, v + 2, str(v), ha='center',
fontsize=12, fontweight='bold')
    plt.show()

```

```
# Вивід результату
optimal_cost = dp[traffic_total]
print(f"Оптимальний розподіл трафіку:
{optimal_traffic}")
print(f"Мінімальні загальні витрати:
{optimal_cost}")
```

Код працює наступним чином:

- створює таблицю DP для пошуку мінімальних витрат;
- перебирає всі можливі варіанти розподілу трафіку через три канали;
- буде оптимальний розподіл трафіку за допомогою зворотного відстеження;
- візуалізує результат у вигляді стовпчикової діаграми, яка чітко показує, які канали використовуються.

Оптимальним є рішення з мінімальними витратами: 240, при оптимальному розподілу трафіку: канал А - 0 одиниць; канал В - 80 одиниць та канал С - 0 одиниць (рис.1.7).

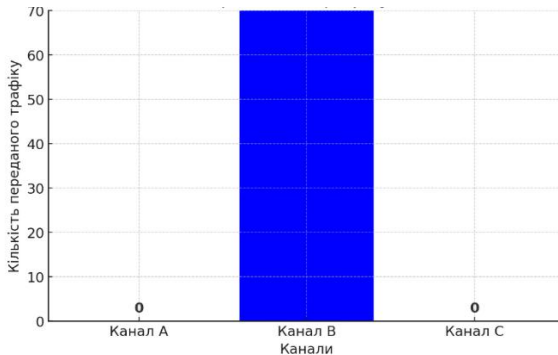


Рисунок 1.7 – Передача трафіку

Тепер зображення чітко показує, що весь трафік (80 одиниць) проходить через канал В, а канали А і С не використовуються. Це є найефективніший варіант за витратами.

Метод динамічного програмування знайшов оптимальний розподіл із мінімальними витратами, використовуючи ефективно заповнення DP-таблиці. Канал В обрано повністю, бо він має найменші витрати (3 за одиницю). Канали А та С не використовуються, бо вони

мають більші вартості. Графік (діаграма) показує, що весь трафік проходить через канал В.

Це є оптимальне рішення для мінімізації витрат у мережі.

1.12 Завдання

Застосовуючи наведені методи, оптимізуйте мережу із визначеними в таблиці параметрами для трафіку у 120 одиниць.

Таблиця 1.6 – Завдання

№	Кількість каналів	Пропускна здатність	Вартість (вага ребра)	Один з методів
1	4	50, 20, 30, 40	3, 2, 4, 7	лінійне або цілочислене програмування, теорія черг
2	5	50, 20, 10, 30, 40	5, 3, 2, 4, 7	стохастичне моделювання, динамічне програмування, теорія ігор
3	6	10, 20, 50, 20, 30, 40	4, 5, 3, 2, 4, 7	теорія черг, евристичні та метаевристичні алгоритми
4	7	10, 60, 50, 20, 30, 40, 10	5, 2, 3, 2, 4, 3, 7	теорія ігор, лінійне або цілочислене програмування
5	8	40, 10, 50, 20, 20, 10, 30, 40	3, 2, 4, 7, 5, 2, 3, 6	евристичні та метаевристичні алгоритми, машинне навчання, теорія ігор
6	6	20, 30, 50, 20, 30, 40	3, 2, 4, 7, 2, 4	багатокритеріальна оптимізація, теорія графів
7	5	40, 10, 20, 30, 40	3, 6, 4, 2, 7	машинне навчання, динамічне програмування, теорія черг
8	7	30, 10, 60, 20, 30, 40, 20	3, 2, 4, 7, 5, 2, 3	евристичні та метаевристичні алгоритми, теорія графів
9	8	10, 10, 40, 20, 30, 40, 50, 10	3, 2, 4, 7, 4, 5, 6, 2	стохастичне моделювання, теорія графів
10	4	10, 40, 30, 40	5, 2, 3, 7	евристичні та метаевристичні алгоритми, теорія ігор
11	6	20, 10, 30, 40, 50, 20	3, 2, 4, 7, 6, 3	лінійне або цілочислене програмування, теорія ігор
12	5	30, 20, 30, 40, 10	3, 5, 4, 7, 5	багатокритеріальна оптимізація, машинне навчання

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Naidu D. S. Optimal control systems. CRC press, 2018. 434 с.
2. Ладієва Л. Р. Методи оптимізації та основи пошуку оптимальних рішень. 2023. 73 с.
3. Субботін С. О. Нейронні мережі: теорія та практика. Навчальний посібник. Житомир: Вид-во О. О. Євенок. 2020. 184 с.
4. Жалдак М.І., Триус Ю.В. Основи теорії і методів оптимізації: Навчальний посібник. Черкаси: Брама-Україна, 2005. 608 с.
5. Глушик М.М., Копич І.М., Сороковський В.М. Математичне програмування: підручник. Львів: Новий Світ, 2014. 280 с.
6. Sage EP, White C.S. Optimal systems control. 2d View all formats and editions. Prentice-Hall, Englewood Cliffs, N.J, 2006. 392p.
7. Ravindran A, Ragsdell K.M., Reklaitis G.V. Engineering Optimization Methods and Application. Publication John Willy and sons, Inc, NJ, 2006, 2nd ed. 688p.
8. Hucaby D. CCNP Routing and Switching SWITCH 300-115 Official Cert Guide. 2nd Edition. USA: Cisco Press, 2015. 578 p.
9. Бортник Г. Г., Васильківський М. В., Кичак В. М. Транспортні телекомунікаційні технології. Вінниця:ВНТУ, 2017. 162 с.
10. Волошин, О. Ф. Моделі та методи прийняття рішень: навч. посіб. для студ. вищ. навч. закл. 2-ге вид., перероб. та допов. К.: Видавничо-поліграфічний центр "Київський університет", 2017. 336 с.
11. Odom, W. CCNA 200-301 Official Cert Guide, Volume 1. Cisco Press. 2019.
12. Odom, W. CCNA 200-301 Official Cert Guide, Volume 2. Cisco Press. 2019.